

# MDA for SoC Design, Intensive Signal Processing Experiment\*

Pierre Boulet, Jean-Luc Dekeyser, Cédric Dumoulin, Philippe Marquet  
Laboratoire d'Informatique Fondamentale de Lille  
Université des Sciences et Technologies de Lille  
France

## Abstract

The development of embedded applications is very difficult. Several different languages are usually used to specify different parts of the application or of the hardware. Dealing with so many languages can be daunting. A separation of the preoccupations: application, hardware architecture, association between them and the simulation or execution technologies are keys to efficient co-design of embedded applications. The Model Driven Architecture can be used to better deal with the reuse of parts of the design and the interoperability between both the implementation technologies and the various simulation levels.

We propose a construction of metamodels to support a co-design methodology. This construction will be experimented on intensive signal processing application co-design to justify the adequacy of this methodology to usual industrial development techniques.

## 1 Introduction

Because of the vast scope of the encountered problems, of the quick evolution of the architectures, we observe a very great diversity as regards the programming languages. Ten years ago each new proposed model (for example within the framework of a PhD) led to the implementation of this model in a new language or at least in an extension of a standard language. Thus 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 architecture model to the application specification model, and to introduce as well a relationship between them. These two models are practically always different, they are often expressed in two different languages.

From this experience, one can derive some principles for the design of embedded application development environments:

- To refrain from designing programming languages to express the two different models, application and 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.

---

\*This work has been supported by the ITEA 99038 project, Sophocles.

- 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.

We believe that the Model Driven Architecture [MM03, Boa01] can enable us to conceive 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.

In this paper, we will first present the Model Driven Architecture in section 2. Then, with some justifications (section 3), thanks to the MDA, we will propose a “Y” construction of our metamodels to support a co-design methodology in section 4. Finally an experimentation of this construction applied to intensive signal processing will justify the adequacy of this methodology to usual industrial development techniques in section 5.

## 2 Model Driven Architecture

The Model Driven Architecture (MDA) [MM03, Boa01] is the OMG proposed approach for system development. It is primarily focused on software development, but can be applied to any system development. 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.

### 2.1 Models and Compilation Techniques

Model transformation is not new. The traditional compilation chain is based on model transformations; a compiler is a translator: A compiler accepts as input a program in some language and produces as output a program in an other language. This translation preserves the “meaning” of the program. That meaning refers to the semantics of the program in the language it is expressed in. This language is described in the language reference manual. This description is actually a model of the language. Despite the language structure may be formally given in a grammar, the model of the language is not formalized. Thus, if the compiler construction may be derived from the grammar with the help of syntactic tools such as the well known Lex & Yacc [LMB92], the rest of this construction is handmade by a specialist who ensures the respect of the language semantics and the validity of the translation process.

In the traditional approach, the implicit model is used to write the compiler. The largest drawback of this approach is the impossibility of automatic compiler/translator construction. The definition of a new language implies the elaboration of the corresponding new implicit model and the construction of a new compiler, even if the use of intermediate languages attempts to alleviate the cost of evolution (a new model only needs a new translation to the intermediate language) and retargetability (a new target only needs a new translation from the intermediate language).

For sure, the definition of transformation tools able to handle whatever model is out of the scope of our research project. Because the MDA approach targets in essence a specific domain, we are convinced of the viability of automatic model transformations in the frame of domain-specific models and languages. This approach is coherent with the Generative Programming [CE00] methodology. Some works have already demonstrated graph to graph transformations parameterized by the input and output graph models (the metamodels) and a set of transformation specifications [ALS<sup>+</sup>02].

## 2.2 MDA Focus

The MDA goals are: to increase the reuse of existing developments; to reduce the time of new developments; to perenize current and future developments; to ease the integration of new technologies with long proven business models. To achieve these goals, the MDA approach promote a clear separation of the fundamental logic of the specification from the particular technologies that implement it. This allows to build business models that can be implemented across different existing platforms and rapidly deployed on new emerging technologies.

According to the MDA board ORMSC [Boa01],

*“The MDA separates the models of the system into Platform Independent Models (PIMs), and Platform Specific Models (PSMs). How the functionality specified in a PIM is realized is specified in a platform-specific way in the PSM”.*

Going from one model to another is done via some transformations. These transformations can be done manually, or automatically with the help of the so called mapping rules (see section 2.4).

## 2.3 Model and Metamodel

A model is expressed with the help of a language, like UML, which can be visual or textual. A language is itself described in a metamodel defining the available elements and the construction rules.

In the MDA, several models are used simultaneously, each one being described by its own metamodel. Models and metamodels are usually represented using the UML standard.

The MDA board defines platform as follows:

*“A platform is the specification of an execution environment for models. The term platform is used to refer to technological and engineering details that are irrelevant to the fundamental functionality of a system.”*

Thus a system or application that is described at the *Platform Independent Level* can be mapped to several *Platform Specific Models*, each one representing a different technological implementation.

## 2.4 Transformations and Mappings

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 metamodel, on which we can define some mapping rules describing how concepts from one metamodel are to be mapped on the other metamodel. From these mapping rules we deduce the transformations between any models conforming to the metamodels.

## 2.5 Use of Standards

The MDA is based on proven standards: UML for modeling and the MOF for metamodel expression. The new coming UML 2.0 [Obj03] standard is specifically designed to be used with the MDA. It removes some ambiguities found in its predecessors (UML 1.x), allows more precise descriptions and opens the road to automatic exploitation of models. The MOF (Meta Object Facilities [Obj00]) is oriented to the metamodel specifications. It groups related specifications under its umbrella:

- The MOF language, which is a meta-metamodel used to describe metamodels. A UML profile exists for the MOF, enabling the description of metamodels with UML.
- The XMI [Obj02] production rules, describing how a metamodel can be saved in the XMI format, and thus how models can be saved in XMI (a particular XML schema).
- The IDL production rules, describing how to generate IDL interfaces allowing the manipulation of the metamodel and of the corresponding models.
- The JMI specification (Java Metadata Interface [Dir01]), describing how to generate Java interfaces and implementations to manipulate the metamodel and the corresponding models. The generated interfaces handle the import and export of models in XMI.

### 3 System-on-Chip Design

SoC (System-on-Chip) 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/instruction), 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.

#### 3.1 Current Practice: The “Y-chart”

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. In the “Y-chart” [GK83] approach, three domains are identified:

- Functional domain: algorithms, flowcharts, functional components.
- Structural domain: processors, memories, busses.
- Physical domain: delays, power consumption, hardware resources, real-time and embedding constraints.

The authors propose to specify specific information in each of these three domains. The design activities match a successive refinement process between each domain according to various abstraction levels. The design flows from the functional domain, to the structural domain, finally to the physical domain, and so on while going down in the abstraction levels.

#### 3.2 Intellectual Properties and Reuse

IPs are block-oriented instantiations of functional or physical units. They are especially designed for reuse. A few characteristics make this possible (standard interface, formal verification, documentation, availability of different abstraction levels). In a codesign system, functional IPs should be included in the functional domain. Physical IPs are used to specify the SoC architecture. Some mapping constraints could appear due to their usage.

## 4 MDA Approach for “Y” Design

Our proposal is partially based upon the concepts of the “Y-chart”. 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, and to ensure interoperability between the different abstraction levels used in a same codesign.

### 4.1 Metamodels for the “Y” Design

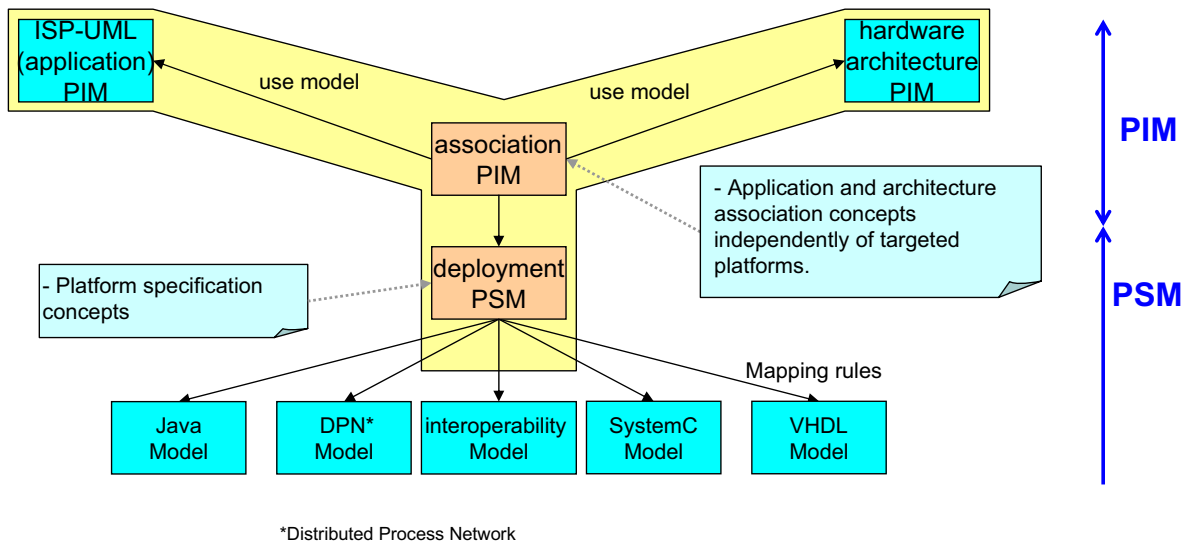


Figure 1: Overview of the metamodels for the “Y” design

We preserve the two following models: application and hardware architecture (functional and structural domains) and we characterize them by different metamodels. Some concepts from these two metamodels are similar in order to simplify their understanding and use. Models for application and hardware architecture are 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.

### 4.2 PIMs and PSMs

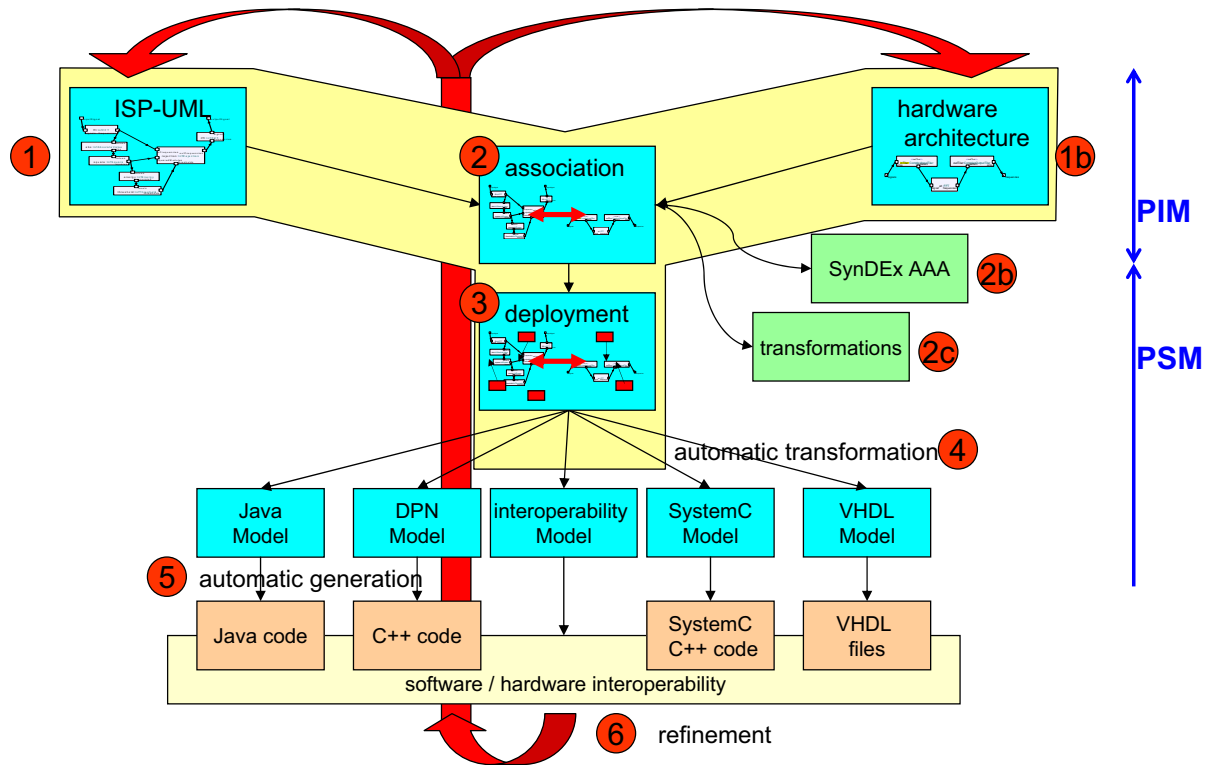
All the previously defined models are platform independent. No component is associated with an execution, simulation or synthesis technology. Such an association targets a given technology (Java, SystemC RTL, SystemC TLM, VHDL, etc). Once all the components are associated with some technology, the deployment is realized.

The diversity of the technologies requires interoperability between abstraction levels and simulation and execution 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, the hardware architecture, the association or the deployment models (see figure 2).



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

Figure 2: Overview of a possible methodology

## 5 Intensive Signal Processing Experiment

We have started implementing our ideas on MDA for SoC design in the particular case of intensive signal processing applications. The application, hardware architecture and association PIMs are under development using UML profiles. We will sketch below the main ideas of these models.

### 5.1 Application PIM

The application metamodel describes a mean to express data dependences. It is based on the ISP UML profile described in [DBDM03]. ISP UML allows the expression of both task parallelism

and data parallelism. A main characteristic of this model is the single assignment form. Thus the time dimension is explicit in the data structures (arrays) and can be infinite.

Modeling is component based. The component models some computation and the input and output capabilities via ports. Those components can be composed, data parallel or elementary.

- A compound component expresses *task parallelism* by the way of a component graph. The edges of this graph are directed and represent data dependences.
- A data parallel component expresses *data parallelism* by the way of the parallel repetition of an inner component on patterns of the input arrays producing patterns of the output arrays. Some rules must be respected to describe this repetition. In particular, the output patterns must tile exactly the output arrays.
- An elementary component is the basic computation unit of the application and has to be defined for each target technology.

This hierarchical description allows to consider the application with different granularities. Indeed, the data dependences expressed at one level are approximations of the real data dependences described at the deepest level of the hierarchy.

## 5.2 Hardware Architecture PIM

The hardware architecture metamodel, though not completely finalized yet, exhibits the same modeling elements as the application metamodel: components, ports and links between ports of components. These modeled physical components are either active components such as processors or direct memory accessors; passive components such as random access memories or sequential access memories; buses; or even compound components. The ports model the communication endpoints of the components and the links between ports model the data paths.

Our description is inherently hierarchical. We introduce a repetitive hierarchy (similar to the data parallel components of the application metamodel) to deal with multiprocessors or memory banks, for example. In the same way as in the pattern extraction of the application metamodel, we are able to describe most of the regular interconnections between the repeated components. As in the application metamodel, the hierarchy allows to view the hardware architecture with several granularities, exposing or not its inner details.

## 5.3 Association PIM

The association metamodel is currently under development. It will allow to express the mapping and the scheduling of an application on a hardware architecture. It imports the two metamodels used to model these application and hardware architecture. It adds an association view to express the mapping by linking the components of the application with active components of the hardware architecture and the ports and dependences of the application model with the passive hardware elements and the buses.

The data parallelism in the application metamodel and the repetitive hardware components can be associated to define parallel schedules. This association is independent (it really is at the PIM level) of the abstraction levels used for simulation or execution.

# 6 Conclusion and Perspectives

We have proposed in this paper to use the Model Driven Architecture to design SoCs. The MDA eases the reuse of application or hardware designs and the interoperability between different simulation levels and technologies.

The separation of the application models, of the hardware architecture models and of the association models allows to follow the current practice of SoC design (the “Y” design). These tree metamodels are PIMs, independent of the technologies used for simulation or execution. The specification of these technologies is done in the deployment metamodel. From this PSM to the execution or simulation code generation, mapping rules allow automation.

To validate our approach, we have sketched application, hardware and association metamodels dedicated to intensive signal processing. To further the experiment, we are studying SystemC [Ope02] PSMs for intensive signal processing. There will be a metamodel for TLM (Transaction Level Model) simulation, another metamodel for RTL (Register Transfer Level) and a third one to model the interoperability between the different simulation levels. We are specifying all our metamodels with the MOF and intend to write a tool [DDK<sup>+</sup>03] to express and apply mapping rules in our particular domain of application: intensive signal processing.

## References

- [ALS<sup>+</sup>02] Aditya Agrawal, Tihamer Levendovszky, Jon Sprinkle, Feng Shi, and Gabor Karsai. Generative programming via graph transformations in the model-driven architecture. In *OOP-SLA 2002 Workshop on Generative Techniques in the context of Model Driven Architecture*, November 2002.
- [Boa01] OMG Architecture Board. Model driven architecture (MDA). Technical Report ormsc/2001-07-01, OMG, 2001.
- [CE00] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.
- [DBDM03] Cédric Dumoulin, Pierre Boulet, Jean-Luc Dekeyser, and Philippe Marquet. UML 2.0 structure diagram for intensive signal processing application specification. Research Report RR-4766, INRIA, March 2003.
- [DDK<sup>+</sup>03] Cédric Dumoulin, Jean-Luc Dekeyser, Boris Kokoszko, Stéphane Pulon, and Gérard Cristau. Interoperability between design and simulation tools using model transformation techniques. In *FDL'03*, Frankfurt, September 2003. ECSI.
- [Dir01] Ravi Dirckze. JSR-000040 java metadata interface (JMI) specification 1.0. Technical report, Java Community Process, 2001.
- [GK83] D. D. Gajski and R. Kuhn. Guest editor introduction: New VLSI-tools. *IEEE Computer*, 16(12):11–14, December 1983.
- [LMB92] John Levine, Tony Mason, and Doug Brown. *Lex & Yacc*. O'Reilly & Associates, 1992.
- [MM03] Joaquin Miller and Jishnu Mukerji, editors. *MDA Guide (Draft Version 0.2)*. <http://www.omg.org/docs/ab/03-01-03.pdf>, 2003.
- [Obj00] Object Management Group, Inc. MOF meta object facility, specification, version 1.3. <http://www.omg.org/cgi-bin/doc?formal/00-04-03>, January 2000.
- [Obj02] Object Management Group, Inc., editor. *XML Metadata Interchange (XMI), Version 1.2*. <http://www.omg.org/technology/documents/formal/xmi.htm>, January 2002.
- [Obj03] Object Management Group, Inc., editor. *U2 Partners' (UML 2.0): Superstructure, 2nd revised submission*. <http://cgi.omg.org/cgi-bin/doc?ad/03-01-02/>, January 2003.
- [Ope02] Open SystemC Initiative. SystemC. <http://www.systemc.org/>, 2002.