An Entirely Model-Based Framework for Hardware Design and Simulation

Safouan Taha[1], Ansgar Radermacher[2], and Sébastien Gérard[2]

[1] SUPELEC Systems Sciences (E3S) – Computer Science Department, France
safouan.taha@supelec.fr
[2] LIST/LISE department of CEA (Commissariat à l'Energie Atomique), France
ansgar.radermacher@cea.fr, sebastien.gerard@cea.fr

Abstract. For a long time, the code generation from domain-specific and/or model-based languages to implementation ones remained manual and error-prone. The use of modeling was required in the early stages of development to ease the design and communicate intents, but because of the manual implementation, there were no traceability and no formal link with the final code. Model-Driven Development (MDD) was unable to win its audience.

Today, models constructed with UML have an equivalent representation in XML. And thanks to XML technologies, manipulating models for data mining, transformation or code generation becomes possible. MDD is now commonly used within the software community.

Next, for the hardware community, this work will empower the use of MDD in hardware design and simulation. It offers a completely operational framework based on OMG standards: UML and MARTE.

1 Introduction

The Object Management Group (OMG) standard UML (Unified Modeling Language) [6] is commonly used within the software community. UML has significantly improved efficiency in software development, thanks to several mechanisms, like generalization, composition, encapsulation, separation of concerns (structure/behavior), abstraction (different views), and refinement. UML is supported by many modeling tools.

By using hardware description languages like VHDL and SystemC, hardware design becomes a programming activity similar to the software development. That eases the hardware design and enables hardware simulation to avoid any risky implementations. But in practice, just like software, hardware programming is implementation-oriented and doesn't match the real issues of hardware design and architecture exploration.

Taking into account this analogy between hardware design and software, we developed an entire and operational framework that is completely based on models of concepts and constructs specific to the hardware domain. Such framework let the hardware designer benefit from all well-known features of DSLs and MDD. Our framework is composed of a standardized Hardware Resource Modeling (HRM) language and a powerful simulation engine.

M. Hinchey et al. (Eds.): DIPES/BICC 2010, IFIP AICT 329, pp. 31–42, 2010.

In this paper, we will first describe a modeling methodology which helps to resourcefully use HRM for building consistent models. This HRM methodology is a set of guidelines within an incremental process of successive hardware compositions. Then, we will illustrate the efficiency of such model-based framework on a large case study: we will apply the HRM methodology to create the model of a heterogeneous hardware platform and we will simulate it.

The paper is organized as follows. The next section introduces in brief the HRM profile. Section 3 describes the modeling methodology based on HRM. Section 4 explains how the simulation engine works. Where the last section depicts the whole design process on a case study.

2 Hardware Resource Model

The purpose of HRM is to adopt UML as a hardware design language to benefit from its features and tools, and to unify the (software/hardware) co-design process of embedded systems. Thanks to the UML extension mechanism, the HRM profile [8] extends UML with hardware concepts and semantics. HRM is part of the new OMG standard MARTE [7] (Modeling and Analysis of Real-Time Embedded systems). HRM is intended to serve for description of existing or for conception of new hardware platforms, through different views and detail levels. HRM covers a large scope:

Software design and allocation: The hardware designer may use a high level hardware description model of the targeted platform architecture, with only key properties of the available resources like the instruction set family, the memory size... Such abstract model is a formal alternative to block diagrams that are communicated to software teams and system architects.

Analysis: Analysis needs specialized hardware description model. The nature of details depends on the analysis focus. For example, schedulability analysis requires details on the processor throughput, memory organization and communication bandwidth, whereas power analysis will focus on power consumption, heat dissipation and the layout of the hardware components. HRM uses the UML ability to project different views of the same model.

Simulation: It is based on detailed hardware models (see section 4). The required level of detail depends on the simulation accuracy. The performance simulation needs a fine description of the processor microarchitecture and memory timings, whereas many functional simulators simply require entering the instruction set family.

HRM is grouping most of hardware concepts under a hierarchical taxonomy with several categories depending on their nature, functionality, technology and form. The HRM profile is composed of two sub-profiles, a logical profile that classifies hardware resources depending on their functional properties, and a physical one that concentrates on their physical nature. The logical and physical views are complementary. They provide two different abstractions of hardware that should be merged to obtain the whole model. Each sub-profile is, in turn, composed of many metamodels as shown in figure 1.

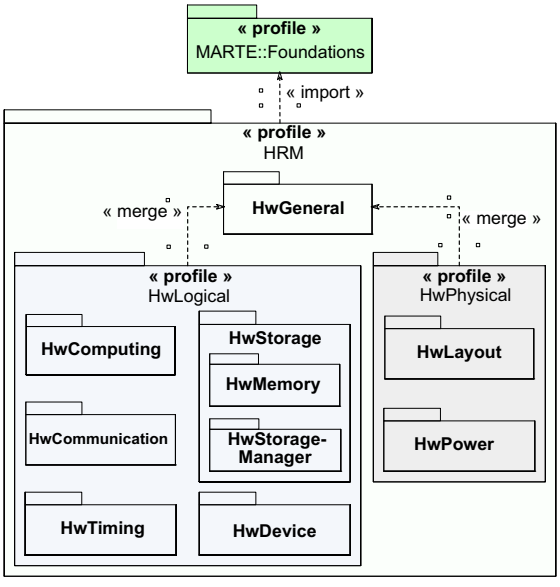

Fig. 1. HRM structure overview

Logical Model. The objective of the logical model is to provide a functional classification of hardware resources, whether if they are computing, storage, communication, timing or auxiliary devices. This classification is mainly based on services that each resource offers. As shown in figure 1, there is a specific metamodel for each hardware logical category.

HRM contains most of hardware resources thanks to a big range of stereotypes that are organized under a tree of successive inheritances from generic stereotypes to specific ones. This is the reason behind the ability of the HRM profile to cover many detail levels. For example, the *HwMemory* metamodel shown in figure 2 reveals the HRM accuracy and its layered architecture. The *HwMemory* stereotype denotes a given amount of memory. It has three attributes, *memorySize*, *addressSize* and *timings*. This latter is a datatype to annotate detailed timing durations. *HwMemory* could be an *HwProcessingMemory* symbolizing a fast and working memory, or an *HwStorageMemory* for permanent and relatively time consuming storage devices...

Physical Model. The hardware physical model represents hardware resources as physical components with physical properties. As most of embedded systems have limited area and weight, hard environmental conditions and a predetermined autonomy, this view enables layout, cost, power analysis and autonomy optimization The *HwPhysical* profile contains two metamodels: *HwLayout* and *HwPower*.

For more details on HRM, please refer to [8] and the MARTE document [7].

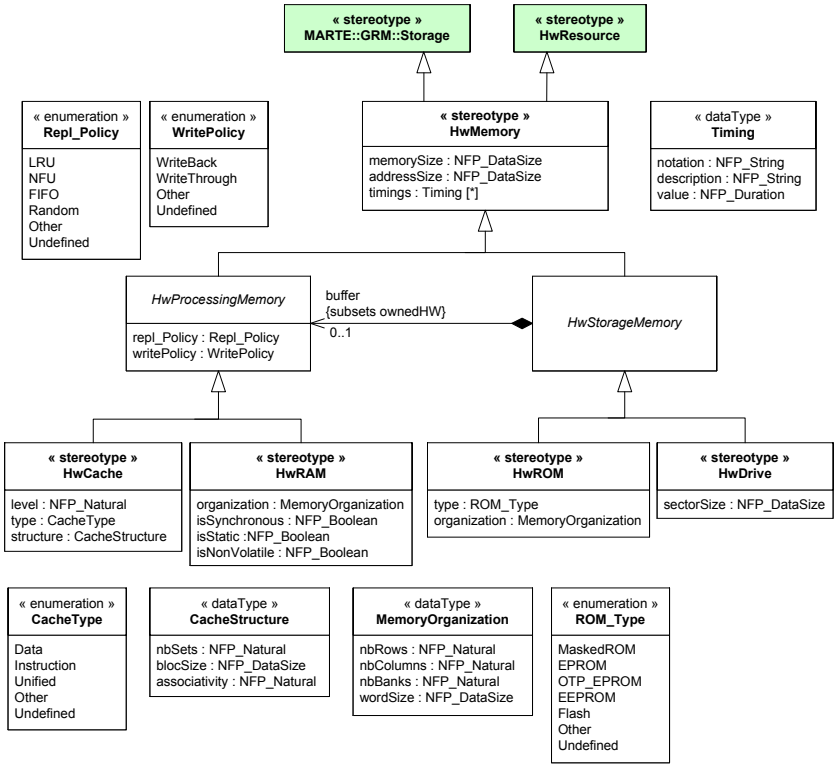

Fig. 2. HwMemory metamodel

As HRM is serialized into the OMG standard XML Metadata Interchange (XMI) [5], it can be used within most UML-based modeling tools. In our case, we use the Papyrus UML tool [3] that is developed within our laboratory (CEA LIST/LISE). Papyrus is based on the Eclipse Modeling Framework and provides a MARTE plug-in.

3 Hardware Modeling Methodology

As the HRM profile extends the generic UML kernel metaclasses, it can be used within all UML diagrams. UML offers a big amount of notations and diagrams, it also includes many variation points. Consequently, it is a common practice to adopt modeling methodologies that restrain the UML mechanisms to use, fixate their semantics and bring consistency rules. Considering that the hardware designers are not used to UML-based modeling, such a modeling methodology is quite necessary.

The HRM modeling methodology is mainly based on the UML2 Composite Structure diagram, since this latter has a clear graphical representation of

composition and supports the *Part*, *Port* and *Connector* concepts that are well-adapted to hardware modeling.

The HRM modeling methodology is iterative, one iteration corresponds to the entire modeling of only one resource from the hardware platform. Each iteration is composed of many modeling steps grouped into three phases: class definition, internal structure modeling and instantiation. Our methodology is also incremental (bottom-up), it starts from modeling elementary resources, and with successive compositions, it reaches the whole platform model.

Class Definition

1. Choose the next resource to model taking into account the incremental order (partial order) of compositions. Create the corresponding class using the resource name. Specify its inheritances from previously defined classes (from previous iterations). Notice that the inheritance mechanism is an efficient way to classify hardware resources depending on their nature.
2. Apply the HRM stereotype matching the resource type. It is a key step where we extend, in a simple manner, the UML class structure and semantics with the hardware specific ones. To avoid useless decompositions, many HRM stereotypes could be applied simultaneously if the current resource plays many roles within the hardware platform (e.g. a typical chipset is either *HwMemoryManager*, *HwBridge*, *HwArbiter*...). Furthermore, stereotypes from different profiles may also be applied if necessary. UML supports these options.
3. Assign values to some of the tag definitions (stereotype attributes), especially those that match the class level and are common to all the resources represented by the current class. For example, if the instruction set of a *HwProcessor* could be assigned at this level, its frequency or its cache size should be specified later within the steps of integration and instantiation. Notice that the HRM tag definitions are optional and they should be specified only if necessary.
4. Even if HRM is very detailed, it is a standard that mainly groups generic and common properties. Therefore, if at this stage of modeling, the hardware designer still needs to specify additional properties of the current resource, he should use UML ordinary, in this step, regardless of HRM.
   - Define specific attributes. They must be strictly typed, and for this, we can exploit the UML typing mechanisms like *DataType* or Enumeration. We can also use the MARTE library of basic types *BasicNFP_Types* or define new complex types (with physical measurements and units) thanks to the NFP profile [7] of MARTE.
   - Add associations when necessary between the current class and the previously defined ones. When an association corresponds to a hardware connection, we can apply corresponding stereotypes (*HwMedia*, *HwBus*...) on it. Notice that class compositions will be defined during the next step.

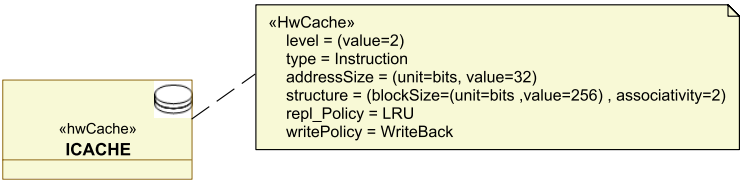

Fig. 3. ICACHE class definition

– Define operations and use the *HwResourceService* stereotype and the *providedServices* tag definition to settle if they are provided services of the current resource.

Figure 3 shows a light model of an instruction cache class that we typically obtain at the end of this first iteration phase.

Internal Structure Modeling

5. To define the internal structure of the resource being modeled, insert UML *Part*(s) typed by resources specified in previous iterations. We see here the reason behind the use of the Composite Structure diagram in our methodology and why we are following the incremental compositions order. Each *Part* has a multiplicity that is a simple and powerful mechanism for the representation of repetitive structures, very frequent in the area of hardware. Each *Part* must also display its ports, which correspond to those of its class type (see step 8).
6. Once *Part*(s) are typed as resources and taking into account their new context, it is important to reapply stereotypes and assign local values to their specific tag definitions. Indeed, if we have previously modeled resource in absolute terms regardless of its enclosing component, it should be now more specifically characterized. The hardware designer is limited to the reapplication of stereotypes previously applied to the typing class or one of its class parents. It is a rule of consistency between the nature of the resource and the roles it can play within different platforms.
7. Connect these parts by means of UML *Connector*(s) linking their ports. Such connectors must be typed by either an association defined in step 4 or a HRM meta-association. They could also be stereotyped as *HwMedia*, *HwBus* or *HwBridge* depending on their role.
8. Define boundary ports. In the UML Composite Structure diagram, a *Port* is a *Property* that is not necessarily typed but has a name and a multiplicity. Under this methodology, we require that each port and/or its class type must be stereotyped as a *HwEndPoint*.
   Use then UML *Connector*(s) to define delegations from the class ports to the ports of its subcomponents.

Figure 4 shows a model of a composite memory class that we typically obtain at the end of this second phase. It contains the ICACHE shown in figure 3.

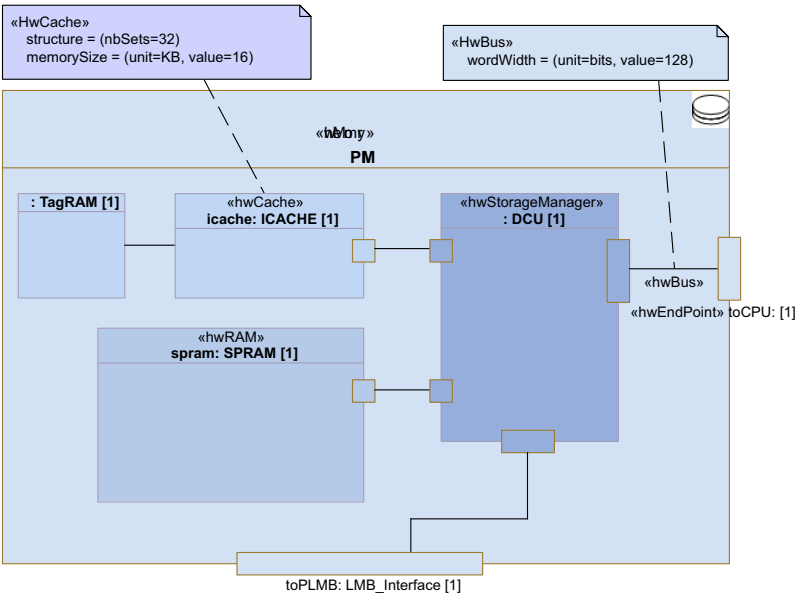

Fig. 4. PMI (Program Memory Interface) internal structure modeling

Instantiation

9. Step 9 is a test step, which unlike other steps, does nothing to the model under construction. Our methodology is iterative and incremental in the sense of composition, since the designer begins with basic resources and then iterates to resources increasingly composite. If the current class represents the entire platform, this means that the model is complete, and that normally at this stage, all resources are referenced from the current platform class. We skip therefore to step 10 for the model instantiation. Otherwise we iterate from step 1, and we choose the next resource to model from those that have all their subcomponents already modeled in previous iterations.

For example, figure 5 represents the block diagram of the complex CPU Subsystem of the *Infineon* microcontroller TC1796 [2]. Figure 6 shows its entire platform class model that was achieved through this methodology. If the first diagram is only a useless drawing, the second one is formal and may be used for analysis and simulation.

10. Finally, once the class of the whole platform is reached, instantiate the model giving values to slots (attributes, parts and ports), linking them and again applying stereotypes (if needed) on instances with assigning tag values corresponding to instance level semantics.

By the several steps of this methodology, we propose an efficient use of the HRM profile. We limit for this, the UML mechanisms to use and we give them

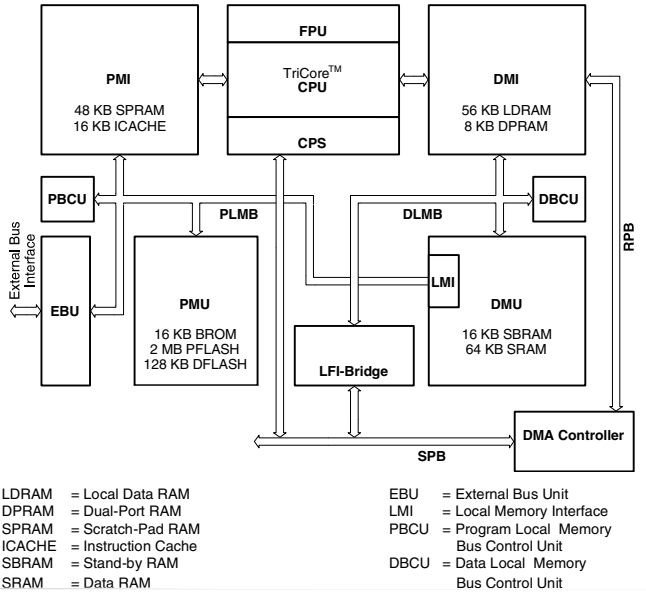

Fig. 5. TC1796 CPU Subsystem (block diagram)

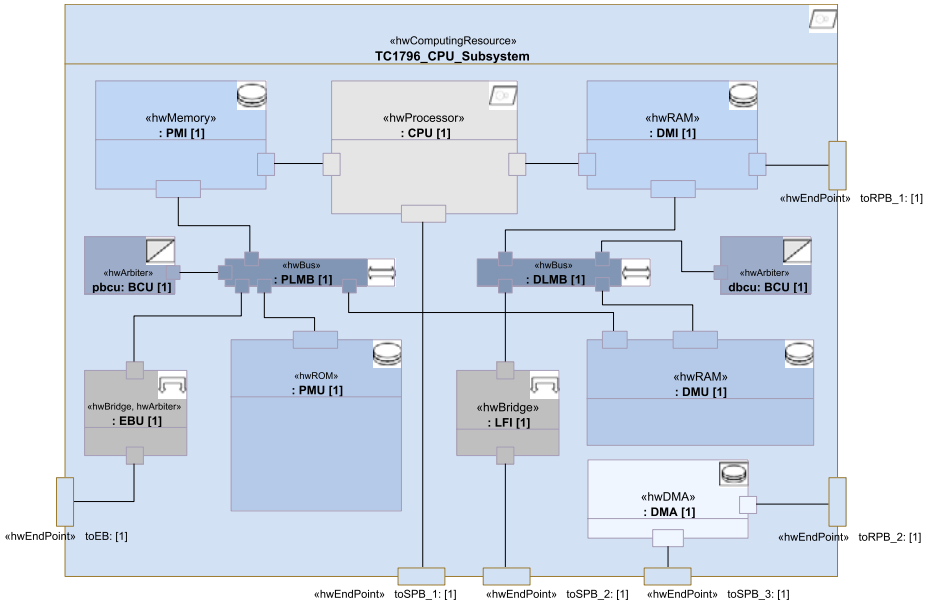

Fig. 6. TC1796 final platform class

clear semantics. However our methodology is advisory, even if it includes good practices, the use of HRM independently of it, is obviously possible. It is nevertheless adapted to new users of UML and ensures consistency of the final model. When a platform model is consistent you can use it for various manipulations, such as simulation that we will detail in the next section.

4 Hardware Simulation

The simulation of a hardware architecture to test its ability to provide an adequate execution platform for the software application, provides many benefits. It improves flexibility, accelerates the development process, saves time and money, and enables effective communication between software and hardware flows. Therefore, developers are no longer dependent on the availability of the physical hardware and they can explore in the early stages of design, several architectures, including new configurations. The simulation also offers several advances in debugging software and hardware.

Designated as one of the three HRM use cases, the idea behind our simulation engine is to use HRM/UML as a common interface to hardware simulation tools. Indeed, the user can take advantage of HRM/UML to describe a hardware architecture in a model that will be automatically translated and interpreted by simulation tools.

Most simulation tools are only Instruction Set Simulators (ISS) that simulate a processor with some RAM running assembler code. However, we simulate a whole execution platform with processors, memory, peripherals, and different means of communication. Such a simulation environment should also run complex software applications without any modification and start operating systems.

After a deep study we adopt Simics [1] as a target of our model-based simulation engine. Simics is capable of simulating the full-system. All common embedded components are available including PowerPC, ARM, SPARC, x86 processors, FLASH memories, I2C busses, serial ports and timers. Also, defining new components is feasible. Simics platform runs the same binary software as would run the real hardware target including operating systems and device drivers. Simics is at the origin a fast functional system-level simulator, it does not handle timing considerations. But recently, a Micro Architectural Interface was designed to overcome these limitations and provides cycle-accurate simulations.

Today, Simics is widely used by the telecom, networking, military/aerospace (including commercial avionics and space systems), high-performance computing, and semiconductor industries.

To start, we modeled using HRM all components supported by Simics, we get then a library of resources' models. This library will be provided to the user who will apply our HRM methodology to create his hardware platform. The user can use the resources of the library as basic components and with successive iterations in the sense of compositions, he can construct the whole hardware platform. Once done, he can automatically generate the equivalent script that will run under Simics. This process is illustrated in figure 7.

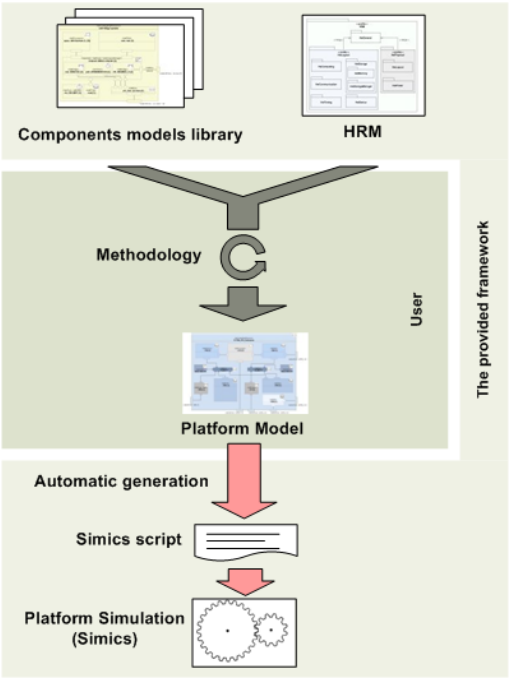

Fig. 7. Simulation engine process

Modeling the Library of Simics Components. As Simics is implemented in Python and C++, our task was easier because Simics has already an object-oriented structure. Nevertheless it has a specific terminology and semantics wider than the object paradigm ones. We had then to translate each of its concepts according to UML/HRM. In brief, the concept of *component* is central in Simics, it denotes a hardware resource that can be used in the construction of a platform (called *machine* or *configuration*), a *component* can be implemented by one or more *classes*.

Code Generation. We had primarily used Acceleo [4] that we reinforced by a set of services we have developed in Java (thanks to the Eclipse UML2 plug-in). Acceleo generates code from models by interpreting a script of declarative rules.

Our first step of code generation is to explore the platform subcomponents and generate the adequate Simics creation commands. To parametrize the Simics components, we have developed indeed a method that checks whether a stereotype is applied and gets the corresponding value of the tag definition when specified.

The second step of code generation is to produce connection commands between the Simics components created during the previous step. To do this, we take one by one all connectors of the platform that are linking the ports of the

various subcomponents. We check that the ports are similar and have consistent directions. Note however that connection commands may be inappropriate for technological or generational reasons and will therefore be rejected by Simics.

The first objective of this simulation engine was to demonstrate that HRM is complete and it offers a level of detail sufficient to interface the most accurate simulation tools. The second objective was to provide the hardware designer with a rich and automated model-based tool to assist him in designing platforms. Let's illustrate it on a real complex example.

5 Case Study

For our case study shown in figure 8, we consider a highly heterogeneous hardware platform, since it combines two very different computing resources: *board* and *boardSMP*. The first is a uniprocessor from the *PowerPC* family and has a *32bits* architecture. While the second is a multiprocessor (SMP) from the *Itanium* family with a *64bits* architecture, it may contain up to 32 processors sharing a *1GiBytes* memory. We connected *board* and *boardSMP* via an *Ethernet* link *ethLink*, but it was necessary, first, to provide the *boardSMP* with an Ethernet card *CardPciEth* that we connected to a PCI port. We also have connected to

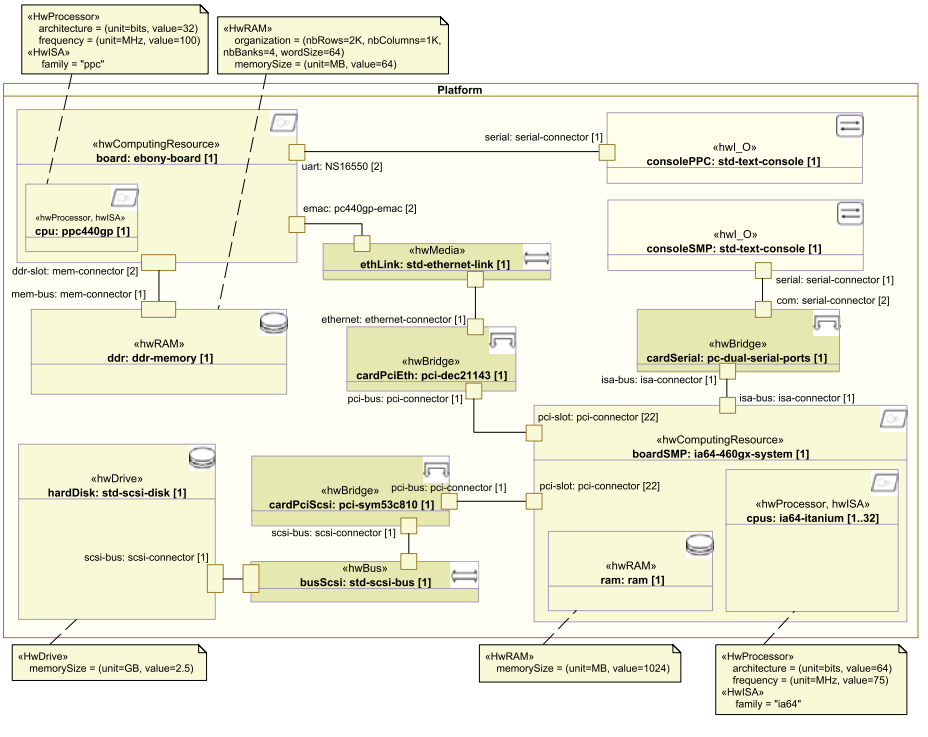

Fig. 8. Heterogeneous platform model

the *boardSMP* a *SCSI* hard disk harddisk via a *HwBridge pci-sym53c810*. *board* has a *64MiBytes ddr* memory and the exact description of its organization is specified in terms of *nbRows*, *nbColumns*, *nbBanks*...

Figure 8 shows the whole platform class that was obtained applying our methodology (within the Papyrus UML tool), we used then our simulation engine to generate the corresponding Simics script. To simulate this platform, we started two different Linux 2.4, the first on *board* was compiled for the *ppc32* instruction set and the second on *boardSMP* was compiled for the *ia64* instruction set with the SMP option activated. Both run and communicate perfectly.

6 Conclusion

Having no equals that meet the needs of high-level description of hardware architectures and with the standardization of MARTE, HRM is dedicated to a massive use within the industry. In this paper, we first describe a modeling methodology which helps to resourcefully use HRM for building consistent platform models. We developed then an innovative simulation framework that is hundred percent model-based and supports the widely-used simulator Simics.

References

1. Simics Platform, `http://www.virtutech.com`
2. TriCore Architecture, `http://www.infineon.com/tricore`
3. CEA LIST: Papyrus UML2 Tool, `http://www.papyrusuml.org`
4. Obeo: Acceleo Generator (2007), `http://www.acceleo.org`
5. Object Management Group, Inc.: MOF 2.0 XMI Mapping Specification, Version 2.1. Tech. Rep. 2005-09-01, OMG
6. Object Management Group, Inc.: OMG UML Superstructure, V2.1.2. Tech. Rep. formal/2007-11-02, OMG (2007)
7. Object Management Group, Inc.: Uml profile for marte, beta 2. Tech. Rep. ptc/08-06-09, OMG (June 2008)
8. Taha, S., Radermacher, A., Gerard, S., Dekeyser, J.L.: An open framework for detailed hardware modeling. In: SIES, pp. 118–125. IEEE, Los Alamitos (2007)