Modèles, systèmes, hétérogénéité

Frédéric Boulanger

Université Paris-Sud XI
UFR scientifique d'Orsay

Modèles, systèmes, hétérogénéité

Mémoire présenté pour
l'habilitation à diriger les recherches

par

Frédéric BOULANGER

Supélec Sciences des Systèmes (E3S)
Département Informatique

Soutenue le 8 juin 2011 devant le jury :

Christian ATTIOGBÉ	Université de Nantes (Rapporteur)
Gérard BERRY	INRIA Sophia-Antipolis
Frédéric BOUSSINOT	INRIA Sophia-Antipolis (Rapporteur)
Edward A. LEE	Université de Californie à Berkeley (Rapporteur)
Pierre-Yves SCHOBBENS	Facultés Universitaires Notre-Dame de la Paix, Namur
François TERRIER	CEA Saclay
Guy VIDAL-NAQUET	Supélec et Université Paris-Sud 11

Remerciements

Au risque d'oublier quelqu'un ou de ne pas respecter certaines préséances, je tiens à souligner ici le rôle important qu'ont tenu certaines personnes, tant dans ma formation scientifique que dans ma façon d'organiser mes idées et d'enseigner.

Pour leur influence sur mon parcours scientifique, par ordre d'apparition dans ce parcours :

- Nicolas Graner (Université Paris-Sud XI)
- Philippe Volle (ESIEA, Paris)
- Guy Vidal-Naquet (Supélec - Université Paris-Sud XI)
- Paul Le Guernic (IRISA Rennes)
- Charles André (INRIA Sophia Antipolis)
- Paul Caspi (Verimag, Grenoble)
- Gérard Berry (INRIA Sophia Antipolis)
- Edward A. Lee (UC Berkeley)
- Stephen A. Edwards (Columbia University)
- Alain Girault (INRIA Rhône-Alpes)
- Xavier Warzee (Thomson-CSF Optronics)
- Denis Aulagnier (Thomson-CSF Detexis)
- François Terrier (CEA)

Tous mes collègues du département informatique de Supélec, pour les nombreuses discussions et les échanges autour de la pédagogie et de la recherche.

Parmi eux, et plus particulièrement pour les discussions parfois très animées qui m'ont permis de tester (quelques fois à leurs dépends) les limites du rôle d'avocat du diable :

- Olivier Friedel (maintenant directeur des études de Supélec)
- le regretté Claude Bocage
- Dominique Marcadet
- Cécile Hardebolle
- Christophe Jacquet

Pour, entre autres, leurs conseils et l'amicale mais ferme pression avec laquelle ils m'ont poussé vers l'HdR :

- Yolaine Bourda
- Jacques Oksman

Table des matières

1 Abstract

This abstract is a very condensed summary of this dissertation. It should allow English speaking readers to get a fair idea of my work, while details can be found in the attached publications. French speaking readers should skip this abstract.

With the increasing complexity of the systems that are designed or analyzed, models have become complex objects which are now considered themselves as systems which must be analyzed, designed and validated. Model engineering has emerged in order to provide the methods and tools required to manage the complexity of nowadays systems. This text deals with model engineering and presents a modeling approach focused on the *behavior* of systems more than on their *architecture*. The aim of this approach is to model complex systems in order to validate their behavior as soon as possible in the development cycle. A key point of this approach is to try to make existing modeling paradigms work together instead of trying to define a new "universal" paradigm. This choice is motivated, firstly by the fact that designing and maintaining such a paradigm is not possible, and secondly by the availability of tools that support existing paradigms, and which are already widely accepted by system designers.

I will first give definitions of what I consider as a model and a system, and of the relationships between systems, models and modeling languages. Then, I will present how our approach has evolved from the specification of synchronous reactive execution machines to the definition of a framework for describing modeling paradigms as well as the way they inter-operate.

1.1 Models and Systems

When speaking about system modeling, we have to wonder about what is a model and what is a system. Both notions are generally not precisely defined, but the various meanings of "model" are the source of a lot of confusion, which Jean-Marie Favre calls the "*meta muddle*" [2006-1]. How can we answer such questions as "Is a model of a model, a model?", "Is it a meta-model?".

I consider here that a model is an abstraction of a system, which is built in order to answer a given set of questions about the system. With this meaning, an UML class diagram is a model of a software system, built to determine the nature of the various components of the system and the services they should provide. Similarly, the equation : $\ddot{\theta} = -\frac{g}{l}\sin(\theta)$ is a model of an oscillating pendulum of length l, built to determine the evolution of the angle θ it makes with the vertical, ignoring friction.

It is a bit more difficult to give a definition of a system. From our point of view, a system is just something we want to model. However, people generally consider that a system has a structure, and that its properties and behavior are obtained by combining the properties and behavior of its components according to some set of laws. Practically, defining a system amounts to define the boundary of what we want to study. We are sometimes interested in its structure, especially when it is too complex to be studied as a whole. It is then decomposed into smaller

subsystems (strictly speaking, it is the model which is decomposed into smaller submodels) that are modeled in turn. Laws must be given to obtain the behavior of the whole system from the behavior of the subsystems. This hierarchical approach is effective for reducing the complexity of a system.

Sometimes, we are not interested in the structure of the system itself. This is the case for instance when, after decomposing a system into smaller and smaller parts, we obtain elementary components. The behavior of such components is described directly as a relation between their inputs and outputs. The model of such a behavior contains components which are not models of parts of the system. For instance, when modeling a system with a state machine, the states are not models of parts of the system. The same is true about state variables in a differential equation model of a continuous behavior. In both cases, the model relies on a state which is an abstraction of the history of the system, from which the response of the system to its inputs can be computed.

In the following, we consider a system as an entity we want to study, which may already exist or still be in the design stage, and which is bounded by an interface. Information, energy and matter may flow through this interface. A model is an abstraction of a system, built for answering questions about the system. A model may describe the structure of the system, or it may describe the transfer function of the system only, i.e. the relations between the that go through its interface.

1.2 The Different Natures of Models

We consider here models that are used for evaluating the behavior or some properties of a system. These are “predictive” models because they allow us to predict the result of observations on the system under given circumstances. Predictive models may also be “executable” when there exists an algorithm to compute the result of the observations. For instance, a model based on differential equations, although it is predictive, may not be executable if it has no analytic solution. However, a numeric solver can provide a good executable approximation of such a model. When a model is executable on the chosen target architecture, it can be considered as a realization of the system. The choice of a target architecture based on a computer, on a programmable automaton or on a card with discrete gates, will therefore lead to very different implementations.

It is possible to build models either for analyzing an existing system or to design a new system. In both cases, we need a predictive model, and we prefer an executable one when possible. When designing a system, model driven engineering leads us to build an implementation through successive refinements of an initial model. According to this approach, the design of a system can be seen as the creation of an analysis model of the requirements, which is then refined by adding details about its structure and its behavior, until we get an executable model for the target architecture. The Model Driven Architecture approach focuses on the most refined model which is still independent from the platform (Platform Independent Model), and on the automatization of the last refinement step toward the implementation (Platform Specific Model), according to a model of the target architecture.

1.3 Relations Between Models, Meta-models

A model is a representation of a system, but a model may also be studied, considered as a system and modeled. Therefore, there exists a relation "is a representation of" between models, and we note it μ as does Jean-Marie Favre in his papers [2006-1]. What is the nature of a model of a model of a system S? Is it a model of S? Is it a meta-model of S?

Let us consider for example a part of the world as the system S, an a map C as a model of this system. There are several kinds of maps (road map, hiking map, geological map, political map), which confirms that there may be several models of a given system. We can write $C \xrightarrow{\mu} S$. Let us now consider a cartographic model M of the map, which describes the cartographic elements in this map. We have $M \xrightarrow{\mu} C$, but also $M \xrightarrow{\mu} S$ because the cartographic elements form a representation of the part of the world. We could therefore think that μ is transitive. However, if we consider another model V of the map, which describes its ID, its date of last update, its price and its availability, we have a model of the map from the point of view of the sales department. We have $V \xrightarrow{\mu} C$, but this model does not represent the part of the world S. The μ relation is therefore not transitive, and a model of a model of a system is not necessarily a model of this system. V is neither a meta-model of S, so a model of a model of a system is not a meta-model of this system.

In order to define the notion of meta-model more precisely, we must examine the way we describe models. A mathematical or a computer model of a system is described using a language, and the structure of valid models is determined by the syntax of the language. A mapping from the elements of this syntax to a semantic domain gives a meaning to the constructs of the language and allows us to determine the meaning of a model. There is a relation "is written in" between models and languages, which is noted ε by Jean-Marie Favre. If M is described using language L, we have $M \xrightarrow{\varepsilon} L$. A language is described by a model which allows us to check if a word belongs to the language and to give a meaning to such words following to their syntactic structure.

Let us examine the relations between a system S, a model M of this system, the language L in which M is written, and a model of this language. For instance, if M is a UML model, L is the UML language, which is modeled by the UML meta-model, which is in turn written in MOF [2005-2]. These relations are illustrated on figure 1 on page 18. This example shows that what we call a meta-model in the context of model driven engineering is a *model of a modeling language*. Since M is written in UML, it conforms to the UML meta-model. This conformance relation is noted χ by Jean-Marie Favre. Note the special relation between MOF and its meta-model, which is the usual way of avoiding to define an infinity of modeling languages.

1.4 Models of Computation, Models of Execution

The conformance of a model to a meta-model ensures that this model is syntactically correct. In order to use this model for obtaining properties of the system it represents, we need to know the semantics of the modeling language. For structural models, which describe a system by assembling components, the semantics of a model can be defined by giving the rules which govern the activation of the behavior of the components and the propagation of the information between components. Such a set of rules is called a *model of computation*. A structural model with a model of computation forms a predictive model since it is possible to predict its properties or its behavior by interpreting its structure according to the model of computation. When the laws of a model of computation are given in the form of an algorithm, the model is executable, and we say that we have a *model of execution*.

1.5 The Sources of Heterogeneity

Most systems are too complex to be studied as a whole. Their models have to be decomposed into simpler models, and each of these models may call for different technical domains. Each model is built for a specific goal. It must therefore be precise enough to achieve its goal, but abstract enough to be dealt with. The most suitable modeling paradigm for each model may therefore depend on the technical domain of the problem, on the nature of the studied properties, and on the level of abstraction at which it is examined.

During her PhD thesis, Cécile Hardebolle did a survey of multi-paradigm modeling and of the reasons for using heterogeneous models [2008-3]. From this survey, it follows that there are four main sources of heterogeneity in the design of complex systems:

- different components of a system may belong to different technical domains;
- a part of a system may be considered under different points of view (for example, functional and non-functional aspects) which belong to different technical domains;
- during the design, a given aspect of a part of a system is modeled at different levels of abstraction, which may require the use of different modeling paradigms;
- a component, considered from a given point of view at a given level of abstraction, may be modeled using different paradigms according to the activity for which the model is built (validation or code generation for instance).

These different sources are illustrated on figure 2 page 20. The first one is the best understood and is handled by current modeling tools. It leads to the use of state machines for modeling the control of a system, and of data-flow diagrams for modeling signal processing for instance.

The second source opens the issue of the consistency of a set of models of a system, which are built as different views for studying different aspects of the system. Such views are generally tightly coupled, and it is an issue to represent such links between models which belong to different semantic domains. For

instance, a power consumption model of a system will have strong dependencies on the functional model of the system, but also on the temperature model of the system.

The third source is a consequence of the refinement process between heterogeneous formalisms. It raises the issue of the conformance of a refinement to its abstract model. This issue is handled for instance by the B method, or by simulation relations between state machines, but these techniques do not take the heterogeneity of the models into account.

Last, the fourth source comes from the different activities in the development cycle of a system. For a given aspect of a part of the system, at a given level of abstraction, we won't necessarily use the same modeling formalism whether we are trying to build an implementation of to validate the design. The issue is the consistency between the various models used for different tasks. What is worth a proof if the model used for the proof is not consistent with the model used to generate code? A solution is to avoid heterogeneity, and to use the same model to validate and to synthesize the system, according to the WYPIWYE principle [1989-4] (*What You Prove Is What You Execute*), but this is not always possible.

1.6 Historical Perspective

I have described the context of my research work, and I have given some definitions that I will use in the following. These definitions are not universal, but I believe they give a sound meaning to words that are important in this text. In the following, I present the evolution of the research I did at the Computer Science Department of Supélec, from the design of execution machines for synchronous programs to the modeling of the semantic adaptation between models of computation. This evolution went through projects, industrial contracts and supervision of PhD students, but it was very strongly influenced by two events: the meeting with synchronous languages during my PhD thesis, and with the Ptolemy environment a few years later. Synchronous languages made me aware of the importance of choosing the right modeling paradigm. Ptolemy showed to me the important role of models and the need for multi-paradigm modeling.

This historical section is summarized much more aggressively than the previous sections because most details can be found in the cited papers in English.

1.6.1 1993–2004: From Synchronous Objects to Heterogeneous Models

My first research work, during the preparation of my PhD thesis, was to "integrate synchronous modules into object-oriented programming" [1993-5]. The need for this integration came from the special modeling paradigm used by synchronous languages [1991-6, 1991-7], which was not directly supported by usual programming languages. This work lead me to adopt a black box approach for modeling systems, and to use a model of computation in order to define how the behavior of the black boxes were combined.

The special nature of the synchronous paradigm required a special runtime environment, called an execution machine [1997-IR1]. It was also necessary to

perform a semantic adaptation between the asynchronous environment and the synchronous code. The details of this work are described in [1995-ED2] and [1997-IR1].

The "interconnected black boxes" approach for describing systems lead to considering models of systems as oriented graphs. Thanks to the precise semantics of the synchronous approach, it was possible to apply transformations to this graph to obtain a new system with a perfectly well defined behavior. We used such a graph transformation approach to add predefined reliability and safety features (replication of components, voters, predictive models etc.) to an application described as a set of interconnected synchronous modules [2008-IR2][1998-IC1].

The Ptolemy project [2003-8] made the notion of model of computation very clear to me. By adapting my previous work on execution machines for synchronous languages, I was able to translate synchronous modules into atomic building blocks (*stars*) for various models of computation of Ptolemy Classic. Following the design of a fixed point execution model for synchronous models by Stephen Edwards [1997-9], and its implementation as the Synchronous Reactive Ptolemy domain (SR), I designed its code generation counterpart as the SRCGC Ptolemy domain in collaboration with Thomson-CSF Optronics. However, the necessity to build a special domain-specific component from a synchronous module for each possible target model of computation was a problem. This lead us to try to insulate two tasks: first, the translation of the synchronous code into a suitable form for Ptolemy, and second, the semantic adaptation of its behavior to the target model of computation. The PhD thesis by Mohammed Feredj [2004-IC2][2009-IR3] lead to an architecture of domain-polymorph components which allows a synchronous kernel to be used in several models of computation thanks to a pair of adaptation components. It was successfully applied to the "Production Cell" test case [1995-10, 1995-11] by embedding Esterel synchronous controllers for the various parts of the production cell into an heterogeneous Ptolemy II model.

1.6.2 2002–2004: Non-Hierarchical Heterogeneous Modeling

Following the Ptolemy approach, we used hierarchical modeling to handle the composition of different models of computation. This allows us to consider the interactions between only two different models of computation at a time: the one used inside the model of a component, and the one used by the model in which the component is utilized. However, this approach cannot handle components with ports that obey different models of computation. For instance, an analog to digital converter as an input which receives a signal modeled as a function of continuous time, a sampling clock modeled as a series of discrete events, and an output which produces a signal modeled as a series of discrete samples. In the PhD thesis by Mokhoo Aimé Mbobi, we showed that models containing such components can be transformed into hierarchical models under some conditions. The main idea in this work is to replace an heterogeneous interface component by homogeneous projections placed in submodels. In order to be able to schedule the submodels, instantaneous dependency loops must not exist in the model [2004-IC3, 2005-IC4][2007-IR4].

1.6.3 2005–2008: A Framework for Describing Models of Computation

After having used Ptolemy to implement execution machines and adapt the semantics of synchronous languages to various models of computation, we wanted to go further in several directions. Firstly, we wanted to be able to describe models of computation in a more formal and structured way. Secondly, we wanted to generalize some mechanisms which are available only in some models of computation (for instance, the validation phase in CT). Thirdly, we wanted to decouple some aspects of models of computation that we consider as hybrid (CT for instance). Lastly, we wanted to describe in a more precise and modular way the semantic adaptation between models of computation.

The first point is very difficult if want to be able to describe any model of computation in order to simulate, validate et generate code for models. We are still using Java for describing models of computation, but we defined an abstract semantics for our models which tries to go further in the direction of the ideas brought by the abstract semantics of Ptolemy.

The second point lead us to define more precisely what our simulation platform should compute, and to integrate into the generic execution engine several steps for scheduling and propagating information between components. We also added a validation operation which allows any component to invalidate the result of a simulation step and to require a new computation. This mechanism is available only in the CT domain in Ptolemy.

The third point is an attempt to define very simple models of computation which can be combined to build complex systems. We consider for instance that the support for both continuous and discrete signals in CT, although convenient for modelers, is a source of complexity in the definition of its semantics. We would like to be able to define a pure continuous time model of computation and to handle the production and handling of discrete events in the semantic adaptation layer between continuous models and discrete one. However, this would make the construction of hybrid model more difficult.

In order to explore these research themes, we developed a modeling and simulation platform named ModHel'X. Its design began during the "Software Factory" project of the System@tic competitiveness cluster, with the PhD thesis of Cécile Hardebolle [2008-3].

ModHel'X relies on a generic meta-model which allows the description of the structure of heterogeneous models. It provides a generic execution engine which interprets the structure of a model according to the associated model of computation [2008-IC5][2009-IR5]. This execution engine relies on three basic operations which define the abstract semantics of ModHel'X:

- *schedule* is the operation that allows the engine to choose a block to observe in the model;
- *update* is the operation used to observe the behavior of a block. During an update, a block should take into account any information available at its interface and publish any additional information produced by its behavior in response;
- *propagate* is the operation used to propagate information produced by a block to the others, according to the relations between their pins.

These operations are similar to the operations of the abstract semantics of Ptolemy. However pins in ModHel'X are just interface points and play no role in the transfer of information between block. The *propagate* operation replaces the *send/receive* pair in Ptolemy. To compute the behavior of a model at a given instant, ModHel'X schedules, updates and propagates information until the value of each pin of each block is determined. If this computation is validated by all the blocks, it become the snapshot of the model at this instant, and the blocks are allowed to update their internal state according to the definitive value of their pins. This abstract semantics is defined as the execution of a synchronous sequential system. The computation of the snapshot is equivalent to the combinatorial computation of the outputs and of the next state. The update of the internal state of the blocks is equivalent to the loading of the registers on the clock edge.

In the "Software Factory project", we were involved in two subprojects, and ModHel'X was developed in the OpenDevFactory subproject. In the MoDriVal (Model Driven Validation) subproject, we applied our approach of heterogeneous modeling to the validation of the control of heterogeneous models. The ADLV (Architecture Description Language for Validation) language and tool developed during the project provide a mean for the separate specification of the control of a system and of the computations it performs on data [2008-IC6]. We rely on this separation to perform a formal validation of the control (we used a synchronous language to model the control and were able to validate properties by model-checking). The novelty of the approach was that, due to the precise modeling of the way the information used to drive the controller is computed from the data, we are able to translate global properties that use data values into controller properties that a model-checker can handle. The same description of the communication between the data processing part and the controller allowed us to build an automatic code generator for gluing the code generated for the controller by the Lustre or Esterel compiler, and the code produced by Simulink or Scilab for data processing [2008-IC7].

Since ModHel'X is only an experimentation platform used to test new ideas, we have to put these ideas at work in industrial projects. The PhD thesis currently done by Abderraouf Benyahia, which I co-supervise with François Terrier from CEA-List in Saclay, consists in formalizing the semantics of the fUML execution model [2010-IC8]. Using ideas from ModHel'X, the standard execution engine of fUML was made more generic, and semantic variation points can be defined by choosing different scheduling and communication policies. Since models are described using UML, profiles are used to specify the precise semantics of the models. This work is performed in the context of the AccordUML project [2003-12].

1.6.4 2006–2010: A Framework for Describing Semantic Adaptation

During the "Software Factory" project, the development of ModHel'X was mainly focused on the core which allows the description of models of computation and their joint use in heterogeneous models. Semantic adaptation between models of computation was supported by this core, but the details of how semantic adaptation was modeled were not clearly specified. During the EDONA

project, we used ModHel'X to describe models of computation used in the automotive industry, as well as the semantic adaptation between these models of computation.

We consider that the semantic adaptation between models that use different models of computation should be explicit because it has a strong influence on the behavior of the global model. This adaptation is often implicit either because it is hard-wired in the modeling tool or because it is delayed until the integration of the components of the system. Our goal is to allow the description of the semantic adaptation at any level of abstraction in the modeling process, in a modular and reusable way. Modularity means that the semantic adaptation should not impact the models it adapts. If we change one model in a system, we surely will have to change how its semantics is adapted to the rest of the system, but this should not lead us to modify other parts of the system. Reusability is achieved by defining adaptation patterns which can be parameterized and applied in different contexts. For instance, periodic sampling is a common adaptation pattern between continuous time and discrete signals, temporal tagging is a pattern for adapting reactive systems to discrete events, etc.

Our study of semantic adaptation during the EDONA project lead us to identify three main directions: adaptation of data, adaptation of time, and adaptation of control. The adaptation of data is the most simple, but is necessary because different models of computation may use different sorts of data to convey information between components. For instance, in a discrete events model, components communicate by exchanging events which are composed of a value and a time stamp. When using a state machine in a DE environment, it is necessary to translate DE events into symbols for the state machine. This can be done by removing the time stamp, and mapping event values to input symbols for the state machine. On the output side, symbols from the state machine can be mapped to event values, and a time stamp can be added, but its value depends on the adaptation of time between the models.

Adapting time between models is more complex. At the beginning, we though it would be possible to express relations between time scales at the boundary between models. For instance, between DE and FSM, when a discrete event is send to the state machine, we can store its date, and if the state machine produces a symbol, use this date as a time stamp for the outgoing DE event. This amounts to consider that the state machine reacts instantaneously in the time of DE. However, this approach is too simplistic to handle more complex cases. It doesn't allow us to specify relations between the clocks of subsystems used in an environment where time doesn't exist or is reduced to a series of discrete instants where events occur. For instance, two continuous time models embedded in a DE model have no mean to synchronize their clock unless we assume the existence of a global reference clock. We have therefore chosen to declare relations between clocks without respect for the hierarchical composition of the models to which they belong. These relations define a system of constraints which is solved at run-time to compute the current time in the model of each subsystem of the model we execute. Synchronization with real time or simulation time is considered as a relation to a special clock which is provided by the simulation engine. These ideas are largely

taken from the notion of multi-form time used in synchronous languages, and from the sub-profile for modeling time in the MARTE profile [2007-13, 2007-14].

Control is the set of instants at which the elements of a model are observed and have the ability to influence the behavior of the system. Control can therefore be considered as a clock, and the adaptation of control between two models of computation is very dependent on the adaptation of the notions of time. For instance, if we consider that a synchronous data flow model works with a sampling period T, we map instants $0, 1, 2, \ldots, n$ of its clock to instants $t_0, t_0+T, t_0+2T, \ldots, t_0+nT$ of the clock of its environment. With regard to control, the SDF model should be observed when time reaches each of these dates in its environment. The flowing of time in a model may therefore create control in another model. Similarly, control is linked to data. When a state machine is used in a discrete event environment, the occurrence of an event for the state machine makes it necessary to observe the state machine in order to handle its reaction to the event. However, there are cases where the production of data or the flow of time does not always produce control. If we consider for example a data flow model which runs periodically in a discrete events environment, the occurrence of an event on an input of the SDF model should not create control if the date of the event does not belong to the periodic sampling clock of the model.

In ModHel'X, semantic adaptation between models of computation is handled by interface blocks. An interface block behaves like a block in its surrounding model, and its behavior is described by an internal model which may obey a different model of computation than the external model. Interface blocks are in charge of adapting data in the internal model from and to the external model. Interface blocks can emit time constraints on clocks to perform the adaptation of time scales. Regarding the adaptation of control, an interface block is in charge of deciding when to update its internal model according to available data and current time, and to emit time constraints to ensure that its internal model is updated when necessary.

1.6.5 Work in Progress

Our approach of heterogeneous modeling is focused on the simulation of heterogeneous models, and has taken into account only one source of heterogeneity until now. However, we are interested in the modeling, validation and realization of heterogeneous systems, and in the other sources of heterogeneity. We studied how muti-view modeling can be handled according to our black-box approach and the use of interface block to adapt semantics between models of computation [2010-ED1]. We choose not to derive views from a global model because this global model should be written in an hypothetical language which could describe all the aspects of the system. Instead, we build separate views for each aspect of the system, using the most suitable modeling language, and we connect these views in order to express the dependencies between their behaviors. The main issue is to preserve the modularity while exposing enough information to synchronize the behaviors of the views.

Regarding the validation of heterogeneous systems, the PhD thesis currently

done by Bilal Kanso takes a coalgebraic approach to the modeling of components [2003-15] that unifies the different ways of modeling state-transition systems. This unified formalization allowed him to generalize the test generation algorithm presented in [2006-16], in order to use it on heterogeneous models as presented in [2010-IC9]. This thesis also treats the issue of integration testing of heterogeneous models. If we have components which conform to their specification, how can we check that a system built from these components conforms to its specification? In this approach, components are composed using integration operators such as cartesian product and feedback. This formal framework allowed us to generalize the ioconf [1996-17] relation, and to prove that cartesian product preserves this relation. Preservation of the conformance relation by the feedback operator is obtained only under some assumptions on the specification of the components. This work is of a very mathematical nature compared to what is done with ModHel'X, but it allows us to define more precisely the nature of components and of their interactions. It also makes a link between connector based approaches like BIP [2008-18] or Metropolis [2007-19], and model of computation based approaches like Ptolemy [2003-8] or ModHel'X [2009-IR5]

1.7 Conclusion

The study of a very specific problem — the integration of a new approach of the modeling of reactive systems into an object-oriented design methodology — lead us to identify the notions of execution machine and of semantic adaptation. At the same time, the increasing complexity of systems was shifting the design effort from code to models, and the joint use of several modeling paradigms for different parts of a system showed the necessity of handling heterogeneous models.

In this context, our initial problem of integrating synchronous modules in object-oriented applications appeared to be a special case of a more general issue. The modeling formalisms used in physics, signal processing, control theory, communication protocols and so on, lead to the same issue of providing a suitable execution environment and adapting the semantics between models. None of these formalisms is suitable for modeling a complete system, so heterogeneous model are necessary.

Following the actor/model of computation approach taken by the Ptolemy project, we chose to describe the structure of systems according to a unique abstract syntax (the meta-model of ModHel'X) and to define the semantics separately in the form of models of computation. The common abstract syntax allows us to use a generic execution algorithm which relies on a very small set of basic operations. These operations are the abstract semantics of models. The concrete semantics is given by models of computation in the form of specializations of the operations of the abstract semantics. This structure allows ModHel'X to execute models that obey any model of computation.

1.7.1 Languages and models of computation

A recurring issue during this work was the nature of models of computation versus the nature of modeling languages. It seems that both are ways of describing

models. A modeling language has operators for combining elementary behaviors, while a model of computation interprets the structure of a model to combine the behavior of its components. It is always possible to represent a modeling language as a model of computation. It suffices to model the operators of the language by components when they perform some computation, and by relations which are interpreted by the model of computation when they are related to control or communication. However, we noticed that some models of computation like state machines, Petri nets, or continuous time, are more naturally considered as modeling languages, while others like Kahn process networks or discrete events are considered as models of computation. The main difference between these kinds of models is that in the second type, the hierarchy of the model describes the hierarchy of the system. Components of the model are models of the components of the system. It is therefore natural to consider that the behavior of the system is obtained by combining the behavior of its parts according to a model of computation. In the first type of models, the components of the model are not models of the components of the system. A state in an automaton does not model a part of the system but a part of its behavior. The hierarchy of these models is therefore a behavioral hierarchy, not a structural one. This leads to consider the components as elements of a language because they have no counterpart in the system. A criterion to distinguish the two types of models is whether new elementary components can be created or not. In a model which obeys a model of computation, with a structural hierarchy, the elementary components of the structure may be opaque behaviors defined outside the modeling environment. For instance, in a Khan process network, a component may have a behavior written in any programming language. It is not necessary to know the details of this behavior to combine it with the behavior of the other components. On the contrary, in a behavioral hierarchy, the precise semantics of each elementary component must be known. In a model of a state machine, it is necessary to know what is a state, what is a transition, and how to interpret the guard. Similarly, in a Petri net, the interpretation of the model relies on the exact knowledge of what is a place, what is a transition, and what are the relations between them. Adding an elementary component with an opaque behavior to such models is nonsense. These modeling paradigms are therefore considered as modeling languages because all the syntactic elements of a model must be known from the interpreter in order to give a meaning to a model.

1.7.2 Universality of Models of Computation

Another open issue is the ability of ModHel'X to describe all models of computation. Is it posible to imagine a model of computation which would be impossible to describe according to the abstract semantics of ModHel'X? To answer this question, it is necessary to consider what we call a model of computation, or more precisely a model of execution. If we call model of execution an algorithm for combining the behavior of the components of a model to compute the behavior of the whole model, then ModHel'X can surely describe any model of computation, just like a Turing machine can execute any algorithm. However, the notion of model of computation relies on the observation of the interface of opaque components,

and this approach has limits and does not allow the computation of the behavior of all systems. Let us consider for instance the electrical circuit shown on figure 5 page 52, which connects a generator of electromotive force e and of internal resistance r to a resistor load R. Figure 6 page 52 shows a "black box" model of this circuit with two components gen and res which represent the generator and the resistor load. Since the behavior of these components is opaque, we cannot compute the value of U and I by solving the system $U = e - rI$, $U = RI$ because these equations are not known outside the components. We can only observe the values published by a component at its interface and propagate them to the interface of the other component. Assuming that the generator initially gives a tension e with a null current, and the resistor answers by giving value e/R to the current, we can iterate and make the generator update the tension according to the new value of the current and so on. In this example, the iteration converges only if $r < R$. So the behavior of the model cannot always be computed using a model of computation. Even when the iteration converges, the values of U and I are know after an infinite series of observations. The reasons for this failure is that the behavior of each component impacts the behavior of the other, and that the behavior of the system is a fixed point of the composition of the open loop behavior of the components. In this example, the fixed point exists and is unique (we can compute it by solving the equations), but it is not necessarily reachable by iteration. Computing the behavior of such systems can only be achieved if we open the black boxes. Therefore, ModHel'X can represent any model of computation, but there are models which cannot be decomposed into opaque components and cannot be handled by ModHel'X.

1.7.3 Future Work

Our approach of the modeling and validation of heterogeneous models consists in defining a general framework with well defined notions of component, model of computation and model of execution. However, this framework cannot be as efficient as ad hoc approaches, which, at the price of some limitations, provide automatic checking of properties or code generation possibilities. Our goal is therefore to work on concrete cases, using industrial tools, so that we can study the real problems. Then, our general framework allows us to experiment new ideas without the constraints of the real cases. Once validated, these new ideas can be used to handle real problems more easily.

Our work on multi-view modeling, heterogeneous model testing and the modeling of semantic adaptation is inspired by real cases. Other issues linked to the heterogeneity of models seem interesting, and we plan to study them in the future:

Heterogenous Model-checking

The work on the coalgebraic modeling of state-based systems is for the moment only used to build a theory of testing heterogeneous systems. However, this kind of systems can also be analyzed through model-checking. It would be interesting to develop model-checking algorithms which rely on the knowledge that some properties hold on a component of the system to avoid exploring domains of the state space which are unreachable.

Such a modularization of the checking of properties would restrict the set of properties that can be checked, but it may allow us to check them on much bigger systems.

Composition of Properties

We defined a model of computation as a set of rules for combining the behavior of components to obtain the behavior of a system. We could also consider a model of computation as set of rules for combining properties of components to compute properties of a model. This leads to consider a model of computation as a logic, and each model which obeys this model of computation as a set of axioms. The global properties of the model should be theorems proven from the axioms according to the rules of the logic. This could also be the basis for a refinement theory for models of computation. Could we define deterministic state machines as a refinement of non-deterministic ones? An advantage of such a theory would be to be able to prove that a model of execution is a refinement of a model of computation. We could then be sure that the execution of a model (according to the model of execution) would satisfy the properties proved according to the model of computation.

Minimal Prerequisites

The opposite approach of the composition of properties would consist in, starting from the properties a model should satisfy, to find, according to the deduction rules associated to the model of computation, the minimal prerequisite that each component should satisfy. This is similar to the determination of the minimal requirements of a program in Hoare's logic [1969-20]. However, the composition of behaviors is more complex than the sequential execution of instructions. Moreover, the syntax of formulae, the axioms and the deduction rules change with each model of computation. It is therefore necessary to define mappings between the different logics used in a model, in a similar way as the semantic adaptation between heterogeneous models.

To conclude, I would say that multi-paradigm modeling is a new research domain which is by nature connected to numerous scientific domains. This is one of its attractive features because it allows us to meet researchers from various domains. This can also be frustrating because it is sometimes difficult to demonstrate the results of a research activity which consists mainly in describing modeling methods. The examples we show most of the time are examples of applications of our work to some domain. This leads us to promote transdisciplinarity in order to make researchers from other domains accept to share one of their multi-paradigm issue with us. The diversity of disciplines within the "Supélec Systems Science" (E3S) team is therefore an advantage for us.

2 Introduction

Avec la croissance de la complexité des systèmes à analyser ou à concevoir, les modèles sont passés du statut de simple outil à celui d'objet complexe, considéré lui-même comme un système à analyser, concevoir et valider. À côté de l'ingénierie traditionnelle qui utilise des modèles pour concevoir des systèmes, s'est développée une ingénierie des modèles [2007-21, 2006-22] dont la tâche est de fournir les concepts et les outils qui permettent d'appréhender la complexité des modèles actuels [2007-23].

Ce texte s'inscrit dans le cadre de l'ingénierie des modèles et présente une approche de la modélisation des systèmes qui privilégie la description du comportement plus que de l'architecture des systèmes. L'objectif de cette approche est de permettre la modélisation de systèmes complexes afin d'en valider a priori le comportement. Un élément clef de cette approche est qu'elle cherche à fédérer les paradigmes existants appliqués aux portions du modèle pour lesquelles ils sont pertinents plutôt que définir un nouveau paradigme « universel ». Ce parti pris tient d'une part à la difficulté de définir et de maintenir un tel paradigme, et d'autre part à la volonté de réutiliser les outils déjà disponibles et auxquels les concepteurs de systèmes sont habitués.

Le reste de cette introduction est consacré à la définition des notions de système, de modèle, et des relations entre modèles, telles qu'elles seront utilisées par la suite. Ces définitions ne prétendent pas être universelles, mais elles synthétisent un certain usage et sont celles sur lesquelles repose notre discours.

Les idées présentées ici se sont en effet formées dans un contexte mouvant, à la confluence entre les approches inspirées de l'électronique, l'automatique et le traitement du signal, et les approches issues du génie logiciel avec l'apparition de l'ingénierie dirigée par les modèles. Cette dernière donne une importance croissante aux modèles au détriment du code, ce qui rejoint les descriptions à l'aide de schémas blocs et de réseaux d'opérateurs utilisées hors du monde logiciel. Partant du problème concret de l'intégration de modules réactifs synchrones dans la programmation par objets, nous avons défini un cadre plus abstrait pour le traiter dans sa généralité et avons ainsi participé à l'émergence d'un nouveau domaine de recherche : la modélisation multi-paradigme [2004-24].

Les sections suivantes mettent notre approche en perspective en présentant son évolution depuis la spécification de machines d'exécution permettant d'intégrer l'approche réactive synchrone dans des programmes non synchrones, jusqu'à la définition d'un cadre général permettant de décrire des paradigmes de modélisation et la manière dont ils s'articulent au sein d'un modèle complexe. Enfin, les travaux actuels et leurs perspectives sont présentés avant de conclure sur les avantages et les limitations de notre approche.

2.1 Modèles et systèmes

Lorsqu'on parle de modélisation des systèmes, on doit se demander ce qu'est un système et ce qu'est un modèle. Si la notion de système n'est pas toujours bien

définie, elle pose rarement des problèmes de compréhension, contrairement à la notion de modèle. Je commence donc par préciser ce que j'entends par « modèle ».

Le terme « modèle » possède de nombreux sens, et son utilisation dans différents contextes est à l'origine de nombreuses confusions. Dans [2006-1] [1], Jean-Marie Favre nomme cette situation confuse le "meta-muddle". Quelle réponse peut-on donner à des questions telles que : un modèle d'un modèle est-il un modèle ? Est-ce un méta-modèle ?

Nous considérons ici qu'un modèle est une abstraction d'un système, conçue pour répondre à un certain nombre de questions sur le système. En ce sens, un diagramme de classes UML peut être un modèle d'un système logiciel, établi pour déterminer la nature des entités de ce système et les services qu'elles peuvent rendre. De même, l'équation : $\ddot{\theta} = -\frac{g}{l}\sin(\theta)$ est un modèle d'un pendule oscillant de longueur l, établi pour déterminer l'évolution au cours du temps de l'angle θ qu'il fait avec la verticale, en faisant abstraction des forces de frottement.

Il est un peu plus difficile de donner une définition satisfaisante de ce qu'est un système. De notre point de vue, un système est simplement quelque chose que l'on souhaite modéliser. Toutefois, on considère généralement qu'un système a une structure [2005-25], et que son comportement est le résultat de la combinaison des comportements de ses composants qui interagissent selon certaines lois. Dans la pratique, définir un système consiste à délimiter un objet d'étude. Dans certains cas, on s'intéresse effectivement à la structure de cet objet, notamment lorsqu'il est trop complexe pour être étudié d'un bloc. On le décompose alors en sous-systèmes (on décompose en fait son modèle en composants) que l'on modélise à leur tour. Il faut alors préciser les règles de composition qui permettent d'obtenir le comportement du système complet à partir des comportements des sous-systèmes. Cette approche hiérarchique est efficace pour réduire la complexité de l'étude d'un système.

Il arrive aussi que l'on ne s'intéresse pas à la structure du système, notamment lorsque, à force de décomposer un système en sous-systèmes, on obtient des composants considérés comme élémentaires. La description du comportement d'un de ces composants peut alors se réduire à des relations entre ses entrées et ses sorties. Les éléments du modèle de ce comportement ne sont pas des modèles des éléments du composant. Par exemple, lorsqu'on modélise le comportement d'un système par un automate, les états et les transitions ne correspondent pas à des éléments du système. De même pour les variables d'état liées par un jeu d'équations différentielles dans le cas d'un système continu. Dans les deux cas, on abstrait l'histoire du système sous forme d'un état qui permet de déterminer la réponse du système à ses entrées.

Par la suite, nous considérons qu'un système est une partie du monde, existante ou à concevoir, qui est objet d'étude et qui est délimitée par une interface à travers laquelle peuvent circuler de l'information, de l'énergie et de la matière.

1. Dans ce mémoire, les références sont composées de l'année de publication suivie d'un numéro. Les références à mes publications comportent de plus deux caractères avant le numéro : IR pour les revues internationales, ED pour les actes de conférence édités, IC pour les conférences internationales, NC pour les conférences nationales, et CM pour les communications.

Un modèle est une abstraction d'un système, construite pour répondre à certaines questions sur le système. Un modèle peut décrire la structure du système et être hiérarchique, mais il peut aussi ne modéliser que la fonction de transfert du système, c'est-à-dire les relations entre les flux qui traversent son interface.

2.2 Nature des modèles

Les modèles que nous considérons ici sont construits pour permettre d'évaluer le comportement ou certaines propriétés d'un système. Ce sont des modèles « prédictifs » car il permettent de prédire les observations qui seront faites sur le système dans certaines conditions. Un modèle prédictif peut aussi être « exécutable » s'il existe un algorithme permettant de calculer le résultat des observations. Par exemple, un modèle à base d'équations différentielles, bien que prédictif, peut ne pas être exécutable s'il n'admet pas de solution analytique. Il peut toutefois être approximé par un modèle exécutable obtenu en résolvant numériquement le système d'équations. Enfin, lorsqu'un modèle est exécutable sur l'architecture cible choisie, il constitue une implémentation, ou une réalisation du système. La nature d'une implémentation est ainsi très différente selon que l'architecture cible est un ordinateur complet, un automate programmable, ou une carte avec des portes logiques discrètes.

La construction d'un modèle peut avoir pour but d'analyser un système déjà existant ou de concevoir un nouveau système. Dans les deux cas, on cherchera à obtenir un modèle prédictif et même exécutable. Dans le cas d'un modèle pour la conception, l'ingénierie dirigée par les modèles favorise la création d'un modèle exécutable sur l'architecture cible par raffinements successifs d'un modèle initial. Selon cette approche, la conception d'un système peut être considérée comme la création d'un modèle d'analyse des exigences, qui est ensuite raffiné (par ajout de détails sur sa structure et son comportement) jusqu'à obtenir un modèle exécutable sur l'architecture cible choisie. L'approche MDA (Model Driven Architecture) consiste à identifier spécifiquement le modèle le plus raffiné qui est encore indépendant de cette architecture (Platform Independant Model, ou PIM) et à automatiser le raffinement vers l'implémentation (Platform Specific Model, ou PSM) en s'appuyant sur un modèle décrivant l'architecture cible.

2.3 Relations entre modèles, méta-modèles

Un modèle est une représentation d'un système, mais un modèle peut lui-même être pris comme objet d'étude et être modélisé. Il existe donc une relation « est une représentation de » entre modèles, que nous noterons μ pour reprendre la notation de Jean-Marie Favre [2006-1]. Quelle est la nature d'un modèle d'un modèle d'un système S ? Est-ce un modèle de S ? Est-ce un méta-modèle de S ? Considérons par exemple le système S constitué d'une région du globe terrestre, et C un modèle de cette région sous la forme d'une carte. Il peut exister plusieurs types de cartes (routière, de randonnée, géologique, politique etc.) ce qui confirme qu'il peut exister plusieurs modèles d'un même système selon l'objectif

poursuivi. Nous pouvons donc écrire $C \xrightarrow{\mu} S$. Considérons maintenant un modèle cartographique M de la carte, qui décrit les éléments cartographiques qui la composent. Nous avons $M \xrightarrow{\mu} C$, mais aussi $M \xrightarrow{\mu} S$ car les éléments cartographiques constituent aussi une représentation de la région. On pourrait donc penser que la relation μ est transitive. Considérons maintenant un autre modèle V de la carte décrivant son numéro d'identification, sa date de dernière mise à jour, son prix et sa disponibilité. Ce modèle représente la carte du point de vue du service commercial, donc nous avons $V \xrightarrow{\mu} C$, par contre, V n'est pas un modèle de la région. La relation μ n'est donc pas transitive, et un modèle d'un modèle d'un système n'est pas nécessairement un modèle de ce système. V n'est pas non plus un méta-modèle [2003-26] de la région. Un modèle d'un modèle d'un système n'est donc pas un méta-modèle de ce système.

Afin d'introduire la notion de meta-modèle, il faut considérer la manière dont sont décrits les modèles. Un modèle informatique ou mathématique est décrit dans un langage, et la structure des modèles valides dans ce langage est déterminée par la syntaxe du langage. À cette syntaxe est associée une sémantique qui donne un sens aux constructions du langage et qui permet de déterminer le sens d'un modèle. Il existe donc une relation « est écrit en » entre modèles et langages, que Jean-Marie Favre note ε. Si M est décrit dans le langage L, nous aurons $M \xrightarrow{\varepsilon} L$. Un langage est décrit par un modèle qui permet de déterminer quels mots appartiennent au langage. Un tel modèle est une description de la syntaxe du langage, et il permet de donner une sémantique aux mots du langage en s'appuyant sur leur structure syntaxique.

Examinons maintenant les relations entre un système S, un modèle M de ce système, le langage L dans lequel est écrit ce modèle, et un modèle de ce langage. Prenons pour exemple un modèle UML [web-27]. Le langage de modélisation L est donc UML, qui est modélisé par le méta-modèle d'UML, lui-même écrit en MOF [2005-2]. Ces relations sont illustrées par la figure 1 ci-dessous, inspirée des mégamodèles [2004-28].

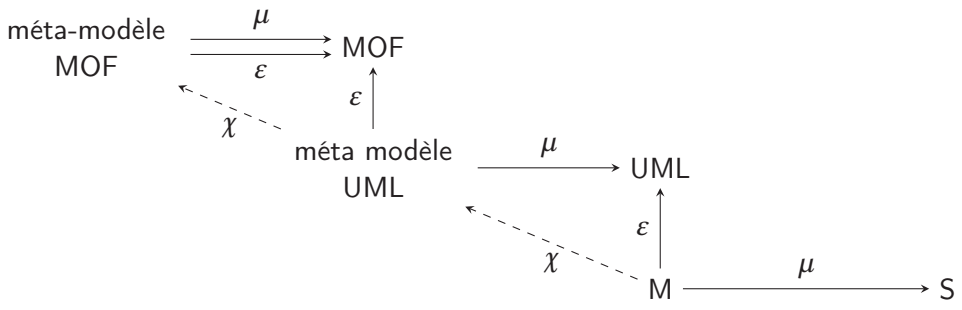


FIGURE 1 – Illustration des relations entre modèles

Cet exemple montre que ce que nous appelons méta-modèle dans le contexte de l'ingénierie dirigée par les modèles est un modèle d'un langage de modélisation. Le modèle M étant écrit en UML, il est *conforme* au méta-modèle d'UML, ce que Jean-Marie Favre représente par la relation χ. On remarquera la relation parti-

culière entre le MOF et son méta-modèle, qui correspond à la manière habituelle d'éviter d'avoir à définir une infinité de langages de modélisation.

2.4 Modèles de calcul et modèles d'exécution

La conformité d'un modèle à un méta-modèle nous assure que ce modèle est syntaxiquement correct. Afin d'utiliser ce modèle pour en déduire des propriétés du système qu'il représente, il faut connaître la sémantique du langage de modélisation utilisé. Pour les modèles structurels, qui décrivent un système par assemblage de composants, la sémantique d'un modèle peut-être définie en donnant les lois qui régissent l'activation du comportement des composants et la propagation de l'information entre composants. Un tel ensemble de règles est appelé *modèle de calcul* citelee :TSM. Un modèle structurel muni d'un modèle de calcul constitue un modèle prédictif : il est possible de prédire ses propriétés ou son comportement en interprétant sa structure selon le modèle de calcul. Lorsque dans ce modèle de calcul, les lois d'interprétation sont mises sous la forme d'un algorithme, le modèle est exécutable. On parle alors de *modèle d'exécution* pour souligner l'aspect exécutable du modèle de calcul.

2.5 Les sources de l'hétérogénéité

La plupart des systèmes étudiés sont trop complexes pour être traités de façon monolithique. Leurs modèles doivent donc être décomposés en modèles plus simples, et chacun de ces modèles peut faire appel à des connaissances métier différentes. Un modèle étant construit pour un objectif particulier, il doit être juste assez précis pour remplir son objectif, et le plus abstrait possible pour faciliter le travail de concepteur. Le paradigme de modélisation le plus adapté à chaque modèle peut donc être différent selon le domaine métier du modèle, la nature des propriétés étudiées et le niveau d'abstraction choisi. Ceci se traduit par exemple en UML par l'utilisation de langages spécifiques à un domaine [2000-29].

Une étude du domaine de la modélisation multi-paradigme et des raisons pour lesquelles les modèles hétérogènes sont nécessaires a été faite au cours de la thèse de Cécile Hardebolle [2008-3]. Il en ressort que l'utilisation conjointe de modèles qui font appel à des modèles de calcul différents peut trouver son origine à quatre sources :

- les différents composants d'un système peuvent relever de différents domaines métier ;
- une partie d'un système peut être considérée sous différents points de vue (aspects fonctionnels et non fonctionnels) qui relèvent de domaines métiers différents ;
- au cours du processus de conception, un aspect d'une partie d'un système est traité à différents niveaux d'abstraction, ce qui peut nécessiter des paradigmes différents ;
- un composant, considéré sous un aspect donné et à un certain niveau d'abstraction, peut être modélisé selon différents paradigmes en fonction de

l'activité (la génération de code ou la validation, par exemple) pour laquelle on réalise le modèle.

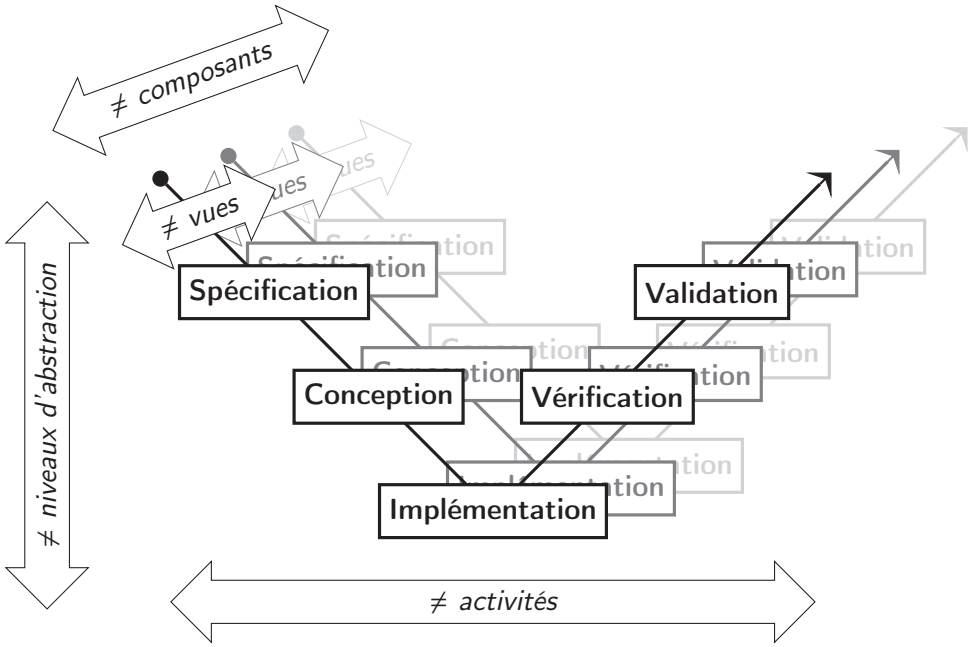


FIGURE 2 – Les sources de l'hétérogénéité

La première source est la plus classique et la mieux traitée par les outils de modélisation actuels. Elle consiste par exemple à utiliser un automate pour modéliser le contrôle d'un système, et des diagrammes à flots de données pour modéliser les traitements appliqués à des signaux. Elle est aussi présente dans le *codesign* qui consiste à concevoir dans un même processus le logiciel et le matériel avec lequel il interagit [1997-30].

La seconde source pose le problème de la cohérence de différents modèles d'un système qui sont conçus comme différentes vues du système prenant en compte des caractéristiques différentes. Il existe généralement un fort couplage entre ces vues, et la difficulté consiste à représenter ces couplages entre des entités qui appartiennent à des domaines sémantiques distincts. Par exemple, un modèle de la consommation d'énergie d'un système aura de fortes dépendances envers le modèle fonctionnel du système, mais aussi envers le modèle thermique du système.

La troisième source est liée au processus de raffinement et pose le problème de la conformité d'un modèle à un modèle plus abstrait. Ce problème est traité par exemple par la méthode B [1996-31], ou par les relations de simulation entre automates, mais sans réelle prise en compte de l'hétérogénéité des modèles. La notion d'approximation conservative est utilisée dans [2001-32] pour modéliser l'abstraction et le raffinement dans Metropolis [2007-19].

Enfin, la quatrième source est liée aux différentes activités du cycle de conception d'un système. Si l'on considère un aspect d'une partie du système à un niveau de raffinement particulier, on n'utilisera pas nécessairement le même formalisme selon que l'on cherche à obtenir une implémentation ou à valider le modèle. Le

problème posé est alors la cohérence des modèles utilisés : que vaut la validation si les modèles utilisés pour l'implémentation et la validation ne sont pas cohérents ? Une solution est d'éviter l'hétérogénéité, et d'utiliser le même modèle pour la synthèse du système et pour sa validation, selon le principe WYPIWYE [1989-4] (*What You Prove Is What You Execute*), mais ce n'est pas toujours possible.

2.6 Conclusion

J'ai exposé dans cette section le contexte des travaux présentés dans ce mémoire, ainsi que le vocabulaire que j'utiliserai par la suite. Les définitions utilisées pour les notions de modèle, de système, ainsi que pour les relations entre systèmes, modèles et langages de modélisation ne sont pas universelles, mais permettent d'éviter les ambiguïtés. Les notions de modèle de calcul et de modèle d'exécution permettent d'associer une sémantique à la structure d'un modèle. Lorsque plusieurs modèles de calcul sont utilisés pour modéliser un système, on obtient un modèle hétérogène. La diversité des paradigmes de modélisation entraîne l'existence de nombreux modèles de calcul, et nous avons identifié quatre sources d'hétérogénéité dans le cycle usuel de conception des systèmes.

Dans la suite de ce document, je présente l'évolution de la recherche que j'ai menée au département informatique de Supélec, depuis la conception de machines d'exécution pour langages synchrones jusqu'à la modélisation de l'adaptation sémantique entre modèles de calcul. Cette évolution s'est faite au cours des projets, contrats et thèses co-encadrées, mais elle a été très fortement influencée par deux événements : la rencontre avec les langages synchrones pendant ma thèse, et celle avec l'environnement Ptolemy quelques années plus tard. Les langages synchrones m'ont fait prendre conscience de l'importance du choix d'un paradigme de modélisation. Ptolemy m'a montré l'importance des modèles et l'intérêt de la modélisation multi-paradigme.

3 1993 – 2004 : Des objets synchrones à l'hétérogénéité des modèles

Mes premiers travaux de recherche, pendant ma thèse, ont consisté à « intégrer des modules synchrones dans la programmation par objets » [1993-5]. Cette intégration était nécessaire car les langages synchrones [1991-6, 1991-33, 1991-7, 1985-34] introduisaient un paradigme de modélisation du contrôle des systèmes enfouis qui n'était pas directement supporté par les langages de programmation usuels. C'est au cours de ces travaux que, partant du cas particulier des machines d'exécution pour modules synchrones, nous avons élaboré le schéma plus général qui consiste à adopter une approche « boîte noire » et à définir la combinaison du comportement des boîtes par un modèle de calcul. Il s'agissait d'une première approche du problème de l'hétérogénéité. Cette section retrace cette évolution, ainsi que les résultats obtenus.

3.1 Intégration du paradigme synchrone

Le paradigme synchrone s'appuie sur des hypothèses (propagation instantanée des signaux, exécution instantanée de certaines opérations) qui permettent de définir précisément la notion d'instant et donc l'état global d'un système réactif [1985-35] au moment où il traite une information. Ceci permet de définir précisément la sémantique des programmes synchrones et leur modèle d'exécution [1988-36]. Toutefois, le code généré à partir d'un module synchrone doit s'exécuter dans un environnement qui lui fournit ses entrées, ses sorties, et son horloge d'activation. Cet environnement est une machine d'exécution [1997-IR1] qui permet de concilier les hypothèses de l'approche synchrone avec l'environnement dans lequel le module synchrone doit s'exécuter. L'aspect « objet » de mes travaux de thèse était principalement méthodologique et permettait d'intégrer la conception synchrone d'un composant d'un système dans la conception par objets du système logiciel complet.

3.2 Environnement d'exécution

L'autre aspect de ces travaux consistait à concevoir un environnement d'exécution dans lequel la notion d'instant était bien définie, à construire des événements pour les modules synchrones à partir des données de l'environnement asynchrone, et à interpréter les événements produits par les modules synchrones dans cet environnement. Il s'agissait donc du traitement *ad hoc* de ce que nous appelons maintenant l'*adaptation sémantique* entre modèles de calcul. Cette adaptation sémantique consiste en effet à adapter les notions de temps et de contrôle, ainsi que le format des données, à la frontière entre des modèles qui obéissent à des modèles de calcul différents.

L'adaptation des données peut paraître très simple lorsqu'il s'agit de détecter le changement de la valeur d'une variable dans le code non synchrone pour générer un événement pour le module synchrone. Mais les hypothèses sous lesquelles le module synchrone a été compilé peuvent créer des contraintes sur le

contrôle qui sont liées aux données. Par exemple, il est possible de spécifier qu'un signal d'entrée *m* d'un module synchrone est toujours présent lorsqu'un autre signal d'entrée *h* est présent. Cette hypothèse peut être prise en compte pour la génération de code et pour les preuves faites sur le module. Lors de l'exécution, l'adaptation sémantique entre le code hôte et le module synchrone doit assurer que l'hypothèse est vérifiée. Donc, si le changement provoquant l'émission du signal *h* est détecté, il ne faudra faire réagir le module synchrone que si le signal *m* est lui aussi présent. On voit ainsi que le contrôle (les instants auxquels réagit le module) peut être lié aux données. Le détail des mécanismes mis en jeu et des outils permettant d'intégrer du code Esterel ou Lustre dans un programme C++ sont décrits dans [1995-ED2][1997-IR1].

3.3 Graphes et modèles

L'approche « boîte noire » utilisée pour intégrer les modules synchrones dans un système hétérogène menait à considérer un système comme un ensemble de composants interconnectés. Chaque composant ne communique avec les autres qu'à travers des ports qui constituent son interface. Des relations entre ces ports modélisent explicitement les connexions entre composants. Un système peut alors être considéré comme un graphe orienté (les communications vont des ports de sortie vers les ports d'entrée) dont les sommets sont les composants. Une transformation du système telle que l'ajout de composants et la modification des connexions entre composants peut alors être considérée comme une transformation de son graphe. En s'appuyant sur la sémantique synchrone des communications entre composants, il est possible de transformer ce graphe de façon contrôlée afin d'obtenir un nouveau système dont le comportement est parfaitement défini. Nous avons appliqué cette approche à la définition de transformations élémentaires pour la sûreté de fonctionnement (réplication de composants, insertion de voteurs, etc.). Ces transformations peuvent être paramétrées et appliquées à des systèmes réactifs synchrones [2008-IR2][1998-IC1]. Implicitement, cette approche s'appuie sur les propriétés du *modèle de calcul* réactif synchrone pour définir la sémantique des transformations appliquées aux modèles.

3.4 Notion de modèle de calcul

C'est grâce à la rencontre avec le projet Ptolemy [2003-8] que la notion de modèle de calcul s'est imposée dans mes travaux. À l'époque, Ptolemy était une plateforme de modélisation et de simulation de modèles hétérogènes écrite en C++ [1991-37]. Selon une analogie cosmologique, les composants y était nommés *Star*, les composites (assemblages de composants) *Galaxy*, et les modèles *Universe*. Les modèles de calcul étaient représentés par des *Domain*, non pas en référence aux « domaines métier », mais en référence à des parties de l'univers qui obéiraient à des lois physiques différentes. L'équipe de Berkeley s'intéressait au paradigme synchrone, et c'est en cherchant à intégrer à Ptolemy des *Star*s dont le comportement était écrit en Lustre ou en Esterel que l'adaptation sémantique

qui était implicite dans mes travaux de thèse s'est avérée être un aspect important pour la conception des systèmes.

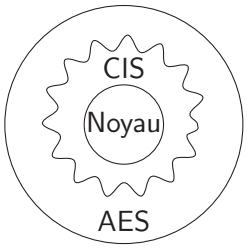
La conception et l'implémentation d'un traducteur de modules synchrones en *Star* Ptolemy adaptées à différents domaines m'a alors permis de percevoir mes travaux de thèse comme un cas particulier d'un problème plus général. Le modèle réactif synchrone n'était qu'un modèle de calcul parmi d'autres, et l'utilisation de plusieurs modèles de calcul dans un modèle d'un système s'avérait utile et indispensable. Le problème était donc de décrire des machines d'exécution pour les modèles de calcul, et d'intégrer ces différentes machines d'exécution pour qu'elle puissent cohabiter dans un même système.

Les travaux de Stephen A. Edwards sur l'interprétation des modèles synchrones par point fixe [1997-9] et la réalisation dans Ptolemy du domaine SR (Synchronous Reactive) m'ont amené à réaliser le domaine de génération de code SRCGC (SR Code Generation C), dual de SR, en collaboration avec Thomson-CSF Optronics. Dans la version *Classic* de Ptolemy, un modèle de calcul pouvait en effet être implémenté sous la forme d'un domaine de simulation et sous la forme d'un domaine de génération de code dual. Au cours de ces travaux, il est apparu que la génération d'un composant (*Star*) spécifique pour chaque modèle de calcul (*Domain*) était non seulement fastidieuse, mais nuisait à l'utilisation de cette approche en obligeant les concepteurs d'un système à maîtriser la chaîne logicielle de production de ces composants « domaine-spécifiques ».

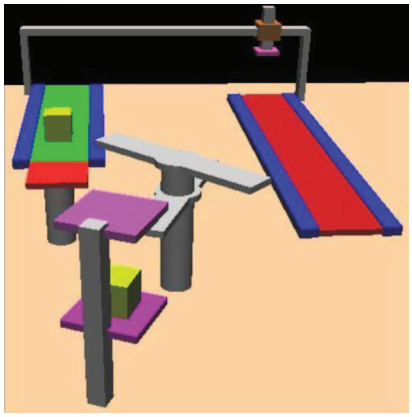
3.5 Composants domaine-polymorphes

La coopération avec Thomson-CSF (devenu Thales depuis) s'est poursuivie dans l'objectif de découpler la traduction du code synchrone et l'adaptation sémantique avec le domaine hôte. Ce découplage permettait d'envisager de ne traduire qu'une fois le code Esterel ou Lustre en C++ ou en Java, et de n'ajouter l'adaptation sémantique qu'au moment où le composant était utilisé dans un domaine particulier.

Les travaux de thèse de Mohammed Feredj [2004-IC2][2009-IR3] ont abouti à ce découplage qui est à l'origine de la notion de bloc d'interface que nous utilisons actuellement dans notre plateforme de modélisation hétérogène ModHel'X.



L'architecture proposée par M. Feredj se compose d'un noyau comportemental synchrone, généré directement à partir d'un module synchrone. Ce noyau nécessite une machine d'exécution synchrone qui lui est fournie par un composant CIS (Conservation of the Internal Semantics) qui préserve la sémantique synchrone dans l'environnement d'exécution du noyau comportemental. Afin de pouvoir être utilisé dans un domaine autre que synchrone, le comportement de la paire noyau-CIS est adapté à la sémantique du domaine hôte par un composant AES (Adaptation to the External Semantics). La principale difficulté est de définir l'interface entre les CIS et les AES puisqu'elle doit permettre à n'importe quelle paire de domaines de s'adapter l'un à l'autre.



La solution obtenue à l'issue de cette thèse s'appuie sur la transformation des données et le filtrage du contrôle (le noyau comportemental n'est activé que lorsque c'est pertinent selon son modèle de calcul) et s'applique bien entre les modèles à flots de données, à événements discrets, et réactif synchrone. Elle a notamment été appliquée avec succès au cas d'étude "Production Cell" [1995-10, 1995-11] dans Ptolemy II. Ce cas d'étude décrit une unité de production constituée de deux convoyeurs à bande, d'un robot, d'une presse et d'un transbordeur. Un des convoyeurs amène les pièces brutes au robot qui les place dans la presse. Une fois pressée, la pièce est placée par le robot sur le second convoyeur. Le transbordeur déplace les pièces qui arrivent en fin du deuxième convoyeur vers le premier convoyeur afin que l'unité puisse fonctionner en continu. Dans une véritable unité de production, le transbordeur serait chargé d'alimenter le premier convoyeur à partir d'un stock de pièces brutes, et de placer les pièces finies dans un autre stock.

L'intérêt de cette approche est qu'il n'est nécessaire de concevoir qu'un seul composant CIS par modèle de calcul intégré, et un seul composant AES par modèle de calcul hôte. En supposant que tous les modèles de calcul sont utilisables dans les deux situations, il ne faut que $2N$ composants CIS et AES pour représenter les $N(N-1)$ combinaisons possibles de N modèles de calcul.

3.6 Conclusion sur les objets synchrones et l'hétérogénéité

La nature très particulière du modèle réactif synchrone ainsi que ses hypothèses d'exécution et de communication instantanées incompatibles avec ce qu'offrent les langages de programmation généralistes, ont mis en évidence la nécessité de machines d'exécution. Ces machines d'exécution s'appuient sur la notion de modèle de calcul défini en tant qu'ensemble de règles d'interprétation de la structure d'un modèle. La disponibilité de nombreux modèles de calcul dans Ptolemy, ainsi que la qualité de l'environnement de modélisation et de simulation qu'il offre, nous ont donné l'opportunité d'étudier l'adaptation sémantique entre modèles de calcul sur de nombreux exemples.

Cette adaptation sémantique, perçue au départ comme une contrainte, et réalisée le plus souvent au niveau de l'implémentation, est devenue un élément clef dans la conception des systèmes une fois que la nécessité et l'utilité de l'hétérogénéité des modèles ont été acceptées.

Toutefois, la nature discrete et multiforme du temps synchrone, tout comme la nature très abstraite du temps dans les réseaux à flots de données, nous ont permis de traiter le temps de manière unifiée avec le contrôle. En effet, dans les réseaux à flots de données, il n'y a activation des comportements que lorsque les données sont en nombre suffisant, et aucune étiquette temporelle n'est associée aux données. De même, dans l'approche synchrone, le temps se résume à une

succession d'événements auxquels le système réagit. Les événements ne portent pas de date, et il n'y a pas de notion de durée entre les événements. Ce n'est qu'avec la prise en compte de modèles à temps continu, et notamment de modèles hybrides, où l'écoulement du temps peut être une source de contrôle, que nous considérerons le calcul du temps comme une opération distincte du calcul du contrôle dans l'adaptation sémantique. Avant cela, nous allons traiter dans la section suivante d'une autre approche de l'hétérogénéité, qui ne s'appuie pas sur l'encapsulation hiérarchique des modèles pour effectuer l'adaptation sémantique entre modèles de calcul.

4 2002 – 2004 : Modélisation hétérogène non hiérarchique

L'approche de la modélisation hétérogène issue de nos travaux sur le synchrone s'appuie sur la composition hiérarchique pour isoler les parties d'un modèle qui obéissent à des modèles de calcul différents. Cette approche, qui est aussi utilisée dans Ptolemy, a pour principal intérêt de limiter la complexité des interactions entre modèles de calcul, puisqu'au plus deux modèles de calcul peuvent être en contact : celui d'un modèle d'un composant et celui du modèle dans lequel ce composant est utilisé. En revanche, cette approche ne permet pas de modéliser des composants dont tous les ports n'obéissent pas à la même sémantique. Par exemple, dans un convertisseur analogique-numérique, l'échantillonneur-bloqueur aura une entrée analogique, qui reçoit un signal modélisé par une fonction continue du temps, une entrée d'horloge, qui reçoit l'événement déclenchant l'acquisition d'un échantillon du signal d'entrée, et une sortie sur laquelle apparaissent les échantillons successifs du signal. Un tel composant se trouve donc à la frontière de trois modèles de calcul, l'un supportant les signaux continus, le second les événements discrets, et le troisième les échantillons de données.

Dans la thèse de Mokhoo Aimé Mbobi, nous avons montré que des modèles faisant appel à de tels composants peuvent être transformés sous certaines conditions en modèles hiérarchiques afin de bénéficier du découplage entre modèles de calcul [2004-IC3, 2005-IC4][2007-IR4]. C'est ce que nous présentons dans cette section.

4.1 Composants à interface hétérogène

Un composant à interface hétérogène est un composant dont tous les ports n'obéissent pas au même modèle de calcul. L'échantillonneur-bloqueur cité plus haut en est un exemple simple. Un tel composant reçoit une fonction du temps continu en entrée, c'est-à-dire une fonction qui associe à chaque réel positif (date) une valeur donnée. La commande d'échantillonnage se fait par des événements discrets : à chaque fois qu'un événement est présent sur cette entrée du composant, la valeur de la fonction d'entrée à la date de l'événement est échantillonnée, et l'échantillon correspondant est placé sur la sortie. La sortie de l'échantillonneur est une suite d'échantillons, c'est-à-dire un flot de données non datées.

Il est bien sûr possible de modéliser un tel composant de façon homogène en représentant les événements discrets par des impulsions de durée nulle, et la suite d'échantillons par une fonction en escalier qui fournit à tout instant la dernière valeur échantillonnée. Toutefois, ce type d'approche qui consiste à choisir le PPCMC (Plus Petit Commun Modèle de Calcul) n'est pas efficace du point de vue de la modélisation. En effet, il est difficile de raisonner sur la notion d'échantillon quand les échantillons sont représentés par une fonction (non continue) du temps continu, et les événements, par nature instantanés, nécessitent un traitement particulier pour être pris en compte par les solveurs de comportements continus. Il est donc important de pouvoir modéliser ce type de composants en utilisant des modèles de calcul appropriés à chaque élément de leur interface.

Le principal problème posé par les composants à interface hétérogène est qu'ils obligent à traiter en même temps les interactions entre plusieurs modèles de calcul, alors que la composition hétérogène hiérarchique permet de ne prendre en compte les modèles de calcul que deux à deux. Cependant, ces composants définissent par leur comportement l'adaptation sémantique entre les modèles de calcul à la frontière desquels ils opèrent. En effet, dans l'approche hiérarchique de l'hétérogénéité, on cherche à découpler les modèles de calcul car on souhaite pouvoir réutiliser un composant dans différents contextes qui peuvent faire appel à différents modèles de calcul. L'adaptation sémantique doit alors être spécifiée indépendamment du modèle du composant. Dans le cas des composants à interface hétérogène, c'est le comportement du composant qui décrit l'adaptation sémantique, et le problème est de trouver une méthode pour exprimer ce comportement sans être obligé de faire intervenir plusieurs modèles de calcul pour interpréter le comportement d'un composant.

4.2 Projections d'un composant

Le principe que nous avons utilisé pour concilier la nécessité de construire des modèles hétérogènes plats et la commodité de la modélisation hiérarchique consiste à réécrire de manière systématique les modèles plats en modèles hiérarchiques. Considérons par exemple le modèle suivant :

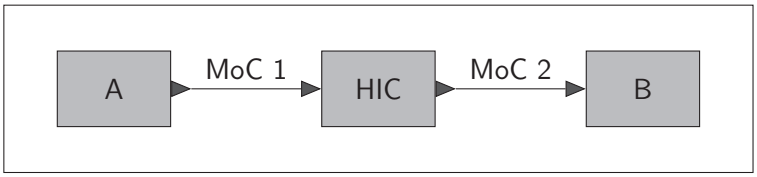


FIGURE 3 – Exemple de modèle hétérogène plat

Ce modèle fait appel à un composant à interface hétérogène HIC qui possède un port d'entrée obéissant au modèle de calcul MoC 1 et un port de sortie obéissant au modèle de calcul MoC 2. Le modèle de calcul auquel obéit ce modèle n'est pour l'instant pas défini et devrait être un hybride des modèles de calcul MoC 1 et MoC 2. Afin de revenir à une modélisation hétérogène hiérarchique, un tel modèle est transformé en créant des sous-systèmes homogènes dans lesquels le composant à interface hétérogène est représenté par une projection qui n'obéit qu'à un seul modèle de calcul :

Le sous-système de gauche Ω_1 obéit entièrement et uniquement au modèle de calcul MoC 1, et le composant à interface hétérogène y est représenté par sa projection $\mathrm{HIC}_{\mathrm{MoC\ 1}}$ qui ne comporte que le port d'entrée obéissant à ce modèle de calcul. De même, le sous-système de droite Ω_2 modélise les échanges effectués selon le modèle de calcul MoC 2. Ces deux sous-systèmes sont couplés par le comportement du composant à interface hétérogène. Ce couplage est représenté par la relation en pointillés entre les sous-systèmes Ω_1 et Ω_2. Une telle relation est un *canal hétérogène abstrait*. Elle ne sert qu'à représenter la dépendance entre les projections du composant à interface hétérogène.

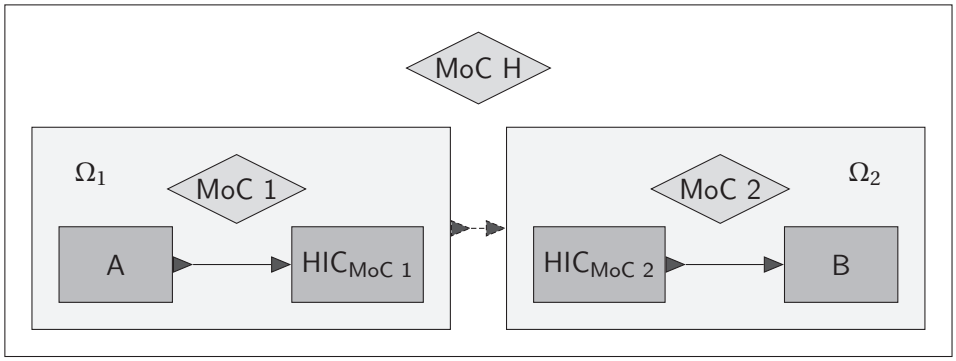


FIGURE 4 – Transformation en modèle hiérarchique

Lors de la partition d'un modèle plat en un modèle hiérarchique, il peut aussi arriver que des composants qui communiquent de façon homogène soient placés dans des sous-systèmes distincts. La relation entre leurs ports doit apparaître entre les sous-systèmes qui les contiennent afin que la dépendance induite puisse être prise en compte. Une telle relation est un *canal homogène abstrait*.

Les composants Ω_1 et Ω_2 doivent être utilisés dans un modèle global du système, qui obéit à un certain modèle de calcul MoC H. Ce modèle de calcul doit pouvoir composer les comportements de composants modélisés selon n'importe quel modèle de calcul. Son principal rôle est d'ordonnancer les sous-systèmes produits par la transformation des modèles hétérogènes plats en modèles hiérarchiques. Ce modèle de calcul hétérogène non hiérarchique est donc par nature très général : il n'interprète les canaux abstraits qui relient les sous-systèmes que pour déterminer leurs dépendances et effectuer leur ordonnancement. Il ne propage notamment pas de données le long de ces canaux car il en ignore complètement la nature et la sémantique. Les transferts d'information entre les sous-systèmes sont effectués par les composants à interface hétérogène qui codent la sémantique des canaux hétérogènes abstraits, et par des composants de communication insérés automatiquement lors de la transformation du modèle afin de prendre en charge des canaux homogènes abstraits.

4.3 Équivalence avec l'approche hiérarchique

Cette approche par transformation de modèles, que nous avons décrite dans [2007-IR4], permet de prendre en compte les composants à interface hétérogène dans le cadre de la modélisation hétérogène hiérarchique. Au prix de certaines limitations que nous verrons plus loin, la composition hiérarchique de modèles hétérogènes, qui permet de préserver la modularité des modèles et limite l'explosion du nombre de politiques d'adaptation sémantique à définir, permet donc de prendre en compte des composants qui opèrent naturellement à la frontière de plusieurs modèles de calcul.

Inversement, ces composants à interface hétérogène ont un comportement qui interagit avec plusieurs modèles de calcul. Il semble donc naturel de les utiliser pour modéliser l'adaptation sémantique. Ce dernier point n'a toutefois pas été abordé dans le travail de thèse de M. Mbobi, mais il est considéré comme une piste

à étudier dans le cadre d'une thèse sur la modélisation explicite de l'adaptation sémantique qui vient de débuter.

4.4 Limites

La transformation d'un modèle hétérogène plat en un modèle hétérogène hiérarchique est toujours possible. Par contre, le modèle hiérarchique obtenu n'a pas systématiquement une sémantique bien définie selon le modèle de calcul hétérogène que nous avons évoqué plus haut. En effet, ce modèle de calcul, afin d'être très général et de pouvoir combiner le comportement de composants modélisés selon n'importe quel modèle de calcul, ne fait aucune hypothèse sur la nature des composants, ni sur celle des données qu'ils échangent. Il ne fait qu'interpréter les relations entre ports comme des relations de dépendance afin d'ordonnancer les composants.

Cet ordonnancement n'est possible qu'en l'absence de boucle dans le graphe de dépendance. En effet, la sémantique des boucles dépend des modèles de calcul. Il serait éventuellement possible de discerner les boucles instantanées des boucles qui contiennent un retard, mais la notion même de retard n'est pas traitée de la même façon selon les modèles de calcul. Il peut s'agir d'un jeton initial sur un canal de communication (par exemple dans les modèles à flots de données), mais il peut aussi s'agir d'un composant dont la sortie ne dépend que de l'état, et qui est donc connue avant que les entrées du composant soient connues. La sémantique des boucles sans retard est encore plus délicate puisqu'elle correspond à un point fixe du comportement du modèle en boucle ouverte. L'existence et l'unicité de ce point fixe, ainsi que sa valeur, dépendent d'une part du modèle en boucle ouverte, et d'autre part des modèles de calcul utilisés pour modéliser des différents composants présents dans la boucle.

La prise en compte des boucles dans les modèles hétérogènes plats nécessite donc de définir précisément leur sémantique, ce qui jusqu'à présent semble nécessiter la définition d'un modèle de calcul englobant tous les modèles de calcul utilisés dans le modèle, ainsi que leurs interactions.

4.5 Conclusion

Ces travaux sur la modélisation hétérogène non hiérarchique ont d'une part confirmé l'intérêt des approches *ad hoc* pour la modélisation d'interactions complexes entre modèles de calcul, et d'autre part montré qu'un cadre plus général permettait de décrire ces approches *ad hoc*. Un outil qui ne prend en compte qu'un nombre limité de modèles de calcul définis une fois pour toutes permet en effet de décrire complètement la manière dont se combinent les comportements selon les différents modèles de calcul, ce qui permet généralement de faire des simulations plus rapides, de générer du code plus efficace, et de prendre en compte des boucles dont la sémantique est connue. Toutefois, notre objectif est de proposer un environnement de modélisation ouvert à tous les modèles de calcul, et permettant d'utiliser conjointement différents modèles de calcul dans un modèle d'un système. Ces travaux montrent qu'en renonçant aux boucles, les modèles

hétérogènes plats peuvent être traités dans ce cadre par une transformation systématique du graphe du modèle. Un effort de recherche sur la modélisation des retards dans différents modèles de calcul permettrait éventuellement de prendre en compte les boucles qui contiennent des retards. De plus, des travaux récents laissent envisager que les composants à interface hétérogène utilisés dans ce type de modèle pourraient fournir un mécanisme intéressant pour modéliser l'adaptation sémantique entre modèles de calcul dans les modèles hiérarchiques.

5 2005 – 2008 : Un cadre pour décrire les modèles de calcul

Les travaux menés jusqu'ici sur les machines d'exécution, l'adaptation sémantique et la modélisation hétérogène non hiérarchique reposaient principalement sur l'infrastructure de modélisation du projet Ptolemy de l'université de Berkeley. Ces travaux nous ont amenés à développer une vision de la modélisation hétérogène plus radicale que celle de Ptolemy. Nous souhaitions notamment :

- décrire plus formellement les modèles de calcul ;
- généraliser des mécanismes qui n'étaient disponibles que pour certains modèles de calcul ;
- mieux découpler certains modèles de calcul hybrides ;
- autoriser des synchronisations complexes entre les horloges des différents sous-systèmes ;
- décrire explicitement l'adaptation sémantique entre modèles de calcul.

Le premier point est assez délicat si l'on souhaite décrire n'importe quel modèle de calcul en vue de simuler, valider et générer du code pour des modèles. Nous décrivons actuellement les modèles de calcul en Java, mais avec un découpage en opérations élémentaires qui définissent une sémantique abstraite. Nous pensons actuellement que la description purement formelle d'un modèle de calcul ne peut pas être directement exploitée de façon systématique pour la simulation, la validation et la génération de code. Toutefois, la représentation d'un modèle de calcul à l'aide des actions élémentaires d'une sémantique abstraite permet d'isoler certaines caractéristiques (ordonnancement, modes de communication entre composants, synchronisation), et qu'il s'agit d'une étape indispensable pour la construction d'une plateforme permettant d'utiliser conjointement des modèles de calcul quelconques.

Le second point consiste à rendre disponibles au niveau de la plateforme de modélisation, des mécanismes introduits de manière ad hoc dans certains modèles de calcul. Par exemple, lors du passage de Ptolemy Classic à Ptolemy II, l'équipe de Berkeley a introduit dans la plateforme la notion d'acteur non strict et la possibilité de calculer le comportement d'un modèle par une suite d'itérations de l'activation de son comportement qui converge vers un point fixe. Ces notions étaient apparues dans le domaine SR (Synchronous Reactive) de façon à supporter la sémantique constructive de la rétroaction telle qu'utilisée pour la compilation d'Esterel [2000-38]. En les rendant disponibles dans la plateforme, la nouvelle version devenait capable de supporter une classe de modèles de calcul plus large.

À l'opposé, le domaine CT (Continuous Time) dispose d'un mécanisme de validation des calculs effectués lors d'un pas de simulation, afin de prendre en compte la précision sur la date des événements discrets en plus de la précision sur la valeur des signaux. Ce mécanisme permet de traiter le cas où un signal varie très lentement, et ne requiert donc qu'un pas d'intégration relativement grand pour obtenir une précision suffisante, et est comparé à un seuil pour générer un événement (par exemple, la commutation d'un thermostat). Le grand pas d'intégration induit une grande incertitude sur la date de l'instant auquel le seuil est franchi. Le domaine CT de Ptolemy permet d'invalider le calcul fait lors du

pas de simulation où le signal franchit le seuil, afin de refaire les calculs avec un pas d'intégration plus faible, jusqu'à obtenir la précision suffisante sur la date de l'événement généré. Nous souhaitions incorporer ce mécanisme au niveau de la plateforme de modélisation, afin de pouvoir en bénéficier dans d'autres modèles de calcul. D'autre part, le domaine CT de Ptolemy traite à la fois les signaux continus et les événements discrets. Nous considérons que ces deux types de signaux relèvent de deux modèles de calcul distincts. Nous préférons donc avoir un modèle de calcul CT dans lequel les signaux sont uniquement continus, et utiliser l'adaptation sémantique pour faire explicitement le lien avec les événements discrets, même si cela induit une plus grande complexité dans le codage des modèles.

Un autre aspect dont nous souhaitions nous affranchir est le couplage entre la hiérarchie des modèles et la hiérarchie des horloges. Dans un modèle hiérarchique, les modèles imbriqués, considérés comme des composants du système, ne sont en général activés qu'à la demande du modèle dans lequel ils sont utilisés. C'est donc le modèle racine, au sommet de la hiérarchie, qui pilote l'écoulement du temps dans tous les modèles imbriqués. Nous souhaitions pouvoir donner à ces modèles un plus grand contrôle sur les instants auxquels ils sont activés, et permettre au temps de s'écouler à des vitesses différentes ou selon des modalités différentes dans les sous-systèmes.

Ces considérations nous ont amenés à concevoir notre propre plateforme de modélisation hétérogène, dont la conception a débuté dans le projet Usine Logicielle du pôle de compétitivité System@tic et avec la thèse de Cécile Hardebolle [2008-3].

5.1 Méta-modèle générique

Cette plateforme, nommée ModHel'X [2008-IC5][2009-IR5] (il s'agit d'un acronyme construit à partir de « Modélisation Hétérogène eXécutable »), définit un méta-modèle générique qui permet de représenter des modèles quelconques. L'unité de comportement est le *bloc*, boîte noire dont l'interface est composée de *pin*s. Plusieurs blocs peuvent être regroupés en un *bloc composite*, et des *relation*s peuvent être établies entre leur pins. Un *modèle* est constitué d'un bloc composite qui définit sa structure, et d'un *modèle de calcul* qui définit sa sémantique. Le comportement des blocs peut être codé en dehors de ModHel'X (code Java, modèles Simulink etc.) Il s'agit alors de blocs *atomiques*. Le comportement d'un bloc peut aussi être décrit par un modèle ModHel'X. On a alors un *bloc d'interface* qui permet de spécifier l'adaptation sémantique entre le modèle de calcul dans lequel il est utilisé et le modèle de calcul de son modèle interne.

Le méta-modèle de ModHel'X permet de représenter des systèmes très variés, depuis les réseaux à flots de données dans lesquels les blocs sont des opérateurs, les pins des séquences d'échantillons et les relations des buffers de communications ; jusqu'aux réseaux de Petri dans lesquels certains blocs sont des places, d'autres des transitions, et où les relations indiquent les liens entre places et transitions.

5.2 Sémantique abstraite

Le sens donné à la structure d'un modèle est donc défini par un modèle de calcul. De manière similaire à Ptolemy, ModHel'X dispose d'un moteur d'exécution générique, qui interprète la structure d'un modèle selon le modèle de calcul associé à cette structure. L'interface entre le moteur d'exécution et les différents modèles de calcul est un ensemble d'opérations qui constituent la sémantique abstraite de ModHel'X. Ces opérations ont en effet un sens très général (par exemple, initialiser le modèle, observer un bloc, propager les données) qui est spécialisé par chaque modèle de calcul pour définir une sémantique concrète. Les principales opérations de base de la sémantique abstraite de ModHel'X sont *schedule*, *update* et *propagate* :

- *schedule* permet de choisir un bloc à observer dans le modèle ;
- *update* est l'opération d'observation, au cours de laquelle un bloc prend en compte les informations disponibles sur son interface et les met à jour en fonction de son propre comportement ;
- *propagate* permet de propager les informations d'un bloc aux autres selon les relations établies entre leurs pins.

Ces opérations ont un rôle similaire à celles de la sémantique abstraite de Ptolemy. Elles sont exécutées en boucle jusqu'à ce que la valeur associée à chacun des pins des blocs du modèle soit déterminée. Il est ainsi possible de décrire des modèles de calcul dans lesquels une observation du modèle est construite par itérations successives vers un point fixe. Une fois que la valeur associée à chaque pin est déterminée, les blocs doivent valider l'observation qui vient d'être faite. Une observation validée est appelée un *snapshot* du modèle. Si l'observation calculée n'est pas validée, le modèle est restauré dans l'état où il était au début du calcul, et un nouveau calcul est lancé avec des paramètres différents (date courante, largeur du pas d'intégration etc.) jusqu'à ce que tous les blocs valident le snapshot. C'est uniquement quand le snapshot est validé que les blocs peuvent mettre à jour leur état interne en fonction des valeurs disponibles sur leurs pins.

Cette sémantique abstraite correspond à l'exécution d'un système séquentiel synchrone. Le calcul d'un snapshot est l'équivalent du calcul combinatoire effectué en fonction des entrées et de l'état courant du système. La mise à jour de l'état interne des blocs en fonction des résultats du calcul du snapshot est l'équivalent du chargement des registres d'un système séquentiel. L'étape de validation permet de prendre en compte certains cas dans lesquels une hypothèse faite pour effectuer les calculs n'est pas vérifiée par le résultat obtenu. Un exemple typique est le franchissement d'un seuil par un signal entre le snapshot précédent et le snapshot courant. Afin de déterminer précisément quand le seuil est franchi, on peut être amené à recalculer le snapshot courant en lui attribuant une date antérieure à celle estimée nécessaire dans un premier temps. Cette approche a été choisie afin de pouvoir définir précisément l'état du système ainsi que la notion d'instant. La succession des snapshots constitue l'horloge de base de ModHel'X. Chaque modèle de calcul peut dériver sa propre horloge de cette horloge de base, et les blocs peuvent imposer des contraintes entre les différentes horloges, par exemple pour assurer un échantillonnage périodique d'un signal par un modèle.

Enfin, lors d'une simulation, il est possible de relier les différentes horloges du système au temps de la simulation, qui peut lui même être lié au temps physique par la plateforme d'exécution.

5.3 Composition hiérarchique

Comme nous l'avons vu précédemment, au prix de certaines restrictions, un modèle hétérogène plat peut être transformé en un modèle hétérogène hiérarchique. ModHel'X s'appuie donc sur la structuration hiérarchique pour construire des modèles hétérogènes. Ce sont les blocs d'interface, qui permettent de décrire le comportement d'un bloc à l'aide d'un modèle ModHel'X, qui autorisent à la fois la structuration hiérarchique d'un modèle et le changement de paradigme de modélisation lors du changement de niveau hiérarchique. Leur fonctionnement sera décrit dans la section suivante. Au début du développement de ModHel'X, leurs fonctionnalités étaient assez réduites, l'objectif principal étant de consolider la sémantique abstraite afin de pouvoir décrire des modèles de calcul.

À la fin du projet Usine Logicielle, la sémantique abstraite de ModHel'X permettait l'exécution de modèles hiérarchiques avec calcul de point fixe, c'est-à-dire avec prise en compte de blocs non stricts (capables de fournir une évaluation partielle de leurs sorties à partir d'une évaluation partielle de leurs entrées). Ce résultat nous fournissait de nouveaux outils pour analyser les relations entre modèles de calcul. Ces outils ont été mis en œuvre dans le cadre du projet EDONA que j'évoquerai dans la section suivante.

5.4 Application à la validation de modèles

Dans le projet Usine Logicielle, nous participions à deux sous-projets. Le développement de ModHel'X se faisait dans le cadre d'OpenDev Factory, dont l'objectif était de réaliser une plateforme de conception de logiciel embarqué fondée sur des outils Open Source comme Eclipse. Notre participation à un autre sous-projet nommé MoDriVal (Model Driven Validation) nous a permis d'appliquer nos idées sur la modélisation hétérogène à la validation de modèles. Le principe du langage et de l'outil ADLV (Architecture Description Language for Validation) que nous avons développé au cours de ce projet [2008-IC6] est de séparer clairement la spécification du contrôle d'un système de celle des traitements qu'il effectue. L'intérêt d'une telle séparation est de permettre la validation formelle du contrôle, et notamment la vérification de propriétés de sûreté. Dans l'application que nous avons faite de ce principe, en modélisant le contrôle grâce à l'approche réactive synchrone, nous pouvions bénéficier des outils de model-checking pour vérifier les propriétés.

Toutefois, le concepteur d'un système veut généralement vérifier des propriétés globales du système, qui font intervenir les données traitées. L'intérêt de l'approche hétérogène, qui favorise l'utilisation du paradigme de modélisation le plus adapté à chaque partie du système, est qu'elle nous a permis d'avoir d'une part une spécification formelle du contrôle, sur laquelle il est possible de faire des vérifications automatiques, et d'autre part une description précise de la manière

dont étaient calculées les informations fournies au contrôle. Cette description nous a permis de concevoir un algorithme capable de transformer une propriété faisant intervenir des données en une propriété du contrôleur, vérifiable par les outils de model-checking. L'effort de conception supplémentaire exigé par la description de l'interface entre le contrôle et les traitements, outre le fait d'autoriser la transformation systématique de propriétés du système en propriétés du contrôle, a aussi permis de générer automatiquement le code *glue* nécessaire pour implémenter le système à partir du code généré par les compilateurs Lustre ou Esterel pour le contrôle, et Simulink ou Scilab pour les traitements [2008-IC7].

5.5 Conclusion

La construction de modèles hétérogènes nécessite la description des interactions entre des modèles qui utilisent des modèles de calcul différents. Notre parti pris, qui consiste à définir un cadre dans lequel tout modèle de calcul peut a priori être utilisé, nous empêche d'utiliser un modèle de calcul « union » et de figer ainsi à la fois la sémantique des modèles de calcul et celle de leurs interactions.

Il est donc nécessaire de pouvoir décrire chaque modèle de calcul indépendamment des autres avant de pouvoir décrire comment ces modèles de calcul interagissent. Notre objectif étant de pouvoir calculer le comportement d'un modèle, notre approche s'est orientée vers la conception d'un moteur d'exécution générique. Ce moteur d'exécution déroule un algorithme dont les opérations élémentaires interprètent la structure des modèles. Ces opérations élémentaires constituent la sémantique abstraite de ModHel'X, de même que le méta-modèle de ModHel'X, qui permet de décrire uniformément la structure des modèles, constitue sa syntaxe abstraite.

La définition d'un modèle de calcul dans ModHel'X consiste donc à donner une sémantique concrète à chacune des opérations élémentaires. Ce faisant, on établit aussi une correspondance entre les concepts du domaine auquel s'applique ce modèle de calcul (états, opérateurs, événements, flots de données, fonctions) et les concepts de ModHel'X (bloc, pin, relation). Définir un modèle de calcul est un travail de spécialiste, mais une fois que la définition est disponible, elle peut s'appliquer à tous les modèles qui utilisent ce modèle de calcul.

ModHel'X n'étant qu'une plateforme expérimentale qui nous permet de mettre en œuvre de nouvelles idées et de tester leur pertinence dans le cadre de notre approche, il reste nécessaire de confronter ces idées à des problèmes réels et de les transposer à des outils et des méthodologies de modélisation utilisées en production. La thèse d'Abderraouf Benyahia, dirigée par François Terrier, du laboratoire LISE au CEA à Saclay, et que je co-encadre, consiste à formaliser la sémantique du modèle d'exécution fUML [2010-IC8]. En s'inspirant des définitions et des mécanismes utilisés dans ModHel'X, le moteur d'exécution standard de fUML a été rendu générique, et les points de variation sémantiques, qui correspondent à des familles de modèles de calcul, peuvent être précisés par le choix de différentes politiques d'ordonnancement et de communication entre tâches. La syntaxe des modèles étant celle d'UML, des profils sont utilisés pour indiquer la sémantique d'exécution à utiliser pour les modèles. Ces travaux se placent de le cadre du projet

AccordUML [2003-12].

Dans la suite, nous examinons comment, une fois les modèles décrits à l'aide d'un unique méta-modèle, une fois les modèles de calcul définis selon la même sémantique abstraite, il devient possible de décrire l'adaptation sémantique à la frontière entre deux modèles qui utilisent des modèles de calcul différents, et en quoi cette description explicite favorise la modularité et la réutilisation.

6 2006 – 2010 : Un cadre pour décrire l'adaptation sémantique

Le développement de la plateforme ModHel'X durant le projet Usine Logicielle avait porté principalement sur l'infrastructure permettant de décrire les modèles et les modèles de calcul. Les mécanismes permettant l'adaptation sémantique entre modèles de calcul étaient intégrés à cette infrastructure, mais la description de l'adaptation sémantique n'était pas développée. Notre participation au projet EDONA du pôle de compétitivité System@tic avait pour objectif de décrire précisément certains modèles de calcul utilisés dans le domaine de l'électronique automobile, ainsi que l'adaptation sémantique entre ces modèles de calcul.

Nous considérons que la description explicite de l'adaptation sémantique entre des modèles qui utilisent des modèles de calcul différents est un point important car le comportement global du modèle en dépend. Cette adaptation est très souvent implicite, soit parce qu'elle est effectuée de manière prédéterminée par l'outil de modélisation, soit parce qu'elle est repoussée à la phase d'intégration des composants. Notre objectif est de permettre la description explicite de l'adaptation sémantique de manière modulaire et réutilisable. La modularité consiste à découpler l'adaptation sémantique et les modèles qu'elle met en relation : si on change la nature d'un des modèles, il faudra éventuellement ajuster l'adaptation sémantique, mais il ne devrait pas être nécessaire de modifier l'autre modèle. La réutilisabilité est obtenue par la définition de patrons d'adaptation sémantique qu'il est possible de paramétrer et d'appliquer de façon à réaliser des adaptations courantes comme l'échantillonnage périodique (adaptation continu-disret), l'étiquetage temporel (adaptation événements discrets-systèmes réactifs), etc. Notre approche ne consiste pas à définir une méthode particulière d'adaptation entre modèles hétérogènes, comme cela est fait par exemple dans [2007-39]. Notre objectif est de définir une infrastructure qui permet l'utilisation de telles méthodes lorsqu'elles sont disponibles pour les modèles de calcul considérés.

Au cours du projet EDONA, nous avons décrit dans ModHel'X les modèles de calcul SDF (Synchronous Data Flow), DE (Discrete Events) et TFSM (Timed Finite State Machines), de façon à pouvoir traiter un exemple significatif. Le système retenu pour l'exemple est un lève-vitre électrique dont les différents éléments communiquent par un bus CAN (modélisé en DE). Le contrôleur du lève-vitre est modélisé par un automate temporisé (TFSM), et la partie electro-mécanique est modélisée en SDF (similaire à un schéma Simulink). L'utilisation conjointe de modèles à temps discret (SDF) activés périodiquement, d'automates temporisés utilisant à la fois des événements non datés et des transitions temporisées, le tout dans un environnement à événements discrets, nous a permis d'identifier trois axes principaux dans l'adaptation sémantique entre modèles de calcul.

6.1 Adaptation des données

Les modèles de calcul n'utilisent pas tous les mêmes types de données pour les échanges entre composants. Par exemple, dans un modèle à événements discrets, les données échangées sont constituées d'une paire (valeur, date), la valeur ayant

elle-même un type propre aux données échangées, et la date étant un réel, ou une paire de $\mathbb{R} \times \mathbb{N}$ dans le cas du temps super-dense qui permet à plusieurs événements de s'enchaîner causalement à la même date physique [2005-40]. Lorsqu'un tel événement discret est transmis à un automate, pour lequel les données sont de simples symboles déclenchant des transitions, il faut effectuer une adaptation des données, c'est-à-dire construire un symbole pour l'automate à partir de l'événement. De même, quand l'automate produit un symbole à destination de l'environnement DE, il faut construire un événement discret à partir du symbole, et notamment choisir une valeur et une date pour cet événement.

Le choix de l'adaptation des données est propre à chaque modèle et fait partie de l'effort de conception du système. Nous considérons que ce choix ne peut pas être fait automatiquement par la plateforme de modélisation, et qu'il doit apparaître explicitement dans le modèle, au même titre que les autres éléments qui sont habituellement considérés comme faisant partie d'un modèle. Il existe pourtant des schémas d'adaptation classiques qui doivent pouvoir être décrits et réutilisés dans plusieurs modèles avec des paramètres différents. Par exemple, dans le cas de l'interface entre DE et FSM, un schéma d'adaptation classique consiste à associer un symbole de l'automate à chaque valeur possible de l'événement discret, et, en sortie de l'automate, à associer une valeur d'événement à chaque symbole de l'automate.

6.2 Adaptation du temps

L'adaptation du temps entre modèles de calcul est plus complexe. Initialement, nous pensions pouvoir exprimer les relations entre les notions de temps deux à deux aux frontières entre les modèles. Par exemple, entre DE et FSM, lorsqu'un événement discret est fourni à l'automate, on mémorise sa date, et si l'automate produit un symbole, on utilise cette date pour l'événement fourni en sortie. Ceci revient à considérer que l'automate réagit instantanément dans le temps de DE.

Il s'est avéré que cette approche était trop simpliste. Elle ne permet notamment pas d'exprimer des relations entre les horloges de deux sous-systèmes plongés dans un environnement où le temps n'existe pas, ou est réduit à une suite d'instants auxquels se produisent des événements (par exemple, deux systèmes à temps continu communiquant dans un environnement DE n'ont pas de moyen de synchroniser leurs horloges, sauf si l'on suppose l'existence d'une horloge globale accessible dans chaque modèle).

Nous avons donc adopté une approche consistant à déclarer des relations entre horloges sans respecter l'imbrication hiérarchique des modèles auxquels elles appartiennent. Ces relations définissent un système de contraintes entre horloges qui est résolu lors de l'exécution pour déterminer la date courante dans chacun des modèles des sous-systèmes du modèle simulé. La synchronisation avec le temps réel lors d'une simulation est considérée comme une relation avec une horloge particulière fournie par la plateforme de simulation.

Ces idées sont largement inspirées de la notion de temps multi-forme utilisé dans les langages synchrones, et du sous-profil pour la modélisation des aspects temporels du profil MARTE [2007-13, 2007-14].

6.3 Adaptation du contrôle

Le contrôle est l'ensemble des instants auxquels les éléments d'un modèle sont observés et ont donc la possibilité d'influer sur le comportement du système. Le contrôle peut donc être considéré comme une horloge, et l'adaptation du contrôle entre deux modèles de calcul est très liée à l'adaptation des notions de temps. Par exemple, si l'on considère qu'un modèle à flots de données synchrones fonctionne avec une période T, cela signifie que l'on fait correspondre les instants $0, 1, 2, \ldots n$ de son horloge avec les instants $t_0, t_0 + T, t_0 + 2T, \ldots t_0 + nT$ de l'horloge de son environnement. En ce qui concerne le contrôle, cela implique que le modèle SDF sera observé quand le temps atteindra chacune de ces dates dans son environnement. L'écoulement du temps dans un modèle peut donc créer du contrôle dans un autre modèle.

De même, le contrôle est lié aux données. Pour reprendre l'exemple de l'automate plongé dans un environnement à événements discrets, lorsqu'un événement est produit, l'automate doit être observé afin que sa réaction au symbole correspondant à l'événement puisse être prise en compte.

Ces liens ne sont pas systématiques, et la disponibilité d'une donnée en entrée d'un modèle n'implique pas nécessairement qu'il faille lui donner le contrôle. Dans le cas d'un modèle à flots de données synchrones fonctionnant périodiquement dans un environnement à événements discrets, l'occurrence d'un événement en entrée du modèle ne doit pas créer de contrôle si la date de l'évenement n'appartient pas à son horloge d'observation périodique.

6.4 Interdépendance entre données, temps et contrôle

L'adaptation des données, qui consiste à déterminer quelles données doivent être communiquées à l'interface d'un composant ; l'adaptation du contrôle, qui consiste à décider de l'existence d'un instant sur l'horloge d'un composant, et des relations de causalité entre cet instant et ceux des autres horloges du modèle ; l'adaptation du temps, qui consiste à déterminer les étiquettes temporelles associées aux instants de chaque horloge du modèle, sont donc liées. Dans ModHel'X, nous avons introduit la notion de bloc d'interface pour permettre de spécifier le comportement d'un composant à l'aide d'un modèle qui peut utiliser un autre modèle de calcul que celui dans lequel est utilisé le composant. Les blocs d'interface sont donc chargés de traduire les données, le contrôle et le temps entre le modèle de calcul extérieur (dans lequel ils sont plongés) et le modèle de calcul interne qu'utilise le modèle du composant.

En fonction des données, du contrôle et de la date qu'il reçoit de son environnement, un bloc d'interface détermine les données et le contrôle qu'il fournit à son modèle interne. Il peut aussi imposer des contraintes sur son horloge ou sur celle du modèle interne. C'est ainsi que le bloc d'interface qui encapsule un modèle à flots de données synchrones dans un environnement à événements discrets peut imposer l'observation périodique de son modèle interne selon l'horloge de son modèle externe. C'est également ce mécanisme de contraintes entre horloges qui permet au bloc d'interface qui encapsule un automate temporisé

d'imposer l'existence d'un instant sur l'horloge de l'automate à la date à laquelle une transition temporisée arrivera à échéance.

6.5 Conclusion

L'identification des trois aspects de l'adaptation sémantique entre modèles de calcul et de leur interdépendance a permis de mettre en place les mécanismes nécessaires à leur prise en compte dans ModHel'X. L'utilisation de contraintes entre horloges d'activation, inspirée de l'approche synchrone et des travaux sur le profil MARTE, permet de modéliser des systèmes dans lesquels le temps prend différentes formes et ne s'écoule pas partout à la même vitesse.

L'utilisation dans ModHel'X d'un élément structurel dédié à la gestion de l'hétérogénéité permet d'y localiser le traitement de l'adaptation sémantique. Les avantages de cette localisation sont les suivants.

- L'adaptation sémantique devient explicite. Elle n'est pas codée en dur dans le moteur d'exécution et fait donc partie du modèle du système. Sa description est donc disponible pour les outils de vérification et de génération de code.
- L'adaptation sémantique se fait en dehors des modèles auxquels elle s'applique. Il n'est donc pas nécessaire de modifier un modèle pour l'utiliser dans un contexte particulier : l'adaptation se fait dans le bloc d'interface. Ceci conduit à une meilleure modularité et à une plus grande réutilisabilité des modèles de composants.
- Des blocs d'interface paramétrables peuvent être définis pour réaliser des adaptations sémantiques typiques entre modèles de calcul. Il est ainsi possible de constituer une bibliothèque de patrons d'adaptation sémantique.

Le projet EDONA nous a permis de mettre en œuvre ces concepts et de constater leur efficacité. Toutefois, il est apparu que le processus de définition de l'adaptation sémantique entre deux modèles de calcul reste délicat, car il fait appel aux opérations primitives du moteur d'exécution de ModHel'X, de manière semblable à la définition d'un modèle de calcul. Or, il serait souhaitable de décrire cette adaptation à l'aide d'un modèle, tout comme on décrit le comportement d'un système par un modèle. Ce modèle devrait alors traiter des informations appartenant à deux modèles de calcul. Il s'agirait donc de réaliser des modèles hétérogènes non hiérarchiques. Une thèse a débuté en octobre 2010 pour étudier la modélisation explicite de l'adaptation sémantique entre modèles de calcul, de façon à permettre au concepteur d'un système, non seulement de construire des modèles et de les assembler, mais aussi de modéliser avec la même facilité le « ciment » avec lequel il les assemble.

7 Travaux en cours

Les travaux de recherche menés jusqu'à maintenant nous ont permis de caractériser les modèles de calcul et de les décrire sous la forme de spécialisations de la sémantique abstraite d'une machine d'exécution générique. Cette description nous permet de simuler le comportements de modèles obéissant à tout modèle de calcul pouvant être décrit dans ce cadre. En nous appuyant sur la sémantique abstraite de ModHel'X, nous pouvons décrire explicitement l'adaptation sémantique entre des modèles qui obéissent à des modèles de calcul différents. Cette description peut prendre la forme de patrons d'adaptation paramétrables. Toutefois, ModHel'X reste une plateforme expérimentale que nous utilisons pour tester de nouvelles idées. Elle n'a pas vocation à devenir un outil de modélisation universel.

Notre approche consiste à appliquer les idées et les principes mis en œuvre et validés dans ModHel'X à des cas concrets pour lesquels d'autres outils sont utilisés. Il s'agit alors de définir les interactions entre ces différents outils pour tirer parti de leur capacités de modélisation et de validation.

Cette approche a ainsi été appliquée à la description d'applications combinant un contrôleur modélisé selon l'approche réactive synchrone, avec des algorithmes de traitement des données modélisés en Simulink. La description de l'adaptation sémantique entre les deux types de modèles a permis, d'une part d'utiliser les outils de model-checking de l'approche synchrone pour vérifier des propriétés du système, et d'autre part de générer le code complet du système à partir du code généré pour ses différents composants (projet MoDriVal).

Notre objectif est d'exploiter la plateforme ModHel'X pour développer des techniques de modélisation et de validation des modèles hétérogènes, en s'isolant dans un premier temps des difficultés techniques propres à l'utilisation de différents outils spécifiques (interconnexion des outils, transformation de modèles, adaptation d'API par exemple). Ces techniques de modélisation seront ensuite mises en œuvre afin de résoudre au cas par cas les problèmes concrets qui se posent aux utilisateurs de ces outils qui sont plus efficaces que ModHel'X en termes de validation, de génération de code ou d'ergonomie dans leur domaine d'application, et auxquels les utilisateurs sont habitués.

Les thèmes exposés ci-dessous sont déjà en cours d'étude dans une thèse ou un projet de recherche, bien qu'à divers degrés d'avancement.

7.1 Couplage entre vues d'un modèle

Nous avons déjà évoqué le fait qu'un même système peut avoir plusieurs modèles selon l'objectif de la modélisation. Nous avons identifié quatre raisons de construire des modèles utilisant différents paradigmes. Trois d'entre elles mènent à construire des modèles différents d'un même système : le changement de niveau d'abstraction, le changement d'objectif de modélisation, et le changement de point de vue. En ce qui concerne ce dernier point, deux approches sont possibles [2008-41] :

– définir un modèle complet du système et en dériver des vues par projection et donc masquage des aspects non pertinents pour chaque vue ;

– définir des modèles partiels du système pour chaque point de vue.

La première solution a l'avantage d'assurer la cohérence des vues puisqu'elle sont obtenues à partir d'un unique modèle. Elle a l'inconvénient de nécessiter un langage de modélisation permettant de prendre en compte les éléments de toutes les vues. Un tel langage est d'une part difficile à définir, et il constitue d'autre part un élément bloquant pour la définition de nouveaux types de vues puisque cela implique d'ajouter des éléments au langage.

La deuxième solution a pour avantage de ne pas nécessiter la construction d'un modèle global explicite du système. Le modèle du système est constitué de l'ensemble des modèles-vues. Par contre, il est tout à fait possible de construire en ensemble de vues contenant des contradictions et qui ne constitue alors pas un modèle global du système. Nous avons toutefois privilégié cette seconde approche et proposé une extension de ModHel'X permettant de synchroniser différentes vues d'un même système [2010-ED1]. Ces travaux montrent que cette synchronisation nécessite des transferts d'information entre vues qui obligent à relâcher l'opacité stricte des composants « boîtes noires ». La solution que nous proposons consiste à définir plusieurs interfaces pour un même composant, chacune étant destinée à échanger des informations soit avec les autres composants de la même vue, soit avec des composants d'une autre vue. Ce mécanisme est à rapprocher des différentes interfaces que propose une classe dans un langage à objets selon que l'on est dans le contexte de la même classe, d'une classe dérivée ou d'une autre classe quelconque (attributs *public*, *protected* et *private* par exemple).

Les travaux réalisés ne sont qu'une ébauche, et ils devraient être poursuivis d'une part pour permettre la simulation conjointe du comportement et des propriétés observables dans les différentes vues d'un modèle ; et d'autre part pour traiter le problème du raffinement d'une vue fonctionnelle sous les contraintes exprimées par des vues non fonctionnelles.

7.2 Test de modèles hétérogènes

Dans le cadre de la modélisation et de la validation des modèles hétérogènes, nous nous sommes intéressés au test de modèles hétérogènes. La thèse de Bilal Kanso, dirigée par Marc Aiguier de l'École Centrale Paris, aborde la modélisation des composants par une approche coalgébrique [2003-15] qui permet d'unifier les différentes représentations des systèmes à base d'états et de transitions. Cette formalisation unifiée a permis de généraliser l'algorithme de génération de tests de conformité présenté dans [2006-16] afin de l'appliquer à des modèles qui obéissent à différents modèles de calcul [2010-IC9].

Cette thèse aborde ensuite le problème du test d'intégration de modèles hétérogènes. Partant de composants conformes à leur spécification, comment vérifier qu'un modèle assemblant de tels composants hétérogène est conforme à sa spécification ? Dans cette approche, la composition des composants se fait à l'aide d'opérateurs d'intégration tels que le produit cartésien (exécution simultanée de composants) et la rétroaction (bouclage de certaines sorties d'un composant sur ses entrées). Ces opérateurs peuvent être combinés pour obtenir la mise en cascade et le produit synchrone.

Le cadre formel choisi nous a permis de définir une relation de conformité (par généralisation de la relation *ioconf* [1996-17]), et de montrer que le produit cartésien préserve cette relation de conformité. Le cas de la rétroaction est plus délicat, et la relation de conformité n'est préservée par cet opérateur que si la spécification de chaque composant accepte tous les événements en entrée dans chacun de ses états. Cette restriction vient du fait que la relation de conformité n'impose pas de contraintes sur les traces qui sont hors spécification : une implémentation peut se comporter librement à partir d'un état non spécifié. Or l'opérateur de rétroaction fait remonter les sorties de tels comportements en entrée des composants.

Ces travaux, qui se placent très en amont des applications réalisées à l'aide de ModHel'X, ont permis de mieux définir les éléments essentiels de la description d'un composant et de son comportement et d'établir un lien entre les approches à bases de connecteurs (comme BIP [2005-42, 2006-43, 2008-18] ou Metropolis [2007-19]) et les approches à base de modèles de calcul (comme Ptolemy II [2003-8] et ModHel'X [2009-IR5]).

8 Conclusion

L'étude d'un problème très particulier — l'intégration d'une nouvelle approche de la modélisation des systèmes réactifs synchrones dans une méthodologie de conception par objets — nous a amenés à identifier les notions de machine et d'environnement d'exécution, puis d'adaptation sémantique. À la même époque, la complexité croissante des systèmes déplaçait l'effort de conception du code vers les modèles. L'utilisation conjointe de plusieurs méthodes de modélisation pour les différentes parties d'un système montrait la nécessité de traiter des modèles hétérogènes. On assistait d'une part à la naissance de l'ingénierie dirigée par les modèles, puis de l'ingénierie des modèles, supportées par des éditeurs et des outils de transformation de modèles, et d'autre part, de façon plus discrète et plus lente, à celle de la modélisation multi-paradigme, sous les noms variés de « modélisation hétérogène », « simulation multi-physique », « co-simulation » etc.

8.1 Généralité du problème de l'adaptation sémantique

C'est dans ce contexte que nous avons perçu la généralité du problème initial de l'intégration de l'approche synchrone dans la programmation par objets. Les formalismes utilisés en physique, en traitement du signal, en automatique, pour les protocoles de communication etc. posent le même problème d'adaptation sémantique et d'environnement d'exécution que le formalisme réactif synchrone, et aucun n'est adapté à la modélisation d'un système complet. La nécessité d'utiliser conjointement plusieurs méthodes de modélisation oblige à décrire comment ces méthodes s'articulent. Notre choix de vouloir garder ouvert le jeu de méthodes disponibles pour modéliser les systèmes nous a amenés à suivre une approche de modélisation par acteurs, illustrée notamment par le projet Ptolemy, et à décrire les méthodes de modélisation sous la forme de modèles de calcul. C'est cette distinction entre la syntaxe utilisée pour décrire la structure d'un modèle et la sémantique utilisée pour interpréter ce modèle, qui nous a permis de décrire tous les modèles avec la même syntaxe abstraite (le meta-modèle de ModHel'X). L'étape suivante consistait à appliquer le même procédé à la sémantique des modèles, et donc à définir une sémantique abstraite, sous la forme d'un jeu d'opérations utilisé par un algorithme d'exécution générique, et dont la sémantique effective est définie pour chaque modèle de calcul.

Cette définition des modèles de calcul en tant que spécialisation des opérations d'une sémantique abstraite n'a pas atténué la difficulté de la définition d'un modèle de calcul. Nous espérions que la distinction entre les aspects liés à l'ordonnancement des composants et ceux liés à la communication entre composants ferait apparaître une structure plus claire des modèles de calcul, mais ces aspects sont en général très couplés, et leur spécification dans des opérations distinctes n'en est pas facilitée. Le véritable avantage apporté par la sémantique abstraite est la possibilité d'exécuter avec le même algorithme des modèles qui obéissent à des modèles de calcul différents.

Les travaux de thèse de M. Feredj et M. A. Mbobi ont permis de cerner les problèmes liés à l'adaptation sémantique entre modèles hétérogènes, et la recherche

de nouvelles solutions nous a poussé à concevoir notre propre plateforme de modélisation hétérogène. La thèse de Cécile Hardebolle, ainsi que notre participation au projet Usine Logicielle de System@tic, nous a permis de créer cette plateforme, mais aussi de mieux définir les concepts de composant (que nous appelons *bloc*), d'interface composée de ports (que nous appelons *pin*), et de connexion entre ports (que nous appelons *relation*). Nous avons choisi des noms différents de ceux employés habituellement afin d'éviter les confusions avec les approches à base de composants CORBA, les ports des diagrammes à flots de données etc. Le métamodèle de ModHel'X, à cause de sa syntaxe similaire aux schémas-blocs peut en effet susciter une interprétation trop « électronique ». L'équipe du projet Ptolemy avait rencontré le même problème, et avait choisi une analogie cosmologique avec des composants *Star*, des modèles *Universe*, et des passerelles entre modèles de calcul *WormHole*.

8.2 Nécessité de la modélisation multi-paradigme

Au début de la thèse de C. Hardebolle, en 2005, l'ingénierie dirigée par les modèles était déjà un domaine de recherche très actif. Mais si l'importance des modèles et des transformations de modèles était bien reconnue, la notion de modèle de calcul et la nécessité d'utiliser conjointement plusieurs modèles de calcul dans un modèle d'un système était moins bien perçue. C'est pourquoi une des contributions importantes de cette thèse est l'analyse du domaine de recherche de la modélisation multi-paradigme (CAMPaM, *Computer Aided Multi Paradigm Modeling*), et l'identification des sources d'hétérogénéité dans l'analyse et la conception des systèmes. Ce domaine de recherche est né à la frontière de l'informatique, de l'électronique et de l'automatique, en réponse à la migration de certaines fonctionnalités vers le logiciel.

L'introduction du logiciel dans les systèmes a permis de leur donner des comportements plus complexes que ceux qui étaient possibles avec la mécanique ou l'électronique. Pour s'adapter à cette complexité croissante, le processus de conception a évolué et nous amène à traiter de façon intégrée des modèles qui appartiennent à des domaines métier distincts. Le savoir-faire de ces domaines métier repose sur des techniques de modélisation pour l'analyse et la conception. Ces techniques doivent être prises en compte, d'une part parce que leurs utilisateurs ont une bonne intuition de la sémantique des modèles qu'ils construisent avec, et d'autre part parce qu'elles sont généralement bien adaptées à la classe de problèmes traités et au type de résultat désiré. La familiarité d'un concepteur avec la sémantique des modèles qu'il construit est extrêmement importante. En effet, la sémantique formelle donnée à un modèle est indispensable à son traitement automatisé par des algorithmes, mais si l'utilisateur a une perception du sens des modèles qui ne correspond pas à cette sémantique formelle, on peut s'attendre aux pires erreurs. Cet argument se décline selon les quatre axes de l'hétérogénéité : décomposition structurelle, niveau d'abstraction, propos du modèle, et aspect considéré.

Une fois convaincus de la nécessité des modèles hétérogènes et de la possibilité de les construire et de les simuler sur la même plateforme, le problème

de l'adaptation sémantique (qui était le problème initial) se posait de nouveau. L'adaptation sémantique implicite effectuée automatiquement par le moteur d'exécution posait problème car sa nature générique ne convient pas à toutes les situations. Il est donc souvent nécessaire d'ajouter des composants aux modèles pour la traiter correctement, ce qui nuit à la modularité et à la réutilisabilité des modèles. ModHel'X disposait des mécanismes nécessaires à la description explicite de l'adaptation sémantique entre modèles hétérogènes, mais cette adaptation n'était pas structurée, et il était difficile de définir des patrons d'adaptation paramétrables et réutilisables. Le projet EDONA nous a permis d'identifier les aspects *données*, *temps* et *contrôle* dans l'adaptation sémantique, et de nous appuyer sur la structuration hiérarchique des modèles ModHel'X pour décrire des patrons d'adaptation sémantique. Il nous reste maintenant à généraliser le traitement de cette adaptation afin de pouvoir le considérer comme un comportement modélisable selon la même syntaxe que les modèles qu'elle relie. La thèse d'Ayman Dogui, qui vient de débuter, a pour objectif de permettre de modéliser l'adaptation sémantique de manière aussi explicite que le comportement des composants.

8.3 Langages et modèles de calcul

Tout au long de ces travaux, le problème des relations entre modèles de calcul et langages de modélisation s'est posé. Il semble que les deux notions ne soient que deux approches de la description d'un modèle. Un langage de modélisation propose des opérateurs pour combiner des comportements élémentaires, alors qu'un modèle de calcul interprète la structure d'un modèle pour combiner le comportement de ses composants. Il est toujours possible de représenter un langage de modélisation sous la forme d'un modèle de calcul, il suffit pour cela de modéliser ses opérateurs par des composants lorsqu'ils effectuent un traitement, et par des relations interprétées par le modèle de calcul lorsqu'ils concernent le contrôle ou la communication. Il nous est toutefois apparu récemment que certains modèles de calcul tels que les automates, les réseaux de Petri, les systèmes à temps continu, étaient plus naturellement considérés comme des langages de modélisation, alors que d'autres, tels que les réseaux de processus de Kahn ou les systèmes à événements discrets étaient plutôt considérés comme des modèles de calcul. La différence fondamentale entre ces types de modèles est que dans les seconds, les composants d'un modèle sont des modèles des composants du système. La hiérarchie du modèle décrit la structure du système. On considère alors naturellement que le comportement du système est issu de la combinaison du comportement de ses composants selon les règles du modèle de calcul. Dans le cas du premier type de modèles, les composants du modèle ne sont pas des modèles des composants du système. En effet, un état ne modélise pas une partie d'un système, il modélise une partie de son comportement. La hiérarchie de ces modèles est donc une hiérarchie comportementale et non structurelle, ce qui tend à faire considérer ses éléments comme ceux d'un langage puisqu'ils n'ont pas de contrepartie identifiable dans le système. Un critère permettant de distinguer les deux types de modèles est la pertinence ou non de composants élémentaires. Dans un modèle régi par un modèle de calcul, avec une hiérarchie structurelle,

les éléments atomiques de la structure peuvent être des comportements opaques définis en dehors du modèle. Par exemple, dans un réseau de processus de Kahn, un composant peut très bien avoir un comportement écrit dans n'importe quel langage de programmation. Il est inutile de connaître les détails de ce comportement pour le combiner à celui des autres composants. Au contraire, lorsque la hiérarchie est comportementale, la sémantique précise de chaque composant doit être connue. Dans un modèle d'automate, il faut savoir ce qu'est un état, une transition, comment interpréter une garde. De même dans un réseau de Petri, l'interprétation du modèle repose sur la connaissance exacte de ce qu'est une place, une transition, et des liens entre les deux. Ajouter un composant ayant un comportement opaque dans ces modèles n'a pas de sens. On considère donc ces paradigmes de modélisation comme des langages car chacun des éléments syntaxiques doit avoir une sémantique connue de l'interpréteur pour qu'il puisse donner un sens au modèle.

8.4 Universalité des modèles de calcul

Une autre question ouverte est celle de la capacité de ModHel'X à décrire tout modèle de calcul. Peut-on imaginer un modèle de calcul qu'il ne serait pas possible de décrire selon la sémantique abstraite de ModHel'X ? Pour répondre à cette question, il faut revenir à la définition de ce que nous considérons comme un modèle de calcul, ou plutôt d'exécution puisque c'est ce dont il s'agit dans ModHel'X. Si l'on appelle modèle d'exécution un algorithme permettant de combiner les comportements des composants d'un modèle pour calculer le comportement global du modèle, alors ModHel'X peut certainement décrire tout modèle d'exécution, de même qu'une machine de Turing peut décrire n'importe quel algorithme. Toutefois, la notion de modèle de calcul s'appuie sur l'observabilité du comportement de composants opaques. Or cette approche a des limites et ne permet pas de calculer le comportement de n'importe quel système. Considérons par exemple le circuit de la figure 5 qui relie un générateur de force électromotrice e et de résistance interne r à une résistance de valeur R.

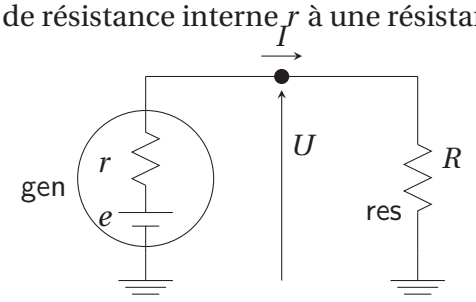


FIGURE 5 – Circuit électrique

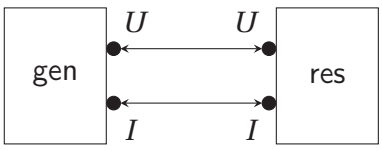


FIGURE 6 – Structure du modèle

La figure 6 présente un modèle « boîte noire » de ce circuit. Il comporte deux composants gen et res qui représentent respectivement le générateur et la résistance. Le comportement des composants étant opaque, on ne peut pas trouver la valeur de U et I en résolvant le système :

$$\begin{cases} U = e - rI \\ U = RI \end{cases}$$

En effet, ces deux équations ne sont pas visibles depuis l'extérieur des composants, on ne peut qu'observer les valeurs que les composants communiquent à leur interface. Chaque composant calcule ainsi la valeur de son port U en fonction de la valeur disponible sur son port I et vice-versa. Lorsqu'aucune valeur n'est associée à ces ports, le générateur gen donne une tension égale à e et un courant nul, tandis que la résistance res donne une tension et un courant nuls. En partant de cette information initiale, et en ne faisant qu'observer et propager l'information produite par les composants, il n'est pas toujours possible de calculer la tension à leurs bornes et le courant qui les traverse. De plus, lorsque le calcul est possible, la valeur exacte n'est obtenue qu'après une infinité d'observations élémentaires. Les détails sont donnés dans le rapport [2009-CM1]. La raison de cet échec est que le comportement de chaque composant influe sur celui de l'autre, et que le comportement global du modèle est le point fixe de la composition des comportements en boucle ouverte de ses composants. Ce point fixe existe (on peut le calculer en résolvant le système d'équations), mais il n'est pas nécessairement atteignable par itérations successives. La résolution efficace du système d'équations représenté par un tel modèle ne peut se faire qu'en renonçant à l'opacité des comportements des composants.

ModHel'X permet donc de représenter tout modèle de calcul, mais certains modèles ne se laissent pas décomposer en composants opaques et ne peuvent donc pas être traités selon cette approche.

8.5 Objectifs pour l'avenir

L'approche de la modélisation et de la validation des modèles hétérogènes que nous défendons consiste à définir un cadre général dans lequel les notions de composant, de modèle de calcul et d'exécution sont bien définies. Toutefois, ce cadre ne peut en aucun cas avoir l'efficacité des approches ad hoc qui, en contrepartie de limitations, offrent des possibilités de validation, de génération automatique de code, ou plus simplement une interface de modélisation adaptée à leurs utilisateurs. Notre objectif est donc de travailler sur des applications concrètes qui utilisent des approches ad hoc afin de mettre en œuvre notre approche et de la nourrir des préoccupations des concepteurs de systèmes. Mais nous avons aussi pour objectif de travailler à un niveau plus abstrait à l'élaboration du cadre général à partir des problématiques rencontrées dans les applications. En retour, le cadre général nous permet de tester de nouvelles idées hors des contraintes d'une application réelle. Une fois maîtrisées, ces nouvelles idées permettent de reconnaître dans des problèmes concrets apparemment différents, des variantes d'un même problème plus abstrait et de les résoudre plus efficacement.

Les travaux en cours sur le test de systèmes hétérogènes, sur la modélisation muti-vues et sur la modélisation de l'adaptation sémantique sont issus de problèmes concrets, et nous rechercherons des occasions d'appliquer leurs résultats à des cas concrets. D'autres problèmes liés à l'hétérogénéité des modèles nous semblent intéressants, et leur étude est envisagée à plus long terme. C'est pourquoi je les cite ci-dessous en guise de perspectives.

8.5.1 Model-checking hétérogène

Les travaux de formalisation coalgébrique des modèles à base d'états et de transitions ont pour l'instant été exploités uniquement pour construire une théorie du test des systèmes hétérogènes. Ce type de systèmes peut aussi être analysé par des techniques de model-checking. Toutefois, les algorithmes de model-checking sont conçus pour analyser des modèles obéissant à une sémantique donnée. Contrairement au test, qui ne fait que stimuler le modèle et observer sa réponse, le model-checking accède à la structure interne du modèle et explore systématiquement ses états. Afin de limiter à la fois l'explosion combinatoire du nombre d'états à explorer et les difficultés liées aux différentes sémantiques données aux états et aux transitions, il serait intéressant d'utiliser la sémantique des connecteurs ou du modèle de calcul utilisé pour assembler des composants afin de « connecter » les algorithmes de model-checking utilisés pour chacun d'eux. Ainsi, un model-checker pourrait effectuer une vérification sur un composant avec la connaissance a priori des propriétés des traces possibles à ses entrées, ces propriétés ayant été vérifiées par model-checking sur les autres composants du système.

Une telle modularisation de la vérification de propriétés restreindra nécessairement le domaine des propriétés vérifiables, mais elle permettra éventuellement de les vérifier sur des modèles beaucoup plus volumineux.

8.5.2 Composition de propriétés

De même qu'un modèle de calcul est considéré dans ModHel'X comme un ensemble d'opérations permettant de combiner les comportements des composants d'un modèle (il s'agit donc à proprement parler d'un modèle d'exécution), on peut considérer un modèle de calcul comme un ensemble de règles qui permettent de combiner les propriétés des composants afin d'en déduire des propriétés d'un modèle. Chaque modèle de calcul est donc une logique, et chaque modèle obéissant à ce modèle de calcul est un jeu d'axiomes particulier. Les propriétés globales du modèle se démontrent à partir des axiomes en respectant les règles de la logique du modèle de calcul.

Pour être totalement satisfaisante, cette approche devrait aussi traiter le problème du raffinement des modèles de calcul. Comment montrer qu'un modèle de calcul préserve les propriétés démontrées dans un autre modèle de calcul ? Est-il possible de définir par exemple un modèle de calcul pour les automates déterministes et de démontrer qu'il est un raffinement d'un modèle de calcul pour les automates non déterministes ? Un des intérêts d'une telle notion de raffinement serait de pouvoir montrer qu'un modèle d'exécution est conforme à un modèle de calcul. Il deviendrait ainsi possible d'appliquer le principe WYPIWYE (What You Prove Is What You Execute) [1989-4] de manière plus générale et d'éviter la déconnexion fréquente entre modèles utilisés pour la validation et modèles utilisés pour l'implémentation.

8.5.3 Prérequis minimaux

L'approche inverse de la précédente serait particulièrement intéressante pour déterminer, à partir des propriétés que doit satisfaire un modèle et des règles de déduction associées à son modèle de calcul, les propriétés minimales que doivent satisfaire ses composants.

Cette approche est comparable à la détermination des pré-requis minimaux d'un programme avec la logique de Hoare [1969-20]. Toutefois, la composition du comportement de composants est généralement plus complexe que l'exécution séquentielle d'instructions. De plus, la syntaxe des formules, les axiomes et les règles de déduction changent avec le modèle de calcul. Il est donc nécessaire d'établir des correspondances entre les différentes logiques utilisées, de façon similaire à l'adaptation sémantique entre des modèles hétérogènes.

8.6 Conclusion sur la modélisation multi-paradigme

La modélisation multi-paradigme constitue un nouveau domaine de recherche qui touche par essence de nombreuses disciplines. C'est un de ses principaux attraits car cela favorise la rencontre de chercheurs d'horizons variés ayant chacun une manière spécifique d'aborder les problèmes de leur domaine. Cela peut aussi être frustrant car il est difficile de mettre en valeur les résultats d'une recherche qui porte principalement sur la description de méthodes de modélisation. La plupart des exemples que nous présentons illustrent des applications de notre travail. Ceci nous amène à cultiver la transdisciplinarité afin d'amener des chercheurs d'autres disciplines à accepter que nous prenions un de leur problème multi-paradigme comme cas d'étude. En cela, la diversité des disciplines présentes à Supélec au sein de l'équipe E3S « Science des Systèmes » est un atout important.

9 Références

[2006-1] Jean-Marie Favre. Megamodelling and Etymology. In James R. Cordy, Ralf Lämmel, and Andreas Winter, editors, *Transformation Techniques in Software Engineering*, number 05161 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006. Internationales Begegnungs und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, ISSN : 1862-4405. http://drops.dagstuhl.de/opus/volltexte/2006/427.

[2005-2] Object Management Group. Meta Object Facility (MOF), 2005. http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF.

[2008-3] Cécile Hardebolle. *Formalisation des modèles d'exécution et de leurs interactions*. PhD thesis, Université Paris-Sud XI, nov 2008. http://wwwdi.supelec.fr/hardebolle/MemoireHardebolle.pdf.

[1989-4] Gérard Berry. Real Time Programming : Special Purpose or General Purpose Languages. In G. X. Ritter, editor, *Information Processing 89*, pages 11–18, North-Holland, 1989. Elsevier Science Publishers B.V.

[1993-5] Frédéric Boulanger. *Intégration de modules synchrones dans la programmation par objets*. PhD thesis, Université Paris-Sud XI, dec 1993. http://wwwdi.supelec.fr/fb/download/Articles/Bou1993Thesis.pdf.

[1991-6] Albert Benveniste and Gerard Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9) :1270–1282, sep 1991, sep 1991.

[1991-7] Nicolas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9) :1305–1320, sep 1991, sep 1991.

[2003-8] J. Eker, J.W. Janneck, E.A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong. Taming heterogeneity – the Ptolemy approach. *Proceedings of the IEEE*, 91(1) :127–144, jan 2003, jan 2003.

[1997-9] Stephen A. Edwards. *The Specification and Execution of Heterogeneous Synchronous Reactive Systems*. PhD thesis, University of California, Berkeley, 1997. http://www1.cs.columbia.edu/~sedwards/papers/edwards1997specification.pdf.

[1995-10] "Production Cell" case study, 1995. http://archive.comlab.ox.ac.uk/procos/prod-cell.html.

[1995-11] Claus Lewerentz and Thomas Lindner, editors. *Formal Development of Reactive Systems : Case Study Production Cell*. LNCS 891. Springer-Verlag, February 1995, ISBN : 978-3540588672. http://www.springer.com/computer/swe/book/978-3-540-58867-2.

[2003-12] Trung Hieu Phan, Sébastien Gérard, and François Terrier. Real-time system modeling with ACCORD/UML methodology : Illustration through an automotive case study. In *FDL*, pages 342–354. ECSI, 2003.

[2007-13] C. André, F. Mallet, and R. de Simone. Modeling Time(s). In *Proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2007)*, volume 4735 of *Lecture Notes in Computer Science*, pages 559–573. Springer, 2007.

[2007-14] Charles André, Frédéric Mallet, and Marie-Agnès Peraldi Frati. Multiform Time in UML for Real-time Embedded Applications. In *IEEE Int. Conf. on Real-Time Computing Systems and Applications (RTCSA)*, pages 232–237, Daegu Corée, République De, 2007. IEEE. http://hal.inria.fr/inria-00204503/PDF/rtcsa07.pdf. The original publication is available at ieee.org (http://dx.doi.org/10.1109/RTCSA.2007.51).

[2003-15] L. Soares Barbosa. Towards a Calculus of State-based Software Components. *"Journal of Universal Computer Science"*, 9(8) :891–909, 2003, 2003. http://www.jucs.org/jucs_9_8/towards_a_calculus_of.

[2006-16] Christophe Gaston, Pascale Le Gall, Nicolas Rapin, and Assia Touil. Symbolic Execution Techniques for Test Purpose Definition. In M. Uyar, Ali Duale, and Mariusz Fecko, editors, *Testing of Communicating Systems*, volume 3964 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin / Heidelberg, 2006, ISBN :. http://dx.doi.org/10.1007/11754008_1.

[1996-17] Jan Tretmans. Test generation with inputs, outputs, and quiescence. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 127–146. Springer Berlin / Heidelberg, 1996, ISBN :. http://dx.doi.org/10.1007/3-540-61042-1_42.

[2008-18] S. Bliudze and J. Sifakis. The Algebra of Connectors – Structuring Interaction in BIP. *IEEE Transactions on Computers*, 57(10) :1315–1330, February 2008, ISSN : 0018-9340. IEEE Computer Society, February 2008. http://www-verimag.imag.fr/~sifakis/IEEEtransactionscomputers-final.pdf.

[2007-19] Alberto Sangiovanni-Vincentelli. Quo Vadis SLD : Reasoning about Trends and Challenges of System-Level Design. *Proceedings of the IEEE*, 95(3) :467–506, March 2007, March 2007. http://chess.eecs.berkeley.edu/pubs/263.html.

[1969-20] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10) :576–580,583, October 1969, October 1969. http://dx.doi.org/10.1145%2F363235.363259.

[2007-21] J. Bezivin, M. Barbero, and F. Jouault. On the Applicability Scope of Model Driven Engineering. In *Proceedings of the 4th International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007), at the European Joint Conferences on Theory and Practice of Software (ETAPS 2007)*, pages 3–7. IEEE Computer Society, March 2007.

[2006-22] J.-M. Favre, J. Estublier, and M. Blay-Fornarino, editors. *L'Ingénierie Dirigée par les Modèles : au-delà du MDA*. Hermès – Lavoisier, 2006. Traité IC2 – Information – Commande – Communication.

[2007-23] T. A. Henzinger and J. Sifakis. The Discipline of Embedded Systems Design. *Computer*, pages 32–40, October 2007, October 2007.

[2004-24] P. J. Mosterman and H. Vangheluwe. Computer Automated Multi-Paradigm Modeling : An Introduction. *Simulation : Transactions of the Society for Modeling and Simulation International*, 80(9) :433–450, 2004, 2004. Special Issue : Grand Challenges for Modeling and Simulation.

[2005-25] Lars Skyttner. *General Systems Theory, second edition*. World Scientific, 2005, ISBN : 978-981-256-467-2.

[2003-26] C. Atkinson and T. Kühne. Model-driven development : a metamodeling foundation. *IEEE Software*, 20(5) :36–41, 2003, ISSN : 0740-7459, 2003.

[web-27] OMG. Unified Modeling Language (UML), version 2.1.2, web. `http://www.omg.org/technology/documents/formal/uml.htm`.

[2004-28] J. Bézivin, F. Jouault, and P. Valduriez. On the Need for Megamodels. In *Proceedings of the OOPSLA/GPCE : Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2004. `http://www.softmetaware.com/oopsla2004/bezivin-megamodel.pdf`.

[2000-29] A. van Deursen, P. Klint, and J. Visser. Domain-Specific Languages : an Annotated Bibliography. *SIGPLAN Not.*, 35(6) :26–36, 2000, ISSN : 0362-1340, New York, NY, USA, 2000. ACM.

[1997-30] F. Balarin, E. Sentovich, M. Chiodo, P. Giusto, H. Hsieh, B. Tabbara, A. Jurecska, L. Lavagno, C. Passerone, K. Suzuki, , and A. Sangiovanni-Vincentelli. *Hardware-Software Co-design of Embedded Systems – The POLIS approach*. Kluwer Academic Publishers, 1997.

[1996-31] J. R. Abrial. *The B-Book – Assigning Programs to Meanings*. Cambridge University Press, 1996.

[2001-32] J. R. Burch, R. Passerone, and A. L. Sangiovanni-Vincentelli. Overcoming Heterophobia : Modeling Concurrency in Heterogeneous Systems. In *Proceedings of the second International Conference on Application of Concurrency to System Design*, page 13, June 2001.

[1991-33] A. Benveniste, P. Le Guernic, and C. Jacquemot. Synchronous programming with events and relations : the SIGNAL language and its semantics. *Sci. Comput. Program.*, 16(2) :103–149, 1991, ISSN : 0167-6423, Amsterdam, The Netherlands, The Netherlands, 1991. Elsevier North-Holland, Inc.

[1985-34] G. Berry and L. Cosserat. The ESTEREL Synchronous Programming Language and its Mathematical Semantics. In *Seminar on Concurrency,*

Carnegie-Mellon University, pages 389–448, London, UK, 1985. Springer-Verlag, ISBN : 3-540-15670-4.

[1985-35] D. Harel and A. Pnueli. *Logics and models of concurrent systems*, volume 13 of *Nato Asi Series F : Computer And Systems Sciences*, chapter On the development of reactive systems, pages 477–498. Springer-Verlag New York, Inc., New York, NY, USA, 1985, ISBN : 0-387-15181-8.

[1988-36] Georges Gonthier. *Sémantiques et modèles d'exécution des langages réactifs synchrones ; application à Esterel*. PhD thesis, Université Paris-Sud XI, 1988.

[1991-37] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy : A Mixed-Paradigm Simulation/Prototyping Platform in C++. In *Proceedings of the C++ At Work Conference*, Santa Clara, CA, November 1991.

[2000-38] Gérard Berry. *The foundations of Esterel*, pages 425–454. MIT Press, Cambridge, MA, USA, 2000, ISBN : 0-262-16188-5. `http://portal.acm.org/citation.cfm?id=345868.345908`.

[2007-39] R. Mateescu, P. Poizat, and G. Salaün. Behavioral adaptation of component compositions based on process algebra encodings. In *ASE '07 : Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 385–388, New York, NY, USA, 2007. ACM, ISBN : 978-1-59593-882-4.

[2005-40] Edward A. Lee and Haiyang Zheng. Operational Semantics of Hybrid Systems. In *Hybrid Systems : Computation and Control ; 8th International Workshop, HSCC*, volume 3414/2005 of *LNCS*, pages 25–53. Springer-Verlag, March 2005. `http://dx.doi.org/10.1007/978-3-540-31954-2_2`.

[2008-41] C. Attiogbé. Mastering Specification Heterogeneity with Multifacet Analysis. In *Proceedings of the MoVaH'08 Workshop on Modeling, Validation and Heterogeneity, held in conjunction with the first IEEE International Conference on Software Testing, verification and validation ICST 2008*, April 2008.

[2005-42] G. Gössler and J. Sifakis. Composition for component-based modeling. *Science of Computer Programming*, 55(1-3) :161–183, March 2005, ISSN : 0167-6423, Amsterdam, The Netherlands, March 2005. Elsevier North-Holland, Inc.

[2006-43] A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-time Systems in BIP. In *4th IEEE International Conference on Software Engineering and Formal Methods (SEFM06)*, pages 3–12, September 2006.

10 Publications

10.1 Articles dans des revues internationales avec comité de lecture

[1997-IR1] Charles André, Frédéric Boulanger, Marie-Agnès Péraldi, Jean-Pierre Rigault, and Guy Vidal-Naquet. Objects and Synchronous Programming. *Journal Européen des Systèmes Automatisés (JESA)*, 31(3/1997) :417–432, 1997, ISSN : 1269-6935. Hermès/Lavoisier, Cachan, France, 1997.

[2008-IR2] Frédéric Boulanger. Integration of Dependability Features in a Synchronous Application. *International Review on Computers and Software (IRECOS)*, 3(1) :31–37, January 2008, ISSN : 1828-6003, Napoli, Italy, January 2008. Praise Worthy Prize.

[2009-IR3] Mohamed Feredj, Frédéric Boulanger, and Aimé Mokho Mbobi. A model of domain-polymorph component for heterogeneous system design. *Journal of Systems and Software*, 82(1) :112–120, January 2009, ISSN : 0164-1212, Amsterdam, The Netherlands, January 2009. Elsevier. `http://www.sciencedirect.com/science/article/B6V0N-4SNWVYR-1/2/b9ac66013d2054d7afe1172b3c3ea8c6`. Special Issue : Software Performance - Modeling and Analysis. Impact factor : 1.340 (2009), Rank : A (CORE 2010).

[2007-IR4] Aimé Mbobi, Frédéric Boulanger, and Mohamed Feredj. An Approach of Flat Heterogeneous Modeling based on Heterogeneous Interface Components. *International Review on Computers and Software (IRECOS)*, 2(2) :179–189, March 2007, ISSN : 1828-6003, Napoli, Italy, March 2007. Praise Worthy Prize. `http://www.praiseworthyprize.com/IRECOS_vol_2_n_2.html`.

[2009-IR5] Cécile Hardebolle and Frédéric Boulanger. Multi-Formalism Modelling and Model Execution. *International Journal of Computers and their Applications*, 31(3) :193–203, July 2009, ISSN : 1076-212X. ACTA Press, July 2009. `http://www.actapress.com/Content_of_Journal.aspx?JournalID=111`. Special Issue on the International Summer School on Software Engineering. Rank : C (CORE 2009).

[2010-IR6] Tudor Ionescu, Géraldine Polaillon, and Frédéric Boulanger. Minimum Tree Cost Quartet Puzzling. *Journal of classification*, 27(2) :136–157, September 2010, ISSN : 0176-4268. Springer New York Publications, September 2010. `http://www.springerlink.com/content/t73284522735217u/`. Impact factor : 0.964 (2009).

[2009-IR7] Cécile Hardebolle and Frédéric Boulanger. Exploring Multi-Paradigm Modeling Techniques. *SIMULATION : Transactions of The Society for Modeling and Simulation International*, 85(11/12) :688–708, November 2009, ISSN : 0037-5497. SAGE Publications, November 2009. `http://sim.sagepub.com/cgi/content/abstract/85/11-12/688`. Impact factor : 0.404 (2007), Rank : B (CORE 2009).

10.2 Articles dans des actes de conférence édités

[2010-ED1] Frédéric Boulanger, Christophe Jacquet, Cécile Hardebolle, and Elyes Rouis. Modeling Heterogeneous Points of View with ModHel'X. In Sudipto Ghosh, editor, *MODELS 2009 Workshops*, volume 6002 of *Lecture Notes in Computer Science*, pages 310–324, Berlin Heidelberg, Germany, 2010. Springer-Verlag, ISBN : 978-3-642-12260-6. http://dx.doi.org/10.1007/978-3-642-12261-3.

[1995-ED2] Guy Vidal-Naquet and Frédéric Boulanger. Integration of Synchronous Modules in an Object-Oriented Language. In R.J. Wieringa and R.B. Feenstra, editors, *Information Systems – Correctness and Reusability, selected papers from the IS-CORE Workshop*, pages 279–291. World Scientific, August 1995, ISBN : 981-02-2240-8.

[2008-ED3] Cécile Hardebolle and Frédéric Boulanger. ModHel'X : A Component-Oriented Approach to Multi-Formalism Modeling. In *Models in Software Engineering : Workshops and Symposia at MoDELS 2007, Nashville, TN, USA, September 30 - October 5, 2007, Reports and Revised Selected Papers*, pages 247–258, Nashville, TN, USA, 2008. Springer-Verlag, ISBN : 978-3-540-69069-6. http://wwwdi.supelec.fr/fb/publis/HarBou2007MPM.pdf.

10.3 Articles dans des conférences internationales avec comité de lecture

[1998-IC1] Frédéric Boulanger and Guy Vidal-Naquet. Integration of Dependability Modules in a Reactive Application. In *Proceedings of the 3rd World Conference on Integrated Design and Process Technology*, pages 313–319, Berlin, Germany, July 1998.

[2004-IC2] Mohamed Feredj, Frédéric Boulanger, and Mokhoo Aimé Mbobi. An Approach of Domain Polymorph Component Design. In *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration (IEEE IRI 2004)*, pages 145–150, Las Vegas, Nevada, USA, November 2004. ISBN : 0-7803-8819-4. http://wwwdi.supelec.fr/fb/publis/FerBouMboIRI2004.pdf.

[2004-IC3] Mokhoo Aimé Mbobi, Frédéric Boulanger, and Mohamed Feredj. Execution Model for Non-Hierarchical Heterogeneous Modeling. In *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration (IEEE IRI 2004)*, pages 139–144, Las Vegas, Nevada, USA, November 2004. IEEE Computer Society, ISBN : 0-7803-8819-4. http://wwwdi.supelec.fr/fb/publis/MboBouFerIRI2004.pdf.

[2005-IC4] Mokhoo Aimé Mbobi, Frédéric Boulanger, and Mohamed Feredj. Issues of Hierarchical Heterogeneous Modeling in Component Reusability. In *Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration (IEEE IRI 2005)*, pages 84–89, Las Vegas, Nevada, USA, August 2005. IEEE Computer Society,

ISBN : 0-7803-9093-8. http://wwwdi.supelec.fr/fb/publis/MboBouFerIRI2005.pdf.

[2008-IC5] Frédéric Boulanger and Cécile Hardbolle. Simulation of Multi-Formalism Models with ModHel'X. In *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST2008)*, pages 318–327, Lillehammer, Norway, April 2008. IEEE, IEEE Computer Society, ISBN : 978-0-7695-3127-4. Rank : B (CORE 2009), Acceptance rate : 20%.

[2008-IC6] Christophe Jacquet, Frédéric Boulanger, and Dominique Marcadet. From Data to Events : Checking Properties on the Control of a System. In *Sixth ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE'2008) ACM-IEEE MEMOCODE'2008*, pages 17–26, ANAHEIM États-Unis, 2008. http://wwwdi.supelec.fr/fb/download/Articles/JacBouMar2008MEMOCODE.pdf.

[2008-IC7] Ahcene Bouzoualegh, Dominique Marcadet, Frédéric Boulanger, and Christophe Jacquet. An Architecture Description Language for Verification in Component-Based Software. In *Proceedings of the IEEE Computer Software and Applications Conference (COMPSAC 2008)*, pages 365–368, Los Alamitos, CA, USA, 2008. IEEE Computer Society, ISBN : 978-0-7695-3262-2.

[2010-IC8] Abderraouf Benyahia, Arnaud Cuccuru, Safouan Taha, François Terrier, Frédéric Boulanger, and Sébastien Gérard. Extending the Standard Execution Model of UML for Real-Time Systems. In Mike Hinchey, Bernd Kleinjohann, Lisa Kleinjohann, Peter Lindsay, Franz Rammig, Jon Timmis, and Marilyn Wolf, editors, *Distributed, Parallel and Biologically Inspired Systems*, volume 329 of *IFIP Advances in Information and Communication Technology*, pages 43–54. Springer Boston, 2010, ISBN : 978-3-642-15233-7. http://dx.doi.org/10.1007/978-3-642-15234-4.

[2010-IC9] Bilal Kanso, Marc Aiguier, Frédéric Boulanger, and Assia Touil. Testing of Abstract Components. In Ana Cavalcanti, David Deharbe, Marie-Claude Gaudel, and Jim Woodcock, editors, *Proceedings of the 7th International Colloquium on Theoretical Aspects of Computing (ICTAC 2010)*, Lecture Notes in Computer Science 6255, pages 184–198, Berlin Heidelberg, Germany, 2010. Springer-Verlag, ISSN : 0302-9743, ISBN : 978-3-642-14807-1. http://wwwdi.supelec.fr/fb/download/Articles/AigKanBouTou2010ICTAC.pdf.

[2009-IC10] Frédéric Boulanger, Christophe Jacquet, Elyes Rouis, and Cécile Hardebolle. Modeling Heterogeneous Points of View with ModHel'X. In *Proceedings of NFPinDSML2009 (2nd Workshop on Non-functional System Properties in Domain Specific Modeling Languages at MODELS 2009)*, volume 553, pages 1–14. CEUR-WS, 2009, ISSN : 1613-0073. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-553/.

[2007-IC11] Frédéric Boulanger, Géraldine Polaillon, Dorin Carstoiu, Alexandra Cernian, and Stefan Bodea. Web Search based on Clustering by Compression. In *Proceedings of the IADIS 2007 e-Society International Conference*, pages 419–423, Lisbon, Portugal, July 2007. IADIS, IADIS Press, ISBN : 978-972-8924-35-5.

[2007-IC12] Cécile Hardebolle, Frédéric Boulanger, Dominique Marcadet, and Guy Vidal-Naquet. A Generic Execution Framework for Models of Computation. In J. M. Fernandes, R. J. Machado, R. Khedri, and S. Clarke, editors, *Proceedings of the 4th International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007)*, pages 45–54, Braga, Portugal, March 2007. IEEE Computer Society, ISBN : 0-7695-2769-8. `http://wwwdi.supelec.fr/fb/publis/HarBouMarVid2007MOMPES.pdf`.

[2006-IC13] Frédéric Boulanger and Mokhoo Mbobi. An Overall Specification of a Meta-Model of Computation For Model-Driven Embedded Systems. In *Proceedings of the 2006 International Symposium on Collaborative Technologies and Systems (CTS 2006)*, pages 194–199, Las Vegas, Nevada, USA, May 2006. IEEE Computer Society, ISBN : 0-9785699-0-3. `http://wwwdi.supelec.fr/fb/publis/HarBouMarVid2007MOMPES.pdf`. Rank : C (CORE 2009).

[2006-IC14] Frédéric Boulanger and Mokhoo Aimé Mbobi. Le paradigme acteur dans la modélisation des systèmes embarqués. In *Proceedings of the 2006 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2006)*, pages 418–421, Ottawa, ON, Canada, May 2006. IEEE Computer Society, ISBN : 1-4244-0038-4. `http://wwwdi.supelec.fr/fb/publis/BouMboCCECE2006.pdf`.

[2006-IC15] Frédéric Boulanger and Guy Vidal-Naquet. A Primitive Execution Model for Heterogeneous Modeling. In *Proceedings of the 2006 International Conference on Software and Data Technologies (ICSOFT 2006)*, volume 1, pages 247–252, Setúbal, Portugal, September 2006. INSTICC Press, ISBN : 972-8865-69-4. `http://wwwdi.supelec.fr/fb/publis/BouVidICSOFT2006.pdf`. Rank : B (CORE 2009).

[2004-IC16] Frédéric Boulanger, Mokhoo Aimé Mbobi, and Mohamed Feredj. Flat Heterogeneous Modeling. In *Proceedings of the 2004 International Conference on Advances in the Internet, Processing, Systems, and Interdisciplinary Research (IPSI 2004)*, Venice, Italy, November 2004. ISBN : 86-7466-117-3. `http://wwwdi.supelec.fr/fb/publis/BouMboFerIPSI2004.pdf`. 7 pages, on CD-ROM.

[2003-IC17] Mokhoo Aimé Mbobi, Frédéric Boulanger, and Mohamed Feredj. Non-hierarchical heterogeneity. In *Proceedings of the 2003 International conference on Computer, Communication and Control Technologies (CCCT'03)*, volume III, pages 430–435, Orlando, Florida, USA, August 2003. International Institute of Information and Systemics, ISBN : 980-6560-05-1. `http://wwwdi.supelec.fr/fb/publis/MboBouFerCCCT2003.pdf`.

[2001-IC18] Charles André, Frédéric Boulanger, and Alain Girault. Software Implementation of Synchronous Programs. In *Proceedings of the 2nd International Conference on Application of Concurrency to System Design (ICACSD2001)*, pages 133–142, New Castle upon Tyne, UK, June 2001. IEEE Computer Society, ISBN : 0-7695-1071-X. `http://wwwdi.supelec.fr/fb/publis/AndBouGirICACSD2001.pdf`.

[1998-IC19] Frédéric Boulanger and Guy Vidal-Naquet. Modular Development of Control and Computational Modules Using Reactive Objects. In *Object Oriented Techology : ECOOP'98 Workshop Reader*, volume 1543/1998 of *Lecture Notes in Computer Science*, pages 515–518, London, UK, July 1998. Springer-Verlag, ISSN : 0302-9743, ISBN : 978-3-540-65460-5. `http://wwwsi.supelec.fr/fb/download/Articles/BouVid1998ECOOP.pdf`. Rank : A (CORE 2007).

[1996-IC20] Frédéric Boulanger and Guy Vidal-Naquet. An Object Execution Model for Reactive Modules with a C++ Implementation. In Max Mühlhäuser, editor, *Special Issues in Object-Oriented Programming, ECOOP'96 Workshop Reader*, pages 443–449, Linz, July 1996. dpunkt.verlag, ISBN : 3-920993-67-5. Rank : A (CORE 2007).

[1996-IC21] Frédéric Boulanger and Guy Vidal-Naquet. Synchronous Reactive Programming in Ptolemy. In *Proceedings of the 2nd World Conference on Integrated Design and Process Technology*, Austin, Texas, USA, December 1996. 6 pages.

10.4 Articles dans des conférences nationales avec comité de lecture

[1999-NC1] Frédéric Boulanger and Guy Vidal-Naquet. Objets réactifs pour le développement modulaire du contrôle. In *Actes du 2e congrès Modélisation des systèmes réactifs (MSR'99)*, pages 333–340, Cachan, France, March 1999. Hermes Sciences Publications, ISBN : 2-7462-0017-1.

[1996-NC2] Charles André, Frédéric Boulanger, Marie-Agnès Péraldi, Jean-Pierre Rigault, and Guy Vidal-Naquet. Objets et programmation synchrone. In *Actes du congrès AFCET Modélisation des systèmes réactifs*, pages 55–62. AFCET, March 1996.

[1994-NC3] Guy Vidal-Naquet, Frédéric Boulanger, and Henri Delebecque. Intégration de modules synchrones dans un langage à objets. In *Actes de la conférence Real Time Systems (RTS'94)*, pages 245–260, Paris, France, January 1994. Teknea, ISBN : 978-2-87717-039-0.

10.5 Communications

[2009-CM1] Frédéric Boulanger. Formalisation de ModHel'X. Rapport interne 2009-09-25-DI-FB, Supélec, September 2009.

http://wwwdi.supelec.fr/internalreports/Rapport_2009-09-25-DI-FB_FormHelX.pdf.

[2005-CM2] Mokhoo Aimé Mbobi, Frédéric Boulanger, and Mohamed Feredj. Integration of a Flat Heterogeneous Domain in Ptolemy II. In *Sixth Biennal Ptolemy Conference*, May 2005.

[2003-CM3] Frédéric Boulanger. Printing Digital Photographs with LaTeX. *Communications of the TeX Users Group (TUGboat)*, 24(3) :453–461, 2003, ISSN : 0896-3207, Portland, OR, USA, 2003. TeX Users Group.

[2001-CM4] Frédéric Boulanger. LaTeX au pays des tableurs. *Cahiers GUTenberg*, 39-40 :7–16, May 2001, ISSN : 1140-9304, May 2001.

[2000-CM5] Frédéric Boulanger and Yolaine Bourda. Documentation de projets en XML. *Cahiers GUTenberg*, 35-36 :15–24, May 2000, ISSN : 1140-9304, May 2000.

[1994-CM6] Guy Vidal-Naquet and Frédéric Boulanger. An Object Oriented Execution Model for Synchronous Modules. In *Synchronous Languages Seminar*, December 1994.

[1994-CM7] Guy Vidal-Naquet, Frédéric Boulanger, and Henri Delebecque. Outil d'intégration automatique de modules de sûreté de fonctionnement pour des applications temps-réel. In *Journée thématique DRET : temps-réel et sûreté de fonctionnement dans les applications de la défense*, 1994.

Composé par pdfTEX 1.4011 le 26 mai 2011 à 11:29