Metamodels in Europe: Languages, Tools, and Applications

Roberto Passerone
University of Trento

Imene Ben Hafaiedh and Susanne Graf
Verimag

Albert Benveniste
INRIA Rennes

**Daniela Cancila, Arnaud Cuccuru,
Sébastien Gérard, and Francois Terrier**
CEA LIST

Werner Damm
Oldenburg University

Alberto Ferrari and Leonardo Mangeruca
Parades

Bernhard Josko and Thomas Peikenkamp
OFFIS

Alberto Sangiovanni-Vincentelli
University of California, Berkeley

Editor's note:
This article provides an overview of current efforts in Europe for using metamodeling in the integrated development of critical systems such as automotive electronics. It distinguishes between lightweight versus heavyweight approaches, surveys a number of related current European projects, and gives details about the Speeds project to illustrate the role of metamodeling-driven system engineering.

—Sandeep Shukla, Virginia Tech

■ **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, reducing the complexity of its representation and manipulation and increasing productivity. This process has been shaped during the past few decades through conceptual representations and languages that are progressively more detached from a given system's implementation because they neglect details that are relevant only in the context of specific realizations. The converse process of refinement fills out those details with tools that can evaluate design alternatives through simulations and analysis and, when possible, through 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 paved the way to the development of the model-based approach to design (for example, see Terrier and Gérard[1]). 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, because 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, 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 designers 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 system description. The same degree of heterogeneity is present when the system description is partitioned into separate orthogonal aspects, or viewpoints. In this case, the fragmentation is at the component level and must be resolved by resorting to appropriate combinations of techniques that account for specification interdependencies.[2]

In this context, researchers have taken a step back and have begun 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 these goals, they used the very same modeling techniques that had proven successful in design to construct models of models, or *metamodels*. These metamodels have quickly been embraced by methodologies such as the *model-driven architecture* (MDA) and *platform-based design* (PBD).[3-5]

In this article, we review the role that models and metamodels have played and are playing in several research projects across Europe. Accordingly, we discuss language design techniques and their use in several industrial applications. We also describe the modeling principles and the infrastructure underlying the Speculative and Exploratory Design in System Engineering (Speeds) European project, and highlight the way metamodeling techniques have helped its implementation and applications.

Language design strategies

Embedded-systems development needs to provide global solutions for reconciling three conflicting concerns: enrichment and refinement of system functionalities, reduction of time to market and production costs, and compliance with nonfunctional requirements.[1] To fulfill these objectives, both academic and industrial communities have been promoting for more than a decade design approaches and methodologies relying on *model-based engineering*.[3,4] MBE methodologies address different problem-related concerns such as model transformations, model repositories, and specific modeling languages.

Metamodeling techniques are the basis for most research efforts in the state of the art in these different areas. A metamodel is the result of capturing concepts and rules of a specific modeling language via more or less formal means.[6] In this context, we can say that a model conforms to a metamodel if the model respects the set of modeling rules defined in the metamodel

("just like a well-formed program conforms to the grammar of the programming language in which it is written").[6] The example shown in Figure 1 illustrates this.

At the figure's top, we show the graphical definition of the metamodel of a simple modeling language for interfaces with operations, where rectangles represent objects, and arrows (with their different heads) represent rectangles' relationships. An *Interface* is a *Named-Element* (arrow pointing up), which can have a number of associated *Operations* (arrow pointing right). These, in turn, may take typed parameters with a specified direction and may return a typed value. Designers are not concerned with the metamodel definition, and instead use graphical tools to represent their specification of an interface, as shown in the bottom left of the figure, where interface *VehicleControllerInterface* has been defined to have two operations. 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 metamodel definition.

There are two strategies for using metamodeling techniques in the design of *domain-specific languages*: a heavyweight and a lightweight approach.

Heavyweight vs. lightweight design

Figure 2 is an overview of the heavyweight and lightweight approaches for defining specific domain languages and their impact on underlying tool architectures. The figure shows the steps and the models involved in the creation of languages and tools corresponding to certain concepts of interest—related in this case to the domain of real-time systems.

In the heavyweight variant, outlined in the left-hand side of Figure 2, designers create a new domain-specific language (DSL) for modeling the domain of interest, characterized by a fully dedicated metamodel. This way, the DSL is optimally suited to the problem at hand.[7] Because 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."[7] This translates, in the realm of tools, to more effort required to obtain an integrated and consistent tool chain, which must be supported by a specific interoperability infrastructure, as illustrated by the bottom-left "Realm of tool" box in Figure 2.

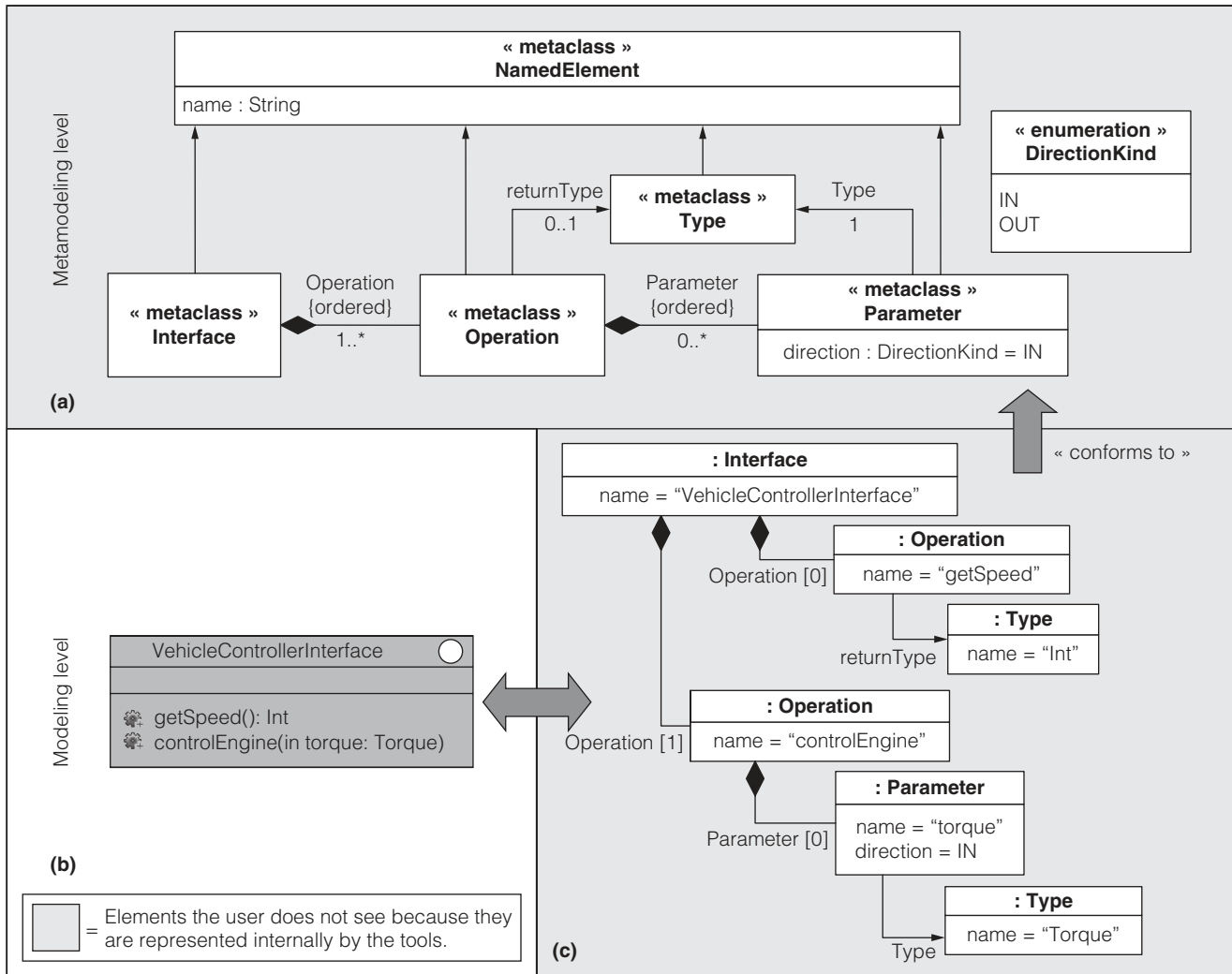The lightweight variant, outlined in Figure 2b, relies on the extension of an existing metamodel.

Figure 1. Language definition and use for a vehicle controller interface specification in model-based engineering form: metamodel definition (a), user-defined model (b), and model repository (c).

This metamodel typically captures the modeling concepts and rules of a more general-purpose modeling language, such as the well-accepted Unified Modeling Language.[8] In the context of UML, this mechanism of lightweight extension is called a *profile*. In our example, the Modeling and Analysis of Real Time Embedded (Marte) systems profile is used to capture the concepts related to real-time systems. Each extension of an element from the UML metamodel is formally captured by a *stereotype* concept. Each stereotype definition can be associated with properties and/or modeling constraints that make sense for the domain targeted by the profile. Stereotypes are then manipulated at the modeling level as annotations on model elements, so that various semiautomatic tools can access the information captured

by the profile (for example, for code generation, verification, or domain-specific analysis). The advantage of this approach is that existing metamodels can be reused and specialized.

The most difficult part when defining a lightweight extension is to determine the most suitable elements of the metamodel that must be extended (that is, the metaclasses for which stereotypes must be defined). This task typically requires a deep knowledge of the metamodel. However, once the profile has been defined, this approach lets designers specialize a general-purpose tool (such as Papyrus or RSA) at low cost.[9,10] 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
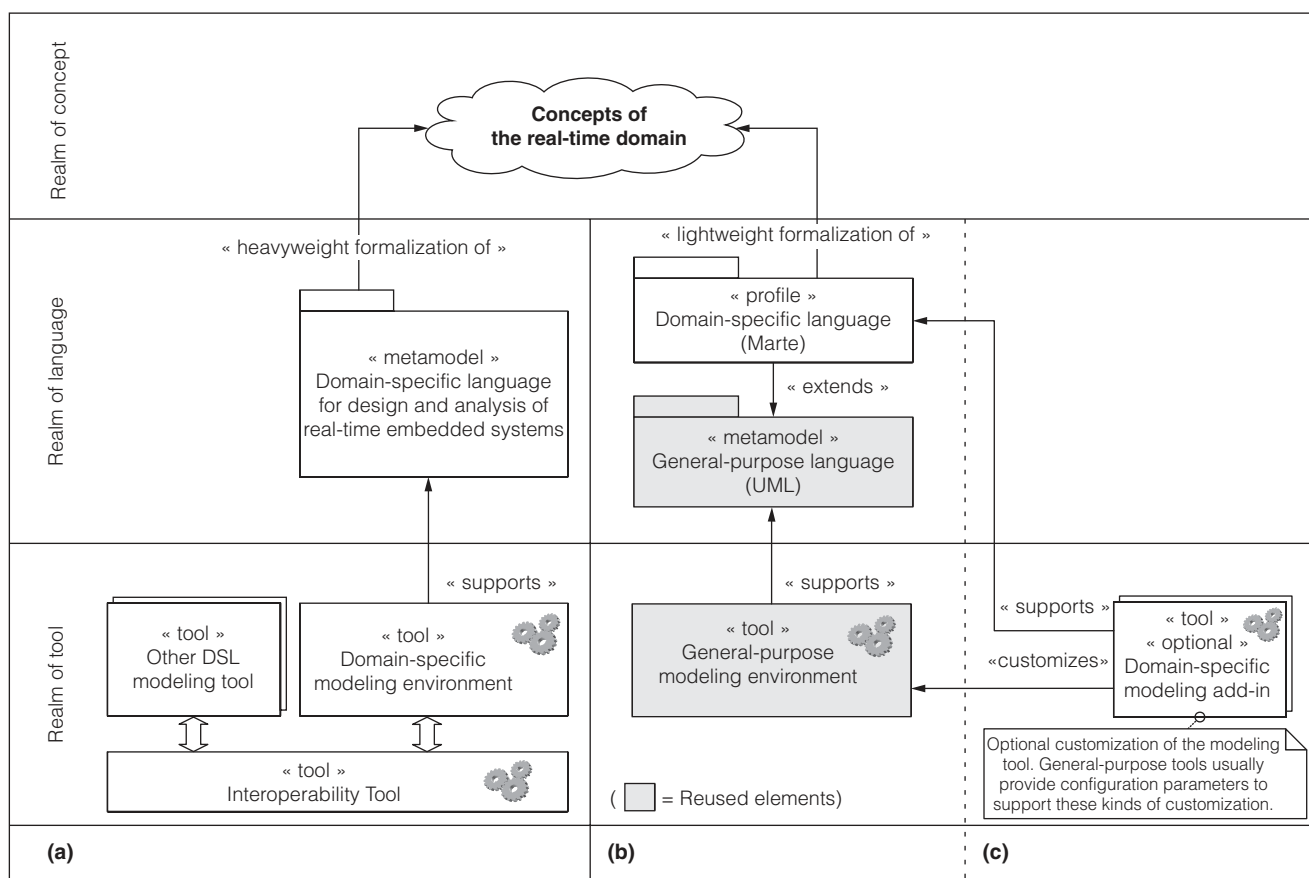
Figure 2. Uses of metamodeling techniques related to concepts of the real-time domain: heavyweight approaches (a), lightweight approaches with reused elements (b), and optional connection to domain-specific modeling add-ins (c).

integration in the form of optional plug-ins, as illustrated in the bottom-right side of Figure 2.

The heavyweight and lightweight variants address the problem of multidomain modeling capabilities differently. Because multiple profiles can be applied to a given model, it can be easier to integrate and combine DSLs for lightweight than for heavyweight approaches. A heavyweight approach requires that a dedicated metamodel and modeling environment be defined for each DSL, and a potentially complex interoperability tool might be required to ensure consistency and interaction between the different model "views." For example, in the Papyrus UML modeler developed at the LISE laboratory in France,[9,11] a designer can automatically import and apply several profiles in a given model and, hence, add information concerning multiple domains to preexisting model elements. Conversely, the heavyweight Speeds approach uses a single "rich" metamodel that takes the role of integration infrastructure and avoids the proliferation of several DSLs.

Profiles as DSL and industrial feedback

The lightweight extension mechanism has been supported by the embedded-systems industry through the development of two important profiles for this 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 limited to software-centric systems development. Instead, SysML addresses a wider understanding of system architectures and interactions with their environment, whether or not they are later realized through software. At the modeling level, two main aspects have been integrated: links with reference requirement documentation, and the capability to describe coarse-grained architectures supporting both discrete (such as messages, data, and material) and continuous (such as energy) interactions. All other aspects (behavioral descriptions, for example) are assumed to reuse largely existing UML constructs and semantics.

■ The Marte systems profile is designed to provide abstract views to support real-time embedded systems (RTES) development as well as unify the various existing approaches through a common language, built by all the actors of the RTES community. Marte's scope is to cover all RTES development activities that require specific constructs or references, not by replacing existing efficient solutions but by mapping them to a reference and global metamodel. Marte is structured as a set of subprofiles that support design, analysis, and the expression of timing characteristics; the subprofiles also support 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. As examples, we will discuss two typical use cases: one for safety-related systems development and one for the automotive domain.

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 for system architecture. As a result, the scientific community attempts to integrate safety requirements in software development as early as possible. However, in the past few decades, the requirements for safety and real-time embedded-system development have been accompanied by the use of heterogeneous methodologies and tools. In addition, safety teams and system development teams are not the same—thus complicating the integration effort. In such a scenario, two directions seem to be possible. The first is driven by tool integration, in which 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 approach lies in the use of existing tools, tailored to the specific application they support. However, the late and often problematic integration implies that the changes in the architecture needed to consider safety requirements 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 we have described. The Imofis industrial European project, which began in mid-2008, fits into this context.[12] The objective is to define a development environment for safety-critical applications. The idea behind Imofis is to define a conceptual data model for safety in strict collaboration with the safety teams of the railway and automotive industries. Concepts in this data model are drawn from an ontology given by the safety teams—concept examples include 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 preexisting profiles, such as SysML and Marte, to import their expressive capabilities. At the modeling level, designers specify information on model elements via a graphical interface.

In addition to the language design technique we have described, Imofis exploits a new conception of a profile,[13,14] which is defined by a "family" of related languages. To tailor a DSL to a given point of view (that is, to address safety analysis, temporal analysis, and so on), the Imofis project keeps only suitable subsets of each preexisting profile. This strategy potentially reduces the possible semantic and syntactical conflicts between profiles. In fact, such an approach is already implicitly adopted by different industrial European projects—for example, Memvatex (Methodology for Modeling, Validation, and Traceability of Requirements), Atesst (Advancing Traffic Efficiency and Safety through Software Technology), and Lambda (Libraries for Applying Model-Based Development Approaches).[15-17]

Specialization for the automotive domain. With the introduction of the Autosar standard,[18] which is an open-system software architecture, many in the automotive domain have stressed the need for solutions to provide standardized support for the first steps in describing an automotive application or function. To this end, the Atesst project was launched to provide an architecture description language for the automotive domain.[16] Thanks to UML profile mechanisms, it was possible, within two years and with limited resources, to both implement the language Electronic Architecture and Software Technology (EAST)-ADL 2 on an existing tool and align it with the Autosar standard. The EAST-ADL 2 profile features the following innovations:

■ the introduction of domain-related vocabulary and concepts with a focus on function descriptions

(in place of software components) by means of communication events and mechanisms dedicated to the domain; and

- the support of a layered development process that distinguishes between the levels dedicated to the vehicle, analysis, design, implementation, and operation level corresponding to the Autosar execution infrastructure.

The EAST-ADL 2 language is currently being extended to support the description of nonfunctional properties (including safety, timing aspects, and product line and variant definition) in new projects such as Atesst 2 and Edona.[16,19] Following the same strategy as in Imofis, the new language is defined by importing capabilities from existing metamodels and profiles. For instance, designers use UML 2 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, describes 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 Memvatex project.[15,20] Memvatex combines three profiles: Marte for temporal analysis, SysML for requirements, and EAST-ADL for the architecture description.

Defining formal execution descriptions. There is a common need for formal execution descriptions to be defined within profiles. Projects such as Imofis and Atesst emphasize the importance of combining multiple profiles. For example, designers might 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 generally, the problem is not only the composition of profiles from a structural standpoint; as we have described, tools like Papyrus already provide support for that. The real problem concerns the ability to integrate execution or behavioral semantic descriptions into profile definitions. The formalism for describing the encapsulated semantics should be standard, to ease the process of combining the execution semantics of multiple profiles. One step in this direction is evident from the definition of the new Object Management Group standard on executable semantics of a foundational UML subset (http://www.omg.org/spec/FUML), which defines operational semantics for a UML subset called fUML. Also, preliminary results on the possibility of encapsulating operational semantics descriptions into stereotype definitions using UML are promising.[21]

In many of the European projects, 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.[12,15-17,19,22-25] One of the main challenges with combining several UML profiles is to ensure the consistency of the resulting modeling language. Lagarde et al. have stressed that this research topic requires new software engineering methods to design good profiles, as well as specific tools for checking the consistency of profiles or for supporting user-defined compatibility and composition rules.[26]

Metamodeling in Speeds

The Speculative and Exploratory Design in System Engineering European project is a concerted effort to define the new generation of end-to-end methodologies, processes, and supporting tools for safety-critical embedded-system design.[27] 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 which enables the sound integration of existing and new tools. Underlying this approach is the definition of a *heterogeneous rich-component* (HRC) model that can represent functional as well as architectural abstractions, such as timing, safety, and other nonfunctional performance metrics. These different

viewpoints can be developed separately in the model, and then integrated and evaluated together to derive the most efficient component-based implementation of the system.[28,29] This custom approach to model design is in contrast to the lightweight variant we presented earlier. This has primarily to do with the requirement of a tight and customized integration between tools, which is supported by the metamodel itself.

Methodological requirements for Speeds

The Speeds methodological requirements are the drivers behind the choices made for the HRC model design, which is targeted to the domain of embedded and reactive systems. The first characteristic to be considered, as we have discussed, is that concurrent development of systems occurs by different teams, which focus on functionality as well as different aspects or viewpoints, such as safety or reliability, timing (for example, in time-triggered development),[30] memory management to ensure segregation of subsystems, and energy. Each aspect requires specific frameworks and tools for analysis and design. Yet, they are not totally independent but rather interact, in ways that are sometimes not obvious. In HRC, these different aspects are expressed in the same model, which is tailored to the different cases by ignoring the nonessential 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 integrating 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 viewpoints and the requirements they express. Early requirement capture today still relies, for the most part, on organized textual descriptions, with little formal support, if any. Advancing beyond this will require formalizing the notation used for individual requirements by relying, for example, on semiformal graphical-scenario languages.[31-33] Similarly, an HRC model can serve as the underlying formal description for system requirements. No matter what model is used, the key point is that several requirements can 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 that jointly contribute to component specification. Consequently, designers need different operators when composing viewpoints 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 an HRC model by including, as part of the component specification, the needed information regarding the possible contexts of use. Thus, designers can establish the responsibilities of suppliers and integrators by explicitly expressing the assumptions under which a component is to be used. This separation between the assumptions and the specification, or promise, of the component is implemented in the form of design contracts, and it is one of the distinguishing features of the HRC approach.

Speeds principles

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 (Workshop for Object-Oriented Design and Development of Embedded Systems) and Omega projects.[34,35] 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 could be easily imported into different verification and analysis tools. In Omega, an explicit effort was made to preserve as many of the original structuring concepts that would be useful for obtaining efficient analysis. Nevertheless, the translation process required additional components and/or the enrichment of both the interfaces and the behavior of existing components. This led to the well-known problem whereby users could not interpret the analysis results, despite significant efforts dedicated to provide this feedback in terms of the original user concepts, whenever possible.

Also in the Speeds framework, the integration of a set of modeling and analysis tools is based on a common intermediate format, HRC, defined in the form of a metamodel. However, several original ideas have been integrated in the Speeds HRC metamodel. For instance, Speeds lets several modeling tools contribute to the global model of a given system and use the component-code generated by some modeling tool
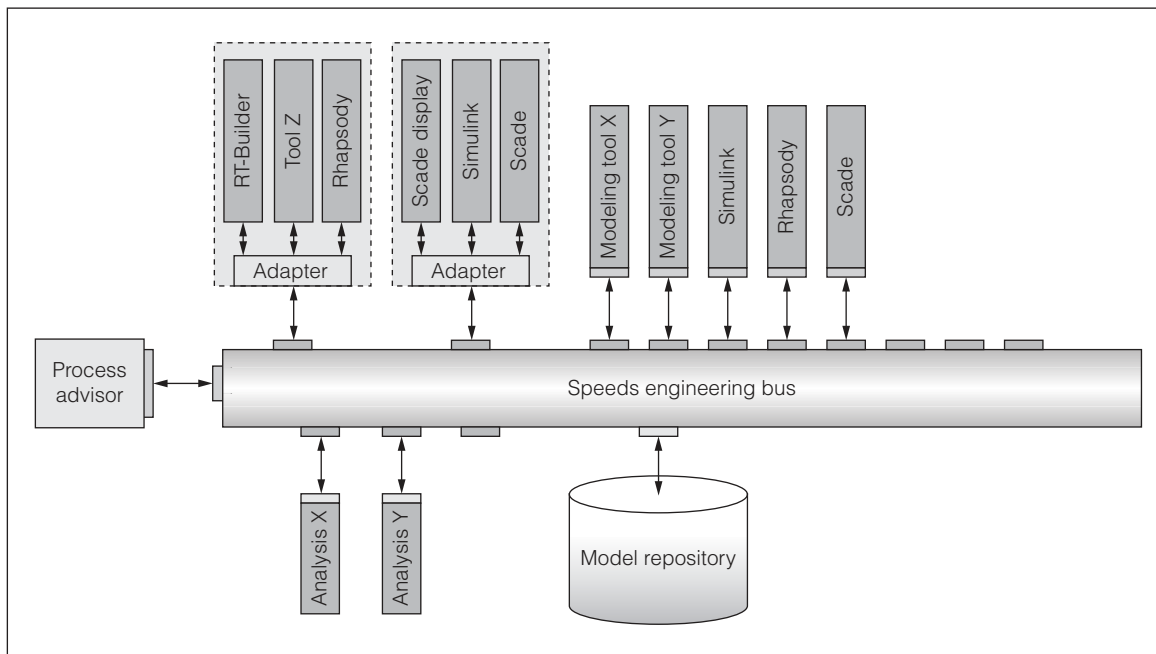
Figure 3. Tool integration via the Speculative and Exploratory Design in System Engineering (Speeds) bus and metamodel.

as their own behavior through the hosted simulation, described later. 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 defined in a layered manner, corresponding to different usages.

Figure 3 shows the structure of the Speeds framework. The Speeds engineering bus and its repository give direct access to multidimensional design data in commercial-off-the-shelf tools, and analysis tools can access the entire virtual model of the system. A process advisor, using the results of the analysis tools, measures design maturity and process convergence, and highlights unresolved assumptions and open design decisions.

The HRC model supports an expressive representation that rests on a semantically well-founded formalism. The formalism describes the abstract behaviors of components or the environment of differently natured systems—software systems as well as physical subsystems, or models of human behaviors, and so on. Such behaviors are described in a form usable by a wide range of analysis tools, as compositions of extended automata describing constraints on discrete and continuous behavior. For these reasons, on top of traditional static component interfaces that define only the interaction points of components, richer

information is exposed to designers as a set of *contracts*. Contracts, associated with a component, abstract constraints on the component and its environment behaviors in the form of assumption-promise pairs. 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 a model of the components exists, and then used throughout the design cycle for verifying or testing the correctness of abstract models or actual implementations. Figure 4 shows the metamodel definition of rich components in the HRC model, highlighting the structure of a contract (made of an assumption-promise pair), which is in turn expressed using a state machine formalism abstracted by HRC blocks.

The HRC metamodel consists of three levels of increasing abstraction, as shown in Figure 5. Level L1 defines the concepts handled by most analysis tools and has been designed for efficient analysis. Different composition modes (asynchronous and synchronous) are expressed here through a rich set of connectors for which there are well-founded theoretical results that can be exploited to make compositional verification efficient.[36] Level L1 is built atop level L0, which provides the basic semantic notions of the metamodel. All L1 concepts can be mapped
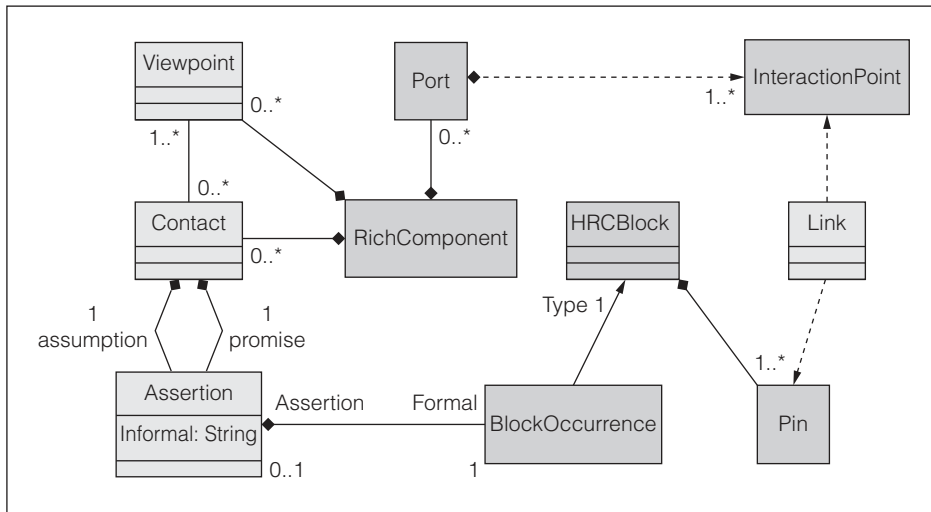
Figure 4. Speeds metamodel: components have contracts.

exchange model for those analysis tools that are tailored toward the verification of synchronous descriptions.[37] Above L1, level L2 is used as a bridge to user-level concepts. Thus, notions 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—which 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 might also choose to handle some of them more efficiently by implementing specific methods tailored toward the L2 layer.

to this semantic layer, although the translation might 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 and is the
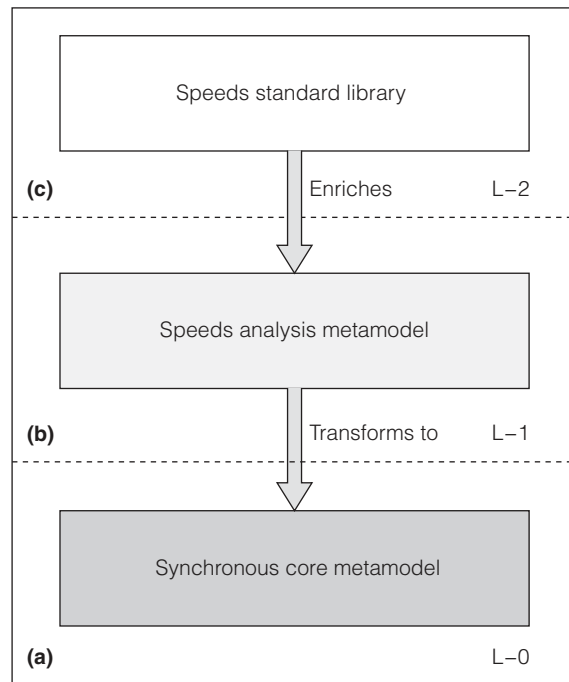
Figure 5. The Speeds-layered metamodel: low-level semantics and concepts used for synchronous modeling (a), abstract semantics and high-level concepts used by analysis tools (b), and generalizations of concepts from user modeling (c).

Comparison with other standards

A relevant question is how the HRC model relates to other standards, and why we defined a stand-alone metamodel rather than a profile, such as SysML and Marte. Initially, we intended to adapt SysML to our needs, because it embodies a general modeling approach that is familiar to most users and is 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 such as Scade and Simulink, our model had to provide some means of expressing a synchronous execution model. Marte, on the other hand, allows 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: whereas the HRC model provides a simple and expressive formalism for constraints, Marte introduces several predefined attributes for expressing standard nonfunctional 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 an HRC model would automatically inherit the definition of the high-level concepts.

Our choice of defining a stand-alone metamodel had several motivations. SysML was already a profile for UML and contains a tremendous number of additional features that could not just be eliminated, nor in a simple way. 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 because many transformations from and to a variety of other formats depend on the HRC metamodel. Also, because the HRC model is only an intermediate representation used in tools, and the user sees models only in terms of some modeling tool or in terms of some analysis or code generation tools, there was no strong need for HRC graphical editors, all of which diminished the pressure for aligning with UML.

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. Here we provide an intuitive understanding of their relationships, illustrating the concepts using a notation based on set theory. More details are available elsewhere.[38,39]

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 that 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 both M and E are defined over the same set of ports and all the behaviors of M are also behaviors of E, that is, when $M \subseteq E$.

We represent properties of components, or *assertions*, as the set components that satisfy the assertions. Exploiting refinement, an assertion E is equal to its largest satisfying component. A contract C for a rich component is a pair of assertions (A, G), 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 determining whether 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 viewpoints 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,

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

Parallel composition of contracts is also needed to formalize the combination of rich components. First, the parallel composition of two contracts must 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

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

The availability of different composition operators makes it possible to develop flexible system

integration flows that focus alternatively on the composition of components and of viewpoints, or combinations of the two.

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, which can be simulated by a single simulation tool.

This approach has several advantages compared to the standard cosimulation approach. First, only one simulation tool is needed, where all animated views are available. Second, there is no overhead associated with the use of a message bus and with the coordination between different simulators. Finally, a designer can simulate the interaction using an HRC component exported from a design tool that has no simulation capabilities. The main drawback of this approach is that a designer can monitor the interactions only between the components, not within each 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 that 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 exchange 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 metamodel. The HRC being exchanged between the tools has two parts. The first is a description of the component based on an HRC model, and contains the interfaces and abstractions describing the component in terms of the metamodel. 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 (dynamic link library) accompanied with header files.

Functional-safety concepts

Functional safety can be expressed in an HRC model 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. We allocate functional-safety requirements to subsystems by associating the corresponding contracts with the system's subcomponents. 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 determined by either the system environment or other components. Thus, either the context should already be contained in the safety goal or it should be 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, a nontrivial dependency structure usually exists between the individual contracts of the components on one side, and between these and the system contracts 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, counterexamples are produced that satisfy all allocated requirements, and do not (fully) satisfy the safety goal defined for the system. These counterexamples pinpoint flaws in the safety concept and provide effective guidance on how to redefine or adapt the safety concept.

To illustrate this, we consider the ISO CD 26262 standard, which allows exploiting redundancy in a process called *ASIL* (Automotive Safety Integrity Level) *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 that *Sys* operates in, and f is the function to be implemented. The underlying black-box view is given in Figure 6a as a SysML block diagram. A typical redundant implementation refines this black-box view as shown in Figure 6b. There are

Figure 6. Black-box view (a) and redundant implementation (b).

three redundant components h_1, h_2, and h_3 (each implementing the function f), and a majority voter *Vote*, which checks for agreement of two output values. For these, we have contracts

$$
\begin{aligned}
C_{h_1} &= (\langle\text{context}\rangle, \, y_1 = f(x)) \\
C_{h_2} &= (\langle\text{context}\rangle, \, y_2 = f(x)) \\
C_{h_3} &= (\langle\text{context}\rangle, \, y_3 = f(x)) \\
C_{Vote} &= (true, (y = y_1 \wedge y_1 = y_2) \vee (y = y_2 \wedge y_2 = y_3) \\
&\qquad \vee (y = y_3 \wedge y_3 = y_1))
\end{aligned}
$$

Three dominance checks yield that

$$C_{h_i} \parallel C_{h_j} \leq C_{Sys} \quad i, j \in \{1,2,3\}, \quad i \neq j$$

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

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. Metamodeling techniques have consequently emerged to organize the landscape of models, and to provide theories and methods for the development of coordinated representations more suitable for the heterogeneous environment in which modern embedded systems operate.

Several new initiatives and funded projects show that much work is still needed, from both a tool support point of view and a fundamental-understanding point of view. Of particular interest are the studies that extend the heterogeneous model integration from the structural to the semantics standpoint while ensuring consistency.[21,25] 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

∎ References

1. F. Terrier and S. Gérard, "Model-Driven Engineering and Prototyping of Real Time Embedded Applications," *Proc. IFIP Working Conf. Distributed and Parallel Embedded Systems* (DIPES 06), Springer, 2006.

2. J.-B. Raclet et al., "Why Are Modalities Good for Interface Theories?" *Proc. 9th Int'l Conf. Application of Concurrency to System Design* (ACSD 09), IEEE CS Press, 2009 (to appear).

3. B. Selic, "From Model-Driven Development to Model-Driven Engineering," keynote talk at *Euromicro Conf. Real-Time Systems* (ECRTS 07), 2007; http://feanor. sssup.it/ecrts07/keynotes/k1-selic.pdf.

4. D. Schmidt, "Model-Driven Engineering," *Computer,* vol. 39, no. 2, 2006, pp. 25-31.

5. A.L. Sangiovanni-Vincentelli, "Defining Platform-Based Design," *EE Design of EE Times,* Feb. 2002; http://www.gigascale.org/pubs/141.html.

6. H. Espinoza, "An Integrated Model-Driven Framework for Specifying and Analyzing Non-functional Properties of Real-Time Systems, doctoral dissertation, CEA LIST, 2007.

7. H. Espinoza et al., *Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems,* tech. report, Center of Nuclear Energy, CEA-LIST, 2008.

8. OMG, Semantics of a Foundational Subset for Executable UML Models (Beta 1); http://www.omg.org/spec/FUML/1.0/Beta1.

9. Papyrus, Open Source Tool for Graphical UML2 Modeling; http://www.papyrusuml.org.

10. Rational Software Architecture (RSA); https://www.ibm.com/developerworks/rational/library/05/510_svc/.

11. LISE, Laboratoire d'ingénierie dirigée par les modéles pour les systémes embarqués (LISE) [Laboratory of Model-Driven Engineering for Embedded Systems], project leader: F. Terrier. LISE is part of CEA LIST [Atomic Energy Commission, Software-Intensive Systems R&D] (in French).

12. IMOFIS Project, Ingénierie des Modéle Fonctions Sécuritaires [Model-Driven Engineering for Safety Functions]; http://www.imofis.org (in French).

13. B. Selic, "On the Semantic Foundations of Standard UML 2.0," *Formal Methods for the Design of Real-Time Systems,* LNCS 3185, M. Bernardo and F. Corradini, eds., Springer-Verlag, 2004, pp. 181-199.

14. C. André, "Time Modeling in MARTE," *Proc. Forum Specification and Design Languages* (FDL 07), CD-ROM, European Electronic Chips & Systems Design Initiative, 2007.

15. MeMVATEX French Project, Méthodologie pour la Modélisation, la Validation et la Tracabilité des Exigences [Methodology for Modeling, Validation, and Traceability of Requirements]; http://www.memvatex.org (in French).

16. ATESST Project, "Advancing Traffic Efficiency and Safety through Software Technology," ATESST Specific Targeted Research or Innovation Project (STREP), 6th Framework Programme; http://www.atesst.org.

17. Lambda Project, Lambda Libraries for Applying Model Based Development Approaches; http://www.usine-logicielle.org/lambda/index_FR.html.

18. Autosar Development Partnership, "Automotive Open System Architecture," Autosar, Munich; http://www.autosar.org.

19. EDONA Project, Environnements de Développement Ouverts aux Normes de l'Automobile [Open Development Platform for Automotive Standards]; http://www.edona.org (in French).

20. A. Albinet et al., "The MeMVaTEx Methodology: From Requirements to Models in Automotive Application Design," *Proc. 4th European Congress Embedded Real Time Software* (ERTS 08), Societe des Ingenieurs de L'automobile.

21. A. Cuccuru et al., "Enhancing UML Extensions with Operational Semantics: Behaviored Profiles with Templates," *Proc. 10th Int'l Conf. Model Driven Engineering Languages and Systems* (MODELS 07), LNCS 4735, Springer, 2007, pp. 271-285.

22. Interested FP7 IP: Interoperable Embedded Systems Tool-Chain for Enhanced Rapid Design, Prototyping and Code Generation; http://www.interested-ip.eu.

23. Genesys Project, Generic Embedded System Platform, a 7th Framework program; http://www.genesys-platform.eu.

24. CESAR Project, "Cost-efficient methods and processes for safety relevant embedded systems," funded project from Artemis Joint Undertaking (JU); http://www.cesarproject.eu.

25. COMBEST, Component-Based Embedded Systems Design Techniques project, Information Society Technologies, STREP 215543, a 7th Framework program; http://www.combest.eu.

26. F. Lagarde et al., "Leveraging Patterns on Domain Models to Improve UML Profile Definition," *Proc. Fundamental Approaches to Software Engineering* (FASE 08), LNCS 4961, Springer Verlag, 2008, pp. 116-130.

27. Speculative and Exploratory Design in Systems Engineering, European Union 6th Framework Project in Embedded Systems Development, IP contract 033471; http://www.speeds.eu.com.

28. A. Benveniste, B. Caillaud, and R. Passerone, "Multi-viewpoint State Machines for Rich Component Models," *Model-Based Design of Heterogeneous Systems,* CRC Press, 2009 (to appear).

29. A. Benveniste et al., "Multiple Viewpoint Contract-Based Specification and Design," *Proc. 6th Int'l Symp. Formal Methods for Components and Objects* (FMCO 07), LNCS 5382, Springer Verlag, 2008, pp. 200-225.

30. H. Kopetz, "The Time-Triggered Model of Computation," *Proc. 19th IEEE Real-Time Systems Symp.,* IEEE CS Press, 1998, pp. 168-177.

31. J.F.M. Burg, *Linguistic Instruments in Requirements Engineering,* IOS Press, 1997.

32. W. Damm and D. Harel, "LSCS: Breathing Life into Message Sequence Charts," *Formal Methods in System Design,* vol. 19, no. 1, 2001, pp. 45-80.

33. ITU-TS, ITU-TS Recommendation Z.120: Message Sequence Chart (MSC), ITU-TS, Geneva, Sept. 1999.

34. WOODDES Consortium, WOODDES project; http://wooddes.intranet.gr.

35. OMEGA Consortium, Information Society Technologies (IST) Omega project for Correct Development of Real-Time Embedded Systems; http://www-omega.imag.fr.

36. S. Bliudze and J. Sifakis, "The Algebra of Connectors: Structuring Interaction in BIP," *Proc. 7th ACM & IEEE Int'l Conf. Embedded Software* (EMSOFT 07), ACM Press, 2007, pp. 11-20.

37. A. Benveniste et al., "The Synchronous Languages Twelve Years Later," *Proc. IEEE,* vol. 91, no. 1, 2003, pp. 64-83.

38. A. Benveniste, B. Caillaud, and R. Passerone, "A Generic Model of Contracts for Embedded Systems," tech. report 6214, INRIA, June 2007.

39. L. Benvenuti et al., "A Contract-Based Formalism for the Specification of Heterogeneous Systems," *Proc. Forum Specification and Design Languages* (FDL 08), CD-ROM, European Electronic Chips & Systems Design Initiative, 2008, pp. 142-147.

Roberto Passerone is an assistant professor at the University of Trento. His research interests include heterogeneous modeling, contract-based reasoning, and system-level design methodologies. He has a PhD in electrical engineering and computer sciences from the University of California, Berkeley. He is a member of the IEEE.

Imene Ben Hafaiedh is a PhD fellow at Verimag laboratory and Université Joseph Fourier. Her research interests include embedded systems design, verification, and validation within a component-based framework. She received a master's degree in computer science (systems and software) from Université Joseph Fourier.

Albert Benveniste is director of research at INRIA Rennes. His research interests include embedded systems development and network and Web services management. He has a PhD in mathematics from Paris 6. He is a Fellow of the IEEE.

Daniela Cancila is a research engineer at CEA LIST. Her research interests include processes and methodologies for safety engineering of embedded systems. She has a PhD in computer science from the University of Udine.

Arnaud Cuccuru is a research engineer at CEA LIST. His research interests include model-driven engineering techniques and their application for real-time and embedded systems development. He has a PhD in computer science from Université des Sciences et Technologies de Lille.

Werner Damm holds the Chair for Safety Critical Embedded Systems at Oldenburg University. His research interests include methods and processes for the development of embedded applications in transportation. He has a PhD in computer science from the RWTH Aachen.

Alberto Ferrari is deputy director of Parades, in Rome. His research interests focus on the design and architectures of safety-critical and real-time distributed systems. He has a PhD in electrical engineering and computer science from the University of Bologna, Italy.

Susanne Graf is research director at CNRS, and she works in the Verimag laboratory. Her research interests include design and verification technologies for embedded and distributed systems. She has a PhD from the Institut Polytechnique de Grenoble.

Sébastien Gérard is leader of the Accord/UML group within CEA LISE at LIST. His research interests include correct-by-construction and model-based design of real-time and embedded systems. He has a PhD in computer science from ENSMA [French Superior School of Mechanics and Aeronautics at Poitiers]. He is cochair of the UML 2 and Marte standardization task forces.

Bernhard Josko is director of the R&D Division of Transportation at OFFIS in Oldenburg. His research interests include development methods for embedded-system design, especially formal specification and analysis techniques. He has a PhD in sciences from RWTH Aachen.

Leonardo Mangeruca is a senior research scientist at Parades in Rome. His research interests include design methodologies, safety-critical hardware-software architectures, and formal methods for embedded systems design. He has a PhD in electrical engineering from the University of Genova, Italy.

Thomas Peikenkamp is manager of the Safety Analysis & Verification Group at OFFIS in Oldenburg. His research interests include model-based safety analysis methods and formal specification techniques. He has a diploma in computer science from the RWTH Aachen.
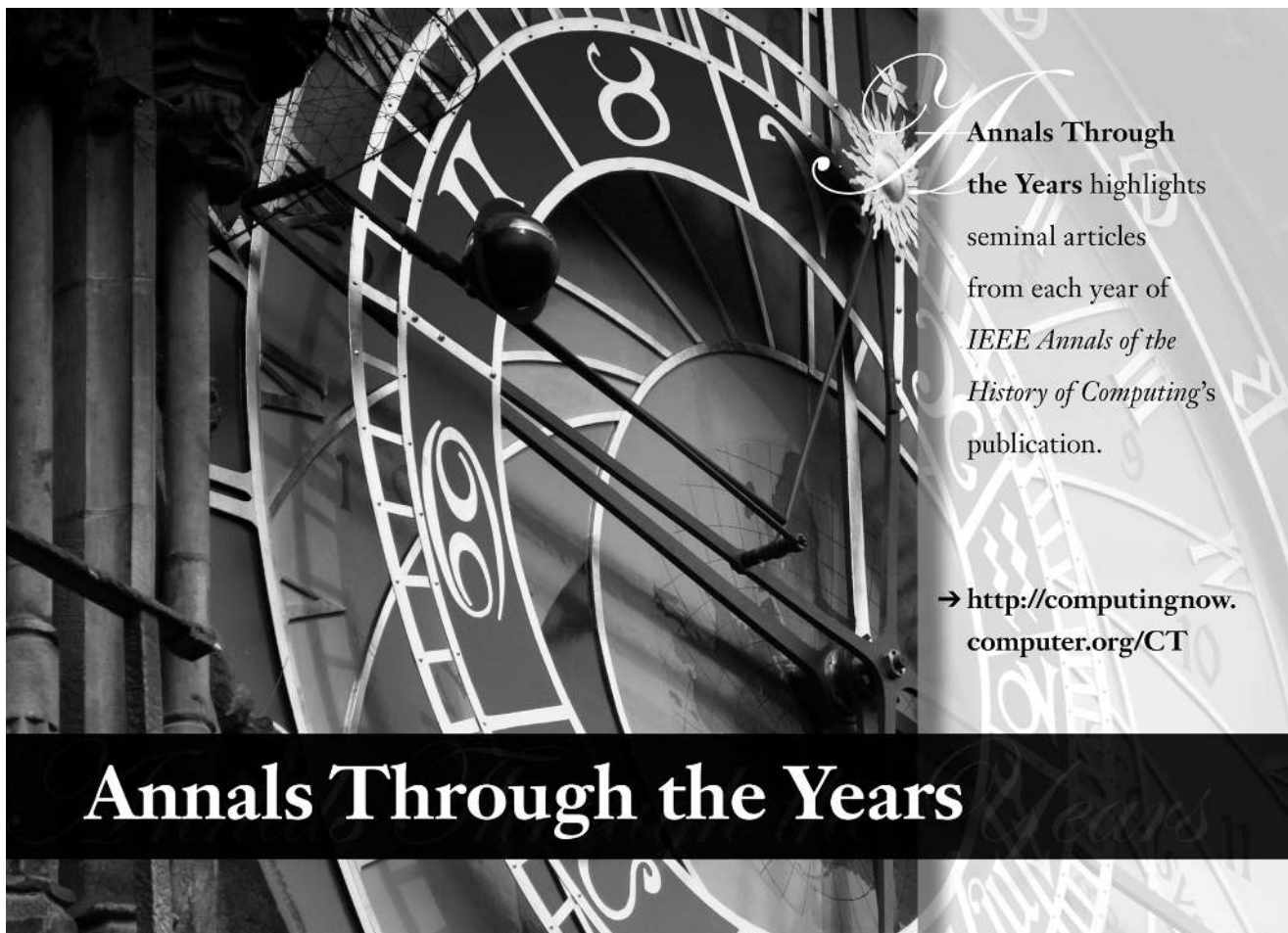
Alberto Sangiovanni-Vincentelli holds the Buttner Chair of Electrical Engineering and Computer Science at the University of California, Berkeley. His research interests include system-level design, embedded and hybrid systems, and EDA. He has a Dr Eng in electrical engineering and computer sciences from Politecnico di Milano. He is a Fellow of the IEEE, and is a member of the National Academy of Engineering and the ACM.

Francois Terrier is leader of the CEA LISE, a part of LIST. His research interests include model-based development and verification of embedded systems. He has a PhD in electronics–artificial intelligence from Université Paris Sud.

■ Direct questions and comments about this article to Roberto Passerone, Dept. Engineering and Computer Science, University of Trento, via Sommarive 14, 38100 Povo di Trento, Italy; roberto.passserone@unitn.it.

For further information on this or any other computing topic, please visit our Digital Library at http://www.computer.org/csdl.

Annals Through the Years highlights seminal articles from each year of *IEEE Annals of the History of Computing*'s publication.

→ http://computingnow.computer.org/CT

Annals Through the Years