Meta-models in Europe: Languages, Tools and Applications

Roberto Passerone, Imene Ben Hafaiedh, Albert Benveniste, Daniela Cancila,
Arnaud Cuccuru, Werner Damm, Alberto Ferrari,
Sebastien Gerard, Susanne Graf, Bernhard Josko, Leonardo Mangeruca,
Thomas Peikenkamp, Alberto Sangiovanni-Vincentelli, Francois Terrier

April 2, 2009

Contents

Abstract

Model-based design methodologies are increasingly finding acceptance in the development of electronics systems thanks to their flexibility and the availability of tools for analysis and implementation. In this landscape, meta-modeling has emerged as an essential method to organize theories and methods for the development of coordinated representations that are more suitable than standard models for the heterogeneous environment in which modern embedded systems operate. We review the role that meta-models have played and are playing in several research projects across Europe. In particular, we discuss language design techniques and language profiling in the context of numerous industrial applications that emphasize safety in heterogeneous specifications. Then we describe in more details the modeling principles and the infrastructure underlying the SPEEDS European project, and highlight the way meta-modeling techniques have helped its implementation and applications.

1 Introduction

Abstraction and refinement techniques are the cornerstone of design methodologies. Abstraction is the fundamental device by which designers extract the essential features of a complex problem, therefore reducing the complexity of its representation and manipulation, and increasing productivity. This process has been shaped during the past decades by the emergence of conceptual representations and languages that are progressively more detached from the implementation of the system, by neglecting those details that are relevant only in the context of specific realizations. The converse process of refinement fills out those details, with tools that are able to evaluate design alternatives through simulations and analysis, and, when possible, with synthesis and compilation techniques. In most cases, the refinement step proceeds by mapping, decomposing and subsequently assembling the system from elementary parts, or *components*, that encapsulate a logical unit of behavior.

The adoption of component-based methodologies has, in fact, paved the way to the development of the *model-based* approach to design (e.g., see [TG06]). This shift was marked by an increased use of concurrency, which more naturally maps on the structure of modern distributed embedded systems, over the traditional software paradigm of sequential execution. Concurrency, however, increases complexity, since the number of interactions that must be considered tends to grow more than linearly with the number of components, and sometimes significantly so. This has led to the proliferation of a host of component *models*, whose primary purpose is to constrain the kind of interaction patterns available to designers, in order to simplify the analysis or achieve a certain degree of expressiveness.

Designers use component models because they are convenient ways to represent a design, and because they can choose the abstraction that best matches the characteristics of the system under development. Convergence of technologies into the same application area, however, results in heterogeneous specifications that use several models simultaneously for the system description. The same degree of heterogeneity can be observed when the description of the system is partitioned into separate orthogonal aspects, or *view-points*. In this case, the fragmentation is at the component level, and must be resolved by resorting to appropriate combination techniques that account for the interdependencies of the specifications [RBB+09].

It is in this context that researchers have taken a step back and began to study and operate on the models themselves, to understand their relationships and to put an order to an otherwise informal collection of methods and tools. To achieve this, the very same modeling techniques that had proven so successful in design were employed to construct models of models, or *meta-models*, which have quickly been embraced by such methodologies as the Model Driven Architecture (MDA) [Sel, Sch06] and Platform-Based Design (PBD) [SV02].

In this paper we review the role that models and meta-models have played and are playing in several research projects across Europe. In the first part, Section 2, we discuss language design techniques and their use in several industrial applications. In the second part, Section 3, we describe in more details the modeling principles and the infrastructure underlying the SPEEDS European project, and highlight the way meta-modeling techniques have helped its implementation and applications.

2 Language design strategies in meta-modeling frameworks

Embedded systems development is being challenged to provide global solutions for reconciling three conflicting concerns: enrichment/refinement of system functionalities, reduction of time-to-market and production costs, and compliance with non-functional requirements [TG06]. In order to fulfil these objectives, both academic and industrial communities have been promoting for

more than a decade design approaches and methodologies relying on Model-Based Engineering (MBE) [Sel, Sch06]. MBE raises different concerns related to problems such as, for instance, model transformations, model repository or also specific modeling languages. Meta-modeling techniques are at the basis of most research efforts in the state of the art in these different areas. A meta-model is the result of capturing concepts and rules of a specific modeling language via more or less formal means [Esp07]. In this context, one says that a model *conforms to* a meta-model if the model respects the set of modeling rules defined in the meta-model ("just like a well-formed program conforms to the grammar of the programming language in which it is written" [Esp07]). The example shown in Figure 1 illustrates this. At the top, we show the graphical definition of the meta-model of

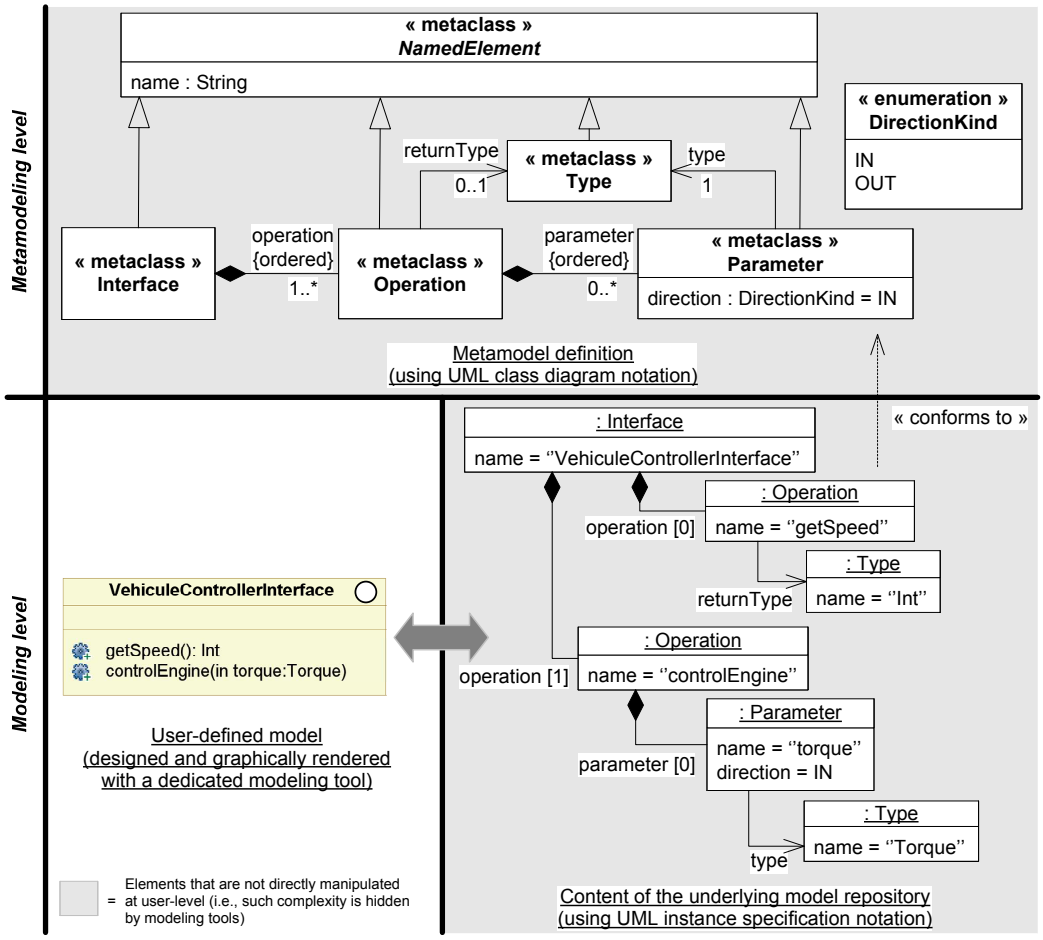


Figure 1: Definition/usage of a language for interface specification in a MDE fashion

a simple modeling language for interfaces with operations. An *interface* is a *named element*, which can have a number of associated operations. These, in turn, may take typed parameters with a specified direction, and may return a typed value. Designers are not concerned with the meta-model definition, and instead use graphical tools to represent their specification of an interface, as shown in the bottom left of the figure. At the repository level, the specification is represented as a particular instantiation of objects derived from the above class diagram, as shown in the bottom right, in a

way that conforms to the meta-model definition.

There are two main strategies for the usage of meta-modeling techniques for the design of domain specific languages (DSL), i.e., the "heavyweight" and the "lightweight" approaches. Section 2.1 below provides a description of these two variants, with a particular focus on the lightweight usage. Then, we report on industrial feedback that highlights the benefits and the challenges of the lightweight approach (Section 2.2). This feedback come from projects in which the LISE laboratory [LIS] is (or has been) involved and they concern multiple application domains. Finally, Section 2.2.3 provides some future perspectives.

2.1 Heavyweight vs. Lightweight DSL design

Figure 2 gives an overview of the heavyweight and lightweight approach for defining specific domain languages, and their impact on underlying tool architectures. The figure shows the steps and the

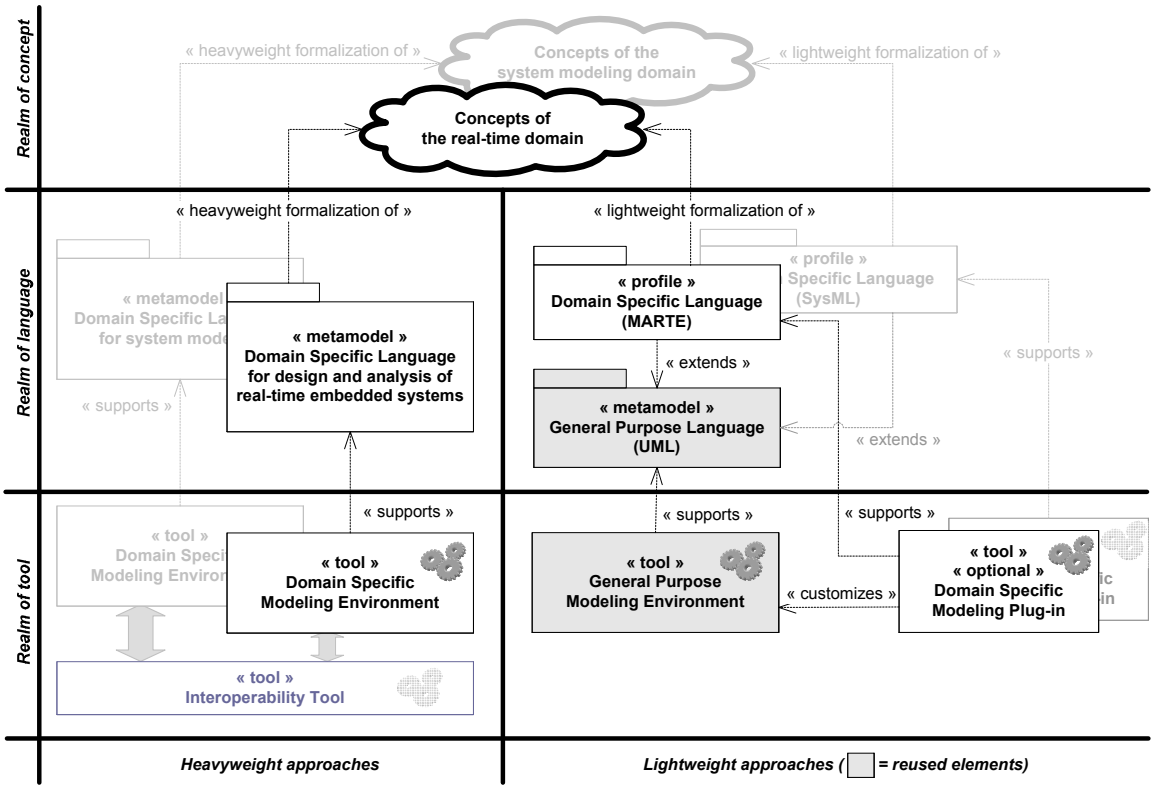


Figure 2: Heavyweight and lightweight usages of meta-modeling techniques

models involved in the creation of languages and tools corresponding to some concept of interest.

The heavyweight variant, outlined in the left-hand side of Figure 2, implies creating a new meta-model fully dedicated to each modeling language under study (such as our interface specification language shown in Figure 1). This leads to the definition of a domain specific language (DSL) that

is optimally suited to the problem at hand [ESCG08]. Since every discipline has its own specific language, the main drawback of this approach is how "to interface the various parts of the design so that integrated systems can be verified, tested, or simply unambiguously understood" [ESCG08]. Therefore, besides the difficulties underlying the creation of a meta-model for each targeted DSL, this approach requires a lot of effort to obtain an integrated and consistent tool chain, as illustrated on the bottom-left part of Figure 2.

The lightweight variant, outlined in the right-hand side of Figure 2, relies on the extension of an existing meta-model. This meta-model typically captures the modeling concepts and rules of a more general-purpose modeling language, such as the well-accepted Unified Modeling Language (UML) [OMGe]. In the context of UML, this mechanism of lightweight extension is called a *UML profile*. Each extension of an element from the UML meta-model is formally captured by a concept called *stereotype*. Each stereotype definition can be associated with properties and/or modeling constraints which make sense for the domain targeted by the profile. Stereotypes are then manipulated at the modeling level as annotations on model elements. General-purpose UML elements of this model may be thus associated with a domain-specific semantics and/or notation, which is explicitly part of the underlying model repository. Technically, this means that various (semi-)automatic tools are able to access the information captured by the profile (e.g., for code generation, verification or domain-specific analysis). The SysML (for system modeling) [OMGc] and MARTE (for real-time embedded design and analysis) [GS08, OMGd] profiles are OMG standards [OMGa] defined as a lightweight extension (i.e., profile) of UML2.

The most difficult part when defining a lightweight extensions is to determine what are the most suitable elements of the meta-model that must be extended (i.e., the meta-classes for which stereotypes must be defined). This is not necessarily an easy task, and typically requires a deep knowledge of the meta-model to be extended. However, once the profile has been defined, this approach allows one to specialize a general-purpose tool (such as Papyrus or RSA [Pap, Rat]) at low cost. General-purpose tools support domain-specific aspects in the sense that stereotypes are made available at the modeling level in the form of annotations. In addition, tools usually support a stronger form of integration with the possibility to define optional "plug-ins", as illustrated in the bottom-right side of Figure 2.

Another advantage of this approach has to do with the fact that real-world models are typically multi-domain (see description of the ATESST project in Section 2.2.2). There is therefore a strong demand from industry for the capability to capture cross-domain concerns in the same model. As multiple profiles can be applied to a given model, integration and combination of DSLs is easier than in the case of heavyweight approaches, where a dedicated meta-model and modeling environment are defined for each DSL and where a potentially complex interoperability tool may be required to ensure consistency and interaction between the different "views" of the model. For example, in the Papyrus UML modeler [Pap] developed at the LISE laboratory [LIS], a designer can automatically import and apply several profiles in a given model and, hence, add information concerning multiple domains on pre-existing model elements.

2.2 Profiles as DSL and industrial feedback

The embedded system industry has already strongly endorsed the lightweight extension mechanism by triggering, supporting and contributing to the development of two important profiles for their application domain: the SysML and MARTE profiles.

- The *SysML (System Modeling Language)* profile has been motivated by the need to provide modeling support that is not only limited to software centric development of systems. Instead, SysML also addresses a wider understanding of system architectures and interactions with their

environment, whether they are later realized through software or not. At the modeling level, two main aspects have been integrated: links with reference requirement documentation and the capability to describe coarse grain architectures supporting both discrete (e.g., messages, data, material, etc.) and continuous (e.g., energy, etc.) interactions. All other aspects (such as behavioral descriptions) are assumed to reuse largely existing UML constructs and semantics.

- The *MARTE (Modeling and Analysis of Real Time Embedded systems)* profile is designed to provide abstract views to support development of RTES (Real-Time Embedded Systems) as well as unify the various existing approaches through a common language, built by all the actors of the RTES community. The scope of MARTE is to cover all activities of RTES development that require specific constructs or references, not by replacing existing efficient solutions, but by mapping them to a reference and global meta-model. MARTE is structured as a set of sub-profiles that support design, analysis, the expression of timing characteristics, and the description of the execution model, the platform and the platform API model libraries.

Several industrial European projects have adopted the lightweight strategy. Such projects focus on real-time embedded systems, with special attention to the railway, automotive and aerospace application domains. Generally, the feedback from industry encourages us to pursue the direction of lightweight extensions. In the rest of this section, we will discuss two typical use cases: one for safety related systems development and one for the automotive domain.

2.2.1 A profile for Safety Analysis

Safety requirements play a crucial role in the railways, automotive and aerospace domains. In many cases, safety requirements have profound implications on the architecture of a system. As a result, the scientific community attempts to integrate safety requirements in the software development as early as possible. However, in the last decades, the requirements for safety and for real-time embedded system development have been accompanied by the use of heterogeneous methodologies and tools. In addition, and to complicate the integration effort, safety teams and system development teams are not the same. In the face of this rhapsodic scenario, two directions seem to be possible. The first is driven by tool integration, where each team develops its own model with its own tool. The issue is then how to proceed with the integration of the results or of the tools themselves. One advantage of this lies in the use of existing tools, that are tailored to the specific application they support. However, the late and often problematic integration implies that the changes in the architecture needed to take safety requirements into account force designers to redesign parts of the system well after the end of the specification phase. This creates long redesign cycles that adversely affect productivity and, in certain cases, correctness.

A different approach relies on the lightweight extension presented above. The IMOFIS industrial European project, which started in the middle of 2008, fits in this context [IMO]. The objective of this project is to define a development environment for safety-critical applications. The idea is to define a Conceptual Data Model (the top part of Figure 2) for safety in strict collaboration with, in this case, the safety team of railway and automotive industries. Concepts in this data model are drawn from an ontology given by the safety teams. Examples of concepts are that of a “hazard”, an “accident”, a “safety barrier” and so on. The language development then proceeds by first creating a profile starting from the Conceptual Data Model. This profile is then integrated with other pre-existing profiles, such as SysML and MARTE, in order to import their expressive capabilities. At the modeling level, designers specify information on model elements via a graphical interface.

On top of the language design technique described above, IMOFIS exploits a new conception of a profile, first introduced by S. Cook and then discussed in various papers [Sel04, And07]. A profile

is defined by a "family of related languages". In order to tailor a DSL to a given point of view (i.e., safety analysis, temporal analysis, and so on), the strategy of the IMOFIS project is to keep only suitable subsets of each pre-existing profile. This approach potentially reduces the possible semantic and syntactical conflicts between profiles. (See the discussion in Section 2.2.3). Such a strategy is in fact implicitly already adopted by different industrial European projects, e.g., MeMVaTEx, ATESST and Lambda [MeM, ATE, Lam].

2.2.2 An experience of specialization for the automotive domain

With the introduction of the AUTOSAR standard [Aut], many of the main actors in the automotive domain have stressed the need for solutions to support, in a way that is "as standard as possible", the first steps of the description of automotive application/function. To this end, the ATESST STREP - FP6 project [ATE] was launched to provide an architecture description language for the automotive domain. Thanks to UML profile mechanisms, it was possible, within two years and with a limited amount of resources, to both implement the language (EAST-ADL 2) upon an existing tool and align it with the AUTOSAR standard. With respect to UML, the EAST-ADL 2 profile brings the following innovations:

- the introduction of domain related vocabulary and concepts with a focus on function description (in place of software component) using communication events and mechanisms dedicated to the domain;
- the support of a layered development process that distinguishes among the levels dedicated to the vehicle, the analysis, the design, the implementation and the operation level that corresponds to the AUTOSAR execution infrastructure.

This language is currently being extended to support the description of non-functional properties (including safety, timing aspects and product line/variant definition) in new projects such as ATESST 2 STREP - FP7 and EDONA of System@tic Paris Région cluster [ATE, EDO]. Following the same strategy as in IMOFIS, the new language is defined by importing capabilities from existing meta-models and profiles. For instance, UML 2 is used for all the basic concepts, SysML for requirements and functional blocks with communication ports and MARTE for the timing aspect, platform allocation and refined communication ports. Finally, a UML profile for AUTOSAR, provided by the AUTOSAR consortium, is used to describe the target architecture of the applications.

Similar work based on profile composition has been done for the sole purpose of requirement modeling and traceability in the ANR MeMVaTEx project [MeM, ABB+08]. MeMVaTEx combines three profiles: MARTE for temporal analysis, SysML for requirements and EAST-ADL for the description of the architecture.

2.2.3 A common need for formal execution descriptions into profile definition

Projects such as IMOFIS and ATESST emphasize the importance of combining multiple profiles. For example, one may want to benefit from the SysML mechanisms for requirement specification, as well as MARTE annotations and concepts for timing analysis or execution resource management. However, the fact that a DSL defined as a profile usually targets a particular application domain does not imply that multiple profiles will necessarily address orthogonal concerns. Concepts and rules defined in these profiles may therefore overlap, potentially raising consistency issues when they are combined in a given model.

For example, SysML and MARTE define their own sets of concepts and modeling rules for component-oriented design, with their own informal descriptions of execution and interaction semantics associated with SysML blocks or MARTE components. More in general, the problem is not just the composition of profiles from a structural standpoint. As described in Section 2.1, tools like Papyrus already provide support for that. Instead, it concerns the ability to integrate execution or behavioral semantic descriptions into profile definitions. The formalism used to describe the encapsulated semantics should be standard, in order to ease the process of combining the execution semantics of multiple profiles. One step in this direction can be seen in the definition of the new OMG standard on the executable semantics of a foundational UML subset [OMGb] (which defines operational semantics for a UML subset called fUML) and in preliminary results on the possibility to encapsulate operational semantics descriptions into stereotype definitions using UML [CMTG07].

We conclude by recalling the aforementioned lightweight approach. In the European scenario, the lightweight approach is one answer to the problems that arise from the continuing integration in one system of various functionalities of increasing complexity. In this context, integration and combination of various UML profiles will play a crucial role in the near future and different large industrial European research projects already require such mechanisms and advances on this topic [ATE, IMO, Int, Lam, MeM, EDO, Gen, CES, com].

One of the main challenges with combining several UML profiles is to ensure the consistency of the resulting modeling language. Lagarde et al. [LET+08] underline that this research topic requires both new software engineering methods to design good profiles, but also specific tools for checking profiles consistency, or for supporting user defined compatibility and composition rules.

3 Meta-modeling in SPEEDS

In this section we give an overview and perspective on the application of modeling and meta-modeling techniques in the context of the SPEEDS European project [spe]. SPEEDS is a concerted effort to define the new generation of end-to-end methodologies, processes and supporting tools for safety-critical embedded system design. One of the technical pillars of the SPEEDS approach is the definition of a semantic-based modeling method that supports the construction of complex embedded systems by composing heterogeneous subsystems, and that enables the sound integration of existing and new tools. At the basis of this approach is the definition of a "heterogeneous rich-component" model (HRC), able to represent functional as well as architectural abstractions, such as timing, safety and other non-functional performance metrics. These different viewpoints can be developed separately in the model, and then integrated and evaluated together in order to derive the most efficient component-based implementation of the system [BCP09, BCF+08].

In the following, we will first discuss the methodological requirements for SPEEDS, the basic principles behind the design of the HRC model and its underlying semantics. We then illustrate applications in the area of heterogeneous simulation and safety analysis.

3.1 Methodological Requirements for SPEEDS

The SPEEDS methodological requirements are the drivers behind the choices made for the design of the HRC model, which is targeted to the domain of embedded and reactive systems. The first characteristic to be considered, as discussed in the introduction and earlier in the paper, is that concurrent development of systems occurs by different teams that, besides the functionality, focus on different aspects or view-points, such as *safety* or *reliability*, *timing* (e.g., in Time-Triggered development [Kop98]), *memory management* to ensure segregation of subsystems, and *energy*. Each

of these aspects requires specific frameworks and tools for their analysis and design. Yet, they are not totally independent but rather interact, in ways that are sometimes non-obvious. In HRC, these different aspects are expressed in the same model, which is specialized to the different cases by ignoring the non-essential features. This approach is justified by the interchange nature of HRC, which is used as an integration model in the SPEEDS infrastructure. In all cases, in fact, the underlying composition semantics is the same, which makes the integration of the aspects easier.

Even under the same interaction model, particular attention must be placed on developing the right operators for composition. This is especially true with view-points and the requirements that they express. Early requirement capture today still relies, for the most part, on organized textual descriptions, with little formal support, if any. Moving ahead can be achieved by formalizing the notation used for individual requirements, by relying, for example, on so-called semi-formal languages [Bur97] or on graphical scenario languages [DH01, IT99]. Similarly, HRC can be used as an underlying formal description for system requirements. No matter what model is used, the key point is that several requirements may be attached to *the same* component. This changes the nature of the interaction, which is not between parallel components exchanging data, but rather between interrelated specifications which jointly contribute to component specification. Consequently, different operators are needed when composing view-points and components.

Similar problems arise during system integration. One important prerequisite of the SPEEDS methodology is that designers should be able to develop subsystems in isolation, and then integrate them correctly. This is achieved in HRC by including, as part of the component specification, the needed information regarding the possible contexts of use. This way, one can establish the responsibilities of suppliers and integrators, by explicitly expressing the assumptions under which a component is supposed to be used. This separation between the assumption and the specification, or promise, of the component, which is implemented in the form of design *contracts*, is, in fact, one of the distinguishing features of the HRC model.

3.2 SPEEDS principles

Since a number of years, several efforts have been undertaken to interconnect design and analysis tools via a common semantic level format. Of particular interest in our context are the WOODDES [Conb] and OMEGA [Cona] projects. In these projects, the chosen user level design notation was a UML profile for real-time component systems with a well-defined operational semantics. This was then expressed in terms of a simpler formalism, based on communicating extended state machines, enriched with timing constraints, which can be easily imported into different verification and analysis tools. In OMEGA in particular, an explicit effort was made to preserve as much of the original structuring concepts as useful for obtaining efficient analysis. Nevertheless, the translation process required the addition of extra components and/or the enrichment of both the interfaces and the behavior of existing components. This lead to the well-known problem that users were unable to interpret the analysis results, despite the significant effort dedicated to provide this feedback in terms of the original user concepts, whenever possible.

Also in the SPEEDS framework, shown in Figure 3, the integration of a set of modeling and analysis tools is based on the use of a common intermediate format, HRC, defined in the form of a meta-model. However, several original ideas have been integrated in the SPEEDS HRC meta-model. For instance, SPEEDS allows several modeling tools to contribute to the global model of a given system and to use the component-code generated by some modeling tool as their own behavior through the so-called hosted simulation (see Section 3.5). In addition, HRC has two distinguishing features with respect to previous semantic level tool exchange formalisms. It provides *additional* structuring constructs with respect to the user level modeling languages of the user tools, and it is

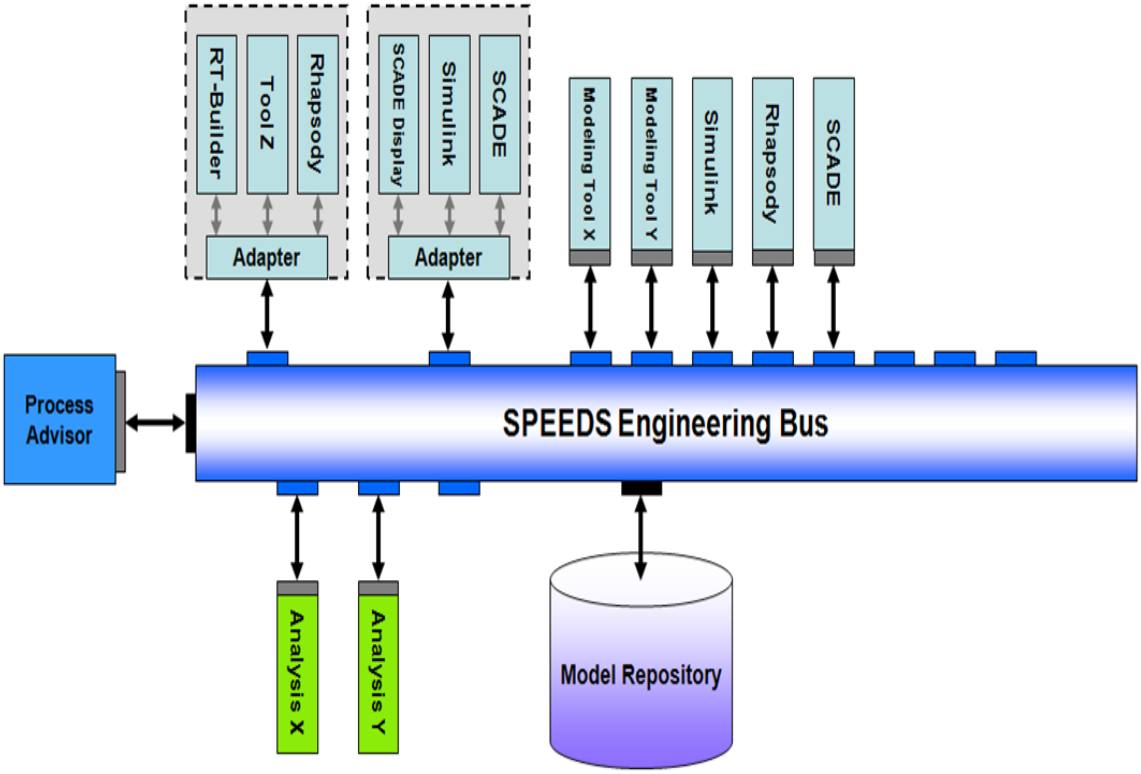


Figure 3: Tool integration via the SPEEDS bus and meta-model

defined in a *layered* manner, corresponding to different usages.

HRC supports an expressive representation that rests on a semantically well-founded formalism used to describe the abstract behaviors of components or the environment of systems of very different nature — software systems, but also physical subsystems or models of human behaviors, etc. Such behaviors are described in a form usable by a wide range of analysis tools, as compositions of extended automata describing constraints on the discrete and continuous behavior. For these reasons, on top of traditional static component interfaces that only define the interaction points of components, richer information is exposed to designers, in the form of a set of *contracts*. Associated to a component, contracts abstract constraints on the component and its environment behaviors in the form of assumption-promise pairs. The interpretation is that in an environment fulfilling the constraint defined by the assumption, the component offers a behavior that satisfies the constraint expressed by the promise. The information in contracts can be used for analysis before any sort of model of the components exists, and then used throughout the design cycle for verifying or testing the correctness of abstract models or actual implementations. The meta-model definition of contracts in HRC is shown in Figure 4.

The HRC meta-model consists of three levels of increasing abstraction, as shown in Figure 5. Level $L1$ defines the concepts that are handled by most analysis tools and has been designed for efficient analysis. Different composition modes (asynchronous and synchronous) are expressed here by means of a rich set of connectors for which there exist well founded theoretical results [BS07]

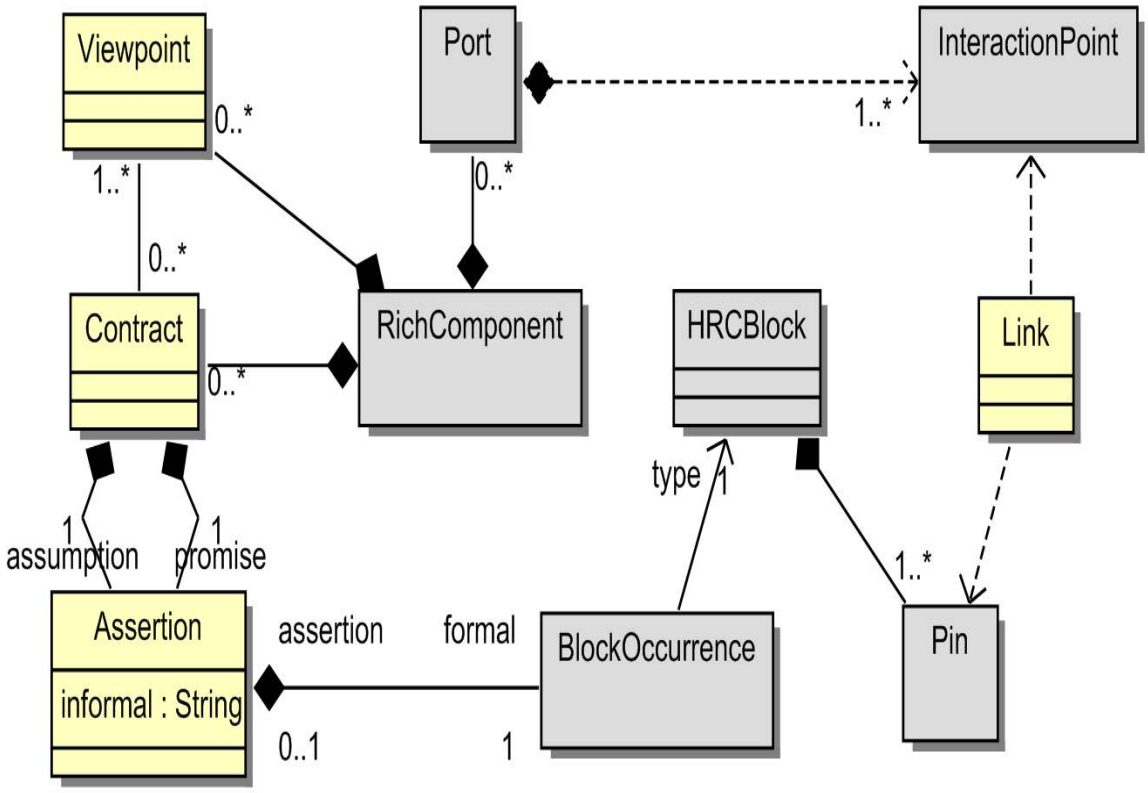


Figure 4: SPEEDS meta model: components have contracts

that can be exploited for making compositional verification efficient. Level $L1$ is built on top of level $L0$, which provides the basic semantic notions of the meta-model. All $L1$ concepts can be mapped to this semantic layer, although the translation may in certain cases introduce a degree of syntactic or behavior explosion. For this reason, several analysis tools work at layer $L1$. The synchronous $L0$-layer has been introduced to provide the underlying interaction mechanism based on synchronous models [BCE+03] and is the exchange model for those analysis tools that are tailored towards the verification of synchronous descriptions. Above $L1$, level $L2$ is used as a bridge to user level concepts. This way, notions that are specific to certain domains need not be directly expressed in terms of $L0$ or $L1$ descriptions. Instead, they are mapped to some intermediate concept — often a generalization of the original — that avoids losing the original structure. These $L2$ concepts are then defined as mappings to the lower layers. Therefore, each $L1$ or $L0$ enabled tool can handle any user level validation problem with some efficiency, but may also choose to handle some of them more efficiently by implementing specific methods tailored towards the $L2$ layer.

3.3 Comparison with other standards

A relevant question is how the HRC model relates to other standards, and why a standalone meta-model was defined rather than a profile. The standards which we had considered as potential alternatives are the SysML and the MARTE profiles (see Section 2.2). Initially, we intended to

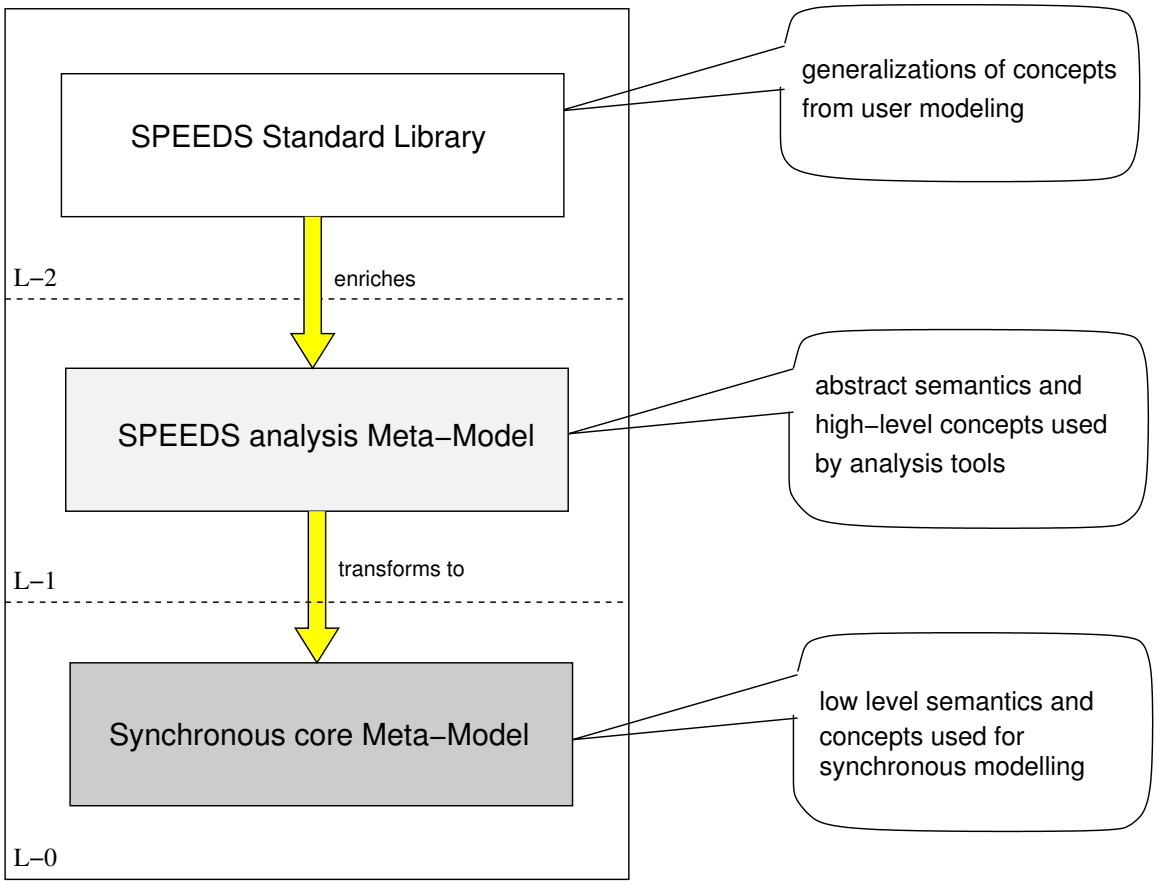


Figure 5: The SPEEDS layered meta-model

adapt SysML to our needs, since it embodies a general modeling approach that is familiar to most users, and already in use by some of them. Its main shortcomings, however, are the absence of certain structuring concepts, such as rich interfaces, contracts and connectors, and our stringent requirements in terms of underlying semantics. SysML, in fact, has no precisely defined semantics, but it specifies that the interaction between components should be asynchronous. As an intermediate representation, and to support tools like SCADE or Simulink, our model had to support some means of expressing a synchronous execution model. MARTE, on the other hand, does allow specifying requirements in a generalized synchronous fashion, but it cannot easily be used to represent SCADE models structurally, other than by using some keyword to tag a component as "synchronous". In addition, the approach to requirement expression is radically different: while HRC provides a simple and expressive formalism for constraints, MARTE introduces a number of predefined attributes for expressing standard non-functional constraints. These are useful, but each tool has to separately provide their meaning. The layered definition of HRC, on the other hand, allows introducing these concepts by giving them a meaning in terms of the lower levels. Thus, tools that can interface to the HRC model would automatically inherit the definition of the high level concepts.

The choice of defining a standalone meta-model has several motivations. SysML is already a profile for UML and contains a tremendous amount of features which were not required and

not simple to just eliminate. We also deemed that it was not convenient to have the important concepts, in particular rich components, contracts and the rich set of $L1$ connectors, as stereotypes. Another motivation was to keep a certain independence from the evolution of a standard as many transformations from and to a variety of other formats depend on the HRC meta-model. Also, as HRC is only an intermediate representation used in tools, and the user sees models only either in terms of some modeling tool or in terms of some analysis or code generation tools, there is no strong need for HRC graphical editors, which diminished the pressure for aligning with UML.

3.4 SPEEDS Mathematical Model

An HRC model is the result of the interplay of several different elements. Rich components are characterized by contracts, which, in turn, are expressed as pairs of assumptions and promises. In this section, we provide an intuitive understanding of their relationships, illustrating the concepts using a notation based on set theory. More details can be found in [BCP07, BFM+08].

A *component* M (typically an implementation of a rich component) consists of a set of ports and variables (in the following, for simplicity, we will refer only to ports) and of a set of behaviors which assign a history of "values" to ports. Behaviors can be represented in various ways, such as (hybrid) automata or as the set of corresponding sequences of values or events. Here, we consider a component as the set of its possible runs. Components can be more or less specific. We say that a component M *refines* a component E whenever they are defined over the same set of ports and all the behaviors of M are also behaviors of E, i.e., when $M \subseteq E$.

We represent properties of components, or *assertions*, as the set components that satisfy it. Exploiting refinement, an assertion E is equal to its largest satisfying component. A *contract* C for a rich component is a pair (A, G) of assertions, where A corresponds to the assumption, and G to the promise. Assertion A and its refinements are the acceptable contexts (or environments) under which the rich component might be used; conversely, G represents the possible behaviors of the rich component under those contexts. A component *satisfies* a contract whenever it satisfies its promise, subject to the assumption. This relation of *refinement under context* can be formally expressed by checking if the composition of a component with the assumptions refines the composition between the promises and the assumptions. Formally, $M \cap A \subseteq G \cap A$. We write $M \models C$ when M satisfies a contract C.

Substitutability, or *dominance*, is the key concept of our contract theory. We say that contract C dominates contract C' whenever the components that satisfy C also satisfy C' under the same or an extended set of contexts. In other words, C can be substituted for C' so that dominance corresponds to a notion of refinement for contracts. Intuitively, dominance is ensured by relaxing assumptions and contextually reinforcing the promises. Formally, we say that $C = (A, G)$ *dominates* $C' = (A', G')$, written $C \preceq C'$, if and only if $A \supseteq A'$ and $G \subseteq G'$.

The semantics of composition of different view-points for the same component corresponds to an operation of *conjunction* of contracts, obtained as the greatest lower bound of the order induced by contract dominance. For contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$, conjunction is obtained by extending the assumptions to all acceptable contexts, and restricting the promises to the guaranteed behaviors. Formally,

$$C_1 \sqcap C_2 \quad = \quad (A_1 \cup A_2, G_1 \cap G_2).$$

Parallel composition of contracts is also needed to formalize the combination of rich components. The parallel composition of two contract must first guarantee the promises of both contracts. Second, the environment should satisfy both assumptions, except that part of the assumptions of a component

is discharged by the promise of the other component. This concurrent strengthening and weakening of assumptions can be represented as

$$C_1 \| C_2 = (A, G) \text{ where } \begin{cases} A & = & (A_1 \cap A_2) \cup \neg(G_1 \cap G_2), \\ G & = & (G_1 \cap G_2) \end{cases}$$

The availability of different composition operators makes it possible to develop flexible system integration flows that focus alternatively on composition of components and of view-points, or combinations of the two.

3.5 Hosted simulation

A common way to validate system models is by using simulation. This approach is however problematic when the system consists of components designed in different tools. The SPEEDS project takes a *hosted simulation* approach to resolve this challenge: the HRC components are exported to a standard format, and then imported to the composed system that can be simulated by a single simulation tool.

This approach has several advantages compared to the standard co-simulation approach. First, only one simulation tool is needed, where all animated views are available. In addition, there is no overhead associated with the use of a message bus and with the coordination between different simulators. Finally, one can simulate the interaction with an HRC component exported from a design tool that has no simulation capabilities. The main drawback of this approach is that one can monitor the interactions between the components only, but has no visibility on the internals of every component.

Hosted simulation works by exporting an executable model of a component from a modeling tool, wrapped inside an adapter that implements the hosted simulation API and protocol. The API includes functions dedicated to setting and reading values on ports, executing a computation step, and advancing the computation time. The protocol determines the sequence of operations that must be performed for a correct model evaluation, and coordinates the interaction between the different components.

The hosted simulation protocol proceeds through iterations that are substantially composed of two nested cycles that compute a fixed point. The inner cycle is a *computation* phase in which components are executed in an arbitrary order to determine the value on ports which are initially undefined. During this phase, components do not update their internal state and do not exchange data. Instead, they iteratively recompute their output based on the additional available input. When the system is stable, the components update their state and exchanged data in the *commit* phase. In addition, they provide a time stamp corresponding to the availability of their next event, which is used to compute the time advance of the simulation. The outer cycle simply iterates these two phases to make the simulation progress until termination.

Hosted simulation works transparently through the SPEEDS infrastructure by taking advantage of the common HRC meta-model. The HRC component being exchanged between the tools has two parts. The first is a description of the component based on HRC, and contains the interfaces and abstractions describing the component in terms of the meta-model. The second is an implementation generated by the tool that exported the component, either as a set of source files or as a compiled DLL accompanied with header files.

3.6 Functional safety concepts

Functional safety can be expressed in HRC by specifying safety goals by contracts. We assume we have identified the safety goal for the system, and we need to derive functional safety requirements

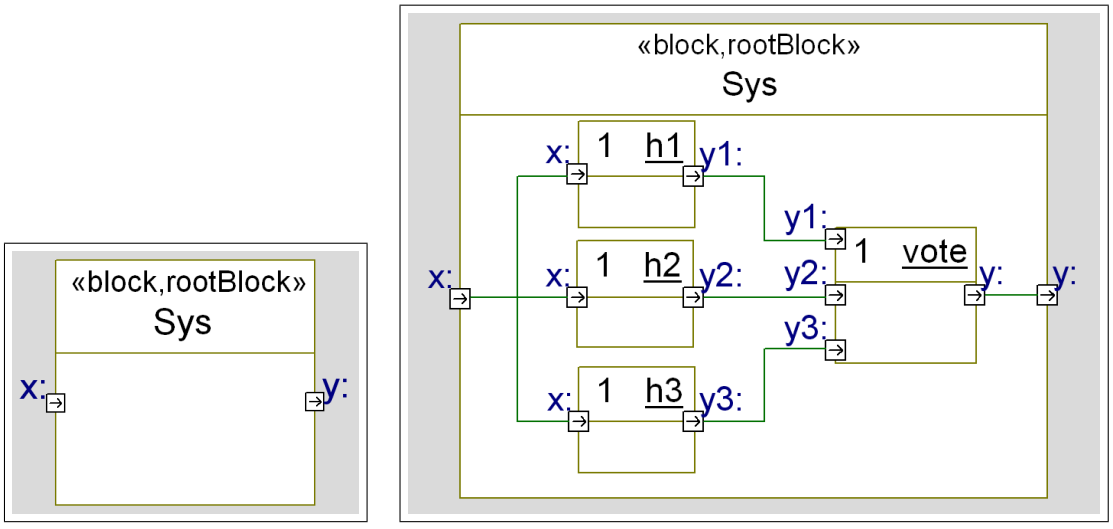
for the subsystems. The allocation of functional safety requirements to subsystems is done by associating the corresponding contracts with the subcomponents of the system. By using contracts, we ensure that for each subcomponent the safety requirements are structured into a promise for the safety function provided by the component and an assumption describing the context in which the function is (safely) provided. This context is either determined by the system environment or by other components. Thus, either the context should already be contained in the safety goal, or it should by traceable to (promises of) other components. In fact, in many situations, only the environment and the other components *together* ensure that the assumptions underlying a particular safety function of a component hold. Thus, there usually exists a non-trivial dependency structure between the individual contracts of the components on one side and between these and the contracts of the system on the other side.

The requirement decomposition can be validated by a dominance check: if the check succeeds, the requirement decomposition complies with the original safety goal. If it fails, counter-examples are produced that satisfy all allocated requirements, and do not (fully) satisfy the safety goal defined for the system. These counter-examples pinpoint flaws in the safety concept and provide effective guidance on how to re-define or adapt the safety concept.

To illustrate this, we consider the ISO CD 26262 which allows exploiting redundancy in a process called *ASIL Decomposition*. The first step to validate a simple redundancy concept is to identify the function to be implemented redundantly. The typical corresponding contract is

$$C_{Sys} = (\langle \text{context} \rangle, y = f(x)),$$

where $\langle \text{context} \rangle$ is some assertion about the behavior of the environment *Sys* operates in, and f is the function to be implemented. The underlying black box view is given in Figure 6.a as a SysML block diagram. A typical redundant implementation refines this black box view as shown in Figure 6.b. There are three redundant components H_1, H_2 and H_3 (each implementing the function f), and a



a) Blackbox view b) Redundant implementation

Figure 6: Blackbox view and redundant implementation

majority voter *Vote* that checks for agreement of two output values. For these we have contracts

$$\begin{aligned} C_{H_1} &= (\langle \text{context} \rangle, y_1 = f(x)), \\ C_{H_2} &= (\langle \text{context} \rangle, y_2 = f(x)), \\ C_{H_3} &= (\langle \text{context} \rangle, y_3 = f(x)), \\ C_{Vote} &= (\text{true}, (y = y_1 \wedge y_1 = y_2) \vee (y = y_2 \wedge y_2 = y_3) \vee (y = y_3 \wedge y_3 = y_1)). \end{aligned}$$

Three dominance checks yield that

$$C_{H_i} \parallel C_{H_j} \leq C_{Sys} \quad i, j \in \{1, 2, 3\},\ i \neq j.$$

This demonstrates that contracts of only two components are required to ensure that the system contract holds.

4 Conclusions

Model-based design methodologies are increasingly finding acceptance in the development of electronics systems, thanks to their flexibility and the availability of tools for their analysis and implementation. Meta-modeling techniques have consequently emerged to organize the landscape of models and provide theories and methods for the development of coordinated representations that are more suitable for the heterogeneous environment in which modern embedded systems operate. In this paper we have provided an overview of several efforts which are ongoing in Europe in the area of modeling and meta-modeling, with an emphasis on language design, applications and tools.

A number of new initiatives and funded projects show that much much work is still needed, both from a tool support and from a fundamental understanding point of view. Of particular interest are the studies that extend the heterogeneous model integration from the structural to the semantics point of view [CMTG07, com] while ensuring consistency. The application of compositional techniques across domains, together with joint performance evaluation and design will be the building blocks for new methodologies able to address and solve the design problems in the emerging area of Cyber Physical Systems.

Acknowledgments

The authors would like to thank all their colleagues who contribute to the projects presented here and would like to acknowledge the contributions of the European Union for funding many of the initiatives presented in this paper.

References

[ABB+08] A. Albinet, S. Begoc, J.-L. Boulanger, O. Casse, I. Dal, H. Dubois, F. Lakhal, Y. Sorel D. Louar, M.-A. Peraldi-Frati, and Q.-D. Van. The MeMVaTEx methodology: from requirements to models in automotive application design. In *4th European Congress ERTS Embedded Real Time Software.*, 2008.

[And07] C. André. Time Modeling in MARTE. In *FDL'07 Forum on specification and Design Languages*, 2007.

[ATE] ATESST Project. Advancing Traffic Efficiency and Safety through Software Technology. ATESST STREP - FP6 project. `http://www.atesst.org`.

[Aut] Autosar Project. Automotive Open System Architecture (Aut@sar). `http://www.autosar.org`.

[BCE+03] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages twelve years later. *Proceedings of the IEEE*, 91(1), January 2003.

[BCF+08] Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. Multiple viewpoint contract-based specification and design. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Formal Methods for Components and Objects, 6^{th} International Symposium (FMCO 2007), Amsterdam, The Netherlands, October 24–26, 2007, Revised Papers*, volume 5382 of *Lecture Notes in Computer Science*, pages 200–225. Springer Verlag Berlin Heidelberg, 2008.

[BCP07] Albert Benveniste, Benoît Caillaud, and Roberto Passerone. A generic model of contracts for embedded systems. Rapport de recherche 6214, Institut National de Recherche en Informatique et en Automatique, June 2007.

[BCP09] Albert Benveniste, Benoît Caillaud, and Roberto Passerone. Multi-viewpoint state machines for rich component models. In *Model-Based Design of Heterogeneous Systems*. CRC Press, Taylor and Francis Group, Boca Raton, London, New York, 2009. In preparation.

[BFM+08] Luca Benvenuti, Alberto Ferrari, Leonardo Mangeruca, Emanuele Mazzi, Roberto Passerone, and Christos Sofronis. A contract-based formalism for the specification of heterogeneous systems. In *Proceedings of the Forum on Specification & Design Languages (FDL'08)*, Stuttgart, Germany, September 23–25, 2008.

[BS07] Simon Bliudze and Joseph Sifakis. The algebra of connectors: structuring interaction in bip. In *7th ACM & IEEE International conference on Embedded software, EMSOFT 2007, September 30 - October 3, 2007, Salzburg, Austria*, pages 11–20. ACM, 2007.

[Bur97] J. F. M. Burg. *Linguistic instruments in requirements engineering*. IOS Press, 1997.

[CES] CESAR Project. Cost-effective Small AiRcraft. `http://www.cesar-project.eu`.

[CMTG07] A. Cuccuru, C. Mraidha, F. Terrier, and S. Gérard. Enhancing UML Extensions with Operational Semantics - Behaviored Profiles with Templates. In *10th International Conference on Model Driven Engineering Languages and Systems (MODELS - 2007)*, 2007.

[com] COMponent-Based Embedded Systems design Techniques, IST STREP 215543. `http://www.combest.eu`.

[Cona] OMEGA Consortium. Ist-omega project webpage. `http://www-omega.imag.fr`.

[Conb] WOODDES Consortium. Ait-wooddes project webpage. `http://wooddes.intranet.gr/`.

[DH01] Werner Damm and David Harel. Lscs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[EDO] EDONA Project. Environnements de Développement Ouverts aux Normes de l'Automobile. `http://www.edona.org`.

[ESCG08] H. Espinoza, B. Selic, D. Cancila, and S. Gérard. Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems. Technical report, Center of Nuclear Energy (CEA - LIST), 2008.

[Esp07] H. Espinoza. *An Integrated Model-Driven Framework for pecifying and Analyzing Non-Functional Properties of Real-Time Systems*. PhD thesis, CEA LIST, 2007.

[Gen] Genesys Project. GENeric Embedded SYStem Platform. `http://www.genesys-platform.eu/`.

[GS08] S. Gérard and B. Selic. The UML MARTE Standardized Profile. In *17th IFAC World Congress, Volume 17, Part 1*, 2008.

[IMO] IMOFIS Project. Ingénierie des MOdèle FonctIons Sécuritaires. `http://www.imofis.org/`.

[Int] INTERESTED FP7 IP: Interoperable embedded systems Tool-chain for enhanced rapid design, prototyping and code generation. `www.interested-ip.eu`.

[IT99] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, September 1999.

[Kop98] Hermann Kopetz. The time-triggered model of computation. In *Proc. of the 19th IEEE Real-Time Systems Symposium*, pages 168–177, 1998.

[Lam] Lambda Project. Lambda Libraries for Applying Model Based Development Approaches. `http://www.usine-logicielle.org/lambda/index\_FR.html`.

[LET+08] F. Lagarde, H. Espinoza, F. Terrier, C. André, and S. Gérard. Leveraging Patterns on Domain Models to Improve UML Profile Definition. In *FASE 2008*, 2008.

[LIS] LISE. Laboratoire d'ingénierie dirigée par les moèles pour les systèmes embarqués (LISE), head leader: F. Terrier. LISE is part of CEA LIST: Commissariat à l'énergie atomique, Département des technologies des systèmes intelligents Service outils logiciels.

[MeM] MeMVATEX French Project. Méthodologie pour la Modélisation, laValidation et la Tracabilité des Exigences (MeMVaTEx). `http://www.memvatex.org/`.

[OMGa] OMG. Object Management Group. `http://www.omg.org/`.

[OMGb] OMG. Semantics of a Foundational Subset for Executable UML Models (Beta 1). `http://www.omg.org/spec/FUML/1.0/Beta1`.

[OMGc] OMG. Systems Modeling Language SysML. `www.sysml.org`.

[OMGd] OMG. UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2. `www.omgmarte.org`.

[OMGe] OMG. Unified Modeling Language UML Resource Page. `www.uml.org`.

[Pap] Papyrus. Open Source Tool for Graphical UML2 Modelling. `www.papyrusuml.org`.

[Rat] Rational Software Architecture (RSA) . `www.ibm.com/developerworks/rational/library/05/510\_svc`.

[RBB+09] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, and Roberto Passerone. Why are modalities good for Interface Theories? In *Proceedings of the Ninth International Conference on Application of Concurrency to System Design (ACSD09)*, Augsburg, Germany, July 1–3, 2009.

[Sch06] D. Schmidt. Model-driven engineering. *IEEE Computer*, pages 25–31, February 2006.

[Sel] B. Selic. From Model-Driven Development to Model-Driven Engineering. Keynote talk at ECRTS'07. `http://feanor.sssup.it/ecrts07/keynotes/k1-selic.pdf`.

[Sel04] B. Selic. On the semantic foundations of standard UML 2.0. In *SFM-RT*, pages 181–199. LNCS, 2004.

[spe] SPEculative and Exploratory Design in Systems Engineering, IP Contract No. 033471. `http://www.speeds.eu.com`.

[SV02] Alberto L. Sangiovanni-Vincentelli. Defining platform-based design. *EEdesign*, February 2002.

[TG06] F. Terrier and S. Gérard. Model-driven engineering and prototyping of real time embedded applications. In *DIPES conference*, 2006.