Exploring Multi-Paradigm Modeling Techniques

Cécile Hardebolle *Frédéric Boulanger*

cecile.hardebolle@supelec.fr frederic.boulanger@supelec.fr

SUPELEC – Computer Science Department
3 rue Joliot-Curie
91192 Gif-Sur-Yvette cedex, France

Abstract

Multi-Paradigm Modeling (MPM) addresses the necessity of using multiple modeling paradigms when designing complex systems. Because of its multi-disciplinary nature, the MPM field involves research teams with technical backgrounds as different as control science, model checking, modeling language engineering or system-on-chip development. In this paper, we propose to explore the MPM domain through a survey of existing techniques from different horizons. Since the heterogeneity of models is at the heart of Multi-Paradigm Modeling, we first identify the sources of this heterogeneity and introduce the problems it raises. Then we show how the different existing techniques address these problems.

Keywords: Multi-paradigm modeling, heterogeneous models, transformation of models, meta-modeling, composition of models, models of computation, heterogeneous interactions, co-simulation, component composition, mega-models, model execution, simulation

1 Introduction

In the context of Model Driven Engineering (MDE), models have taken an increasingly important role at the heart of the development processes in system engineering. This evolution has come along with the significant multiplication of modeling languages dedicated to particular domains or applications, called "domain specific modeling languages" (DSMLs or DSLs) [1]. Capturing specific knowledge and adapted know-how, these languages are considered as a key feature to gain efficiency and productivity as well as to improve the quality of systems [2]. MDE is now being applied to the design of *complex systems*, one characteristic of which is their inherent heterogeneity. Because of this characteristic, their design requires inevitably the joint use of several DSLs. Therefore, in this context, the heterogeneity of systems entails the *heterogeneity of models*.

The heterogeneity of models raises many issues and makes the development process particularly difficult. The necessity to address these issues is now widely admitted [3, 4, 5] and research on this subject recently gave birth to a domain called *"Multi-Paradigm Modeling" (MPM)* [6]. However, it is important to note that the problem of the heterogeneity of models is not new. Techniques have been developed to solve heterogeneity problems in several fields even before MDE existed. But the problem of heterogeneity has spread with the development of MDE and requires now a more global treatment. Therefore, one of the main challenges of MPM is to bring a global vision of the issues, of the state of the art and of the terminology that relate to the heterogeneity of models.

In this paper, we explore the MPM domain through a survey of different existing techniques which deal with the heterogeneity of models[1]. To better understand the problems raised by the heterogeneity of models, we had to get back to its origins. The results we obtained on this subject, which are also presented in this paper, help putting this survey into perspective and we believe that they complement the existing definitions of the MPM domain.

The paper is organized as follows. First, in Section 2, we highlight the causes of the heterogeneity of models with regard to the use of modeling techniques during the typical development cycle. Then, in Section 3, we propose a definition of the domain of Multi-Paradigm Modeling which focuses on the identified causes of the heterogeneity of models. Section 4 presents a survey of existing techniques from different fields which we consider as MPM techniques. We propose a set of different qualities which we consider important for MPM approaches in Section 5. We conclude this paper in Section 6.

[1] It is important to note that our work focuses on the modeling of the *behavior* of systems.

2 The heterogeneity of models: back to the basics

The problem of the heterogeneity of models is at the center of multi-paradigm modeling. In order to emphasize the issues in multi-paradigm modeling, we propose to turn to the elementary causes of the heterogeneity of models. Our goal in this section is to illustrate on an example why different models of a given system are used throughout the design cycle. But before describing the example, we first precisely define what we mean by "modeling language". Indeed, if it is usually agreed that modeling languages are the source of the heterogeneity problem, it seems that there is no clear consensus on the definition of a modeling language. We also discuss the meaning of the words "formalism" and "paradigm", which are often used alternatively in the MPM domain.

2.1 Modeling language, modeling formalism and modeling paradigm

A model is necessarily expressed in a language, which is usually called in the context of modeling, a "modeling language". Since MDE models are digital models[2], the modeling languages that we consider in this paper are computer languages — which are sometimes also called "software languages". There are many different ways to define a computer language. Following [7] and [8], we assume that the definition of a modeling language consists of[3]:

- an *abstract syntax*, which defines the abstract concepts from which models are built. In the context of MDE, the abstract syntax of a modeling language is often described by a meta-model. For DSLs, the concepts that are part of the abstract syntax are very close to domain concepts.
- a *semantics*, which defines how the abstract concepts are to be interpreted (by humans and/or by computers). Semantic definitions may take several forms: specifications written in natural language or in another computer language, formal semantics, mathematical definitions, etc. The semantics of a modeling language may even be defined by a tool implementation or by model transformations [10].
- a *concrete syntax*, which defines a notation for the language and gives a concrete representation of each element of the abstract syntax in this notation. A concrete syntax can be textual, graphical, or both.

Some languages used to model the behavior of systems are *executable*. This means that there is an algorithm which is able to compute a behavior of a model according to the semantics of the modeling language.

We call *modeling formalism* a modeling language which has a formal syntax and a formal semantics. The usual meaning of "formal" involves a mathematical definition. For instance, Petri nets are a modeling formalism. However, work such as [7] consider that "formal" means precisely and unambiguously defined (which may not necessarily involve a mathematical definition). We consider this nuance interesting since it emphasizes the need for the precise definition of the semantics of modeling languages, which is of prime importance in the context of heterogeneous modeling. As we will see in the following, mathematical definitions are very useful but not absolutely required.

The word "modeling paradigm" is much more abstract, therefore it cannot be defined with as much precision. According to G. G. Nordstrom [11], a modeling paradigm is a "a set of requirements that governs how any system in a particular domain is to be modeled". Our own interpretation is that a *modeling paradigm* can be considered as a mindset for modeling. As such, it can be implemented by several languages or formalisms. For instance, the synchronous reactive paradigm is implemented by Lustre [12], Esterel [13] and Signal [14]. It is important to note that, on the other hand, a given language or formalism may be based on several modeling paradigms. This happens in particular in the case of languages which are the union of other (and older) languages. A typical example is VHDL-AMS, which relies both on a discrete time modeling paradigm and on a continuous time modeling paradigm.

In the following, we will sometimes use the word "modeling technique" when a more encompassing meaning is needed to denote a modeling mean which can be a language, a formalism, a paradigm, or a tool.

2.2 On the use of modeling languages

At some point in the design of a system, the use of a given modeling language instead of another one depends on many criteria. In this section, we try to illustrate that, while some of these criteria may be subjective (i.e. may basically depend on the preferences of the designer), others are more rational and can therefore be identified rather precisely. These criteria can then help characterize the causes of the heterogeneity of models.

[2] As opposed to models on paper or to prototypes.

[3] This definition, which is quite general, also holds for programming languages. Actually, we consider that the difference between a programming language and a modeling language is only a matter of abstraction level. Therefore, as R. France and B. Rumpe do in [9], we consider that, in the case of software, "source code is a model of how the system will behave when executed".

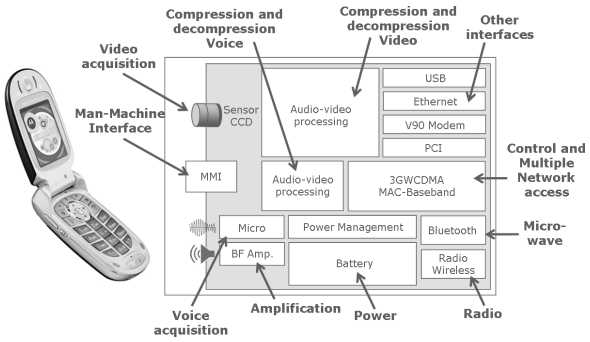

Fig. 1: A cell-phone example

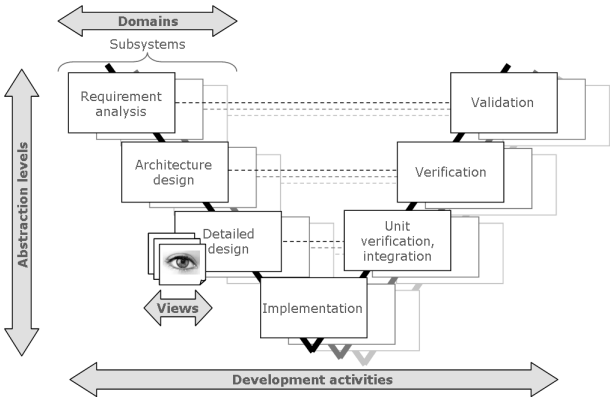

Fig. 2: Four causes for the heterogeneity of models

2.2.1 Introductory example

We consider the example of a multimedia cell-phone as shown on Figure 1. This cell-phone is a complex system since, even if it is not composed of a high number of elements, it involves several technical domains such as signal processing, software, opto-electronics, electronics, radio, power systems, networks etc. In this example, we adopt a development process which follows the traditional V cycle (see Figure 2). We believe that the results obtained on this example can be adapted to other development cycles. In the following, we detail what types of models are generally necessary to design this system in a MDE context.

First, for the design, this system will typically be split into subsystems for study by different dedicated expert teams. For example, one team will work on the signal processing chain while another will work on the embedded software applications. In signal processing applications, dataflow-oriented modeling tools such as Simulink by The MathWorks are commonly used. For embedded software design, many modeling languages can be used, among which UML (which may be used through a specific profile such as MARTE[4]), Statecharts, Communicating Sequential Processes or synchronous reactive languages such as Lustre/Esterel [13] (supported by the SCADE tool suite by Esterel Technologies).

Another important aspect of the development process is that the design of each subsystem progresses through different levels of detail. Designers start with specifications with a high level of abstraction and then refine and transform the subsystem models until they lead to an implementation. During the refinement process, a change in the modeling language may be necessary to capture particular design details. For example, if Finite State Machines are used to represent the logic of the software controller of the cell-phone, in a refinement step a timed version of Finite State Machines could be used to model timeouts, delays, etc.

Furthermore, the development process usually goes through distinct stages. The adequacy of a modeling technique depends very often on the considered stage of the process and on the activities which are carried out during this stage. For instance, during the requirement analysis phase, UML Sequence diagrams would be suitable for representing usage scenarios regarding the software controller of our cell-phone. For a static deadlock check in the verification stage, a representation of the behavior of the software controller and its environment as Petri nets would be more appropriate.

At last, besides the functional point of view, each subsystem also has to be studied from several other points of view in order to determine its non-functional properties — such as the performance, the energy consumption, etc. For instance, if we consider one objective of the design of our cell-phone which is the determination of the capacity of its battery, a power consumption-oriented view of the behavior of its software controller may be necessary. Such a model would probably involve differential equations or hybrid automata.

On this example, we observe that the choice of the appropriate modeling technique depends mainly on four criteria in addition to the designer's own preferences:

- the different technical or application domains involved in the system under design [15]: software, hardware, control, signal processing, etc.
- the different levels of abstraction at which the system is studied [4]: system level, algorithm level, register transfer level, etc.
- the different stages of the development cycle and their specific activities [15]: requirement analysis, design, verification, etc.

[4] MARTE is a UML profile for Modeling and Analysis of Real-Time and Embedded systems (`http://www.omgmarte.org/`).

- the different specific aspects to analyze during the design, which require a study from different points of view [16]: function, performance, power consumption, etc.

2.2.2 Modeling goals

These four criteria — *domain, abstraction, activity, view* — which are strongly tied to each other, characterize the purpose of a designer when he realizes a model of a system. These criteria form what we call a *modeling goal*. We represent a modeling goal by a 4-tuple <*domain, abstraction, activity, view*> where:

- *domain* is the technical or application domain of the considered system or subsystem;
- *abstraction* refers to the level of detail at which the system or subsystem is studied;
- *activity* is the current activity of the designer in the development process;
- *view* corresponds to an aspect under which the system or subsystem is considered.

During the whole development process (see Figure 2), changes are necessary:

- in the *level of abstraction* when refining models;
- in the *view* when exploring the different aspects of a system;
- in the *domain* when using techniques that are suitable to a particular field. Such a change may also happen jointly with a change of view or a change of the level of abstraction (e.g. when refining a model based on logical equations into a model in which the equations are implemented by transistors).
- and in the *activity* when going through the different stages of the development cycle.

Each of these changes generally requires a change of modeling technique in order to use the technique which suits the modeling goal best.

Such changes are both *unavoidable* and *essential*. They are unavoidable because, for instance, an embedded software developer obviously cannot use the same modeling technique as an electronics engineer. They are also essential because the use of a well mastered modeling technique allows a specialist to concentrate on its design problem and therefore to avoid design mistakes.

As a consequence, at each moment in the development process, a system is described by a collection of models expressed in different languages, i.e. *heterogeneous models*.

2.3 The heterogeneity problem

Heterogeneous models which represent the same system at a given moment in the development process do not form a global model of this system. Yet, such a global model is necessary to study global properties of the system. Indeed, such properties are very hard to infer from several isolated models, in particular in the case of a complex system. The manual integration of heterogeneous models to obtain a global view of the system is both tedious and error prone, not to mention issues related to traceability and maintainability. The analysis, the verification and the validation of properties of the whole system, in particular properties on its behavior [5], are therefore major difficulties.

In Section 2.1, we pointed out the difference between a modeling language and a modeling paradigm. In our opinion, this difference entails different levels of difficulty for the heterogeneity problem.

2.3.1 Several levels of difficulty

We recall here that we consider a modeling paradigm as a mindset for modeling. Thus, we consider that a modeling paradigm can be implemented by several languages (or formalisms). Moreover, we consider that a given modeling language can be based on several paradigms. As a consequence, several cases of heterogeneity are to be distinguished depending on the languages and paradigms involved in the considered models: (1) models described in different languages but with a unique underlying paradigm; (2) models described in different paradigms but with a unique representation language; and (3) models described in different paradigms and different languages. These three cases are presented in Table 1.

	Identical paradigms	**Different paradigms**
Identical languages	homogeneous	heterogeneous (2)
Different languages	heterogeneous (1)	heterogeneous (3)

Tab. 1: Types of heterogeneity

(1) Models described in different languages but with a unique underlying paradigm.

In this case, the languages which are used in the different models all rely on (quasi) identical paradigms. As an example, let us consider a model described in SyncCharts [17] and a model described in Esterel [13]. Both SyncCharts and Esterel rely on the synchronous-reactive paradigm but in SyncCharts a system is represented by state machines whereas in Esterel a system is represented by synchronized concurrent processes. Since both SyncCharts and Esterel rely on the synchronous-reactive paradigm, it is possible to rewrite a SyncCharts model into an Esterel model without loosing any information. As a consequence, the heterogeneity in this case is rather the heterogeneity of format.

(2) Models described in different paradigms but with a unique representation language.

This happens when a unique language is used in the different models but the underlying paradigms are different. Two different types of situations lead to this case:

- In the first type of situation, the models are described in a language which supports several paradigms, such as VHDL-AMS for instance. In such a case, the language creates a semantic bridge among the underlying paradigms. As a consequence, the integration of the models to form a global model of the system generally does not present important difficulties.
- The second type of situation regards models which were originally described in different languages with different underlying paradigms and which were translated into a common language. Let us consider the example of a model described in Simulink and a model described in Esterel. If we generate C code for both models, we obtain representations of these models in a common language. However, because the underlying paradigm of Simulink is radically different from the underlying paradigm of Esterel, the generated C code is designed completely differently. As a consequence, the integration of these representations to form a global model of the system in the C language can be very difficult. It requires, in particular, the adaptation of the data structures and the control flows (function calls) of the different models.

(3) Models described in different languages with different underlying paradigms.

This last case regards models which are described both in different languages and in different paradigms. This case presents the higher level of difficulty, since such languages usually share almost no concepts and almost no semantics. Let us illustrate this case using a SyncCharts model and a Simulink model. These models are described both in different paradigms and in different languages. However, it may be desirable to compose these models in order to obtain a global description of the behavior of a system. For instance, the Simulink model may describe the behavior of the system when it is in a particular mode, which is represented by a "state" in the SyncCharts model. A number of questions have to be answered for such a composition to have a well defined semantics. For instance, the way the discrete events from the SyncCharts model should be interpreted in the Simulink model, which uses continuous time, must be defined; the same is true of what happens to the state of the Simulink model at the entry or at the exit of a mode, etc. This last case of heterogeneity is, to our opinion, the most difficult to handle.

In conclusion, we observe in this table that the heterogeneity of paradigms (represented by the cases of the second column of the table) is the major source of difficulty in the heterogeneity problem.

2.3.2 One research domain?

In literature, approaches which deal with the different facets of heterogeneity are presented under different names such as "multi-paradigm modeling", "multi-formalism modeling", "multi-language modeling", "multi-view modeling", "multi-modeling", etc. In this paper, we advocate for a "unified" vision of heterogeneity, with different types of causes, different levels of difficulty and different types of approaches (which are presented in Section 4). We believe that a research field which would encompass the several types of existing approaches would therefore provide a more coherent environment for research on the heterogeneity problem. The work by P. Mosterman and

H. Vangheluwe in [6] is a first step toward such a field, which they call "Computer Automated Multi-Paradigm Modeling (CAMPaM)". In the following section, we present CAMPaM and we propose to complement its definition using the results presented in the previous sections in order to form the *Multi-Paradigm Modeling* domain.

3 The Multi-Paradigm Modeling domain

3.1 Existing definitions

In [6], P. Mosterman and H. Vangheluwe propose to define Computer Automated Multi-Paradigm Modeling (CAMPaM) as the field relying on three research directions: (1) model abstraction, i.e. dealing with the relationship between models at different levels of abstraction; (2) multiformalism modeling, i.e. coupling and transforming models described using different formalisms; and (3) meta-modeling, i.e. describing modeling formalisms.

In our point of view, the coupling and the transformation of models as well as the description of modeling formalisms are techniques which can be used to achieve multi-paradigm modeling. We consider that these two topics already are research fields in the domain of Model Driven Engineering in general [9]. We believe that the Multi-Paradigm Modeling domain would benefit from a focus on the causes of the heterogeneity problem as introduced in Section 2. The heterogeneity of levels of abstraction in models as cited in the research axes of CAMPaM is one of these causes, and we propose to extend the definition of MPM to the heterogeneity of domains, of views and of activities. We think that such a modification in the definition of the MPM domain would not only help to differentiate the MPM domain from other research domains such as MDE, but it would also help to select among the various existing techniques depending on the cause(s) of heterogeneity they address (i.e. not only on their underlying mechanisms). Moreover it would take into account in a better way other work such as the work by E. A. Lee on the heterogeneity of models of computation [18] (which we detail in Sections 4.2.3 and 4.3.3).

We propose to provide a general definition of the *Multi-Paradigm Modeling (MPM) domain* as the field which addresses the issues resulting from the heterogeneity of models with the primary objective of easing the joint use of heterogeneous models during the development cycle. Achieving this goal involves the automation of different actions on models such as the transformation, the composition, the co-execution, etc. We detail the research axes that we foresee for this research field in the next section.

3.2 Research axes

Following from the four causes of the heterogeneity of models that we have introduced in Section 2.2.2, we identify four main sources of difficulties in the MPM field: (1) the composition of models from different domains, (2) the formal or automatic processing of the abstraction/refinement relationship between heterogeneous models, (3) the joint use of multiple views of a given system, and (4) the use of related models of a system for several activities (e.g. for design, for test generation, for code generation and for model-checking). We give an insight of the respective issues of these four research axes in the following sections.

3.2.1 Composition of models from different domains

As we have seen in Section 2.2.2, the modeling techniques which are used in different domains or different technical field are very often completely different. Composing models from different domains is necessary in particular when dealing with models of different parts of a system.

Several issues arise when composing models from different domains. The first questions to answer are (a) can the models be composed and (b) if yes, will the composition yield the expected result. These questions arise because of the semantic differences which may exist among the modeling paradigms involved in the models to compose. Solving these issues requires the comparative analysis of the considered modeling paradigms. Assuming that these first two issues are solved, the next problem is to determine how to realize the composition. Several possibilities exist, as we will see in Section 4. Then, once models are composed, one remaining issue is the verification of the preservation of the semantics of the models in the composition. Indeed, if the semantics of the models involved in the composition is modified by the composition process, their maintenance and their evolution become major difficulties and may cause design incoherences in the modeled system.

3.2.2 Abstraction/refinement relationship between heterogeneous models

As introduced in Section 2.2.2, during the typical development process, models are progressively refined until they lead to an implementation. The "refinement" mechanism is used to build a model by adding details to a model whereas the "abstraction" mechanism is used to build a model by hiding details in a model. In these mechanisms, the considered models are linked by a particular relationship which we call the abstraction/refinement relationship

— but which is specifically called the “conformance” relationship in the B method [19]. The abstraction/refinement relationship sets preservation constraints on the properties of the involved models (this characteristic differentiates the abstraction/refinement relationship from the view relationship).

In the context of a manual refinement process, a well-known issue is to prove that a model obtained by refining a source model is really a refinement of this source model. In other words, the main problem in this context is to prove that the behavior specified by the more detailed model conforms to the behavior specified by the more abstract model. In an homogeneous context, behavioral refinement can be defined through the notion of simulation for state-based models, or through the preservation of logical properties, as in the B method for instance. For state-based models, the more detailed model is considered as a behavioral refinement only when it is simulated by the more abstract one. In methods such as B, a model is said to be a behavioral refinement of another model when it satisfies all the logical properties which hold for the more abstract model. In an heterogeneous context, defining the conformance relation between models may be an issue in itself.

Other issues in this context relate to the automation of the abstraction or of the refinement process through the use of model transformations, a topic which closely relates to the OMG’s MDA (Model Driven Architecture). In an heterogeneous context, such model transformations are to be designed not only to hide or to add details to models but also to translate them.

At last, difficulties also arise from the composition of models at different levels of abstraction. The heterogeneity of abstraction levels brings compatibility issues, which often come in addition to issues caused by other types of heterogeneity (this is true in particular in the case of the heterogeneity of domains as detailed in the previous section).

3.2.3 Joint use of multiple heterogeneous views

As illustrated in Section 2.2.2, several views are used when studying different aspects of a given system (in particular for studying non-functional properties). We call views models which represent the same part of the same system but from different perspectives. These views are generally linked to each-other by common elements which influence the observation of the system which is made through each view. For instance, the power consumption of a software controller which has different functioning modes may vary depending on its mode. Therefore, the function influences the power consumption and the notion of mode is shared between the functional view and the power consumption view of the software controller.

A first category of difficulties in this context regards the consistency among views. Two main trends for the handling of the consistency among views can be distinguished: the detection of inconsistencies a posteriori, i.e. after having built several views of a system, and the use of a derivation process for building consistent views of the system. Consistency problems also arise when changes in the design of a system impact multiple views at the same time. The consistency must be preserved in spite of the change, and the automation of the propagation of changes among views is highly desirable in this case.

Other types of issues relate to the merging of overlapping views to form a model of a system which takes all aspects into account. This action of merging views in a model is often named “model weaving” [20]. Difficulties in this domain regard the identification of the overlapping parts in the different views to ensure their correct integration in the resulting model. This category of issues closely relates to aspect oriented development which, applied to models, is addressed in a research field known as Aspect Oriented Modeling [21].

Solutions to the issues in this particular research axis lie in the formalization of the relationship between views. Of course, the heterogeneity of the modeling languages used in the different views makes these categories of problems more complex. In this context, we foresee a potential interest in exploring solutions in which views are not merged but rather used jointly to answer questions on the global properties of the behavior of a system (for simulation or for verification, for instance). We believe that this would help limiting maintenance and evolution problems on the considered views.

3.2.4 Use of related models for different activities

The last of our four research axes focuses on the fact that, even when considering only one aspect of one part of a system, different models of this part of the system are generally required for different design activities. For instance, the model of the specifications of a component used in the “implementation” activity will generally be different from the model of the same specifications used in the “verification” activity, where an implementation is checked against these specifications. At the origin of this difficulty is the fact that the tools which are used in the different activities of the design cycle are based on different theoretical foundations. Since the use of different theoretical foundations leads to different modeling formalisms, it is difficult to maintain the consistency of the models of a given component used in different activities.

At typical example of the use of several related models of a system for different activities occurs when the behavior of the system must be modeled for both model-checking and code generation. There exists languages such as Esterel which have a mathematically defined semantics and can be used for execution, model-checking and even to express the properties to be checked. However, their use is not always possible, and they are not suitable for all stages of the development cycle. In the general case, different activities such as verification and code generation call for different models. Therefore, more global solutions are needed to ensure the consistency of all the models used for different activities all along the development process.

4 Multi-paradigm modeling techniques

Now that we have introduced the Multi-Paradigm Modeling (MPM) domain, we propose to explore existing techniques which address its issues. Because of its inherent multi-disciplinary nature, MPM involves research teams with technical backgrounds as different as control science, signal processing, model checking, modeling language engineering or system-on-chip development. We have identified several types of techniques, from different horizons, which help achieving the objectives of Multi-Paradigm Modeling as defined in Section 3. We introduce these techniques in the following sections. We present the principles of each category of techniques and give examples of existing approaches. We provide details only for illustrating the various approaches, more information can be found in the cited papers. We do not pretend that this survey is exhaustive, however, it is an attempt to give an overview of approaches which we consider as part of the state of the art of MPM.

4.1 Specific approaches

Numerous approaches target the combination of specific paradigms, in particular to solve issues related to the heterogeneity of time and synchronization.

The combination of continuous and discrete time in models has been extensively investigated in approaches for modeling hybrid systems. Overviews of these key techniques are provided in [6] and in [22]. The cited approaches include hybrid automata, VHDL-AMS, Simulink/Stateflow and Modelica [23]. The DEVS formalism [24], proposed by B. P. Zeigler, can also be classified in this category since it allows the description of systems based on both state transitions and differential equations.

With regard to the heterogeneity of synchronization, several approaches address the issues related to "Globally Asynchronous - Locally Synchronous" (GALS) systems [25]. A survey of the state of the art in GALS architectural techniques, design flows and applications is proposed in [26].

At last, research on hardware/software codesign and system-on-chip design has also lead to approaches combining specific paradigms, one example of which is proposed in [27]. We would like to emphasize here that, due to the specific role that hardware plays for software — the role of "execution platform" — the design cycle for hardware/software codesign and system-on-chip design is different from the typical V cycle that we have introduced in this paper. The hardware/software heterogeneity causes particular problems in the context of this specific design cycle (e.g. problems related to allocation, synthesis, etc.), which are not the main topic of this paper.

Pros and cons We call the approaches introduced in this section "specific approaches" because they address the heterogeneity problem in an ad-hoc manner and allow the combination of restricted sets of modeling paradigms or languages. These restrictions allow such approaches to provide very efficient support for the specific heterogeneity they address. However, in the context of MDE, the number of modeling languages tends to increase with the generalization of language definition techniques. That is why more flexible approaches are needed in order to address the heterogeneity problem for an open set of modeling languages. However, when a modeling problem fits the constraints of a specific approach, this approach will generally be more efficient than a more flexible one.

4.2 Approaches addressing the capture of modeling languages

Approaches which support an open set of modeling language must provide a way to capture the syntax and the semantics of modeling languages. Several techniques exist for addressing this issue. Strictly speaking, these techniques are not part of the state of the art of MPM. However, we consider that an overview of these techniques is necessary for the understanding of the rest of the paper.

Well known techniques for describing the semantics of modeling languages formally include *logics*, *algebras* and *co-algebras*[5] [28]. In the context of MDE, as emphasized by P. Mosterman and H. Vangheluwe in [6], *meta-*

[5] The notion of co-algebras is the dual of the notion of algebra. Algebras are meant to describe how to construct objects whereas co-algebras are meant to describe observations of objects. It is possible to compose co-algebras like it is possible to compose algebras, by using the dual composition operations.

modeling is a key technique to capture the abstract syntax of modeling languages. Several possibilities exist to capture the semantics of a language whose abstract syntax is expressed as a meta-model. We present some of them in the following sections.

4.2.1 Kermeta

Using Kermeta [29], for instance, it is possible to give an executable semantics to a modeling language by defining methods with an imperative semantics for the elements of its meta-model. Each element of the meta-model is provided an execution method. For instance, in the meta-model of a language based on a simple version of finite state machines, an execution method can be provided to the element which represents the notion of transition in order to specify what happens when a transition is fired. Since the execution operations are defined at a "meta" level, it is possible to execute any model which conforms to the meta-model of the language.

4.2.2 Semantic Units (SUs) and semantic anchoring

Semantic Units, which are introduced in [30], are another method to describe the semantics of a language whose abstract syntax is defined using a meta-model. The principle of Semantic Units is similar to the principle of Kermeta since the goal of a Semantic Unit is to attach an execution semantics to the elements of the meta-model which represent the abstract syntax of a modeling language.

A Semantic Unit is composed of (a) an abstract data model and (b) a set of operational rules defined on the elements of (a). Both the abstract data model and the operational execution rules are described in AsmL (Abstract State Machine Language) [31]. A mapping associates the elements of the meta-model of the language to the elements of the abstract data model of the Semantic Unit. Therefore, this mapping anchors the abstract syntax of the language — the meta-model — to the operational semantics defined by the execution rules in AsmL.

4.2.3 Models of Computation (MoCs)

Another way to define a modeling language (syntax and semantics), is based on the concept of *Model of Computation (MoC)*. We will see in this section that MoCs present interesting features for addressing heterogeneity.

The concept of MoC as used in this paper is the concept introduced by E. A. Lee. In his work, a MoC is viewed as a set of characteristics which are common to several (component oriented) modeling languages. For instance, different languages for describing sequential processes which communicate by rendezvous involve the same model of computation — Communicating Sequential Processes (CSP). In the same way, languages for describing processes which communicate synchronously through signals involve another model of computation — Synchronous Data Flow (SDF). In summary, a MoC corresponds to a class of modeling languages for which the computation and the communication characteristics are similar. In most cases, a model of computation corresponds to a modeling paradigm (as defined in Section 2.1).

In [32], E. A. Lee and A. L. Sangiovanni-Vincentelli describe and compare the properties of several models of computation in a theory called the "Tagged Signal Model". In this theory, the behavior of a system (or of a component) is viewed through the notion of "signal". A signal is made of events, which are value-tag pairs. A model of a system (or of a component) is a set of signals which represent the possible behaviors of the system (resp. component). A model of computation restrains the sets of possible values and of possible tags on which signals are expressed, and defines composition rules which determine how the behaviors of components are composed in the model.

The concept of model of computation has been implemented in the Ptolemy II project [5]. The Ptolemy approach relies on (a) an actor-oriented abstract syntax and (b) the notion of "domain". The abstract syntax of Ptolemy is made of few simple elements, as illustrated on Figure 3. In this syntax, the primary elements are actors [33], a special kind of components. Actors communicate through ports and interact through relations. A Ptolemy "domain" defines the set of rules which define how a model is to be interpreted according to a particular model of computation. Each domain is implemented in Ptolemy by a "director" class and a "receiver" class. A director is in charge of controlling the execution of a model according to the computation rules (the "execution model") of a MoC whereas the role of a receiver is to implement the communication rules (the "interaction model") of a MoC. In the context of Ptolemy, *heterogeneous models* are models which involve different domains, that is to say different models of computation.

The major advantage of this approach is that the same abstract syntax is used for all models. The semantics of each model is defined by its associated domain. To give an intuitive illustration of this point, let us consider the example presented on Figure 4. In this example, we consider three models which are syntactically identical. However, their associated domains (i.e. models of computation) are different. We give an intuitive graphical representation of the semantics of each model as it is interpreted according to its domain. The first model, which is

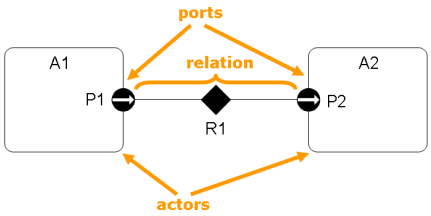

Fig. 3: Abstract syntax of Ptolemy

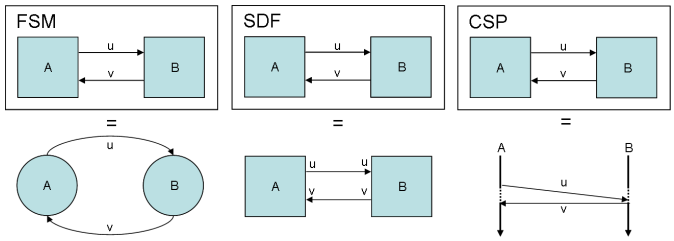

Fig. 4: Interpretation of the same model by different MoCs

interpreted according to the Finite State Machine domain (FSM), represents an automaton with two states, A and B, and two transitions triggered by the events u and v. The second model, which is interpreted according to the Synchronous DataFlow domain (SDF), represents two processes, A and B, which exchange synchronously u and v data samples on communication channels. The third model, which is interpreted according to the Communicating Sequential Processes domain (CSP), represents two processes, A and B, which exchange the values u and v by rendezvous.

Having a unique abstract syntax for representing models simplifies the heterogeneity problem since there is no need to define mappings between the elements of the abstract syntax used in the different models. The heterogeneity remains "only" at the semantic level (which we admit is not the easiest problem!). We present how models involving different models of computation can be composed, using Ptolemy and other approaches, in Section 4.3.3.

We conclude here our section on the capture of the syntax and semantics of modeling languages. Capturing the semantics of modeling languages is a mandatory step in heterogeneous modeling because the composition of heterogeneous behavioral models can have a well defined meaning only if the behavior of the composed models is well defined. We will see in section 4.3.3 that the approaches presented in this section differ greatly in the means that they provide to address the heterogeneity problem. Other techniques exist but we focused on techniques which we find of particular interest in the context of multi-paradigm modeling. We now explore techniques which address directly the heterogeneity problem.

4.3 Approaches addressing the heterogeneity problem

We have identified several types of techniques which address the heterogeneity problem in different ways. We propose to classify these techniques according to the following categories:

- Techniques based on the **translation of models**, which aim at supporting the translation between modeling languages;
- Techniques based on the **composition of modeling languages**, which allow the construction of composed modeling languages;
- Techniques based on the **composition of models**, in which heterogeneous models are assembled together;
- Techniques based on the **joint use of modeling tools**, which target the correct interoperation among heterogeneous modeling tools;
- Techniques based on a **unifying semantics**, which propose a unique but customizable semantic support for describing heterogeneous models.

This classification shares similarities with the classifications proposed in [34] and [35]. In particular, the way we group some of the reviewed approaches is comparable to the grouping which is proposed in these classifications. However, the classification that we propose can be considered as an adaptation and an extension of these former works in the MPM context as presented at the beginning of the paper. First, we show how the approaches presented in each category of the classification address the different facets of the heterogeneity problem introduced in Section 3.2. Moreover, our classification includes approaches which have been developed in the context of Model Driven Engineering (MDE). In addition, we have tried to review approaches which cover different activities of the development cycle — e.g. specification, design, simulation, model-checking, etc.

We detail the different categories of techniques which we propose in the following sections and provide examples of existing approaches for each category.

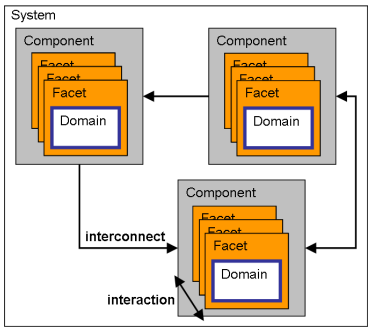

Fig. 5: Combination of facets with Rosetta

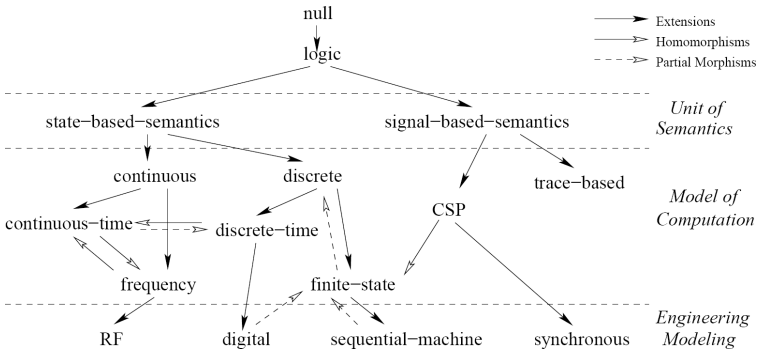

Fig. 6: Lattice of the Rosetta domains

4.3.1 Techniques based on the translation of models

The techniques which we present in this section help translating models between modeling languages. Considering the different cases of heterogeneity described in Section 2.3.1, this method corresponds conceptually to a transition from the second line of Table 1 (models described using different modeling languages) to the first line of the table (models described using a single modeling language). Therefore, this method enables a reduction of the effort needed to use heterogeneous models together.

In the context of the translation of heterogeneous models, the main concern is the choice of the modeling language which should be the target of the translations. The underlying problem is of course the preservation of the semantics of the models in the target language. There are three possibilities: (1) choosing one of the modeling languages involved in the heterogeneous models as a target, (2) choosing another modeling language as a target or (3) composing the modeling languages involved in the heterogeneous models and choose this composed modeling language as a target. In this section, we deal only with the first two options, we discuss the third one in Section 4.3.2 in which we introduce approaches targeting the composition of modeling languages.

Several techniques exist for translating models. Most of them are now generally classified under the term "model transformation techniques". An introduction to model transformation and a survey of model transformation techniques can be found in [36] and [37]. In this section, we present approaches which propose techniques for translating models which describe the behavior of systems in the case of graph based, logic based and co-algebra based representation of models.

ATOM3 ATOM3 [38] is a tool which implements a model transformation technique based on graph rewriting. In this approach, models are represented internally using graphs and model transformations are modeled by graph grammars which specify rewriting rules. To guide the choice of the target language of the transformation, the authors propose a "Formalism Transformation Graph" (FTG) which formalizes the relations existing between well-known modeling formalisms. This method is flexible since the meta-model which is the target of the transformation can be chosen according to the current objective of the designer. Moreover, this method can support a wide range of modeling languages, provided that their abstract syntax is described by a meta-model and that a transformation can be written between the meta-model of this language and the target meta-model (we emphasize here that this may be particularly difficult for certain types of languages). The main drawback of this method is that the number of transformations to design increases exponentially with the number of languages to take into consideration.

HETS (Heterogeneous Tool Set) HETS [39, 40] is a tool which implements an approach based on the notion of colimit (as defined in category theory). This approach uses a graph of logics and logic translations, formalized as institutions and comorphisms. Based on this graph and on the notion of colimit, this approach allows the automatic determination of a logic in which two models described in two different logics can be translated. Therefore, this approach supports heterogeneous models through the translation between logics. HETS, the tool set which supports this approach, integrates parsers, static analyzers and theorem provers for several logics, thus allowing proof management on heterogeneous models. The major advantage of such an approach compared to ATOM3 is that the graph of logics is used to provide automated support for the transformations.

Rosetta Rosetta [41, 42] is a system specification language with a formal semantics based on co-algebras. We introduce Rosetta in this section on approaches based on the translation of models, but Rosetta also provides features for the composition of models which are particularly interesting. We present here the general principles of Rosetta, we will insist on its composition features in Section 4.3.3.

Rosetta is a language for expressing constraints in the context of the description of the specifications of a system. Rosetta introduces the notion of "facet", which models a particular aspect or view of a component, e.g. its function or its energy consumption. Rosetta allows the combination of facets either in order to assemble models of components or to model different aspects of one component, as it is illustrated on Figure 5. In Rosetta, "domains", which are special kind of facets, encapsulate vocabulary and semantics for domain specific modeling. Domains are related to each other by extension relations, thus forming a lattice which is shown on Figure 6. The semantics of a Rosetta facet is denoted by a coalgebra defining observations on changes over an abstract state. This is the key concept underlying this approach: since the state of a facet is only an observation of an abstract state, it is possible to define multiple state observations with multiple semantics and to keep them related. Composition operations as well as transformations are defined on facets thanks to the domain lattice. Therefore, Rosetta provides a formal support for multi-paradigm modeling through both model composition and model transformation.

Pros and cons The purpose of translation-based approaches is to allow the translation of models between different modeling languages, whether they are dedicated to different domains, to different levels of abstraction, to different views or to different activities. Therefore, they address some of the issues caused by the four causes of heterogeneity. However, the transformation of models may be very complex depending on the semantic gap between the underlying paradigms of the source and target modeling languages. Moreover, these approaches do not provide support for "gluing" models together once they have been translated, which is necessary to run global analyses. As illustrated in Section 2.3.1, a semantic gap may remain between models even after translation, which may make this task difficult. Approaches based on the composition of models such as Rosetta address this issue and will be discussed in Section 4.3.3.

4.3.2 Techniques based on the composition of modeling languages

As introduced in the previous section, when dealing with heterogeneous models, an idea is to compose the modeling languages involved in the heterogeneous models and to choose the resulting composed modeling language as a target for a model transformation. The composition of modeling languages whose abstract syntax is described by a meta-model can be achieved using meta-model composition techniques.

Composition of meta-models In [43], M. Emerson and J. Sztipanovits present several techniques for composing meta-models. In such approaches, the elements contained by different meta-models are assembled using several types of operators (equivalence, merge, etc.). The resulting meta-model therefore contains a mixture of the elements present in the source meta-models.

The main advantage of using a composed meta-model as the target of the transformation is that the information loss can be reduced since the target meta-model contains elements of the source meta-models. However, this method is not scalable since it implies the modification of the target meta-model and of the associated model transformations each time an additional modeling language is taken into consideration. Moreover, it is important to note that for the moment most methods for composing meta-models do not provide support for handling the semantics which may be attached to the elements of the meta-model (and which may be defined using one of the approaches introduced in Section 4.2 for example). In the next section, we present an approach which allows the composition both at the syntactic level and at the semantic level.

Semantic Units (SUs) As introduced in Section 4.2.2, Semantic Units can be used for semantic anchoring of modeling languages whose abstract syntax is defined by a meta-model. In [44], the authors show how SUs can be used to support heterogeneous modeling. The authors first define the notion of "primary" SU. A primary SU captures the minimal semantic elements of a particular class of modeling languages. As such, the notion of primary SU is similar to the notion of model of computation as defined in Section 4.2.3. Then the authors propose mechanisms for composing SUs in order to form more complex SUs. SUs which are built from primary SUs are called "derived" SUs. Derived SUs can also be composed with other SUs and so on. Two steps are necessary for composing SUs. First, correspondence tables have to be created to define relations between the abstract data models of the constituent SUs. Then execution rules have to be created to define how the execution rules of the constituent SUs interact. These additional execution rules play the role of execution orchestrators.

This approach applies to multi-paradigm modeling since a composed SU defines a semantics which is made of heterogeneous primary semantics. As a consequence, a modeling language whose semantics is anchored on a composed SU is a "multi-semantic" modeling language. The main asset of the SU approach is that it offers the reusability of the semantic descriptions of modeling languages. This approach can be used in association with model transformation techniques as mentioned earlier, but it can also be used to build new DSMLs (Domain Specific Modeling Languages) with a well defined semantics "on demand".

Pros and cons By allowing the creation of hybrid modeling languages, approaches based on the composition of modeling languages can be used to create "multi-domain" or "multi-abstraction" modeling languages. It is also possible to use this type of technique to extend a modeling language with concepts for representing heterogeneous views. The main drawback of these methods is their lack of scalability. Each time an additional modeling language is taken into consideration, the target hybrid language must be rebuilt. Moreover, existing models described using this language must also be updated, as well as model transformations targeting this language.

4.3.3 Techniques based on the composition of models

The third type of techniques which we have identified is based on the composition of models. By "composition of models" we mean the coherent coupling of several heterogeneous models.

We begin this section by presenting techniques which are based on the concept of Model of Computation (MoC) introduced in Section 4.2.3. As previously mentioned, MoCs present characteristics which facilitate the support of heterogeneity. In this context, heterogeneous models are models which involve different MoCs.

Ptolemy II Ptolemy, which was introduced in Section 4.2.3, is specifically designed to support heterogeneous modeling and is oriented toward simulation. In Ptolemy, heterogeneous models are organized into hierarchical layers, each layer involving only one MoC, i.e. one Ptolemy domain. This is achieved through the use of "opaque composite actors". The principle, which is illustrated by the actor named A2 on Figure 7, is the following: (a) an actor which is used in a model involving a domain (i.e. a director and corresponding receivers) D1 can contain other interconnected actors and (b) these internal actors can be interpreted according to a domain D2 which is different from D1. An actor such as A2 is said to be "composite" since it contains other actors, and it is said to be "opaque" because it appears as an atomic actor from its outside environment. Thanks to this hierarchical structure, each hierarchical level in a model can involve a different MoC and MoCs are combined in pairs at the boundary between two hierarchical levels.

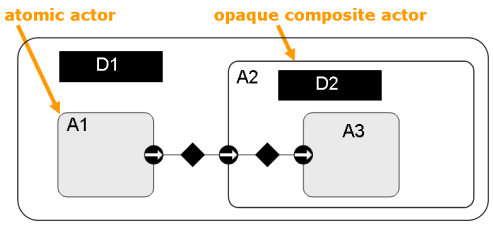

Fig. 7: Atomic and opaque composite actors in Ptolemy

To our knowledge, Ptolemy is the approach which supports the widest range of models of computation. The list of supported MoCs includes continuous-time modeling, several types of dataflow semantics, finite state machines and modal models, various kinds of process network semantics and different versions of discrete-events semantics. From our point of view, the main drawback of the Ptolemy approach is that the way MoCs are combined at a boundary between two hierarchical levels is fixed and coded into the Ptolemy kernel. This implies that a modeler has either to rely on the default adaptation performed by the tool, or to modify the design of parts of its model (by adding adaptation components) in order to obtain the behavior he expects. ModHel'X, which we describe in the following paragraph, addresses this issue.

ModHel'X In ModHel'X [45], models are made of "blocks". The notion of block is a generalization of the notion of actor, the main difference being that the behavior of actors is *fired* whereas the behavior of blocks is *observed*. This means that blocks are considered as "complete" black boxes whose internal control flow remains hidden. If a block needs to receive control information to trigger internal mechanisms, it uses ports, just like for data. As a consequence, in ModHel'X, a MoC is seen as the set of rules which define how to combine the observations made on the blocks of a model to obtain an observation of the model. An important advantage of the ModHel'X approach lies in the notion of "*interface block*". The behavior of an interface block is described by an internal model which is hidden from its outside environment. The MoC involved in this internal model can be different from the MoC involved in the model in which the interface block is used. Therefore, the notion of interface block allows hierarchical heterogeneity, like the notion of opaque composite actor of Ptolemy. However, unlike an opaque composite actor, an interface block includes an "adaptation layer" which allows the modeler to specify explicitly how the semantics of the MoCs are adapted at the boundary between the two heterogeneous

hierarchical levels. This allows the flexible combination of models involving different MoCs in a hierarchical structure without modifying the models which are assembled.

"42" In both Ptolemy and ModHel'X, describing the execution and communication rules defined by a MoC is a difficult task. The choice made by the authors of the "42" approach [46] is to automate the description of these rules as much as possible. In 42, which relies on the synchronous paradigm, the code of the MoCs (called "controllers") is generated from the contracts of the components (described using automata), from the relations between their ports and from additional information related to activation scheduling. The strength of this approach resides in the description of the behavioral contract of components. However, such a description may not be available (in the case of an external IP — Intellectual Property — i.e. a component provided by a third party for instance) or may not be easy to establish, in the case of continuous time behaviors for example. Moreover, the approach seems to restrict the set of supported MoCs.

Rosetta A last approach to cite in this section is Rosetta. As introduced in Section 4.3.1 when detailing approaches of the translation of heterogeneous models, Rosetta relies on the notion of "facet" for modeling different aspects or parts of a system. Rosetta is particularly interesting for the composition of models because it allows the "vertical composition" of facets. Vertical composition consists in assembling facets which represent multiple views of the same component. Using vertical composition, a component is represented by a "laminated model", which is made of facets representing different views of the component which interact with each other. Thanks to these particular concepts, Rosetta addresses both the heterogeneity of domains and the heterogeneity of views.

Pros and cons The composition of models addresses issues related to the four types of heterogeneity and seem very versatile. MoC-based approaches such as Ptolemy, ModHel'X or 42, address the issues related to the heterogeneity of domains and to the heterogeneity of levels of abstractions thanks to hierarchical composition. Both the heterogeneity of domains and the heterogeneity of views are addressed by Rosetta. In addition, Rosetta also aims at addressing the issues caused by the heterogeneity of activities by proposing a set of tools for simulating Rosetta models and for generating test vectors using Rosetta specifications (even if this tool support does not seem very mature for the moment). As a conclusion, by avoiding model translation, model composition techniques provide a way to address the heterogeneity problem while preserving the modularity of models.

4.3.4 Techniques based on the joint use of modeling tools

We have focused so far on approaches which help handling the heterogeneity problem either at the level of *modeling languages* or at the level of *models*. Other approaches exist which help handling the heterogeneity problem at the level of *modeling tools*. The general idea is to take advantage from the performance and accuracy of tools which are optimized for specific modeling languages and to build "bridges" which facilitate the use of these tools in an heterogeneous context. Cosimulation techniques[6], which we introduce in this section, are based on this idea.

Cosimulation has been extensively investigated in literature. Examples of existing approaches can be found in [48]. Some cosimulation environments integrate a fixed and restricted set of simulators. For instance, in the MUSIC approach [15], SDL-Matlab cosimulation is used for system-level validation. Specific hardware/software cosimulation environments are also proposed as commercial tools such as Seamless by Mentor Graphics[7]. Other cosimulation approaches aim at supporting the connection of any type of simulators so that they execute different models in a consistent manner. The main issue of this kind of approaches lies in the synchronization of the heterogeneous simulators as well as the communication of heterogeneous data among them during the simulation. In the following, we present several approaches which provide support for this type of cosimulation.

High Level Architecture (HLA) HLA [49] is a standard initially developed by the US Department of Defense (DoD) in an effort to facilitate the interconnection of simulators. The goal of HLA is to standardize the way simulators, which are called "federates", can connect on a "Run-Time Infrastructure" (RTI), i.e. a middleware, to form a "federation" (see Figure 8). The HLA specification includes "Rules" that simulators must obey to be compliant to the standard and defines an "Interface Specification" which tells how HLA compliant simulators interact with the RTI. HLA also defines an "Object Model Template" (OMT) which describes how information is communicated between federates. In conclusion, HLA provides a standard architecture which eases the interconnection of

[6] The term "cosimulation" is sometimes given a wider meaning than the meaning we use in this work. In particular, it may refer to approaches in which heterogeneous models are transformed to be simulated by a unique simulation environment (like in the POLIS approach [47] for instance).

[7] `http://www.mentor.com/seamless`

simulators. It is not an implementation itself, but several implementations of the RTI exist and recent simulators are HLA compliant.

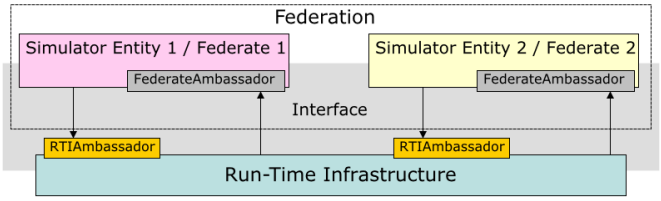

Fig. 8: Interconnection of two simulators with the High Level Architecture (HLA)

It is necessary to emphasize here that simulators which are dedicated to different modeling paradigms usually have communication and execution models which differ a lot (this is the case in particular when considering domain-specific modeling paradigms). Therefore, their correct synchronization and coordination for the co-execution of heterogeneous models are sources of major difficulties [3]. An illustration and formalization of this problem using DEVS is presented in [50] in the case of continuous and discrete simulators. Therefore, the goal of several cosimulation approaches is to propose automated support for the coherent synchronization and coordination of heterogeneous simulators as well as for their coherent communication. Such work complements the work done on HLA which provides an infrastructure for the interconnection of simulators.

Cosimulation bus As an example, in [51, 52], the authors present a cosimulation approach based on SystemC in which wrappers allow the connection of heterogeneous "modules" (i.e. heterogeneous simulators for models of parts of the system) on the SystemC environment which is used as a "backplane". A wrapper consist in a simulator interface that adapts a given simulation environment to the cosimulation bus and a communication interface dedicated to a communication protocol. These interfaces are automatically generated using a library. This approach targets in particular simulators for continuous time modeling and discrete time modeling. Moreover, it allows the cosimulation of models at different levels of abstraction.

Pros and cons Approaches based on the joint use of modeling tools present attractive characteristics in terms of modularity (neither the modification nor the merging of models are needed), of reusability (the same tool is used for an unlimited set of models) and of performance (dedicated tools are highly optimized). They address the heterogeneity of domains and of levels of abstraction and we believe that they could also be used in the context of the heterogeneity of views (by cosimulating the models of different views of a system) and even of activities (for instance by cosimulating an implementation and an observer which checks a property). However, the adaptation layer which is required to connect a tool to the others is specific to this tool, not to the underlying modeling paradigm. Therefore, an adapter built for one tool may not be easily reused for other tools based on the same modeling paradigm. Another issue is that the interface of a tool may not provide access to all the information which is necessary for its co-execution with other tools.

4.3.5 Techniques based on a unifying semantics

The last category of our classification concerns approaches which propose a unique but customizable semantic support for describing heterogeneous models. These approaches are quite specific in the sense that they allow the expression of semantic variations on a unique and fixed semantic basis. The following descriptions of two different approaches illustrate this principle.

Metropolis In the Metropolis approach [53], which is oriented toward hardware/software co-design, the semantic basis which has been chosen is process networks. Therefore, in Metropolis, models are made of "processes" associated with "threads", and processes communicate by "events" exchanged through "ports" via a "media". However, Metropolis allows modelers to customize the way threads synchronize and communicate, therefore allowing the use of variations of the process networks semantics.

The semantics of models in Metropolis is formalized using trace and trace structure algebras [54]. This approach, as well as the Tag Machines [55] approach, was inspired by the work of E. Lee and A. Sangiovanni-Vincentelli on the Tagged Signal Model [32] (see Section 4.2.3). In Metropolis, a trace corresponds to the observable behavior of an "agent", i.e. a process, an actor, a module, etc. A trace structure defines a model of an agent and consists of the set of the possible traces of the agent. A trace structure algebra is used to define how

these models are composed. Thanks to this formal underlying semantics, Metropolis provides a quite complete framework of modeling methods and modeling tools for verification, simulation and synthesis.

"Inframodels" "Inframodels" [4, 56] is an approach which is particularly interesting because it supports a wide range of model analysis techniques, from reachability analysis to model execution. This approach is based on a special type of labeled directed hypergraphs[8] which are called "inframodels". In this approach, a labeled directed hypergraph represents all the possible behaviors of a component in terms of successive possible states. Hyperarcs represent transitions from a set of possible source states to a set of possible target states. As such, this representation shares similarities with Petri Nets. However, this approach extends Petri Nets by giving the possibility to define customized firing rules which are used to capture different notions of synchrony.

Pros and cons A common drawback of the approaches presented in this section is that the range of the modeling paradigms they support is quite limited compared to other types of approaches. However, thanks to their well defined underlying semantics, they provide an interesting tool support, thus helping to address the issues related to the heterogeneity of activities.

4.4 Other approaches

After this presentation of the different categories of approaches which we consider in the MPM state of art, we propose to extend the scope of this survey to approaches which we do not consider as multi-paradigm modeling approaches but which provide interesting features for handling heterogeneity.

4.4.1 Component-based approaches

In the context of component and service-oriented software development, an important goal is to enable the composition of components or services developed by third-parties, which are therefore potentially heterogeneous. When integrating such components, mismatches between their interface signature, their behavior, their protocol, their requirements in terms of quality of service or their semantics may arise.

In this section, we review different techniques that have been developed to handle these issues. We believe that these techniques could apply to heterogeneous modeling since, in the context of model composition, models could be considered as software components. Therefore, for each technique, we try to emphasize its interest for multi-paradigm modeling.

Compatibility checking A first idea to prevent the mismatch between components is to check their compatibility by using the description of their behavior or of their interface. The *compatibility* between components ensures that their interaction is *possible*, considering the available information on their interface and on their behavior.

Several approaches exist in this domain. For instance, in [57], "interface automata" are used to capture the temporal I/O behavior of components, i.e. assumptions about the order in which the methods of a component are called and the order in which the component calls external methods. Interface automata interact through the synchronization of input and output actions. They allow the automatic checking of the compatibility of components using the composition of the automata which represent their interfaces: two components are said to be compatible when the composition of their interface automata has a nonempty set of states.

In [58], component interfaces are defined by L. de Alfaro and T. Henzinger as sets of constraints that apply to the environment of the components for guaranteeing their behavior. As such, interfaces express the assumptions the designer makes about the environment in which the component is to be deployed whereas the model of the component specifies how the component behaves in an arbitrary environment. These definitions are used as a basis for defining interfaces theories, which consist of interface algebras, component algebras and implementation relationships. With interface theories, it is possible to manipulate concepts such as implementation, refinement, composition and decomposition in the context of component based design and verification. For instance, if we decompose an interface into smaller interfaces and if we build components that implement these interfaces, then the interface theory guarantees that the components can be composed and connected to form a system which implements the initial interface.

Interface automata and interface theories provide general and powerful mechanisms for compatibility and composability checking in component-based design. An interesting application of interface automata to compatibility checking for Ptolemy II components and directors is presented in [59]. This work shows how these concepts can be applied to heterogeneous model composition when based on the concept of model of computation.

[8] Hypergraphs are graphs in which edges, called hyperarcs, can connect any number of vertices, called nodes.

Component adaptation (software adaptation) When the behavior of components is not fully compatible, another idea is to correct their behavior to enable their composition. This is the goal of software adaptation [60, 61].

Two different types of correction are usually cited: the generation (as automatic as possible) of static adaptors and the dynamic correction at run time. The generation of static adaptors is now a quite well known subject, with formal definitions [61], formal methodologies [62] and enforcement of liveness or safety properties [63]. However, the computation cost in most of these approaches is usually high and the size of the generated adaptors important. Solutions are explored to enable the incremental computation of the adaptors or the on-the-fly modification (e.g. the removal of incorrect interactions or the behavioral reduction) of adaptors [64].

Approaches in which heterogeneous models are coupled to form composed models (see Section 4.3.3) could benefit from the application of software adaptation techniques. The concept of semantic adaptation as proposed in ModHel'X (presented in Section 4.3.3) shows the potential of such an idea.

4.4.2 Heterogeneous interactions

In component oriented modeling approaches, the computation and the communication aspects are orthogonalized. The description of the computation is split into parts which correspond to the models of the components whereas the communication is described by one or more interaction models. Various types of interaction exist, including synchronous or asynchronous message passing (as in usual component models such as the CORBA Component Model), unbounded buffers (as in some process networks models) or shared variables. Interaction is usually represented in a model by a connection/link between components. Assigning semantics to connections in a model can be achieved either in a global way by defining a common semantics that applies to every connection, or connection by connection with the ability to mix different types of interaction. The former method is similar to the use of a model of computation (see Section 4.2.3) whereas the latter defines heterogeneous interactions.

BIP (Behavior, Interaction, Priority) BIP (Behavior, Interaction, Priority) [65, 66] provides formally defined mechanisms for describing combinations of components in a model using heterogeneous interactions. In BIP, components are described using three layers: one layer defining the behavior of the component (in the form of a labeled transition system), one layer defining the connectors of the component (i.e. the way it is possible to interact with it) and a third layer defining priorities among the possible interactions. In a given BIP model, two links between two components may denote different types of interaction, depending in particular on the nature of the connectors. Components can be composed to form new "compound" components. BIP supports model execution and model-checking. Moreover, thanks to its layered and formal representation of components and interactions, BIP allows correctness by-construction for properties such as deadlock-freedom or liveness.

Architectural Interaction Diagrams (AIDs) Another approach allowing heterogeneous interactions in a model is proposed in [67]. It introduces "buses" as first-class entities into a modeling paradigm called Architectural Interaction Diagrams (AIDs). In this approach, buses represent the notion of interprocess communication. The framework is extensible, allowing the user to implement new interprocess primitives. A mathematical definition of AIDs is given as a foundation for analyzing the behavior of the AIDs elements.

Both BIP and AIDs share very similar concepts. The main difference between them is, from our point of view, the underlying mathematical formalization: the advantage of BIP is that it has been designed to enable correctness by-construction.

Pros and cons The main interest of the approaches presented in this section lies in the flexibility they offer to modelers for the description of the interaction mechanisms between components of systems. The heterogeneity of interaction models is difficult to compare to the different types of heterogeneity introduced at the beginning of the paper. This is mainly due to the fact that a given interaction model is usually associated to a given modeling paradigm. For instance, Communicating Sequential Processes (CSP) is associated to the rendezvous interaction model. Providing the ability to mix interaction models in models could eventually be considered as an intermediate approach between the composition of modeling languages (see Section 4.3.2) and the use of a unifying semantics (see Section 4.3.5).

4.4.3 Megamodels

The last approach we present in this extension of our MPM survey is "megamodels". The concept of "megamodel" [68, 69] is quite recent. According to J. Bézivin and J. M. Favre, megamodels aim at providing structures for representing the global relationships between models, metamodels, modeling languages, models of transformations, etc. In [69], six types of relations are proposed: (1) decomposition, (2) representation, (3) belonging

(to a set), (4) conformance, (5) transformation and (6) semantics. Megamodels therefore allow the definition of meaningful links between potentially heterogeneous "*-model" elements (model, meta-model, etc.).

Strictly speaking, megamodels cannot be considered as a multi-paradigm modeling technique. Rather, we consider them useful to study the respective roles of models, meta-models and modeling languages which are used during the development process of a given system. We believe that this may have interesting applications in model repositories and that the information provided by a mega-model could be used by multi-paradigm modeling tools.

4.5 Conclusion of the survey

This section on megamodels ends our survey of the techniques which relate to MPM. We have seen that many techniques have been developed to address the heterogeneity problem. We have classified these techniques depending on their underlying mechanism and we have tried to show how these different types of techniques address the different types of issues raised by the different sources of heterogeneity. Still, work remains to be done to provide engineers with a precise and detailed reference grid for choosing a particular approach when considering a particular problem. In the next section, as a first step in that direction, we discuss the qualities which we consider important for MPM approaches from an engineering point of view.

5 Qualities of MPM approaches

In the survey presented in the previous section, we have reviewed different types of techniques and we have seen how they address the different facets of the heterogeneity problem. In this section, we focus on the qualities which MPM approaches should have from an engineering point of view, independently from their underlying mechanisms or from the type of heterogeneity they address.

From our survey, we have extracted two criteria which are significant to us: (1) support for an open set of modeling languages, (2) support for formal verification of properties. In the following, we detail the reasons that lead us to choose these criteria and we list the approaches that match them.

5.1 Support for an open set of modeling languages

As we have seen in Section 4.1, the ad-hoc combination of a finite set of modeling paradigms is a well addressed issue. However, the development of MDE entails the use of many types of modeling languages — in particular Domain Specific Languages (DSLs). It is recognized now that DSLs are a key feature to gain efficiency and productivity as well as to improve the quality of systems [2]. Thus, the development of MDE leads to the need for more flexible tools which are able to support the use of such languages. Thanks to metamodeling techniques, tools such as GME [70] or the Eclipse Modeling Project (EMF, GEF, GMF)[9]now allow the automatic generation of tool support for user defined modeling languages. In the same way, MPM tools should be easily and automatically extended for supporting additional modeling languages. This is possible only if the approach is generic enough and has been designed to support an open set of modeling languages.

Model transformation approaches such as [38] fit this criterion of genericity. The cost for supporting an additional modeling language, provided that its meta-model is available, is mainly due to the design of the transformation(s) from this language to the target modeling language(s).

Approaches based on the composition of modeling languages are generic in the sense that they allow the composition of any languages provided that their syntax and semantics are expressed in a given format. However, as introduced in Section 4.3.2, their practical application can be very costly.

Most of the model composition approaches such as Ptolemy II and ModHel'X also match this criterion of genericity. With model composition approaches, the cost of adding support for an additional modeling language corresponds to the description of the semantics of the modeling language using the provided tools (MoCs, coalgebras, etc.). This task can be very difficult. However the ability to glue models described using this new language with other models is automatically provided by the underlying mechanism of the approach (which is the main interest of this kind of approaches).

Cosimulation approaches which support the addition of any type of simulator are generic. For instance, the approach described in [48] allows the connection of an additional simulator by generating a wrapper for plugging it to the simulation bus. The cost of this addition comes from the necessary analysis of the simulator for adding the corresponding elements to the library which is used for the generation of the wrappers.

[9] `http://www.eclipse.org/modeling/`

5.2 Support for formal verification of properties

The verification and validation phases are of crucial importance in the development cycle, especially when designing critical systems. When considering heterogeneous models, the verification and validation tasks, which are already highly complex by nature, require extraordinary efforts (when they are not merely impossible). Several attempts have been conducted to address the multiple issues in this domain. In this last section, we review approaches that allow reasoning about the formal properties of heterogeneous models, therefore providing an essential support for the verification and validation tasks.

With respect to this criterion, we have found Rosetta (see Section 4.3.1) and BIP (see Section 4.4.2) of particular interest. The distinctive feature of Rosetta is the ability to *formally* combine different views of a given component. Thanks to coalgebras, Rosetta seems to support a very wide range of semantics, including non state-based semantics. However, Rosetta has been designed as a specification language with a high level of abstraction and Rosetta models are not directly executable. This means that the designer cannot use Rosetta during the design phases of the development process and has to change for another language. In addition, its tool support does not seem to be mature for the moment. Parsers, interpreters (for a subset of the Rosetta language) or test vector generators have been developed but seem to be disconnected initiatives, thus leading to a multiplication of the tools to use when dealing with Rosetta specifications.

As for BIP, in our opinion, its main strength is the ability to obtain correctness by construction for several properties such as mutual exclusion or deadlock-freedom. In the context of complex systems, such an ability can be a serious advantage since usual model-checking methods reach their limits. However this feature seems to be experimental for the moment and restricted to a subset of the BIP language [71]. In a more traditional way, BIP is supported by an execution platform and a model-checking tool called IF. The fact that the hierarchy used in BIP is not strict (i.e. components are not considered as black boxes) can be considered as a drawback: the formal power of BIP is at the expense of modularity. Moreover, the way the behavior of components has to be expressed using labeled transition systems limits the range of supported semantics.

Among other interesting approaches, we also have selected Metropolis and interface automata. Metropolis provides the designer with a model verification feature based on its formal semantics (trace structure algebras). The main interest of Metropolis lies in its tool support. As a counterpart, it is limited to process network-oriented languages. In the domain of component-based design, the work by L. de Alfaro and T. Henzinger on interface automata and interface theories defines a solid formal basis for compatibility checking. Interface automata have been used in Ptolemy II for checking the compatibility between actors and domain directors. As such, interface automata seem promising for MPM.

In conclusion, it is obvious that reasoning formally about properties at a global level on a set of heterogeneous models is a challenge. As we have seen in this section, research in this domain is ongoing. Several important advances have been made, but there are still important limitations. In the next section, we conclude this paper and give some perspective for this work.

6 Conclusion

It is now widely admitted that there is no single modeling technique which is suitable and convenient for all the stages in the development of all the parts of a complex system. It is also generally agreed that the resulting heterogeneity of models causes major difficulties for system engineering. However this consensus has not lead to a coherent research effort in this direction yet. We believe that the role of the Multi-Paradigm Modeling research field is to bring a global vision of the issues and of the state of the art that relate to the heterogeneity of models.

In this paper, we have identified and illustrated the causes of this problem which we call *the "heterogeneity problem"*. We have used these results to propose an extended definition of the Multi-Paradigm Modeling (MPM) domain with research axes which correspond to the different facets of the heterogeneity problem. After a presentation of the MPM context, we have explored the existing MPM approaches in a survey in which we identified different types of techniques. In the presentations of the different types of techniques, we have highlighted how these techniques address the different facets of the heterogeneity problem. We have concluded the survey by presenting a set of qualities which we consider important for MPM approaches from an engineering point of view.

The work presented in this paper lays the foundation of a framework for comparing MPM approaches. This framework certainly needs to be further studied and refined, but we think it opens some tracks which may help to build a research roadmap for Multi-Paradigm Modeling. The preliminary results presented in this paper show that there is no single type of technique which solves all the categories of problems. This seems very natural to us, in fact: just like different modeling paradigms are used for different modeling goals, it seems reasonable to use different techniques for solving different heterogeneity problems. The important thing is that these different techniques, used in conjunction, should help modelers to use different modeling techniques all along the design

cycle. Much work remains to be done on the interoperability of MPM techniques to achieve such an objective. We believe that this work should be done while keeping in mind the different causes of the heterogeneity problem in order to improve the adequacy of MPM techniques to the different facets of the heterogeneity problem.

References

[1] A. van Deursen, P. Klint, and J. Visser. Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, 35(6):26–36, 2000.

[2] R. B. Kieburtz, L. McKinney, J. M. Bell, J. Hook, A. Kotov, J. Lewis, D. P. Oliva, T. Sheard, I. Smith, and L. Walton. A software engineering experiment in software component generation. In *ICSE '96: Proceedings of the 18th international conference on Software engineering*, pages 542–552, Washington, DC, USA, 1996. IEEE Computer Society.

[3] P. K. Davis and R. H. Anderson. Improving the Composability of DoD Models and Simulations. *The Journal of Defense Modeling and Simulation (JDMS)*, pages 5–17, april 2004. The Society for Modeling and Simulation International (SCS).

[4] M. Pezzè and M. Young. Generation of multi-formalism state-space analysis tools. In *ISSTA '96: Proceedings of the 1996 ACM SIGSOFT international symposium on Software testing and analysis*, pages 172–179, New York, NY, USA, 1996. ACM.

[5] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity – The Ptolemy approach. *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, 91(1):127–144, January 2003.

[6] 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. Special Issue: Grand Challenges for Modeling and Simulation.

[7] D. Harel and B. Rumpe. Meaningful modeling: what's the semantics of "semantics"? *Computer*, 37(10):64–72, October 2004.

[8] Anneke Kleppe. A Language Description is More than a Metamodel. In *Fourth International Workshop on Software Language Engineering (ATEM 2007) at the 10th IEEE/ACM International Conference on Model-Driven Engineering Languages and Systems (MODELS 2007)*, Nashville TN, United States, 2007.

[9] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society.

[10] G. Karsai, A. Agrawal, F. Shi, and J. Sprinkle. On the Use of Graph Transformations for the Formal Specification of Model Interpreters. *Journal of Universal Computer Science, Special issue on Formal Specification of CBS*, 9(11):1296–1321, 2003.

[11] G. G. Nordstrom. *Metamodeling – Rapid Design and Evoluion of Domain-Specific Modeling Environments*. PhD thesis, Vanderbilt University, Electrical Engineering, May 1999.

[12] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[13] 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.

[14] 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.

[15] P. Coste, F. Hessel, Ph. Le Marrec, Z. Sugar, M. Romdhani, R. Suescun, N. Zergainoh, and A. A. Jarraya. Multilanguage design of heterogeneous systems. In *CODES '99: Proceedings of the seventh international workshop on Hardware/software codesign*, pages 54–58, New York, NY, USA, 1999. ACM.

[16] S. E. Schulz. An introduction to SLDL and Rosetta. In *Proceedings of the Asia and South Pacific Design Automation Conference 2000 (ASP-DAC 2000)*, pages 571–572, 2000.

[17] C. André. Representation and analysis of reactive behaviors: A synchronous approach. In *CESA'96 (Computational Engineering in Systems Applications)*, pages 19–29. IEEE-SMC, 1996.

[18] E. A. Lee and H. Zheng. Leveraging Synchronous Language Principles for Heterogeneous Modeling and Design of Embedded Systems. In *Proceedings of the 7th ACM and IEEE international conference on Embedded software (EMSOFT'07)*, pages 114–123, New York, NY, USA, 2007. ACM.

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

[20] Jules White and Jeff Grayand Doug Schmidt. Constraint-Based Model Weaving. *Transactions on Aspect-Oriented Software Development*, Special Issue on Aspects and MDE (Robert France and Jean-Marc Jezequel, eds.), 2009.

[21] Tzilla Elrad, Omar Aldawud, and Atef Bader. Aspect-Oriented Modeling: Bridging the Gap between Implementation and Design. In *ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering, ACM SIGPLAN/SIGSOFT (GPCE 2002)*, Lecture Notes in Computer Science, pages 189–201. Springer, 2002.

[22] L. P. Carloni, M. Di Benedetto, R. Passerone, A. Pinto, and A. Sangiovanni-Vincentelli. Modeling techniques, programming languages and design toolsets for hybrid systems. Technical report, IST - Columbus Project, 2004.

[23] P. Fritzson and V. Engelson. Modelica – A Unified Object-Oriented Language for System Modeling and Simulation. In *European Conference on Object-Oriented Programming (ECOOP98)*, pages 67–90, July 1998.

[24] Bernard P. Zeigler and Sankait Vahie. DEVS formalism and methodology: unity of conception/diversity of application. In *Proceedings of the 25th conference on Winter simulation (WSC 93)*, pages 573–579, New York, NY, USA, 1993. ACM.

[25] D. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems*. PhD thesis, Dept. of Computer Science, Standford University, 1984.

[26] M. Krstic, E. Grass, F.K. Gurkaynak, and P. Vivet. Globally Asynchronous, Locally Synchronous Circuits: Overview and Outlook. *Design & Test of Computers, IEEE*, 24(5):430–441, 2007.

[27] B. D. Theelen, O. Florescu, M. C. W. Geilen, J. Huang, P. H. A. van der Putten, and J. P. M. Voeten. Software/hardware engineering with the parallel object-oriented specification language. In *MEMOCODE '07: Proceedings of the 5th IEEE/ACM International Conference on Formal Methods and Models for Codesign*, pages 139–148, Washington, DC, USA, 2007. IEEE Computer Society.

[28] B. Jacobs. Introduction to Coalgebra. Towards Mathematics of States and Observations. In preparation, draft electronically available at http://www.cs.ru.nl/B.Jacobs/CLG/JacobsCoalgebraIntro.pdf.

[29] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel. Weaving Executability into Object-Oriented Meta-Languages. In *Proceedings of the 8th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS/UML 2005)*, pages 264–278, 2005.

[30] K. Chen, J. Sztipanovits, S. Neema, M. Emerson, and S. Abdelwahed. Toward a semantic anchoring infrastructure for domain-specific modeling languages. In *Proceedings of the Fifth ACM International Conference on Embedded Software (EMSOFT'05)*, pages 35–44, New Jersey, September 2005.

[31] E. Börger and J. K. Huggins. Abstract State Machines 1988-1998: Commented ASM Bibliography. *CoRR*, cs.SE/9811014, 1998.

[32] E. A. Lee and A. L. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12):1217–1229, 1998.

[33] Gul Agha. *Actors: a model of concurrent computation in distributed systems*. MIT Press, Cambridge, MA, USA, 1986.

[34] Rolf Ernst and Ahmed A. Jerraya. Embedded system design with multiple languages. In *Proceedings of the 2000 conference on Asia South Pacific Design Automation (ASP-DAC '00)*, pages 391–396, New York, NY, USA, 2000. ACM.

[35] Hessam S. Sarjoughian. Model composability. In *Proceedings of the Winter Simulation Conference (WSC 2006), Monterey, California, USA, December 3-6, 2006*, pages 149–158, 2006.

[36] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45(3):621–645, 2006.

[37] T. Mens and P. Van Gorp. A taxonomy of model transformation. *Electr. Notes Theor. Comput. Sci.*, 152:125–142, 2006.

[38] J. de Lara and H. Vangheluwe. $ATOM^3$: A Tool for Multi-formalism Modelling and Meta-modelling. In *5th Fundamental Approaches to Software Engineering International Conference (FASE 2002)*, pages 595–603, April 2002.

[39] Till Mossakowski, Christian Maeder, and Klaus Lüttich. The Heterogeneous Tool Set – HETS. In *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2007), memeber conference of ETAPS 2007*, volume 4424/2007 of *Lecture Notes in Computer Science*, pages 519–522, Braga, Portugal, March 2007.

[40] M. Codescu and T. Mossakowski. Heterogeneous colimits. 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.

[41] J. Streb and P. Alexander. Using a Lattice of Coalgebras For Heterogeneous Model Composition. In *Proceedings of the MoDELS Workshop on Multi-Paradigm Modeling*, pages 27–38, Genova, Italy, October 2006.

[42] C. Kong and P. Alexander. The Rosetta Meta-Model Framework. In *Proceedings of the IEEE Engineering of Computer-Based Systems Symposium and Workshop (ECBS'03)*, April 2003.

[43] M. Emerson and J. Sztipanovits. Techniques for metamodel composition. In *OOPSLA – 6th Workshop on Domain Specific Modeling*, pages 123–139, October 2006.

[44] K. Chen, J. Sztipanovits, and S. Neema. A case study on semantic unit composition. In *MISE '07: Proceedings of the International Workshop on Modeling in Software Engineering*, page 3, Washington, DC, USA, 2007. IEEE Computer Society.

[45] C. Hardebolle and F. Boulanger. ModHel'X: A Component-Oriented Approach to Multi-Formalism Modeling. *Models in Software Engineering - Workshops and Symposia at MoDELS 2007, Nashville, TN, USA, September 30 - October 5, 2007, Reports and Revised Selected Papers*, 5002/2008:247–258, June 2008.

[46] F. Maraninchi and T. Bouhadiba. 42: Programmable Models of Computation for a Component-Based Approach to Heterogeneous Embedded Systems. In *6th ACM International Conference on Generative Programming and Component Engineering (GPCE'07)*, pages 53–62, October 2007.

[47] 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.

[48] P. Gerin, S. Yoo, G. Nicolescu, and A. A. Jerraya. Scalable and flexible cosimulation of SoC designs with heterogeneous multi-processor target architectures. In *ASP-DAC '01: Proceedings of the 2001 conference on Asia South Pacific design automation*, pages 63–68, New York, NY, USA, 2001. ACM.

[49] Defense Modeling and Simulation Office of the US Department of Defense. IEEE standard for Modeling and Simulation High Level Architecture (HLA) – Framework and Rules. IEEE Std 1516-2000, Sep 2000.

[50] L. Gheorghe, F. Bouchhima, G. Nicolescu, and H. Boucheneb. Formal Definitions of Simulation Interfaces in a Continuous/Discrete Co-Simulation Tool. In *Seventeenth IEEE International Workshop on Rapid System Prototyping (RSP 2006)*, pages 186–192, June 2006.

[51] G. Nicolescu, S. Yoo, and A. Jerraya. Mixed-level cosimulation for fine gradual refinement of communication in soc design. In *DATE '01: Proceedings of the conference on Design, automation and test in Europe*, pages 754–759, Piscataway, NJ, USA, 2001. IEEE Press.

[52] G. Nicolescu, S. Martinez, L. Kriaa, W. Youssef, S. Yoo, B. Charlot, and A. Jerraya. Application of multi-domain and multi-language cosimulation to an optical mem switch design. In *ASP-DAC '02: Proceedings of the 2002 conference on Asia South Pacific design automation/VLSI Design*, pages 426–434, Washington, DC, USA, 2002. IEEE Computer Society.

[53] F. Balarin, L. Lavagno, C. Passerone, A. L. S. Vincentelli, M. Sgroi, and Y. Watanabe. Modeling and designing heterogeneous systems. In *Concurrency and Hardware Design, Advances in Petri Nets*, volume 2549 of *Lecture Notes in Computer Science*, pages 228–273, London, UK, 2002. Springer.

[54] J. R. Burch, R. Passerone, and A. L. Sangiovanni-Vincentelli. Using Multiple Levels of Abstractions in Embedded Software Design. In *EMSOFT '01: Proceedings of the First International Workshop on Embedded Software*, pages 324–343, London, UK, 2001. Springer-Verlag.

[55] A. Benveniste, B. Caillaud, L. P. Carloni, and A. L. Sangiovanni-Vincentelli. Tag machines. In *Proceedings of the 5th ACM International Conference On Embedded Software (EMSOFT 2005)*, pages 255–263. ACM, September 2005.

[56] Mauro Pezzè and Michal Young. Constructing multi-formalism state-space analysis tools: using rules to specify dynamic semantics of models. In *ICSE '97: Proceedings of the 19th international conference on Software engineering*, pages 239–250, New York, NY, USA, 1997. ACM.

[57] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 109–120, New York, NY, USA, 2001. ACM.

[58] L. de Alfaro and T. A. Henzinger. Interface Theories for Component-Based Design. In T. A. Henzinger and C. M. Kirsch, editors, *Embedded Software, First International Workshop, EMSOFT 2001*, volume 2211, pages 148–165, Tahoe City, CA, USA, October 2001. Springer.

[59] E. A. Lee and Y. Xiong. A Behavioral Type System and Its Application in Ptolemy II. *Formal Aspects of Computing*, 16(3):210–237, August 2004.

[60] B. Morel and P. Alexander. Automating Component Adaptation for Reuse. In *18th IEEE International Conference on Automated Software Engineering (ASE 2003)*, pages 142–151, Montreal, Canada, October 2003. IEEE Computer Society.

[61] D. M. Yellin and R. E. Strom. Protocol specifications and component adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333, 1997.

[62] A. Bracciali, A. Brogi, and C. Canal. A formal approach to component adaptation. *Journal of Systems and Software*, 74:45–54, January 2005.

[63] P. Inverardi, L. Mostarda, M. Tivoli, and M. Autili. Synthesis of correct and distributed adaptors for component-based systems: an automatic approach. In *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 405–409, New York, NY, USA, 2005. ACM.

[64] 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.

[65] 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.

[66] G. Gössler and J. Sifakis. Composition for component-based modeling. *Science of Computer Programming*, 55(1-3):161–183, March 2005.

[67] A. Ray and R. Cleaveland. Architectural Interaction Diagrams: AIDs for system modeling. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 396–406, Washington, DC, USA, 2003. IEEE Computer Society.

[68] 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.

[69] J.-M. 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 fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

[70] J. Davis. GME: the generic modeling environment. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 82–83, New York, NY, USA, 2003. ACM.

[71] G. Gössler, S. Graf, M. E. Majster-Cederbaum, M. Martens, and J. Sifakis. Ensuring properties of interaction systems. In Thomas W. Reps, Mooly Sagiv, and Jörg Bauer, editors, *Program Analysis and Compilation, Theory and Practice, Essays Dedicated to Reinhard Wilhelm on the Occasion of His 60th Birthday*, volume 4444 of *Lecture Notes in Computer Science*, pages 201–224. Springer, 2007.