Multi-View Power Modeling based on UML MARTE and SysML

Carlos Gomez, Julien Deantoni, Frédéric Mallet

Multi-View Power Modeling based on UML MARTE and SysML

Carlos Gomez, Julien Deantoni, Frederic Mallet

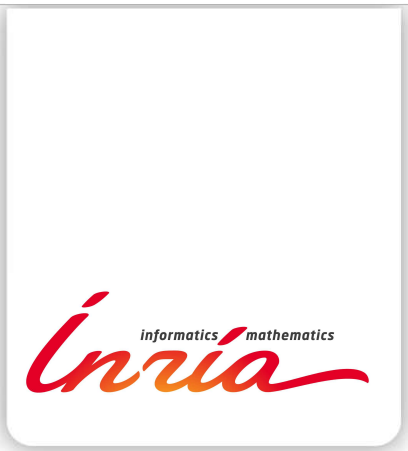
RESEARCH REPORT

N° 7934

April 2012

Project-Team Aoste

ISSN 0249-6399 ISRN INRIA/RR--7934--FR+ENG



Multi-View Power Modeling based on UML MARTE and SysML

Carlos Gomez*, Julien Deantoni*, Frederic Mallet*

Project-Team Aoste

Research Report n° 7934 — April 2012 — 19 pages

Abstract: The development of SoC involves different activities, usually driven by specialists. These specialists use specific languages and tools to manipulate their specific concepts. The problem is that the multiple views of the system are split into different tools with redundant information. It makes it difficult to ensure consistency as well as to change from one tool to another. We propose a multi-view model where each view represents the specialist concepts in a tool-agnostic manner. The model can be kept consistent by using explicit associations instead of redundancy and tool transformation can be performed to analysis-specific tools. The approach is based on UML and two of its extensions: MARTE and SysML. It is illustrated by adding specific views to specify power management techniques. The resulting model is then transformed into a tool-specific model; *i.e.,* a model for Docea Aceplorer, a power analysis tool.

Key-words: Multi-View Modeling, Power Reduction Techniques, UML, MARTE, SysML

* Université Nice-Sophia Antipolis

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Multi-View Power Modeling based on UML MARTE and SysML

Résumé : Le développement de SoC implique des activités différents, habituellement manipulées par des spécialistes. Ces spécialistes utilisent des langages spécifiques et des outils pour manipuler leurs concepts spécifiques. Le problème est que les plusieurs points de vue du système sont répartis dans différents outils avec des informations redondantes. C'est difficile d'assurer la cohérence ainsi que le passage d'un outil à l'autre. Nous proposons un modèle multi-vue où chaque vue représente les concepts spécialisés d'une manière d'outil agnostique. Le modèle peut être maintenue constant en utilisant des associations explicites au lieu de la redondance, et l'outil de transformation peut être utilisée à l'analyse des outils spécifiques. L'approche est basée sur UML et deux de ses extensions: MARTE et SysML. Celle-ci est illustrée en ajoutant vues spécifiques pour spécifier les techniques de gestion d'énergie. Le modèle résultant est ensuite transformé en un modèle d'outil spécifique, c'est à dire, un modèle pour Docea Aceplorer, un outil d'analyse d'énergie.

Mots-clés : Modélisation Multi-Vue, Techniques de Réduction d'énergie, UML, MARTE, SysML

1 Introduction

The modeling of System on Chip (SoC) is a complex activity because it relies on several expertise domains. For instance, a SoC has a physical (layout) and logical (gates/IP) representation. Each representation has a structural definition, where SoC components and their connections are specified from a different point of view: the logical representation serves to ensure the functionality of the system, whereas the physical one aims at performing the timing closure. Additionally to such functional concerns, both representations must take performance and extra-functional concerns in consideration (like the heat dissipation and the energy consumption or timing properties). Such performance concerns also impacts the behavioral definition, where the functionality of each component and their interaction are possibly modified by extra-functional concerns (*e.g.,* power management techniques). In addition to the various activities involved in the development of SoCs, the system is usually modeled at different abstraction levels, starting with high abstraction modeling and refining it into a detailed description.

The study of functional and extra-functional properties is usually done through the analysis of specific domain models where the concerning properties are defined. For instance, to study the power consumption of a system, domain experts develop a system model which concepts represent the power consumption properties of each component of the system. This model is a specific abstraction of the system, representing the system from the power consumption point of view.

Depending on the points of view, different languages are used, each one dedicated to a specific tool. It makes difficult the analysis of interactions between properties from different points of view. It also makes it difficult to ensure a consistent model of the SoC. In the study of power consumption, in the one hand the functional properties are defined in a model, *e.g.,* SystemC and on the other hand power experts build the power consumption model of each system component using a power domain language specific to a tool (*e.g.,* for the Aceplorer tool[1]). Aceplorer uses scenarios to analyze the power consumption. These scenarios have to be created according to the power model description specified in this tool but are usually provided by the simulation of the functional specification. In this case, the SystemC and Aceplorer models have to be consistent and aligned to be used conjointly.

A first solution to specify the various SoC points of view is to defined them in a unique model. For instance, by using the UML or one of its specific profiles (*e.g.,* SysML and MARTE). However, in traditional approaches, concerns are usually mixed up together even though they are, during a traditional development process, driven by very different designers, specialists of their domain. A key issue for the acceptance of high level design for SoC is, on one hand the ability to keep consistency between different models while keeping each concern separated from the other, and on the other ones, the ability to use already existing analysis-specific tools or the extraction of information from these models.

As an answer to this problem, we propose a multi-view modeling approach to specify each system concerns in a dedicated view. This framework aims at describing the functional and extra-functional properties by using the concepts well accepted in the view domain. We start from an architectural system model based on the component paradigm and then we add other views as specific concerns based on the first view. In a specific view, a component is abstracted and associated with "view specific" components and equations. Each view explicitly exhibits its control possibilities through control ports. A specific view is then in charge of keeping the consistency between all the control ports of each view. Finally, by using transformations, we build analysis/tool specific models, which extract properties from the different views to enable the use of specific tools for analysis.

To support our approach, we use the modeling language (UML) together with standard UML

profiles, namely MARTE and SysML. MARTE is a profile used to specify real-time and embedded systems. SysML is a language used in the system engineering domain. We use MARTE for its hardware and software component model and SysML for its acausal model (so-called parametrics).

Our approach is illustrated by the realization of the views needed to implement power consumption reduction techniques. After a description of the related works, we give a brief overview of the views used in this paper. Then we detail each of the view domain model. In section 5, we present a profile that specifies the minimal extensions of UML needed to support multi-view modeling. Before concluding, we present the models of the view involved in a simple example and the transformations used to make power consumption analysis in the Aceplorer tool.

2 Related Work

In the past few years, two languages have emerged to specify power reduction techniques at register transfer level (RTL) adding power elements to the exiting Hardware Description Languages like VHDL [2] or Verilog [3]. The first language is UPF (Unified Power Format) [4], initially developed by the Accellera consortium [5]. It has now become an IEEE standard named IEEE 1801 [6]. The second language is CPF (Common Power Format) [7], created by Cadence and then transferred to Si2 [8]. Both languages are very similar and the IEEE is currently making efforts to align the two languages. UPF and CPF mainly describe specific elements to control power sources, *e.g.,* switches, by putting existing components in specific power domains. The control of these elements is left aside and supposed to be added in the hardware description language itself.

In the literature, there are an increasing number of approaches that make efforts to describe power issues at a higher level of abstraction. Among these approaches, two OMG UML profiles are interesting, MARTE [9] and SysML [10]. These standards already define some features to characterize power consumption in systems. MARTE proposes in its Hardware Resource Modeling and Non-Functional Properties packages, a power package (HW_Power) where the power consumption and the heat dissipation are defined for each hardware component and its provided services. MARTE also defines components that supply energy (HW_PowerSupply and HW_Battery), and components that reduce the heat dissipation (HW_CoolingSupply). This standard aims providing a power specification to execute power consumption analysis of the systems. However, MARTE does not provide any way to specify power reduction techniques and does not support power control elements such as the ones defined in IEEE 1801 and CPF; it only provides power consumption values, fixed all along the component use. It is then not possible to specify different modes of the component where its consumption changes. The possible power consumption analysis is then really limited. MARTE also proposes specific types to define extra-functional properties. These types are specified as Non-Functional Properties (NFPs). Using these types, extra-functional properties such as power, current and voltage can be specified. SysML proposes a similar type definition to specify extra-functional properties. In SysML these types are called *TypeValues*. SysML also defines *Constraint Blocks* to specify equations. To specify systems of equations, SysML uses *Parametric Diagrams*, an acausal diagram making the originality of the profile since it allows the description of physical phenomenon by using equations over continuous variables.

Recent works propose extensions to MARTE to add the missing concepts for the definition and analysis of power consumption aspects. Arpinen et al. [11] propose an MARTE extension to model dynamic power management based on finite state machines. They define a power state machine where each state represents a power specification of a system component. *Power System*

Configuration, a MARTE *Configuration* extension, contains the active power states of the system components. When the application is specified by use cases, they are allocated to a power system configuration. When a use case is active, the associated system power configuration is activated and the power states associated with this configuration holds. In the same way, Hanger et al. [12] propose a power consumption analysis view profile based on MARTE. They define stereotypes to specify a power model of the system components and the executed tasks. Each element contains specific power features that are used to evaluate the power consumption of the system and to explore the optimal power solution.

3 Power Modeling Overview

3.1 Power Estimation and Power Reduction Techniques

In general, the power consumption of a components is dictated by equations, proposed by specialists. At a finer level, these equations depend on the transistor technology used. However, less precise equations can be used to give first results at a higher abstraction level. For instance, the power can be characterized by equation 1:

$$P_{total} = P_{stat} + P_{dyn} \tag{1}$$

Where P_{total} is the total power consumed by the component, P_{stat} is the power consumed when the component is in idle state, and P_{dyn} is the power consumed when the component is in active state. P_{stat} depends on the technology that is used to build the physical component. It also depends on the temperature of the component. P_{dyn} can be approximated by equation 2, defined in [13][14].

$$P_{dyn} = \alpha.C_{load}.V_{dd}^2.f \tag{2}$$

Where α is the switching activity, C_{load} is the load capacitance of the component, V_{dd} is the voltage source, and f is the frequency of the clock signal. Frequency and voltage are the two properties modified by power reduction techniques to reduce the dynamic power consumption. Note that the activation frequency of a component depends on its voltage. The most representative techniques to reduce dynamic power is Dynamic Voltage Frequency Scaling (DVFS) [15], which controls the voltage and frequency values to reduce consumption while still ensuring the temporal properties. To control the static power consumption, the most effective technique is to turn off the component when it is not in use. This technique is called Power-Gating [16]. The same technique can also be applied on the clock instead of the power (*i.e.,* clock gating [17]).

This section highlights some issues for the designers when they want to model power reduction techniques of a SoC system. A first designer should design the system itself. This system can contain temporal properties such as deadline, periodic activation, etc. Based on this, an power management expert can provide the structural elements to allow power management. Then, the associated equations must be defined for each "power state" specified by the power management elements. These equations can also be impacted by other physical phenomenon from other views, like for instance the heat dissipation view. Finally, the power controller that controls the power elements must stay consistent with regards to the other system constraints (temporal ones but possibly many other ones). In the remainder of the paper we propose to specify each concern in a specific view but with explicit link to the relevant other view elements of the system.

3.2 Multi-View Power Model

In our multi-view system, there is always a backbone view. This view is the logical representation of the system under development. This view, in the domain of SoC design is typically represented by an interconnection of components, where components represent the IPs (Intellectual Property) of the system. In each other view, an abstraction of these components is used to specify their property with regards to the view under consideration. For instance, in the power view, a component is abstracted by a state machine representing its power states. The power states represent states where the component has a specific behavior on the extra-functional properties related to the layer (in this case, energy consumption). The evolution of the extra-functional properties in a state can rely on an equational view if needed. The equations are then described in a specific view (power characterization view) but the power view elements are explicitly linked to the corresponding equations. The same pattern applies to the other views (like the clock view, for instance). Finally, when a view has control possibilities, it possesses control ports linked to a control view, in charge of the consistency of the control between the different views so that (temporal) requirements are fulfilled (Fig. 1).

4 Views Definition

We create a domain model definition for each view to specify clearly the concepts and the relationships between the view elements. This definition is specified through a meta-model. In the following, we suppose that the backbone view is already defined by using the MARTE profile (*e.g.,* by using HW_Logical definition) [9]. It consequently defines a *FunctionalView* in terms of the structure and behavior of the system functionality. In this view, the extra-functional properties are not defined, they are specified in dedicated views. These dedicated views share a common definition that is specialized for each view. presents the common view elements.

4.1 View Element

A *ViewElement* defines the common concepts shared by elements from multiple views.

We follow a component approach to describe our views to be consistent with the representation adopted in MARTE for hardware and software descriptions, with which we want to be compatible. A *ViewElement* contains *features*, *nets* and can be hierarchical, *i.e.,* it can contain nested *viewElements* (cf. right-hand side of Figure 2). A *Feature* has a type and can be either shared with another view element (it is a *Port*) through connectors (*i.e., Net*) or can be an internal attribute (*i.e.,* a *Property*). *ControlPort* is a specialized port that receives only control events. These events are all sent by the control view to ensure the consistency of the control. No assumption on the structure of the control is given since several strategy are possible (*e.g.,* hierarchical control or not). Note that a *Net* can connect only ports of the same type. *Port* is an abstracted entity that may be extended in each view, according to its specification.

A *ViewElement* contains a state machine with its respective states and transitions. This state machine must abstract the states of a specific component according to the kind of view it is used into. For instance, in a power view states represent the power states of the component. A transition between two states is fired when a specific control event occurs on a control port.

4.2 Power View

The elements of this view intend to supply power and control power features of system components defined in the functional view. The elements from these views are abstractions of

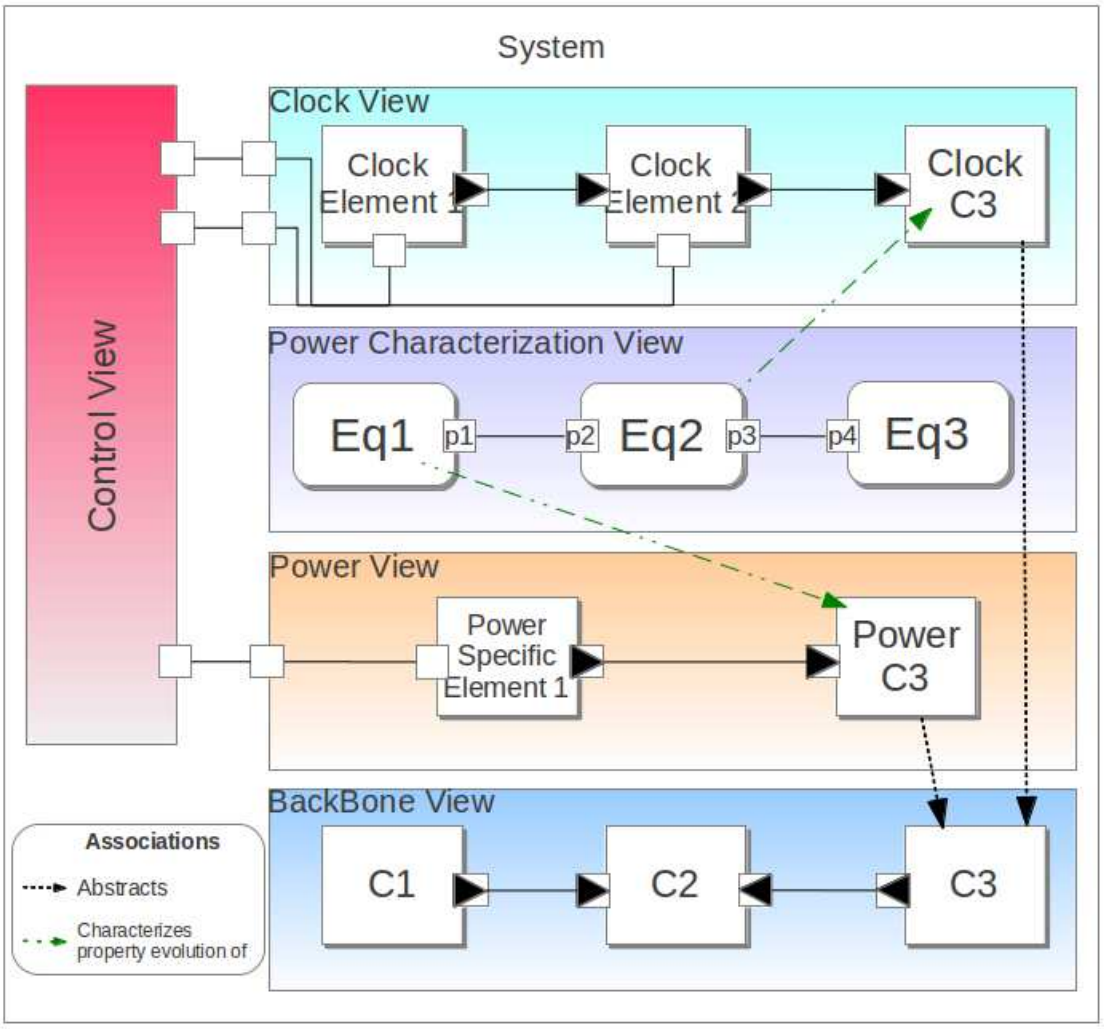


Figure 1: System model overview

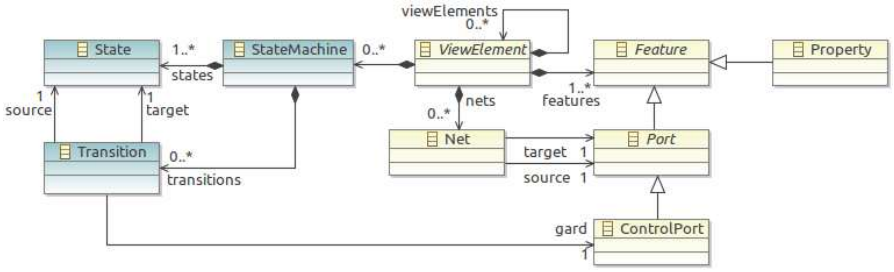


Figure 2: Simplified View element meta-model.

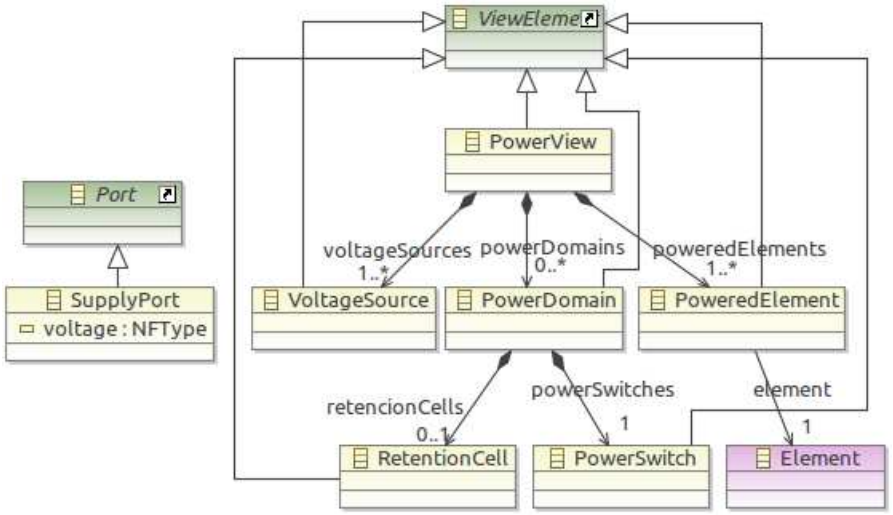


Figure 3: Simplified power view meta-model.

the concepts defined in the CPF and UPF languages. The *PowerView* element inherits from *ViewElement* and contains control ports and other view elements (see Figure 3). We define a specialization of *Port* called *SupplyPort*. This specialization represents the power nature of the information shared between the power view elements. The type of the associated port value is *NFType*. A NFType is a tuple with a unit of type *PowerUnit* and a value. *PowerUnit* is defined according to the International System of Units. The elements of this view are: *VoltageSource*, *PowerDomain* and *PoweredElement*.

A *PoweredElement* is an abstraction of a functional view element from a power point of view (see *Element* association in Figure 3). A powered element contains a supply port that receives a voltage value from a *PowerDomain*, and it also possesses enough control ports to change the current power state of its state machine. A *PowerDomain* is the component that provides the voltage values to a set of powered elements. A *PowerDomain* contains two kinds of view elements: *PowerSwitch* and *RetentionCell*. A *PowerSwitch* contains two supply ports and a control port. The supply ports are used to receive a voltage value from a voltage source and to send a voltage value to a powered element according to its active state. The behavior is defined by a state machine where there are two states: *Open* to indicate that the voltage value sent to the powered element is zero, and *Close* to show that the received voltage value is re-transmitted to the powered element. The state change is done via a control port. A *RetentionCell* saves information of the functional view element associated with the supplied powered element before this element is turned off. When the element is turned on, the *RetentionCell* restores the saved information. Power domain also contains control ports and nets that transmit the control event sent from the control view to its internal view elements. Additionally, a power domain has *SupplyPorts* to receive and to transmit voltage values. Using power switches and retention cells, we can implement the power-gating technique. Low abstraction level elements from UPF and CPF like isolation cells and level shifters are not specified here because they can be automatically generated from the power view model definition.

A *VoltageSource* is the power source. This element contains a supply port, to transmit the voltage values to a power domain or directly to a powered element. The voltage source behavior definition is a state machine which states represent the different voltage values that

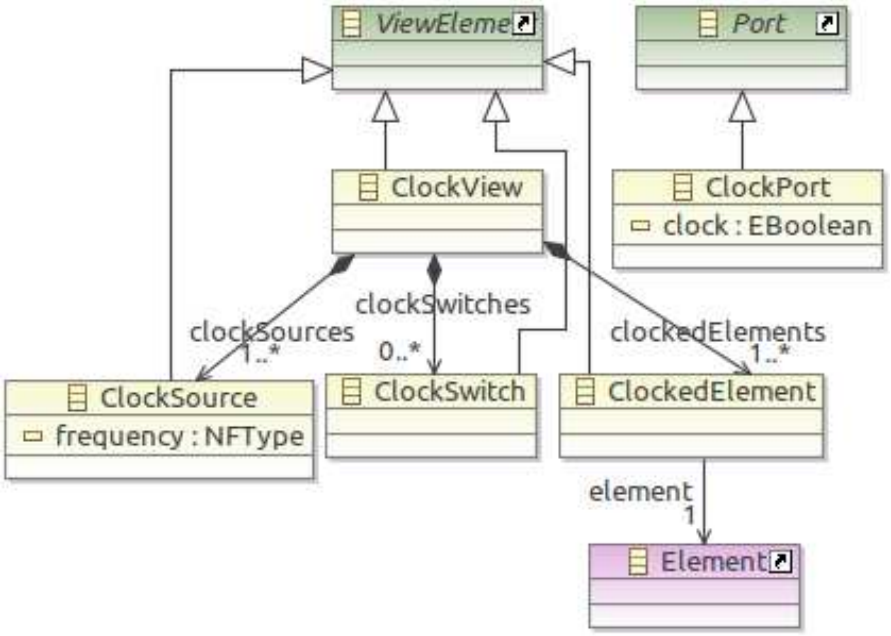


Figure 4: Simplified Clock view meta-model.

can be provided by the component. Voltage sources may have a control port that receives a control event from the control view to fire transitions between states. Dynamic voltage scaling techniques can be implemented on a functional view element using a voltage source.

4.3 Clock View

This view specifies the elements that provide and control the supply clock signals that activates the functional view elements. Figure 4 presents a simplified domain model of the clock view. It is quite similar to the power view but with a Boolean value on the *ClockPort* which repetitive change of value creates a clock. The main difference with the power view is that the clock state of the *ClockSource* state machine specifies the frequency of the signal in the clock port instead of a value change on the port.

4.4 Equational View

The main importance of this view is to define the evolution of the extra-functional properties specified in the other views. For instance, if we have a voltage source that represents a battery, the intensity of the voltage intensity level increases or decreases depending on the state of the battery (on charge or not). This evolution is represented by equations. Note that such equations can also be used to compute extra-functional properties of the system like the global energy consumption for instance.

The *EquationalView* contains hierarchical *EquationalModels*, themselves composed by *Equations*, *Parameters* and *Bindings*. An *Equation* is an acausal relation among the associated parameters. The acausal relation is given by a definition (not detailed here), in the form of a mathematical relationship between the formal parameters. A single parameter can be used in various equations by using *Bindings*. It is important to notice that this equational view is linked

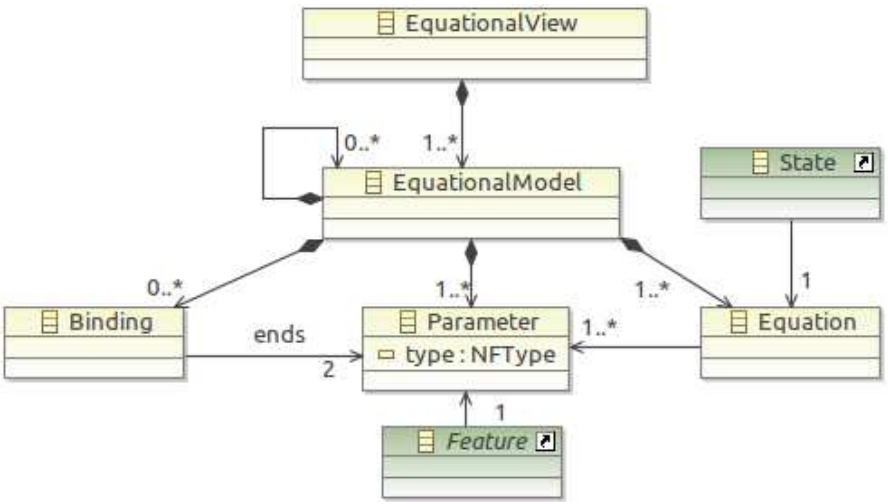


Figure 5: Equational View meta-model.

to the other views by two mechanism. First, the features of a *ViewElement* can be associated with a parameter in the equational view (their values are always identical). Second, a *ViewElement* state is linked to an equation so that, in this state, the feature of the view element is defined by the associated equation. For instance, in Figure 6, Voltage Source contains a voltage feature called *vout*. This feature is associated with the parameter V. This parameter belongs to an *EquationalModel* called *Voltage*. Voltage Source also contains a state machine with two states $s1$ and $s2$ respectively associated with voltage *Equation* $v1$ and $v2$. These *Equations* define the parameter V depending on the current state in the *VoltageSource*. For instance, $v1$ specifies the equation $V = 1.2V$ and the equation holds when the current state of the voltage source state machine is $s1$. Another equation can be defined for $v2$ and hold when the current state is $s2$. Additionally, V and *vout* have the same value because they are linked together.

The value of V can also be used in another equation, for instance to compute the power according to the current consumed by the powered element. V is then binded to the parameter of another equation.

4.5 Control View

This view coordinates the actions of the control-oriented element in each view in order to fullfill the system requirements. No description of its internal elements is done since it strongly depends on the kind of control realized. It could be done by state machine as well as by opaque actions in a specific language. However, it is important that the implementation fulfills the system requirements as well as causalities of the system (*e.g.,* a component can not go in the idle state if a critical action is currently executing). Consequently, we propose to specify only constraints on the schedule of the control events. These constraints can for instance be given by MARTE CCSL [18]. CCSL is a declarative language used to specify causal and temporal relationships between events. The main advantage of using CCSL is that it is well integrated in a model driven process (initially introduced in the MARTE time model) but it can also be used to generate observers that check the correctness of an hardware implementation [19]

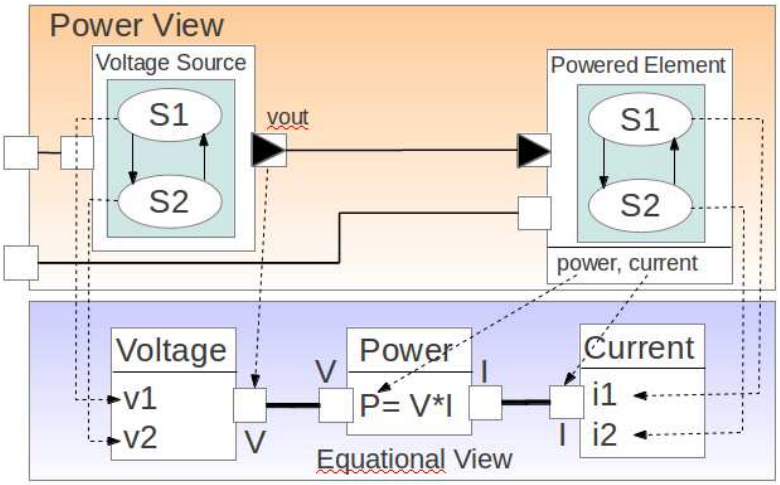


Figure 6: Equational View Example.

5 Multi-view modeling in UML

To ease the use of our approach, we propose an integration in UML by using the notion of UML profile. We added as few stereotypes as possible to ease the modeling of the multi-view concepts. Additionally, because MARTE and SysML already provide lots of relevant stereotypes, we reuse them as much as possible.

The component approach of the multi-view modeling UML composite structures. Its main element is the *StructuredClassifier* therefore the notion of *View* and *ViewElement* are specified by the *StructuredClassifier* definition. To organize the views in a unique structure, we extend a Class by the *System* stereotype. The owned *part* of a *System* are its views. For each view and each view element from the domain model, a stereotype is created. It allows adding OCL constraints on it. For instance, a class stereotyped as *VoltageSource* must contain at least two ports: one *supplyPort* and one *controlPort*.

The multi-view meta-models define three kinds of ports: *supplyPort*, *controlPort* and *clockPort*. MARTE specifies that a *flowPort* represents the flow of information between two components. We specify a *supplyPort* as a *flowPort* which carried information is a voltage value. A voltage is a measurable properties expressed by a quantity and a unit. We specify voltage values by typing a property by MARTE non-functional property (NFP). MARTE defines a library that contains basic NFP types commonly used in the design of real-time systems. We extend this library to add new NFP types that are necessary to type the view element properties; *NFP_Voltage* is an example of the ones we added.

MARTE also specifies the *Clock* stereotype that is the representation of discrete time or physical time. *ControlPort* and *clockPort* have these time notions and they are represented by this stereotype. The connection between port is done by *Nets* according to the view element meta-model. We specify *Nets* by UML connectors.

In the equational view, we use SysML *ConstraintBlocks* to specify the *EquationalModels*. To define the relationship between *EquationalModels*, we use parametrics. The binding between parameters is the *BindingConnector* defined in SysML.

There exist two associations between different views. The first one is an abstraction association between powered and clocked elements to a functional element. Consequently we reuse the

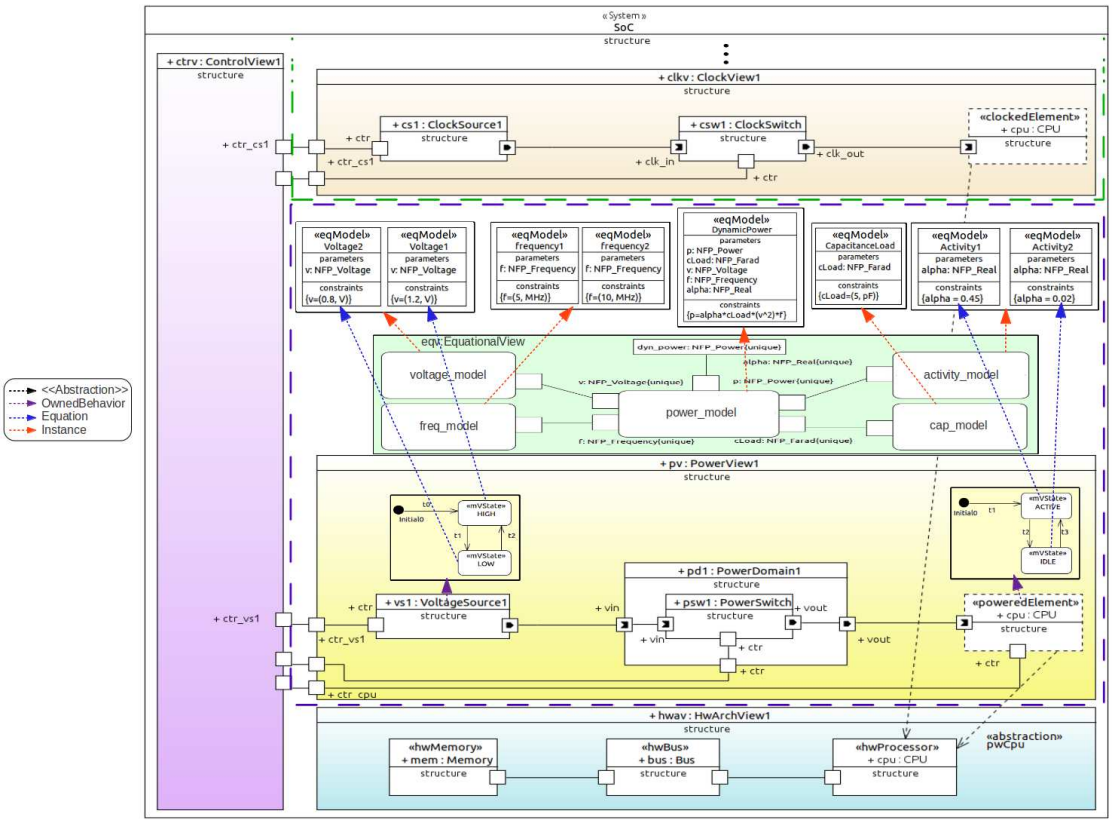


Figure 7: Multi-View System Example.

UML abstract association. The second one is an association between view element features and equational model parameters. In the UML, the *Use* represents this second association (*i.e.,* it specifies that the value of the equation is the value of the view element feature).

In power and clock view meta-models, we define that powered and clocked elements are the functional view elements from the power and clock points of view, respectively, therefore we specify this element in UML as a shared *Reference* of the part defined in the functional view.

The behavior of the view element are specified in UML by a state machine. However, we extend the UML *State* by *MultiViewState* to add a stereotype property that references the *Equation* as explained in section 4.4.

Finally, view elements such as *PowerSwitch* and *RetentionCell* are elements whose structure and behavior is the same in every model. Thus, we define a component library in UML to easily use them in a specific model. Each designer, if it reuse "classical" view element could put them in the library for future design.

6 Multi-view Model Example

We illustrate the use of the multi-view profile on a simple example of a system consisting of a CPU, a bus and a memory. Figure 7 presents the multi-view model of this example. As a first step, we create the main component, named SoC, and tag it with the *System* stereotype. Each

part of this component are then the views of the system. During the second step, the views are created as *Components* and tagged with the corresponding view stereotype (*e.g., ClockView* for ClockView1, *PowerView* for PowerView1). The part of the SoC system are then typed with the component created for the view. The first view to be defined is the functional view of the system (*HwArchView1*). This view is specified by using MARTE stereotypes: *cpu* is an *hwProcessor*, *bus* is an *hwBus* and *mem* is an *hwMemory*. This view is the backbone from which we create the other views.

The other layers could be specified concurrently. We focus now on the power view (*PowerView1*). In this example, we show only the *cpu* power management, even though the same definition can be applied to the other functional view elements through powered elements. It is important to note that every element defined in the functional view has a powered element in the power view to make explicit its supply. In this view, we define two parts (*vs1*, *pd1*). They are instances of the components *VoltageSource1* and *PowerDomain1* respectively. *VoltageSource1* and *PowerDomain1* use the *VoltageSource* and *PowerDomain* steteotypes to identify and to force the conformance to the view constraints (specified in OCL). The *cpu* reference uses *powerElement* stereotype and is an explicit reference to the *cpu* part from *HwArchView1*. *PowerDomain1* contains a part called *psw1*, which is an instance of *PowerSwitch*. The latter is defined in a library and it is imported to the model to be used.

VoltageSource1 contains a state machine that defines two states: *HIGH* and *LOW*. These states represent the possible voltage levels that *VoltageSource1* can generate. The *cpu*'s *poweredElement* contains also a state machine to define the activity state of the *cpu*. Its states are *ACTIVE* and *IDLE*. These states are linked to equational models from the equational view by using *equation* tagged values defined in the *MultiViewState* stereotype. They specified the voltage and activity values that hold for each state.

Similar to the power view definition, the clock view is specified (*ClockView1*). This view contains a clock source (*cs1*), a clock switch (*csw1*) and a clocked element (*cpu*). While not represented on Figure 7, the *cs1* is also associated with a state machine and with an equational model to specify the clock frequency according to its state.

Finally, we define the equational view where the dynamic power consumption of *cpu* is specified (*EquationalView*). We define an *equationalModel* called *Power_model* to add the parameters and the equation of the *cpu* power consumption. In the figure, the constraint block that defines *power_model* is *DynamicPower* and is defined by equation 2 in section 3.

In the system component, the *ctrv* part is also defined, which is the instance of *ControlView1*. Each time a control event is generated from *ctrv*, a state is changed in a view element, a value is changed in an equational model, and a new power consumption is evaluated. Note that the allowed frequency for the CPU depends on the value of its power supply. Such constraints are expressed to ensure the consistency of the control between the power view and the clock view.

7 Multi-view Transformation

Our multi-view model intends to capture various concerns of a system while allowing specialists to deal only with the concepts from their own specialty. The multi-view system model can then be used to drive specific analysis; *i.e.,* to use an analysis specific tool. To do it, To do it, we extract only the information needed by the tool we want to use and we transform it to a tool specific model. Next, we launch the process to analyze the studied property on the analysis tool.

We define this transformation in three steps as it is indicated on the 8: The first step is to transform the UML multi-view model to a multi-view domain model. This transformation reduces the UML model complexity to search the multi-view concepts creating a model that is conformed

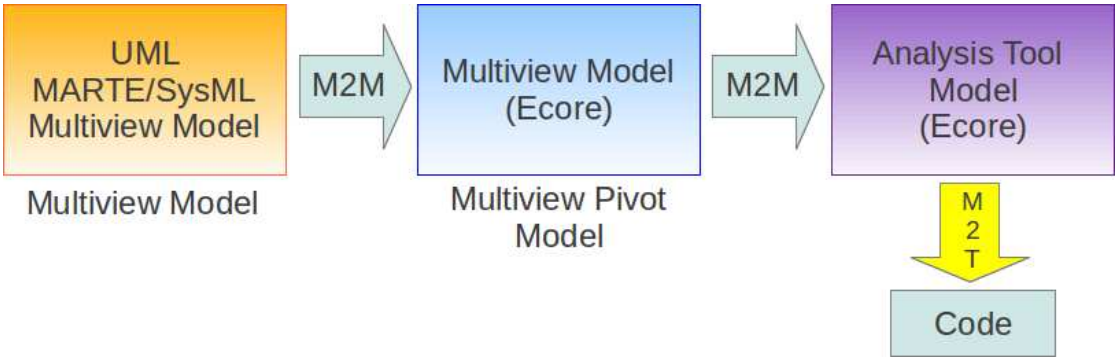


Figure 8: Transformation Overview.

to the multi-view meta-model from Section 4. In other words, we create a simplified model from the UML model easing the transformation to analysis tool models. We use *Ecore* as modeling language to create the multi-view meta-model. The second step is to transform multi-view model to analysis tool model. To define this transformation, a meta-model from the analysis tool model is defined. Next, the views that are concerned to the property that will be analyzed are identified and their concepts are transformed to the target model. Finally, the last step is to generate the model code to be imported on the analysis tool. We apply this transformation chain to analyze power consumption using Aceplorer.

7.1 Multi-View uml Profile To Multi-View Domain Model Transformation

During the transformation deployment, there are certain difficulty to locate the multi-view concepts in the UML model, because there are many UML concepts that are not used in the multi-view model and they have to be filtered to get the multi-view elements. Each time that a transformation from the UML multi-view model to an analysis tool model has to be defined, a multi-view filter must be used and then to execute the transformation process to the analysis tool model. To split the filter and transformation process from one domain to the analysis one, we create a sub-transformation from the stereotyped UMLmulti-view model to the multi-view domain model. This domain model is defined using *Ecore*.

The sub-transformation rules are defined as the identification of the multi-view concepts on the UML model, *i.e.,* the UML elements that have the multi-view stereotypes and the standard elements that are identified to be used in the multi-view (*e.g.,* Connectors, Abstract and Use association). Once these concepts are identified, they are transformed to its specified multi-view concept on the *Ecore* multi-view model creating a replica of the UML multi-view model to the specific multi-view model.

7.2 Aceplorer Domain Model

Aceplorer uses its own language to create the power model of a system to evaluate the power consumed by each system component. Figure 9 presents a simplified Aceplorer meta-model. An Aceplorer model has four elements: *Constraints*, *Components*, *Links* and *Types*. Module is an abstract element of *Constraint* and *Component* to define the base elements that is specified in a Aceplorer model. *Constraint* is a module that represents a value generator, *e.g.,* a power source

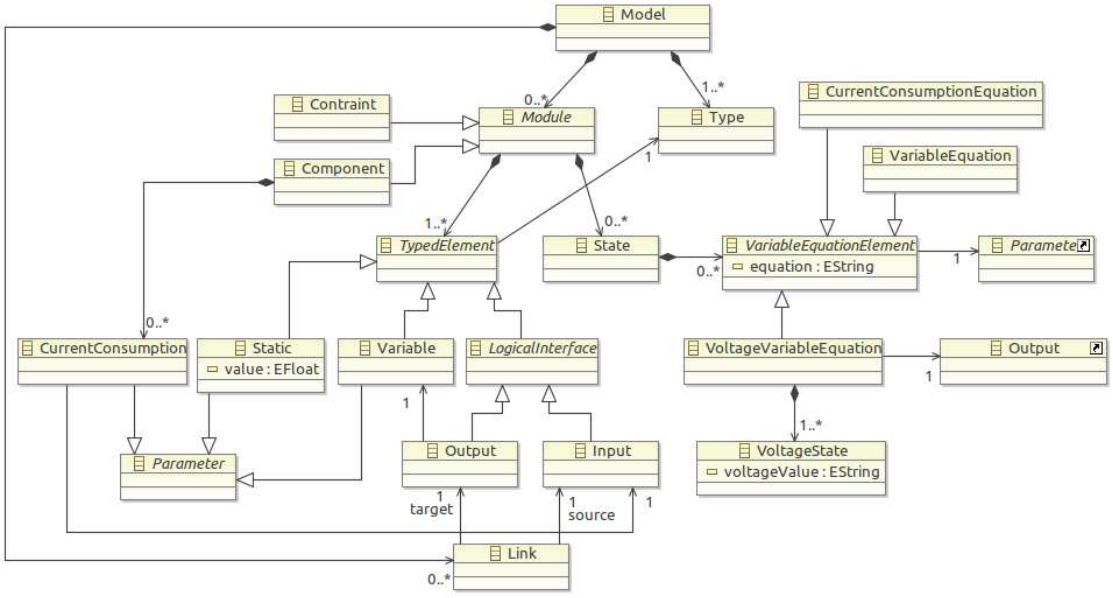


Figure 9: Simplified Aceplorer meta-model.

and a clock source. *Component* is a module that uses the generated values from constraints to evaluate the power consumption of a system element. A module contains *TypedElements* and *States*. Typed elements are associated to a type and they are specialized in *Static*, *Variable* and *LogicalInterface*. The latter is the abstraction of *Output* and *Input*, which are the module output and input ports to share variable values. *Link* connects an input with an output interface. As their name implies, *Static* is a typed constant and *Variable* is a typed variable. Component has *CurrentConsuptions* in addition to the module elements. Current consumption is associated to an input, more precisely to a voltage input. A state contains *VariableEquationElements*, which specializations are: *CurrentConsomptionEquation*, *VariableEquation* and *VoltageVariableEquation*. Variable equation contains the equation definition and this element is associated with a *Parameter*. Parameter is the unknown element in the equation and it can be a Current Consumption, a Static or a Variable. Voltage variable equation contains *VoltageState* where a *voltageValue* is defined. Voltage variable equation is associated to an Output, which represents that each time that a voltage stage changes, the voltage value assigned to output will be changed.

7.3 Multi-View to Aceplorer Transformation

We identify the corresponding elements from multi-view model to Aceplorer model in order to define the transformation rules between the two models. In this section, we present the transformation from a multi-view source to Aceplorer constraint and the transformation from a multi-view powered element to Aceplorer component.

Figure 10 depicts the first transformation. To build an Aceplorer constraint module, we identify the views that are concerning to this transformation from the multi-view model. We identify that Power View, Characterization View and Function View are involved in this transformation. In Power View, voltage source is transformed to Aceplorer Constraint. Additionally, voltage source port is transformed to Aceplorer constraint output port. The states that define the voltage level generated by the voltage source are transformed to constraint's states. In the

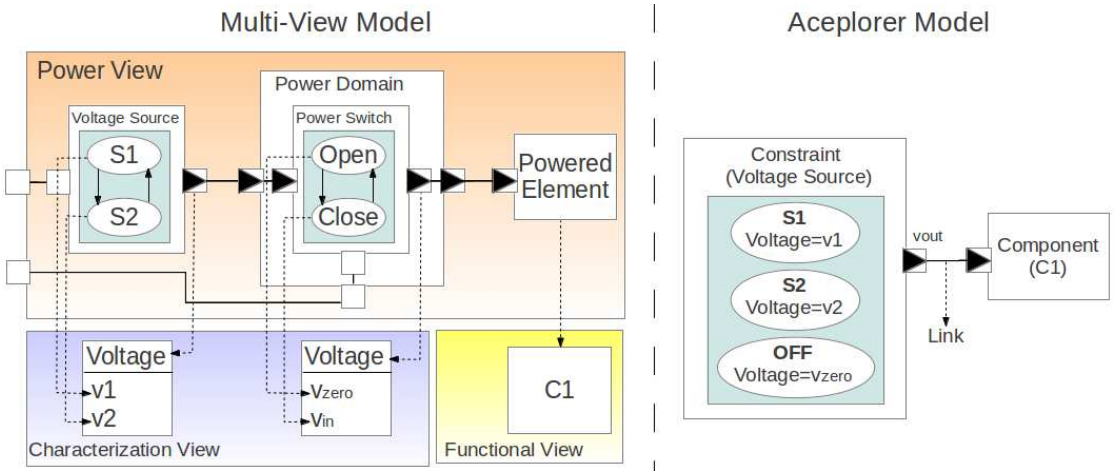


Figure 10: Transformation from Multi-view power source, power domain and powered element to Aceplorer Constraint Component and Link.

figure, voltage source port is associated to *Voltage*, which is a parameter defined in a characterization model in Characterization View. This parameter is transformed to Aceplorer voltage variable including its type. Voltage source's states are associated to the characterization model properties that define the voltage values. The equation that defines the voltage value is transformed to Aceplorer voltage state in Aceplorer constraint state. When a Power switch is defined between voltage source and a powered element, it is transformed in *Off* state in Aceplorer constraint. *Vzero*, which is the characterization model property that is associated to power switch *Open* state, is the value assigned to the voltage variable in *Off* state voltage state. Powered element is transformed to an Aceplorer component, including its supply port that is transformed to Aceplorer input voltage port. The component takes the name of the associated component of the powered element, according to the figure, *C1*. The supply net between power domain and powered element is transformed to Aceplorer link. Similar transformation is defined for Clock source in Clock view.

Figure 11 presents the transformation from the internal powered element description to Aceplorer component. Powered element is transformed to Aceplorer component using the name of the associated view element (*C1*). In Characterization View, different characterization models are defined to enable the power evaluation of *C1*. Power characterization model is defined as an equation that is function of voltage and current. Current value is given by a Current characterization model, which is a function of α and frequency. α value is given by Activity characterization model and frequency is given by Frequency characterization model. Powered element contains a state machine, whose states are associated to properties of activity characterization model ($\alpha 1$ and $\alpha 2$). An activity value is given from the current characterization model according to the active powered element state. The powered element states are transformed to Aceplorer component states. As each power element state is associated to an activity characterization model parameter and this characterization model is associated to the current characterization model, the equation of the current characterization model is transformed to current consumption equation in each Aceplorer component state and the α value is the activity characterization model value associated to the corresponding powered element state.

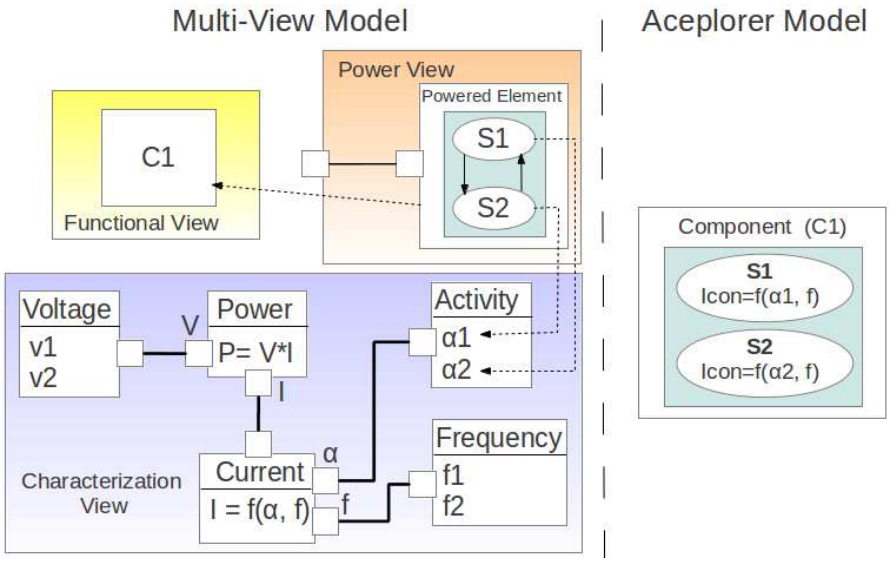


Figure 11: Transformation from Multi-view powered element to Aceplorer Component.

7.4 Aceplorer Code Generation

Once the transformation between multi-view model and Aceplorer model is done, we generate the Aceplorer model in Python code according to the Aceplorer library. This code is charged on Aceplorer tool and it is executed in order to create the Aceplorer model on the tool environment. This model contains the structure, states, variables and equation that are need to evaluate the system power consumption. To execute the evaluation, it is necessary to create an test scenario that is generated from the control view specification. For the moment, this transformation is not implemented.

8 Conclusion and Future Work

The development of SoC involves different activities, usually driven by specialists of different domains. A specialist is used to its domain specific concepts and the associated language. It leads to a model of the system split into different models, languages and tools so that their consistency is difficult to ensure. To avoid such issues, we propose a multi-view modeling approach where the specific specialist concepts are specified in a specific view, independently of a specific tool. It is then possible to federate the different point of views in a single model. Then, to benefit from existing tools, one can develop, once and for all, a transformation to an analysis specific tool that can be used almost transparently by designers. This multi-view approach is implemented as a minimal UML profile based on elements from the MARTE and SysML profiles. We implement views to model power and clock management techniques and we propose a transformation of such models to the Aceplorer power analysis tool.

Our future works are twofold. On one hand we will investigate the interns of the control view by using CCSL and others UML behavioral descriptions. On the other hand, we would like to study how the explicit links between the views together with the transformation to analysis tools could be used to parameterize a model bus and ease the co-simulation/analysis between different tools.

Contents

References

[1] Docea Power, "Aceplorer," http://www.doceapower.com/products-services/aceplorer.html, 2012, [Mar. 7,2012].

[2] "Ieee standard VHDL language reference manual," *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)*, 2009.

[3] "IEEE standard for Verilog hardware description language," *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, 2006.

[4] Accellera, "Unified power format 1.0," http://www.accellera.org/activities/p1801_upf, 2007.

[5] Accellea, www.accellera.org.

[6] IEEE, "IEEE standard for design and verification of low power integrated circuits," *IEEE Std 1801-2009*, pp. C1–218, 2009.

[7] S. I. Initiative, *Common Power Format Specification 2.0*, Silicon Integration Initiative, Inc., Feb 2012.

[8] "Silicon integration initiative," www.si2.org.

[9] OMG, "UML profile for MARTE," *Object Management Group*, vol. v1.1, Oct. 2010.

[10] OMG, "Systems Modeling Language," *Object Management Group*, vol. v1.2, Jun. 2010.

[11] T. Arpinen, E. Salminen, T. D. Hmlinen, and M. Hnnikinen, "MARTE profile extension for modeling dynamic power management of embedded systems," *Journal of Systems Architecture*, 2011.

[12] M. Hagner, A. Aniculaesei, and U. Goltz, "UML-based analysis of power consumption for real-time embedded systems," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, nov. 2011, pp. 1196 –1201.

[13] D. Helms, E. Schmidt, and W. Nebel, "Leakage in CMOS circuits an introduction," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, ser. Lecture Notes in Computer Science, E. Macii, V. Paliouras, and O. Koufopavlou, Eds. Springer Berlin / Heidelberg, 2004, vol. 3254, pp. 17–35.

[14] W. Zhang, J. Williamson, and L. Shang, "Power dissipation," in *Low-Power Variation-Tolerant Design in Nanometer Silicon*. Springer Berlin / Heidelberg, 2010, pp. 41–80.

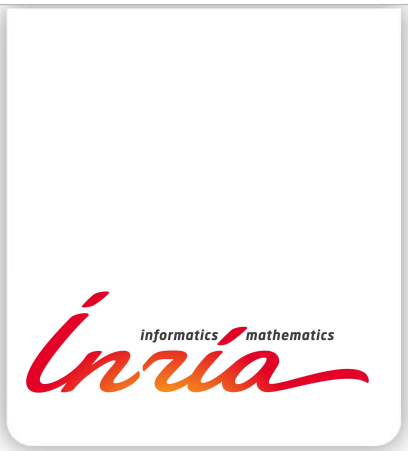
[15] D. Ma and R. Bondade, "Enabling power-efficient DVFS operations on silicon," *Circuits and Systems Magazine, IEEE*, vol. 10, no. 1, pp. 14–30, 2010.

[16] M. D. Grammatikakis, G. Kornaros, and M. Coppola, "Poweraware multicore SoC and NoC design," in *Multiprocessor System-on-Chip*, M. Hbner and J. Becker, Eds. Springer New York, 2011, pp. 167–193.

[17] S. Ahuja, A. Lakshminarayana, S. K. Shukla, S. Ahuja, A. Lakshminarayana, and S. K. Shukla, "Power reduction using high-level clock-gating," in *Low Power Design with High-Level Power Estimation and Power-Aware Synthesis*. Springer New York, 2012, pp. 119–129.

[18] C. André, J. DeAntoni, F. Mallet, and R. de Simone, *The Time Model of Logical Clocks available in the OMG MARTE profile*. Springer Science+Business Media, LLC 2010, July 2010, ch. 7, pp. 201–227.

[19] C. André, F. Mallet, and J. Deantoni, "VHDL Observers for Clock Constraint Checking," in *Symposium on Industrial Embedded Systems*. trento, Italie: IEEE computer society, Apr. 2010.



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399