Modeling Heterogeneous Points of View with ModHel'X

Frédéric Boulanger, Christophe Jacquet, Elyes Rouis, and Cécile Hardebolle

SUPELEC, 3 rue Joliot-Curie, 91192 Gif-sur-Yvette Cedex, France
e-mail: {firstname}.{lastname}@supelec.fr

Abstract. Non-functional properties (NFPs) concern various characteristics of a system (cost, power, QoS). These characteristics belong to different models of the system, built by different design teams, using different formalisms. Therefore, the design of a system includes a number of domain-specific modeling languages, used to express various NFPs. This paper focuses on the heterogeneity of the points of view on the system. We show that "multi-view" approaches which do not rely on a unique underlying model appear better-suited to express NFPs than model weaving or annotations. However, existing approaches in this category do not yet support model execution. We introduce a multi-view extension to ModHel'X, a framework for executing heterogeneous models, and we show how it can be used for modeling non-functional characteristics of a system and expressing NFPs. A key point of this approach is that it relies only on the core concepts of ModHel'X, but uses them in new ways.

1 Introduction

The design of a system requires to take into account different concerns besides the core function of the system. Such concerns comprise cost, reliability, performance, or power consumption. Concerns are generally classified as functional or non-functional. While functional concerns used to be the main matter in the development process, non-functional concerns are now taken into consideration earlier because they constrain the design of the functional part of the system. This is why, in the context of Model Driven Engineering, different techniques have been developed to model non-functional characteristics of systems. One of the goals of these techniques is to allow the analysis of non-functional properties of a system together with its functional behavior during design.

Obviously, different concerns require different types of models since they rely on different domains (power, security, etc.), which therefore implies the use of suitable modeling languages or paradigms (differential equations, state machines, statistics). In consequence, several domain-specific modeling languages (DSMLs) are used when modeling a system with respect to different concerns. We focus here on the problems caused by the heterogeneity of the modeling languages which are used to model functional and non-functional features.

In this paper, we present an approach in which the system is modeled from different points of view which correspond to different concerns. A view of the system may regard either its functional or its non-functional characteristics. Each

view is modeled using a modeling language which is suitable for the considered concern. Since different views of the system may be interdependent (for instance, the power consumption of a system, which is studied in one view, may depend on the state of the system, which is studied in another view), we provide means to express dependencies among views in order to ensure the coherence of the different models of the system. Our goal is to enable the simultaneous execution of different models which represent the system under different points of view. Our approach relies on an existing framework, called ModHel'X [1].

First, we present related work and motivate our approach in section 2. Then, in section 3, we introduce the approach we propose. We briefly present ModHel'X, a framework built for assembling heterogenous models through hierarchical composition in 3.1. Then, starting from the basic concepts of ModHel'X, we propose an architecture for defining the coherence of several views of a system in 3.2. This architecture introduces no new concepts in ModHel'X, but uses the composition mechanisms of ModHel'X in two ways: for aggregation (building more complex systems by assembling simple systems), and for superposition (adding layers for defining different aspects of a system). We detail the mechanisms that we propose to model interdependencies among views. In section 4, we illustrate the principles of our approach on an example. Finally, we discuss open issues in section 5, before concluding.

2 Related Work

Non-functional characteristics of a system may be expressed in a number of ways. First, crosscutting concerns (including non-functional features) may be modeled separately from the functional core of the system, and later be "woven" into it. This constitutes the aspect-oriented approach. Second, non-functional features may be expressed as annotations to the core model, as done in UML with the MARTE profile. Third, the features of the system, both functional and non-functional, may be constructed in parallel, in relation to one another, through various "views" on the system, corresponding to various fields of concerns.

2.1 Aspects for Weaving Non-Functional into Functional Features

In the field of programming, object-oriented programming allows the structuration of an application in a hierarchical fashion. However, non-functional, crosscutting concerns are interwoven in the object-oriented code. For instance, in an application that performs logging, virtually every class is "polluted" with logging-related code, which increases complexity. To allow for more modularity, aspect-oriented programming (AOP) was introduced [2]. Crosscutting concerns like logging are extracted from the main code and called *aspects*. An aspect is composed of an *advice*, a portion of code to add to the main program, and a *pointcut*, an expression which specifies at which points in the main program the advice is to be added. The action of automatically combining a base program with a number of aspects to get a new program is known as *weaving*. Well-known

implementations of aspect-oriented programming include, for the Java language, AspectJ [3] and Hyper/J [4].

AOP enables one to weave non-functional behaviors into a core program. Likewise, one may wish to be able to weave non-functional aspects into a functional model. To this end aspect-oriented modeling (AOM) was introduced [5]. An aspect-oriented design model consists of a primary (functional) model and a set of aspect (non-functional) models. Each aspect model describes a feature that crosscuts elements in the primary model. The primary model and the aspect models can be woven automatically to obtain a global model which contains both functional and non-functional features [6].

Theme/UML [5] extends the UML standard to explicitly support modularization and the composition of concerns. The extensions include a new type of classifier called *theme* and a new type of relationship called *composition relationship*. A theme contains two types of UML diagrams, class diagrams and sequence diagrams. Themes correspond to functional or non-functional concerns in the design of the system. When a theme corresponds to a crosscutting concern, it is called an *aspect theme*. Composition relationships define how themes are woven with each other: concepts and behaviors may be composed between themes [7].

RAM (Reusable Aspect Models) [8] follows the same philosophy as Theme/UML, but crosscutting concerns can be described using three types of UML diagrams: class diagrams, sequence diagrams and state diagrams.

2.2 Annotations for Expressing Non-Functional Properties

Non-functional features are not necessarily intended to be woven into the system. Instead, one may wish to express properties to be verified on the system. Starting from a functional model of the system, which states what the system shall do, one can add *annotations* to state how the system shall do it – non-functional properties. For instance, when using UML to model a system, one can use various profiles that allow the addition of domain-specific annotations [9], for instance SPT (UML Profile for Schedulability, Performance and Time Specification) or more recently MARTE (Modeling and Analysis of Real-Time and Embedded systems) for real-time constraints, QoS&FT (UML Profile for Modeling Quality of Service and Fault Tolerance) for QoS constraints, etc.

In this context, annotating a UML model amounts to attaching information to model elements, which can be done in two ways [9]. First one can use tagged values, which are value slots associated to stereotype attributes. Hence a tagged value applies to one element only. Second, one can use OCL constraints, which are more flexible, and apply to several elements. A syntax must be given for non-functional properties, in order to define their types, their values and possibly the source of the values (required values, assumed values, achieved values, etc.).

The annotations may be used for two purposes: analyzing a model, and constraining the synthesis of the actual system from the model. For instance, MARTE annotations to a UML model may be used to check the model for schedulability (analysis), but also to choose a scheduler or allocate resources when generating an implementation (synthesis).

2.3 Multiple Views for Representing a System from Various Angles

Adding annotations to a core model does not allow one to create complex descriptions of the system from the non-functional points of view. Instead, a designer or a team of designers may wish to be able to build full-fledged models of a system from several points of view in parallel, while ensuring consistency among the points of view. This is called *multi-view modeling*, and it can be approached in two ways [10]. The first and most common one is to consider that views are projections of a hybrid *reference model*. In this approach, the reference model aggregates all the information about the system. Views are queries on the reference model; they perform *projections* hiding irrelevant information when studying a particular aspect of the system. The second approach uses views as partial definitions or expected observations of the system. Therefore, there is no explicit reference model of the system. However, information about the system can be obtained dynamically by combining information observed in different views. The different views must be linked so that global properties can be computed from properties which come from different models. This approach is similar to the drawing of a front view, a top view and a side view in order to model a building: it provides global information about the building without requiring to build a 3D global model.

The work of Attiogbé et al. [11] and View-based UML (VUML) [12] belong to the first approach: they are centered around a reference model.

In the framework of Attiogbé et al., there is a clear distinction between an *abstract reference model*, which concentrates all information available about the system, and *specific models*, which are built from the reference model. The specific models constitute various points of view on the system. Analysis (verification of properties) can be performed on specific models. Modifications on one specific model are fed back into the reference model and then to the other specific models.

VUML is a UML profile for multi-view modeling. It is based on the concept of *multi-view class*. A multi-view class consists of a default (common) view and a set of specific views related to the common view through extension relations. OCL constraints enforce consistency among views. As UML is the underlying framework, there is indeed a reference model, the UML model itself.

Maintaining a reference model requires to update this model each time a view is changed, as well as to propagate the change to the other views. Moreover, the reference model must be an instance of a meta-model which encompasses the semantics of every possible view of the system. Defining the semantics of such a union language is very difficult, and reference models are therefore purely syntactic. This makes the verification of the global coherence of a model a difficult problem. That is why the second approach, in which there is no explicit reference model, seems promising. Let us now see two examples of this approach: the work of Benveniste et al. [13] and Rosetta [14–16].

Benveniste et al. introduce a framework for system design, inspired by the Tagged Signal Model [17], in which components are characterized by *contracts*. A contract comprises assumptions and guarantees, which are sets of acceptable

behaviors, or "traces". An implementation of a component conforms to a contract if the guarantees are satisfied whenever the assumptions are satisfied. Several contracts can be provided for a given component, which allows the designer to express both functional and non-functional properties.

Rosetta is a system specification language based on formal (coalgebra-based) semantics. Central in Rosetta is the notion of *facet*. A facet corresponds to an observable aspect of a component, either functional or non-functional. Facets which model different components can interact with each other. Also, several facets of the same component may be combined with a product operator, meaning that all of them must be satisfied simultaneously, which ensures consistency among views. Rosetta can therefore be used to model the structure of systems, as well as different points of view on system components.

However, the framework of Benveniste et al. and Rosetta remain at the level of specifications, which means that models cannot be executed directly. Therefore a designer who uses one of these tools to specify a system will have to use another tool in order to get an executable implementation.

In the remainder of this paper, we introduce a proposal which adds multi-view modeling capabilities to ModHel'X, an existing framework for heterogeneous modeling. Contrary to the aforementioned frameworks, ModHel'X is capable of *executing* a model: it can calculate its behavior. ModHel'X belongs to the second approach, since it considers views as layers that add information to each other in order to build an *implicit* model of a system. The originality of the approach presented here is to specify the links between views using the *same mechanisms* that were designed for the hierarchical composition of heterogeneous models. It therefore provides a uniform framework for combining heterogeneous models both for hierarchical composition and for the superposition of views.

3 Multi-view Modeling with ModHel'X

3.1 ModHel'X

We start here with a very short introduction to the basic concepts of ModHel'X, so that we can show later how these concepts can be used for multi-view modeling. ModHel'X [1] is a framework for modeling heterogeneous systems. It relies on a generic metamodel for describing the structure of models, and on a generic execution engine for interpreting such structures.

In ModHel'X, the interpretation of a model by the generic execution engine is directed by a *model of computation*. A model of computation is a set of rules for combining the behaviors of a set of components into the behavior of a model. Well-known examples of models of computation include discrete events, synchronous data-flows, finite state machines and continuous time [18]. In ModHel'X, a model of computation dictates the rules for scheduling the components of a model, for propagating values between components, and for determining when the computation of the state of a model is complete. The concept of model of computation is essential in ModHel'X because it allows ModHel'X to support the execution of models described using different modeling languages as

well as the execution of models composed of sub-models which are described using different modeling languages (which are called "heterogeneous models"). In order for ModHel'X to support a given modeling language, an expert of this language must describe the corresponding model of computation. Once described in ModHel'X, this model of computation is used by the generic execution engine in order to interpret any model described using the chosen modeling language. This mechanism is further detailed in [1].

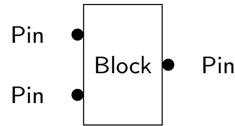

Fig. 1. A block

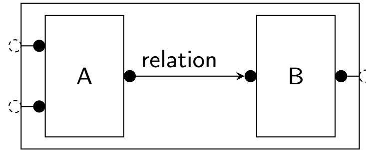

Fig. 2. A composite

The elementary unit of behavior in ModHel'X is the *block*, as shown on figure 1. A block is defined (a) by its interface which is composed of *pins*, and (b) by an `update` operation which allows the observation of the behavior of the block through its interface. Pins are used to send and receive information and they define what is observable from a block. The behavior of a block is observed by requiring an update of its interface. The update operation requires that the block take its inputs into consideration and update its outputs according to the inputs and its current state.

Blocks can be assembled by setting *relations* between their pins. A *composite block*, as shown on figure 2, is composed of a set of blocks with relations between their pins. Its interface is a subset of the union of the interfaces of its blocks (it can hide their pins or make them visible from the outside). On the figure, the pins exposed at the interface of the composite block are shown as dashed circles in order to emphasize that they are just pins of the internal blocks which are made visible through the interface of the composite block. A *model of computation* is used to define the semantics of the interconnected blocks. A composite associated to a model of computation forms a *model*, as shown on figure 3.

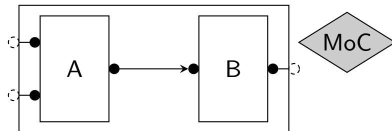

Fig. 3. Model

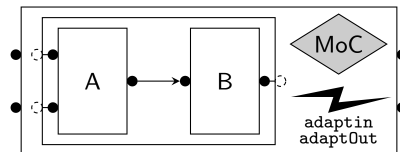

Fig. 4. Interface block

ModHel'X has support for hierarchical composition and heterogeneity thanks to *interface blocks*. An interface block is a kind of block whose behavior is defined by a model (which is therefore "inside" the block). An example is shown on figure 4. The model which defines the behavior of an interface block may use a different (internal) model of computation than the (external) one used for the model in which the interface block is used. Hierarchy through interface blocks is the mechanism used to combine heterogeneous models in ModHel'X. The semantic adaptation between the two models of computation is specified by defining the `adaptIn` and `adaptOut` operations of the interface block. These operations are in charge of adapting the control (scheduling, synchronization) and the data between the internal model of computation and the external one. An interface block has its own pins, and the data available on those pins is processed by its `adaptIn` and `adaptOut` operations. Typically, `adaptIn` uses the valuation of the pins of the interface block to compute scheduling and status information as well as a valuation of the pins of the composite block. `adaptOut` uses status information and the valuation of the pins of the composite block to give a valuation of the pins of the interface block.

3.2 Multiple Views of a System in ModHel'X

A block in ModHel'X represents an observable behavior. When we consider a component that we want to model from different points of view, each view can be considered as an observable behavior and represented as a block in ModHel'X. With this approach, a "real world" component may be modeled by a block C_{φ} for its functional behavior, by a block C_{P} for its power consumption, and by a block C_{T} for its thermal behavior.

By considering that a view of a component is a ModHel'X block, we can build composite views by assembling components and interpreting this composition according to a model of computation. This is *composition by aggregation*, the usual way of building models with block diagram tools, for which the metamodel of ModHel'X was built.

Actually, there can be *several* of these views for a given system: in the same way we build a functional view of a system by assembling blocks which represent the functional behavior of the components of the system, we can also build a power consumption view of the system by assembling the blocks which represent the power consumption behavior of these components. Each of these two models computes one aspect of the behavior of the system, and they are interdependent.

3.3 Multi-view Models and Coherence among Views

Indeed, the power consumption of a system may depend on its functional behavior, and this dependency should be stated in the model of the system. This leads us to consider that blocks may be combined in two ways: *by aggregation*, in order to build larger models (as seen in the previous section); and *by superposition*, in order to build multi-view models. Composition by superposition amounts

to maintaining coherence between models which represent different views of a system so that the behaviors observed in the different views are coherent.

The abstract syntax (block, pin, relation, MoC) of ModHel'X has been designed for composition by aggregation, but we will now show that it can also be used for composition by superposition. In a multi-view model, coherence between views can be denoted by relations between the ports of the views, just as the composition of the behaviors of several blocks is denoted by relations between their ports. Figure 5 shows two views of a component C, which are superposed so that observations on view C_P are coherent with observations on view C_φ. The relations between the pins of the two views represent the *coherence relation* between their behaviors, and the semantics of the interconnection of these views is defined by a model of computation, as for any model in ModHel'X.

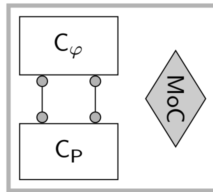

Fig. 5. Superposition of views

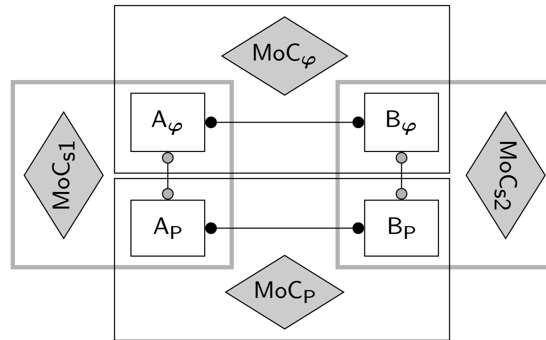

Fig. 6. Superposition and aggregation

Figure 6 illustrates the general pattern for using both composition by aggregation and composition by superposition. Views A_φ and B_φ are composed by aggregation according to MoC_φ in order to build a functional view of a system. Views A_P and B_P are also composed by aggregation according to MoC_P to build a power consumption view of the system. The coherence of these views is obtained by maintaining the coherence of A_φ and A_P, and of B_φ and B_P. This is achieved by composing these views according respectively to MoC_{s1} and MoC_{s2}.

4 Illustrative Example

In order to illustrate our approach of multi-view modeling, we present on figure 7 a simple example in which a thermostat regulates the temperature in a room by switching a heater on or off. For the sake of simplicity, the border and pins of the composite blocks are omitted on the figure, and the internal model of interface blocks is represented in a form (state machine, pseudo code) which is easier to understand than a strict ModHel'X model. In the functional view, a thermometer (denoted by T°) provides the thermostat with the ambient temperature. The

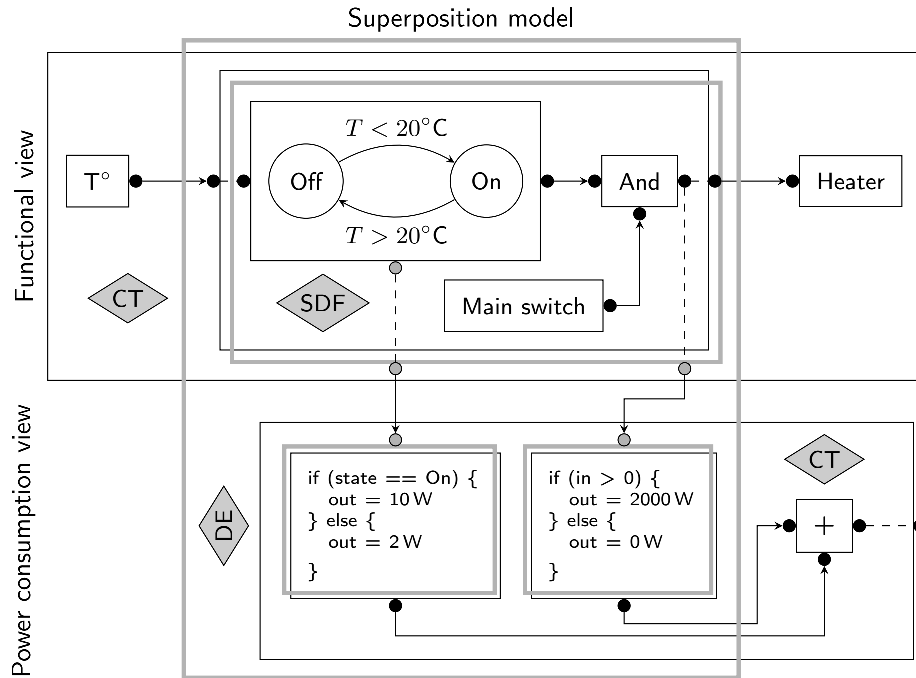

Fig. 7. Example multi-view model

thermostat is modeled as a state machine which switches between the On and Off states when the temperature goes below or above 20°C. When the main switch is on, the output of the thermostat is used to drive a heater. A synchronous data flow (SDF) model of computation is used for the model of the thermostat, and the continuous time (CT) model of computation is used to combine the behaviors of the thermometer, the thermostat and the heater.

In the power consumption view, the left block models the power consumption of the thermostat: 10 W when in the On state and 2 W when in the Off state. The power consumption of the heater is modeled by the next block (2000 W when powered). The last block in the power consumption view is an adder which computes the power consumption of the system by adding the power consumption of the thermostat and of the heater. The continuous time (CT) model of computation is used for this view.

Let us examine now the coherence of the views. The power consumption of the thermostat depends on its internal state, so this information must be provided by the functional view to the power consumption view. However, for the sake of modularity, the internal state of the thermostat is an implementation detail that should not be revealed to other blocks in an aggregation model. We can consider it as protected information that should be available only for blocks whose behavior is tightly coupled to the functional behavior of the thermostat.

The model of the thermostat has therefore two interfaces, one for composition by aggregation, and one for composition by superposition. ModHel'X has support for this since a given model may be wrapped into several interface blocks. On figure 7, aggregation interfaces are shown using solid black rectangles and pins, and superposition interfaces are shown using thick gray rectangles and gray pins.

In a similar way, the power consumption view of the heater needs to know if the heater is powered. This is determined by the output of the And block in the functional view, which is therefore published in the superposition interface of the thermostat. This illustrates that pins that appear in the superposition interface of a model can provide information which comes either from aggregation interfaces (the output of the And block) or from superposition ones (the state output of the thermostat).

In ModHel'X, the role of the interface blocks is not only to define different interfaces for a given model, but also to adapt the behavior of this model to different models of computation. Here, the coherence of the two views of the system is expressed using the discrete event (DE) model of computation. The superposition interfaces therefore perform the semantic adaptation of the model of computation used for these models with DE. The aggregation interfaces of the same models also perform a semantic adaptation, but with CT. Let us examine the semantic adaptation of the model of the thermostat, which uses SDF. Its aggregation interface adapts between SDF and CT by making the internal model react only to significant changes in the continuous input signal, and by holding a constant continuous output when no data sample is produced by the model. Its superposition interface adapts between SDF and DE by producing output events only when the automaton changes its state or when the output of the And block changes.

5 Discussion

We have shown that the basic meta-model designed to represent the structure of hierarchical heterogeneous models in ModHel'X can also be used to represent the coherence relations between several views of a system. However, we did not give the precise semantics of this representation. The generic execution algorithm of ModHel'X was designed with the hypothesis that a given block belongs to only one model because only composition by aggregation was possible. Adding composition by superposition allows a block to belong to several models, behind different interfaces. Since the observations that are made of this block through different interfaces are interdependent, the result of the execution of a multi-view model may depend on the order in which the different views are observed. In the simple example of the thermostat, the power consumption view depends on the functional view, and there is no other dependency, so we can merely compute the behavior of the functional model, then the behavior of the superposition model, and last, the behavior of the power consumption model. However, if there were cyclic dependencies between these different views, defining the semantics of the multi-view model would be more difficult. Actually, the current version of the

execution engine of ModHel'X does not support it. In the following, we illustrate the problem and present some possibilities to address it.

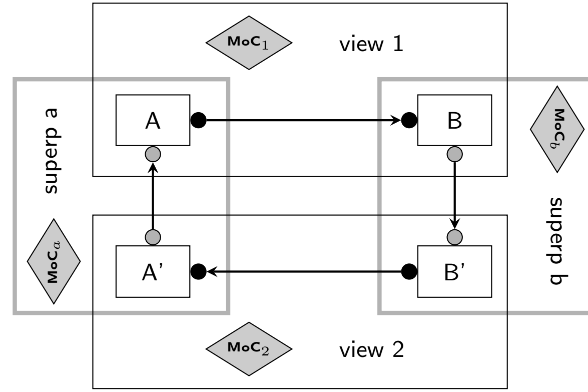

Fig. 8. Cyclic dependencies

First, to show what a multi-view model with cyclic dependencies looks like, let us consider the example of figure 8. view 1 and view 2 model the behavior of the system according to different points of view. superp a and superp b are used to maintain the coherence between the views. One may consider that A and B are views of some components of the system according to the point of view of view 1, and that A' and B' are views of the same components according to the point of view of view 2. Arrows are used to show that the behavior of B depends on data produced by A, the behavior of B' on data produced by B, the behavior of A' on data produced by B', and finally, the behavior of A depends on data produced by A'.

Such cyclic dependencies are common and appear naturally in models of systems. In the example of the thermostat and the heater, we could have modeled the fact that the heater makes the temperature increase when it is on. Such a model is shown on figure 9. It uses three views to model the functional behavior of the system, its power consumption behavior, and its thermic behavior. The functional and power consumption views are similar to the ones shown on figure 7, with simplified schematics for complex blocks. The thermic view models the production of heat by the heater, as well as the thermic behavior of the room. A second superposition model is used to maintain the coherence between the power consumption view and the thermic view. A third one is added to maintain the coherence between the thermic view and the functional view, because the temperature measured by the thermometer depends on the thermic behavior of the room. The relation between the output of the room thermic model and the input of the thermometer in the functional view closes a dependency loop: the temperature in the room depends on the output of the thermostat (because it drives the heater), which depends itself on the temperature. However, when

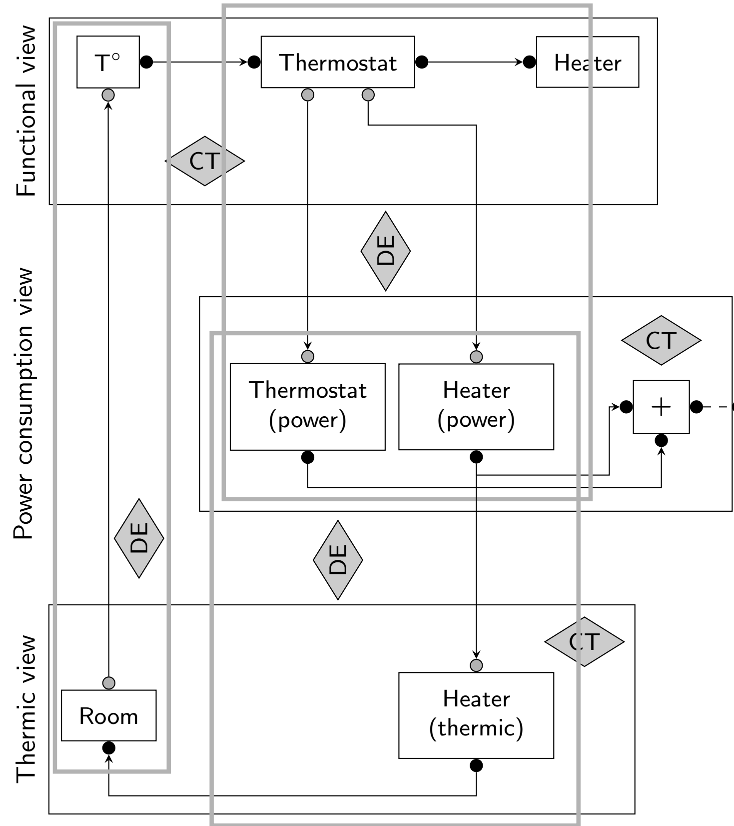

Fig. 9. Model of the thermostat with a dependency cycle

the heater is switched on, the temperature does not rise instantaneously, so the dependency cycle is broken by a delay: the output of the thermometer depends on the *current* room temperature, but the heater acts on the *future* temperature.

Introducing delays in communications between ports is a way of solving the issue of cyclic dependencies. However, we have seen that the semantics of the relations between blocks is determined by a model of computation. Therefore, in general, and for instance on figure 8, we cannot assume that the models of computation used in view 1, view 2, superp a and superp b all have "propagate with delay" semantics for relations.

Another way of considering cyclic dependencies is to regard the cycle as an equation of the form $\vec{x} = f(\vec{x})$, with $\vec{x}$ the vector of values given to their pins, and to choose the value of the pins in order to solve this equation. The f function represents the behaviors of the blocks and the rules used by the model of computation to combine these behaviors. If the equation has no solution, the model has no behavior and is ill-formed. If there are several solutions to the equation, we can either consider the model as ill-formed, or pick a random

solution (this allows non-deterministic behaviors), or choose a solution according to some criterion (for instance, the smallest or the largest solution according to some order). Indeed, ModHel'X allows cyclic dependencies in models, provided that a solution can be computed by repeated observations of the blocks. With the equation above, this means that ModHel'X allows cyclic dependencies when $\exists k_0 \in \mathbb{N}, \forall k \geq k_0, f^k(\vec{x_0}) = f^{k_0}(\vec{x_0})$. Starting from an initial guess $\vec{x_0}$ for the values of the pins, we repeatedly apply f, which represents the observation of the blocks and the propagation of the observed values according to the relations between pins. When two successive series of observations yield the same values for the pins, a fixed point $\vec{x_f}$ has been reached and gives the behavior of the model for this observation.

The fixed point which is reached depends on the individual behavior of the blocks, but also on the model of computation, which determines the order in which blocks are observed, and how the values associated to the pins are propagated according to the relations. When ModHel'X computes the behavior of a model, it relies on the MoC of the model to compute $f(\vec{x})$. In our multi-view example, four models of computation are involved (one for each view, and one for each superposition model), so it is much more difficult to determine f, and therefore to find its fixed-point, if it exists. We are studying a new version of the execution engine of ModHel'X which observes the structure of multi-view models in order to schedule the observations of their views and superposition models in order to compute the behavior of the whole model as a fixed-point.

6 Conclusion

Non-functional aspects of a system can be captured in models which focus on a specific point of view on the system. Such models often require different modeling formalisms or DSMLs because they belong to different domains. In consequence, different models which represent views on different aspects of a system are often heterogeneous. Since non-functional properties may be interdependent (e.g. temperature, dissipated power and clock frequency in a CPU), and may also depend on the functional behavior of the system, it is necessary to model these dependencies so as to ensure the coherence of the different models of the system.

The approach of multi-view modeling presented in this paper relies on ModHel'X, a framework for heterogeneous modeling and design, to model each view of a system. Each view combines the behavior of components according to the rules of a model of computation which is equivalent to the semantics of a DSML. Components are aggregated into models and a model can be wrapped in a component to give a hierarchical structure to a larger model.

In the work presented in this paper, we introduce superposition, a new way of combining components to express the coherence between different views of a system. In this approach, a multi-view model of a system consists of several views (which are models of the system built according to different points of view), and of superposition models which propagate information between the views in order to ensure their coherence.

This approach has the main advantage of reusing the basic mechanisms designed in ModHel'X for hierarchical heterogeneous modeling. It only assumes that a given model can be encapsulated in several interfaces for composition by aggregation and superposition. It therefore provides the base of a unified framework for building heterogeneous multi-view models.

References

1. Boulanger, F., Hardebolle, C.: Simulation of Multi-Formalism Models with ModHel'X. In: Proceedings of ICSTW'08, IEEE Comp. Soc. (2008) 318–327
2. Elrad, T., Filman, R., Bader, A.: Aspect-oriented programming: Introduction. Communications of the ACM **44**(10) (2001) 28–32
3. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.: An overview of AspectJ. Lecture Notes in Computer Science (2001) 327–353
4. Ossher, H., Tarr, P.: Multi-dimensional separation of concerns and the hyperspace approach. Kluwer Int. Series In Engineering And Comp. Science (2002) 293–324
5. Clarke, S., Baniassad, E.: Aspect-oriented analysis and design. Addison-Wesley Professional (2005)
6. Reddy, Y., Ghosh, S., France, R., Straw, G., Bieman, J., McEachen, N., Song, E., Georg, G.: Directives for composing aspect-oriented design class models. Lecture Notes in Computer Science **3880** (2006) 75–105
7. Cahill, V., Clarke, S.: Separation of distributed real-time embedded concerns with theme/UML. In: Proc. of the MOMPES 2008 Conference, IEEE (2008) 27–33
8. Kienzle, J., Al Abed, W., Klein, J.: Aspect-oriented multi-view modeling. In: Proc. of AOSD'09 (Aspect-oriented software development), ACM (2009) 87–98
9. Espinoza, H., Dubois, H., Gérard, S., Medina, J., Petriu, D., Woodside, M.: Annotating UML models with non-functional properties for quantitative analysis. Lecture Notes in Computer Science **3844** (2006) 79–90
10. Périn, M.: Spécifications graphiques multi-vues: formalisation et vérification de cohérence. PhD thesis, Université de Rennes I (2000)
11. Attiogbé, J.: Mastering Specification Heterogeneity with Multifacet Analysis. In: Proceedings of ICSTW'08, IEEE Comp. Soc. (2008) 121–130
12. Nassar, M.: VUML: a Viewpoint oriented UML Extension. In: Proc of the 18th IEEE Int. Conference on Automated Software Engineering. (2003) 373–376
13. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple Viewpoint Contract-Based Specification and Design. In: Proceedings of FMCO 2007. Number 5382 in LNCS, Springer (2008) 200–225
14. Alexander, P., Kong, C.: Rosetta: Semantic support for model-centered systems-level design. Computer **34**(11) (2001) 64–70
15. Kong, C., Alexander, P.: The rosetta meta-model framework. In: Proc. of the IEEE Engineering of Computer-Based Systems Symp. and Workshop. (2003) 133–141
16. Streb, J., Alexander, P.: Using a lattice of coalgebras for heterogeneous model composition. In: Proc. of the Multi-Paradigm Modeling Workshop. (2006)
17. Lee, E., Sangiovanni-Vincentelli, A.: A framework for comparing models of computation. IEEE Transactions on computer-aided design of integrated circuits and systems **17**(12) (1998) 1217–1229
18. Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity – the Ptolemy approach. Proceedings of the IEEE **91**(1) (2003) 127–144