UML 2 AS AN ADL HIERARCHICHAL HARDWARE MODELING

Arnaud Cuccuru
Arnaud.Cuccuru@lifl.fr

Philippe Marquet
Philippe.Marquet@lifl.fr

Jean-Luc Dekeyser
Jean-Luc.Dekeyser@lifl.fr

Laboratoire d'informatique fondamentale de Lille
Université des sciences et technologies de Lille
France

Abstract Taking into account the hardware architecture specificities is a crucial step in the development of an efficient application. This is particularly the case for embedded systems where constraints are strong (real-time) and resources limited (computing, power). This approach is called co-design, and it is found more or less explicitly in ADLs. Much work have been done around co-design and ADLs, but no standard notation and semantics have emerged. Concerning software engineering, UML has become a recognized standard language for modeling, proving the need of users for common syntax and vocabulary to specify their applications. We believe that it would useful to use the well achieved syntax and vocabulary of UML for both applications and hardware architectures, that is to say using UML as an ADL. Our approach consists in a clear specialization of an UML subset via a the proposition of a generic profile that allows the definition of precise semantic and syntaxic rules. The generic profile can then be extended to suit the need of the user. To illustrate our subject, we give a refinement example of the profile to get relevant informations for a simulation at the TLM level (Transaction Level Modeling). The modeling of the Texas Instrument OMAP2410 and OMAP2420 is provided as an example.

1. Hardware Modeling and UML

The usage of an ADL permits to represent static or dynamic characteristics of a system. This system is either a software or a hardware system. For embedded systems, both system are defined: you need to co-design your application and your hardware platform in the same time with respect to specific constraints.

Concerning software engineering, UML (Unified Modeling Language) [Object Management Group, Inc., 2003] has become a standard language for modeling. It does not present a particular methodology and it can be used to model different point of view of the same model. UML 2 introduces the component notion and structure diagrams that facilitate architecture modeling. With deployment diagrams, UML 2 also considers hardware descriptions and mapping. Unfortunately, the model for applications differs from the model for hardware. The same comment applies for the mapping of an application on a particular hardware, for example a System on Chip: we want to benefit from the component notion, the hierarchical constructs of the structural and behavioral diagrams for the hardware design as well as for the software design.

We define our hardware description metamodel based on the UML 2.0 component notion. The structural specification is sufficient to generate SystemC [Open SystemC Initiative, 2002] code to produce a TLM (Transaction Level Modeling) simulation once the mapping of an application on this hardware model is achieved.

1.1 Related Work

Several proposals, emerged from the OMG world or not, introduce hardware modeling techniques and/or concepts. Only a few ones are "sold" as ADLs.

AADL [Feiler et al., 2003] (Avionics Architecture Description Language) is the only proposal which clearly advocates the use of UML as an ADL. This language is based on MetaH. It is used to describe the structure of an embedded system as a gathering of software and hardware components. It can describe both functional (data inputs and outputs) and non functional (such as timing) aspects of components. A UML profile for AADL (based on UML 2) is under standardization, and will be soon submitted to the OMG. Concerning the hardware modeling part of this proposal, four concepts are introduced: *memory*, *processor*, *bus* and *device*. We will see later in this paper that our proposal is not semantically far from this one. However, the goal of hardware modeling (called platform) in AADL is not the same as ours. The platform specification is more or less used to apply schedulability and fault tolerance analysis verification tools (that is to say to verify that a software associated to one platform meets the requirements that the system must satisfy), whereas the tools we plan to use are rather optimization tools (for mapping of computing, data,

and communications). Moreover, we can't clearly see in the specification of the language how the hardware concepts can be composed and/or assembled.

The **UML SPT Profile** [Object Management Group, Inc., 2002] (Scheduling, Performance, and Time analysis) is an UML 1.x OMG standard profile for embedded systems modeling. It introduces various hardware (or more generally platform) modeling concepts and an interesting resource classification according to three criteria: *purpose* (processor, communication and device), *activeness* (a resource is active when it is able to generate stimuli, and passive when it is only able to react when prompted by stimuli), and *protection* (a resource is protected when the access to services it offers is restricted according to some control access policy). However, the profile gives no clear orientation and methodology on the way it can be used. Moreover, the underlying execution model (a resource is acquired and then released) is too restrictive and unfortunately does not suit our needs.

HaSoC [Green and Edwards, 2002a, Green and Edwards, 2002b] (Hardware and Software Objects on Chip) is a platform-based design methodology using UML to represent high-level structure and behaviour of the hardware architecture model of an application platform. The hardware architecture model of HASoC distinguishes general-purpose hardware (processors, memory), programmable logic (FPGAs), fixed-function (custom) hardware, and interconnection elements. This model enables to generate SystemC code for simulation and particularly for hardware synthesis. For our purpose, distinction between programmable and non programmable units is not necessary [1], as hardware synthesis is not one of our goal. Moreover, this model doesn't seem to exploit hierarchical capabilities offered by UML 2.

The **SLOOP** [Zhu et al., 2002] (System Level design with Object-Oriented Process) design process integrates a methodology based on UML for both SoC modeling and performance evaluation at system level. The hardware model proposed is similar to those previously presented: it proposes hardware elements such as processors, memories, buses and hardware devices. However, the hardware architectures are modeled via deployment diagrams, with various stereotypes applied to "Nodes". This choice has the benefit of being simple and relatively natural for people coming from the UML world. Indeed, deployment diagrams are used to coarsely describe execution platforms for applications. But we believe that deployment diagrams are too restrictive to model architectures (no hierarchy, no encapsulation via ports and interfaces, no behavior description...), and that the same model must be used for both hardware and software.

1.2 Proposal

Our approach consists in a clear specialization of an UML subset for which we can defi ne precise syntactic and semantic rules. This prevents from all kind of ambiguities in the model[2].

The abstract syntax of our model is described by a MOF metamodel defi ned as an extension of the UML 2 metamodel. The 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 refi nements between abstraction levels via mapping rules, to ensure interoperability between the different abstraction levels used in a same codesign, and to ensure the opening to other tools, like verifi cation tools, thought the use of standards. The concrete syntax is defi ned by a profi le for UML 2. This profi le is, for the moment, only based on UML 2 class diagrams and internal structure diagrams[3]. In this paper, we focus on the description of the profi le(there is almost a "one to one" equivalence between concepts introduced in the metamodel and stereotypes introduced in the profi le). The generic rules and concepts are described in a generic part of the profi le. This generic elements are then refi ned via an extension of the profi le to fi t to the needs of the user (documentation, simulation at various abstraction levels, optimization...).

The fi rst part of the paper summaries the so-called "Y model approach" [Dumoulin et al., 2003], and describes the generic part of the profi le, focusing on the set of "basic blocks" that defi ne the syntax and semantics of our hardware architecture model. Therefore, we propose a classifi cation, at the same time both functional and structural, in order to identify each kind of components[4] of our model. Then, we specify the construction rules which govern components assembling and composition.

The second part of this document shows how an extension of the profi le can be used to refi ne the "basic blocks", in order to produce models suited to a particular use. For that, we give an extension of the used profi le to gather relevant informations of a simulation at a TLM level in an environment such as SystemC. As a conclusion, we illustrate our subject with a case study: the TI OMAP2410 and OMAP2420 [Texas Instruments, 2004].

2. Y Model Approach

Our proposal is partially based upon the "Y-chart" concepts [Gajski and Kuhn, 1983]. A clear distincition is made between application and hardware, which are related via an explicit mapping.

The application and hardware architecture are described by different metamodels. Some concepts handle within these two metamodels are similar in order to unify and so simplify their understanding and use.

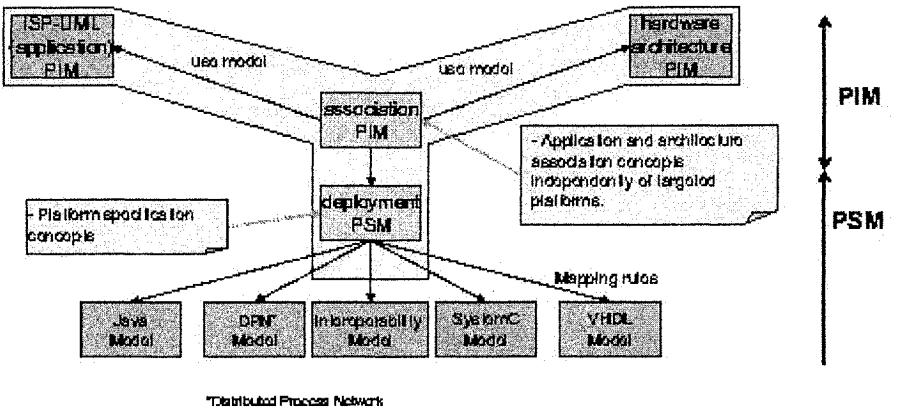

Figure 1. Y Model Approach

The proposed methodology is to separately design the two models for the application and the hardware architecture (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, the so-called association metamodel, to express the associations between the functional components of the application model and the hardware components of the hardware model. This third metamodel imports the first two metamodels.

3. Hierarchical Hardware Architecture Model

We present here the different hardware components that we introduce in our model, and we give the rules on the way to compose and assemble them with each other.

The hardware components represent abstractions of physical hardware architecture elements. A hardware component owns an interface materialized by its ports, and a structure defined by an assembly of components via an internal structure diagram.

3.1 Resource Classification

We propose to classify the resources according to two criteria: a functional criterion, and a structural criterion (Fig. 2). Each resource is characterized by a composition of these two criteria. In UML, it comes to apply two stereotypes to each component.

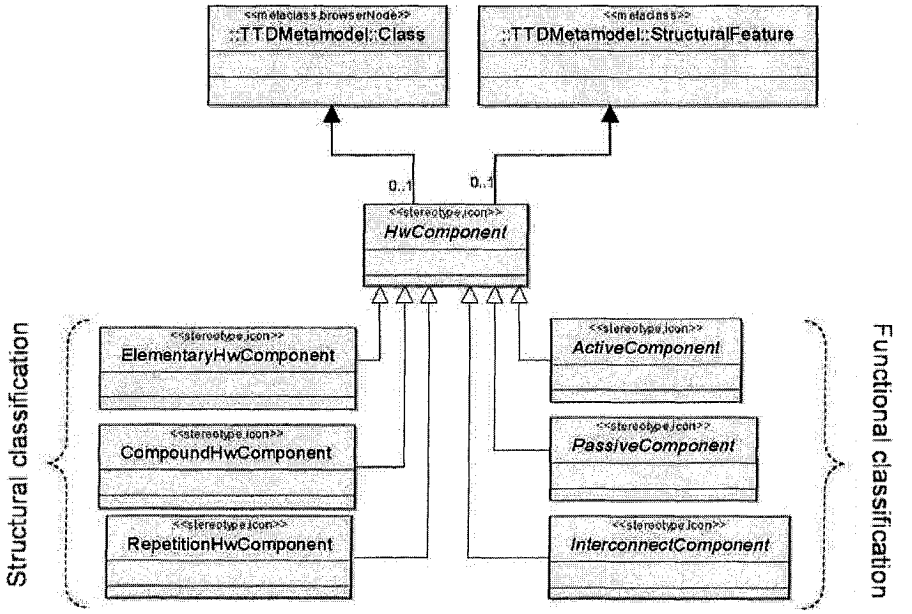

Figure 2. Resource classification

Functional Classification. We identify three kinds of function for a component (three stereotypes in UML): a component can be either passive, active, or for interconnection.

"`PassiveHwComponent`" A passive component symbolizes a resource which has the function of supporting data. Typically, we find in this category elements such as RAMs, ROMs, sensors, or actuators.

"`ActiveHwComponent`" An active component symbolizes a resource which has the function or reading or writing into passive resources. It may modify, or not, the data. This category includes elements such as CPUs or DMAs. It also includes more coarse-grained elements, such as a SMP node inside a parallel machine.

"`InterconnectHwComponent`" An interconnection resource connects active and passive components, or active and active in the case of a distributed memory architecture. This category includes element as simple as a bus, or as complex as an omega interconnection network.

Structural Classification. The structural classification is used ease introspection of models and, as a consequence, automatic or manual exploitation of gathered data (for example to optimize the mapping of an application on a

hardware architecture). We identify three kinds of structure for a component (three stereotypes in UML): a component may be either elementary, passive, or for repetition.

"ElementaryHwComponent" An elementary component is a component without an internal structural description. For example, it could be used in the case where we have an hardware IP for this component, or in the case where we don't want to model the component more finely.

"CompoundHwComponent" A compound component is a component with an internal structure description. A compound component can represent an "executable architecture" (fully defined architecture) or a part of an architecture that will be reused in other contexts. A compound component may be defined with several hierarchy levels: a compound component may be described as the gathering of other "sub" compound components. The benefits of composition are numerous, and they are not specific to our model: encapsulation of structural details not needed at a given hierarchical level, reuse or repetition (in the case of a "RepetitionHwComponent") of predefined blocks to model other architectures...

"RepetitionHwComponent" A repetition component[5] is a particular case of the compound component. The repetition component structure contains a regular repetition of a single component. This kind of component is well suited to the modeling of massively parallel architectures and is motivated by the recent introduction of such architectures in the design of SoC such as the picoChip PC101 [picoChip, 2003].

3.2 Construction Rules

The construction rules (that is to say the composition and assembling rules) of our model are based on UML 2.0 rules. From generic UML rules, we define coherent rules for the possible direction of ports ("required" or "provided") according to the function of each component ("active", "passive" or "interconnection"). Then, we propose rules for components assembling, based on rules given for ports direction. We also lightly modify the semantic of connection concept, but we remain coherent with definition given in UML 2 specification. Finally, we give rules concerning component composition.

Assembling Rules Defined in UML 2. The components are linked together via connectors. The ports materialize the connection points. One or more required or provided interfaces are associated to each port. Two ports can be connected only if they have compatible interfaces, and if one port is "required"[6], and the other is "provided"[0].

The methods associated to each interface represent services. A component with a required port is able to emit services requests to its environment, whereas a component with a provided port provides a set of services to its environment.

Ports "direction". We use assembling rules defined in UML to restrict possible directions of ports (required or provided) associated to each kind of components of our architecture model.

A passive component only owns provided ports, as it can only propose services for data support (read/write) to active components. An active component typically owns required ports, as it can emit read or write services requests to passive components. It can also own provided ports to receive requests emitted by other active components, for example in the case of a distributed memory architecture. An interconnection component owns required and provided, as it has the function of linking active and passive components, or active and(Fig. 5) active components.

Assembling Rules. Respecting rules previously mentioned, two passive components can not be connected together (on both sides, there are only provided ports). We add the constraint that all connections between active and passive components or between active and active components must be done via an interconnection component. This constraint enables to ease and systematize the parsing of models. Moreover, a clear identification of the interconnection resources enables to associate properties to these resources (such as bandwidth, or latency).

Connections Semantics. Services associated to ports are implicit according to the function of the component and the definition we gave of these functions (the active components require read/write services to passive components via interconnection components). As a consequence, an incomplete model in the standard UML formalism (that is to say with no interfaces associated to ports) can be interpreted without ambiguities in the context of our profile. The user of the profile is free to add the interface suited to the case he is modeling.

Connections between ports are interpreted as potential data paths offered by architecture, more than paths for services exchanges. This semantics is close to the semantics of connections between modules in SystemC.

In our case, the association of interfaces to ports in a hardware architecture is a way to refine its specification according to application mapped on this architecture.

Composition Rules. A passive compound component may only contain other passive components. A active compound component can only contain

other interconnection components. An active component can contain all kinds of component. The top level component in the hierarchy is necessarily a active compound (or repetition) component. By this way, the parsing of an architecture consists in a traversal of active compound or repetition components.

4. A Profile Refinement for TLM Level Simulation

In this section, we show a refinement of the generic profile designed to get relevant informations from a simulation at a TLM level, such as execution time, load of interconnection elements, load balancing of active elements... The profile refinement consists in extending stereotypes defined in the generic part by adding them properties. First of all, we describe data types associated to properties of various modeling elements. Then, we show successively a refinement for active elements, for passive elements, and finally for interconnection elements.

4.1 Data Types

Data types described here are used to type properties added to various refined model elements:

`TimeExpression` is an expression representing a duration *("13ns")*

`FrequencyExpression` is an expression representing a clock frequency *("1.2MHz")*

`CapacityExpression` is an expression representing a capacity *("16Mo")*

`BandWidthExpression` is an expression representing a bandwidth *("3.5Go/s")*

4.2 Active Component Refinement

To refine the concept of active component, we introduce two kinds of components: "`CPU`" and "`DMA`" (Fig. 3). These components extend (in a UML point view, but particularly in a semantic point of view) the definition of the active component.

"`CPU`" Component. A CPU represents a resource able to read and write in passive resources, with or without data modifications. It also symbolizes a resource able to execute functions defined in the application model. It is potentially able to execute all functions, except if the list of functions is explicitly restricted[7]. CPU is a generic term gathering CPUs (strict meaning), DSPs, or even FPGAs (i.e. all kind of programmable or not programmable resource able to realize a computation on data). A CPU is characterized by four attributes:

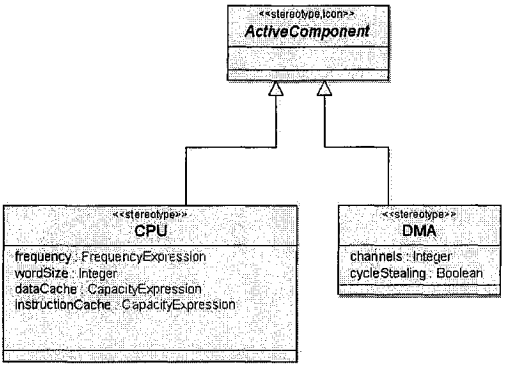

Figure 3. Active component refinement

`frequency: FrequencyExpression`. This property represents the clock frequency of the CPU. For example, it can be used to determin the execution duration of a function, in the case where the number of cycles necessary to execute this function on this CPU is known.

`wordSize: Integer`. This property represents the size of the words handled by the CPU. The size expression unit is the bit.

`dataCache: CapacityExpression`. This property represents the size of the data cache of the CPU.

`instructionCache: CapacityExpression`. This property represents the size of the instruction cache of the CPU.

"DMA" Component. A DMA is characterized by its number of channels and by the policy it uses: whether it is based on cycle-stealing or not.

4.3 Passive Component Refinement

To refi ne the concept of passive component, we introduce three kinds of component: "`Memory`", "`Sensor`" and "`Actuator`" (Fig. 4). These components extend (in a UML point view, but particularly in a semantic point of view) the defi nition of the passive component.

"`Memory`" Component. A memory has the function of supporting data. It is characterized by six attributes:

`capacity: CapacityExpression`. This property represents the size of the memory.

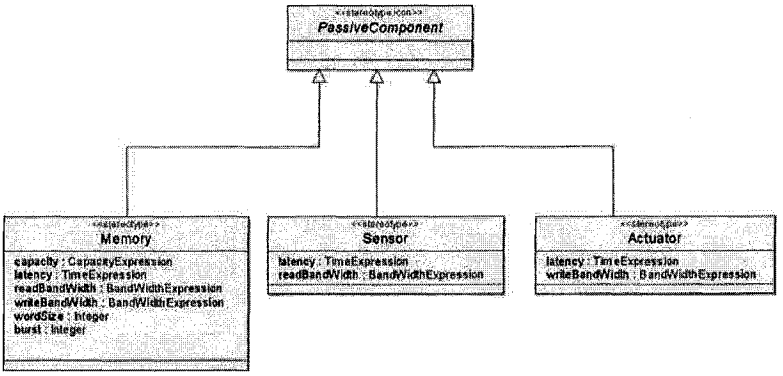

Figure 4. Passive component refinement

`latency: TimeExpression`. This property represents the latency related to a memory access. The value is expressed in time, and not in cycles, in order to make easier the reuse of memory component for several architecture modelings, or in the case of heterogeneous architectures (several kinds of CPU accessing the same memory component).

`readBandWidth: BandWidthExpression`. This property represents the read bandwidth of the memory.

`writeBandWidth: BandWidthExpression`. This property represents the write bandwidth of the memory.

`wordSize: CapacityExpression`. This property represents the size of a memory word.

`burst: Integer`. This property represents the number of memory word collected when an access in burst mode occurs.

"`Sensor`" Component. A sensor represents a resource able to periodically catch data from its environment, and put it at hardware architecture disposal. Sensors are classifi ed in the category of passive elements, in the way that they can be considered as read only memories. A active element can periodically get (read) data to compute or move from it. A sensor is characterized by two attributes:

`latency: TimeExpression`. This property represents the latency related to a read access.

`readBandWidth: BandWidthExpression`. This property represents the read bandwidth of the sensor.

"`Actuator`" Component. An actuator is the opposite of a sensor, that is to say a resource that provides data from the hardware architecture to its environment. Actuators are classified in the category of passive elements, in the way that they can be considered as write only memories. An active component can periodically give (write) data to it. An actuator is characterized by two attributes:

`latency: TimeExpression`. This property represents the latency related to a write access.

`writeBandWidth: BandWidthExpression`. This property represents the write bandwidth of the actuator.

4.4 Interconnection Component Refinement

To refine the concept of interconnection component, we introduce one kind of component: "`Interconnect`" (Fig. 5). This component extends (in a UML point view, but particularly in a semantic point of view) the definition of the interconnection component.

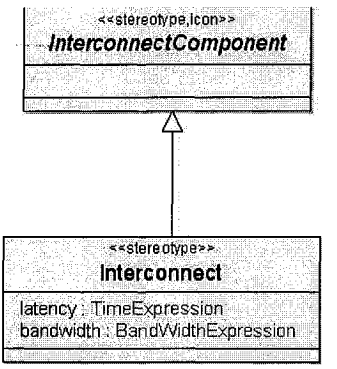

Figure 5. Interconnection component refinement

"`Interconnect`" Component. The interconnect is not more precise semantically than the interconnection component defined in the generic part of the profile. It's just an interconnection enriched with properties used for a simulation at a TLM level. It is characterized by two attributes:

`latency: TimeExpression`. This property represents the latency related to an access to the interconnection component.

`bandWidth: BandWidthExpression`. This property represents the bandwidth of the interconnect.

5. Modeling examples: The TI OMAP2410 and OMAP2420

We present in this section two modeling examples. Specifications are deliberately not complete, and are just introduced for an illustration purpose. We illustrate the use of various modeling elements (refined or not) of our model,

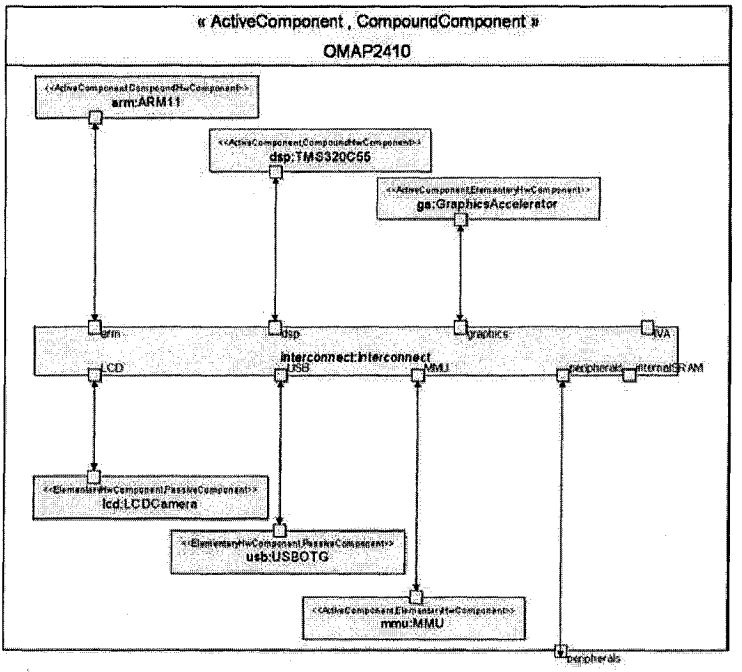

Figure 6. OMAP 2410 Structure

and the use of the inheritance mechanism of UML to defi ne an hardware architecture by extension of another one.

The OMAP2410 and OMAP2420 processors are based on OMAP2 "All-in-One Entertainment" architecture [Texas Instruments, 2004]. They are designed to be used into smart-phones and portable media devices. They support high-end features such as multi-megapixel cameras, hifi music with 3D sound effects, high-speed wireless connectivity and more.

The OMAP2410 processor (Fig. 6) consists of an ARM11 that handles the operating system tasks and a TMS320C55 that works on the audio and video applications. It also contains additional features such as a hardware engine for three-dimensional images, a MMU (memory management unit), and interfaces to LCD/Camera, USB devices, and peripherals. Communications are made via a low-latency interconnection component.

The OMAP2420 processor (Fig. 8) is an extension of the OMAP2410. It extends the features of the 2410 with an IVA (Imaging Video Accelerator) and a 5 Mbit SRAM supporting a VGA display. As illustrated in Fig. 7, the OMAP2420 inherits the defi nition of the 2410.

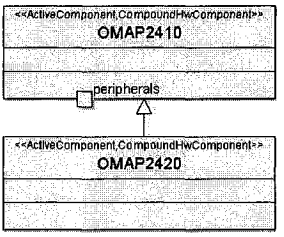

Figure 7. Inheritance link between OMAP 2410 and OMAP 2420

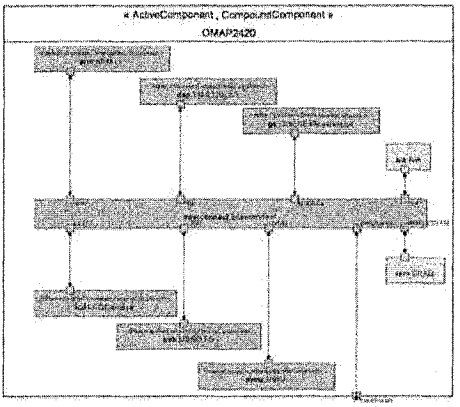

Figure 8. OMAP 2420 Structure

6. Conclusion

We have shown how we use UML 2 as an ADL, focusing on the hardware model of our approach. The model we have presented enables a description of hardware architectures at a high abstraction level. This model is implemented in a UML profi le. This profi le introduces generic concepts, with clear semantic and syntaxic basis, that can be refi ned for particular use. We gave an example of profi le refi nement to get relevant informations from a simulation at a TLM level, and we illustrated our subject with the modeling of the TI OMAP2410 and OMAP2420.

This is only the fi rst step of our approach. We are currently working on mechanisms to express repetition of architectural elements. It enables for example to easily model regular hardware architectures such as hypercubes, or complex interconnection networks such as Omega networks. The repetitions are expressed in the context of the "`RepetitionHwComponent`", with mechanisms similar to ones used in the application model to express data parallelism via dependency expressions.

Moreover, we plan to soon introduce behavioral descriptions of hardware components. Typically, such informations would be taken into account in our simulation environment. For example, it would be useful the describe the behavior of a shared interconnect element (priority policy) to get informations about latency.

Notes

1. As we'll see later, we just tell which functions can be executed by a processing unit

2. This restriction does not mean that the usage of the other UML elements are forbidden. It only means that all the elements added by a user in a model will not be taken into account by tools which exploit the profi le.

3. We are investigating the introduction of activity diagrams, and possibly state diagrams, to model hardware architecture behavior

4. In this document, the term "Component" refers to UML 2 "StructuredClass".

5. This kind of component is part of a work in progress and will not be detailed in this paper.

6. A "required port" refers to a port with a required interface , and a "provided port" refers to a port with a provided interface.

7. The way to restrict a list of executable functions is not described here. This problem is rather related to the association model.

References

[Dumoulin et al., 2003] Dumoulin, Cédric, Boulet, Pierre, Dekeyser, Jean-Luc, and Marquet, Philippe (2003). UML 2.0 structure diagram for intensive signal processing application specification. Research Report RR-4766, INRIA.

[Feiler et al., 2003] Feiler, Peter H., Lewis, Bruce, and Vestal, Steve (2003). The SAE avionics architecture description language (AADL) standard : A basis for model-based architecture-driven embedded systems engineering. In *RTAS 2003 Workshop on Model-Driven Embedded Systems*.

[Gajski and Kuhn, 1983] Gajski, D. D. and Kuhn, R. (1983). Guest editor introduction: New VLSI-tools. *IEEE Computer*, 16(12):11–14.

[Green and Edwards, 2002a] Green, Peter and Edwards, Martyn (2002a). The modeling of embedded systems using hasoc. In *Design, Automation and Test in Europe Conference and Exhibition (DATE'02)*, Paris, France.

[Green and Edwards, 2002b] Green, Peter and Edwards, Martyn (2002b). Platform modeling with UML and systemc. In *Forum on specification and Design Languages (FDL'02)*.

[Object Management Group, Inc., 2002] Object Management Group, Inc., editor (2002). *(UML) Profile for Schedulability, Performance, and Time Specification.* `http://www.omg.org/cgi-bin/doc?ptc/2002-03-02/`.

[Object Management Group, Inc., 2003] Object Management Group, Inc., editor (2003). *(UML 2.0): Superstructure Draft Adopted Specification.* `http://www.omg.org/cgi-bin/doc?ptc/03-07-06/`.

[Open SystemC Initiative, 2002] Open SystemC Initiative (2002). SystemC. `http://www.systemc.org/`.

[picoChip, 2003] picoChip (2003). PC101 and PC102 datasheets. `http://www.picochip.com/technology/picoarray`.

[Texas Instruments, 2004] Texas Instruments (2004). OMAP 2 architecture. `http://focus.ti.com/docs/general/splashdsp.jhtml?&path=templatedata/cm/%splashdsp/data/omap2`.

[Zhu et al., 2002] Zhu, Qiang, Matsuda, Akio, Kuwamura, Shinya, Nakata, Tsuneo, and Shoji, Minoru (2002). An obect-oriented design process for system-on-chip using UML. In *Proceedings of the 15th international symposium on System Synthesis*, pages 249–259, Kyoto, Japan.

SESSION 3: DOMAIN SPECIFIC ARCHITECTURE DESCRIPTION LANGUAGES