6 Modeling Languages for Real-Time and Embedded Systems

Requirements and Standards-Based Solutions*

Sébastien Gérard[1], Huascar Espinoza[2], François Terrier[1], and Bran Selic[2]

[1] CEA LIST, Laboratory of Model Driven Engineering for Embedded Systems (LISE), Boîte courrier 65, Gif sur Yvette Cedex, F-91191 France
{Sebastien.Gerard,Huascar.Espinoza,Francois.Terrier}@cea.fr
[2] Malina Software Corp., Nepean, Ontario, Canada
selic@acm.org

Abstract. Development of increasingly more sophisticated dependable real-time and embedded systems requires new paradigms since contemporary code-centric approaches are reaching their limits. Experience has shown that model-based engineering using domain-specific modeling languages is an approach that can overcome many of these limitations. This chapter first identifies the requirements for a modeling language to be used in the real-time and embedded systems domain. Second, it describes how the MARTE profile of the industry-standard UML language meets these requirements. MARTE enables precise modeling of phenomena such as time, concurrency, software and hardware platforms, as well as their quantitative characteristics.

6.1 Introduction

It is helpful to start with a clear definition of what is meant here by real-time and embedded systems (RTES). To that end we provide below a taxonomy of the different kinds of real-time and embedded systems that are of interest in this chapter. There is no generally agreed on classification of systems in the real-time and embedded domain. For our purposes, we shall use the following taxonomy (NB: this categories are not mutually exclusive) [1]:

- The *embedded* domain – This covers systems composed of both hardware and software components.
- The *reactive* domain – This sub-category covers systems thatrespond to discrete stimuli generated by their environment.
- The *command and control* domain – These systems are usually built to manage the running of a physical process or other systems.
- The *intensive data flow computation* domain – These systems generally deal with large amounts of physical data for applications such as signal processing, image processing, or various mobile device functions.
- The *best-effort service* domain – These are real-time systems which do not guarantee meeting all their timing and safety constraints for every individual input.

H. Giese et al. (Eds.): MBEERTS, LNCS 6100, pp. 129–154, 2010.

The use of abstraction as a means for coping with complexity when designing large technological systems has always been a common and effective strategy in engineering. It has proven particularly effective in the design and realization of software-intensive systems through the use of computer languages of increasing levels of abstraction, starting with assembly languages, followed by third-generation languages such as C, and on to object-oriented languages like C++ and Java. However, faced with unrelenting demands for ever more sophisticated and more dependable systems as well as for shorter time to market intervals, we seem to be approaching the limits of effectiveness achievable by using traditional code-based approaches.

Model-based design is considered by many as a suitable approach to overcoming these limits, particularly in the embedded systems domain. One of the expected advantages of this approach is the ability to exploit correct-by-construction incremental design processes, which rely on extensive use of automated transformations and synthesis, as well as formalized computer-based analyses of correcteness.

Undoubtedly, much effort is required to develop the tools and methods necessary to bridge the gap between the very optimistic vision of Jacobson, who advocated that "software development is model building" [2], and the views held by more conservative software programmers, who often feel that "they don't have time to waste on modeling". In the past decade significant progress has been made in this direction, most notably the emergence of meta-modeling and practical model transformation techniques. These and related innovations are at the core of a new approach to system and software design and development often referred to as model-based engineering (MBE) or model-driven development (MDD).

The incremental nature of model-based engineering approaches is based on progressive refinement of an abstract design or system model through the gradual inclusion of more and more detail. Supported by automation-based verification and validation, this refinement is performed until the model is either (a) sufficiently detailed for relatively straightforward trouble-free implementation or (b), in case of software systems, it actually becomes the system that it was modeling. The latter in particular relies on appropriate tools that can automatically transform a model expressed using an abstract modeling language into a corresponding concrete technology-specific implementation. Thus, there are two key aspects to model-based engineering: one is the issue of selecting the right abstractions for a modeling language and the other is the matter of tool design. In this chapter, we will focus exclusively on the first aspect: the requirements for and design of modeling languages suitable for real-time and embedded systems.

For the real-time and embedded system domain, a major source of design complexity comes from the intrinsic heterogeneity of these systems. Indeed, design of modern real-time and embedded systems depends more and more on effective interplay of multiple disciplines, such as mechanical, control, electronics, and software engineering. These systems are compositions of different inter-related parts (also called components), some of which may have already been designed

while others need to be designed. Given their heterogeneous nature, the parts are typically designed by different design teams, possessing different expertise, and using different tools. This is often done through vertical design chains such as, for example, in the avionics and automotive industries. A complete development chain typically involves a multitude of tools and data that are today still poorly integrated. In particular, the lack of a common modeling language to specify the overall system architecture hampers reasoning about solution trade-offs during early development phases. This results in high development costs due to long feedback cycles for issues uncovered during the integration phase.

Examples of integration needs in this area include: bridging the gap between both software and hardware models, or between software models expressed in a systems language and their implementation in terms of a target programming language. Other examples include coordinating modeling and design tools with specific engineering analysis tools (e.g., for safety or performance analysis), or connecting control engineering tools (such as Matlab/Simulink [3] or tools supporting the Modelica [4] language) with architectural design tools. Such integrations are usually complex, inefficient, and error-prone resulting in the infamous "islands of automation".

Given the importance that sharing knowledge has in embedded system development, we subscribe to the view that both system design and integration will be reduced significantly by the use of a common modeling formalism. In particular, we believe that the widespread acceptance of UML (Unified Modeling Language) [5] by both the industrial and academic communities, along with the use of UML *profiles*[1] for domain-specific purposes will considerably ease integration difficulties.

In the following section, we summarize some key requirements for modeling embedded systems. In section 6.3, we describe a standard modeling language that meets these requirements. The profile mechanism is explained first, since it is used to derive the domain-specific modeling language (DSML) out of standard UML. This is followed by an introduction to the language itself, the UML profile for modeling and analyzing real-time and embedded systems, MARTE. This profile has been adopted by the OMG as a standard technology recommendation that deals with modeling of time- and resource-constrained characteristics of systems, and includes a detailed taxonomy of relevant hardware and software patterns along with their non-functional attributes. Among other things, MARTE enables state-of-the-art quantitative analyses (e.g., performance or schedulability analysis). Section 6.4 concludes with a description of some typical scenarios that illustrate the value of MARTE in specifying real-time and embedded systems. Section 6.5 discusses contributions and shortcomings of other modeling languages for the same domain, and section 6.6 summarizes the conclusions of this chapter.

[1] A profile is the mechanism standardized by the OMG for creating domain-specific modeling languages by refining the concepts of an existing standard language such as UML.

6.2 Two Main Architectural Styles for Dealing with Abstraction

To cope with the complex nature of the real-time and embedded systems and their ever increasing sophistication and more stringent requirements, it is helpful to use higher levels of abstraction when specifying them. Since abstraction is one of the most powerful benefits of using models and modeling languages, we concur with the view described in [6], section 3.1.8 on page 3-13, that, when modeling a system, abstraction can be applied both vertically and horizontally.

Vertical Abstraction (Layering)
This is one of the most popular architectural patterns. It provides a graduated form of abstraction across multiple discrete levels. Two primary forms of this pattern can be identified:

Refinement layering is needed to support the iterative refinement process flows which occur during development; each layer focuses on a different level of detail. From the language point of view, what is needed is the ability to trace between corresponding model elements at two different levels of abstraction (vertical layers). From the modeling language point of view as well as from a formal perspective, it should be feasible to relate these different layer models in order to (1) enable conformance verification between them and (2) ease derivation of model elements from one level to the next. From the tooling point of view, the goal is to define and automate the process of deriving one specialized model from another (code generation is a typical example of this).

Concrete layering is used to deal with horizontal separation-of-concerns. It comes from the recurrent need in system development to model relationships between applications and their underlying software platforms (e.g., real-time operating systems or dedicated middleware) and hardware implementation platforms (e.g., SystemC and VHDL). They identify dependencies between application models and implementation choices/constraints.

Horizontal Abstraction (Slicing)
When considering abstraction from a horizontal point of view, we will use the term "slicing". Indeed, in vertical abstraction the system is divided into layers, whereas in horizontal abstraction, the system is represented as comprising different slices (i.e., partitions), with potential relationships between them. In [6], this aspect was referred to as the peer-interpretation of the client-server relationship, in contrast with its layered-interpretation that matches the layering introduced above. Again, as we did for layering, we can classify slicing into *abstract slicing* and *concrete slicing*. Slicing is intended to be used for grouping the components of a system into different sets, called slices. The rationale for grouping entities into a particular slice may vary. For example, it may be for some project-related organizational reasons or based on the need to separate distinct functional concerns. Whatever the rationale, it is important to remember that all components, regardless of which slice they belong to, share a common feature: they all coexist at the same level of abstraction!

In addition, slicing can sometimes be associated with an abstraction operation at a higher level. For example, this is typically required when it is desired to view a system from a specific perspective. Consequently, abstract slices are sometimes called *views* or *projections*. Such *abstract slicing* may contain slices that are quite different from and unrelated to the concrete slices of a system. Examples of such abstraction include task models for a schedulability analysis, or architectural models centering on system functions and scenarios models for system testing.

In contrast to a slice, a layer is a view of a complete system, but at a specific level of abstraction that is different from other layers. For example, one layer (which RM-ODP would call the "computational" layer) might show a system as a network of concurrent logical components, whereas a lower layer would represent the very same system as a set of operating system tasks (the RM-ODP "technology" layer). In that sense, layers represent different viewpoints.

6.3 Modeling Needs for Real-Time and Embedded Systems Design

In this section, we identify a set of requirements that a modeling[2] language must fulfill to support the design of real-time and embedded systems. These needs are grouped into two categories following the dichotomy introduced in the previous section.

6.3.1 Layering and Needs for RTES

First, we focus on the implications that different forms of layering have for a real-time and embedded modeling language.

Refinement
Clearly, a modeling language must support the refinement relationships between two model layers. In particular, in the real-time and embedded domain, it is necessary to be able to attach non-functional properties to such refinement relationships.

Resource
Real-time and embedded systems are computer-based systems that interact with the physical world. This means that they are not only coupled to the physical world but that they are also constrained by the physical capacities of their underlying hardware and/or software platforms. Hence, these systems are typically resource limited. It is therefore crucial that the modeling language provide facilities for a precise modeling of such resources. Specifically, this facilitates two very important capabilities.

[2] In this paper, the term *modeling* refers to the process of describing a system architecture and its features. *Design* refers to the activities involved in making solution-oriented decisions that satisfy given requirements and constraints for the intended product. *Analysis* is the process of verifying how well the resulting system satisfies these requirements (usually before the actual systems is fully implemented).

First, since the application software will be embedded in a specific software and/or hardware platform, the code that is generated from a model of the application must be easily interfaced with a variety of potential computing platforms, such as a real time operating systems (RTOS), middleware, micro-controllers, or specific hardware (e.g., ASIC and FPGA).

Second, achieving a balance between the need to optimize the utilization of resources for cost reasons while meeting an application's functional and non-functional (e.g., quality of service) requirements can be a very challenging design task. Consequently, real-time and embedded design generally requires facilities to perform resource optimizations.

Both of these point to a need for a modeling language that can accurately model computing platforms and other kinds of resources commonly encountered in the real-time and embedded domain.

Allocation

The design of real-time and embedded systems often follows the well-known Y-Chart scheme [7]. This approach specifies how both the application model and the resource platform model are combined to provide the full system model. This is then achieved by means of a third model, often called the mapping model, allocation model, or *deployment model*. This kind of model specifies how the elements of an application model are allocated to elements of the platform. Since an application is simply a software specification, it is the platform elements that are ultimately responsible for its physical realization. In other words, the allocation model identifies which elements of the platform are used to execute a given part of the application specification. Moreover, for our specific domain, it is very important to be able to specify the associated non-functional characteristics of the allocation. For example, when deploying the execution of a behavior on a given processor, one may need to specify its worst-case execution time.

Refinement Modeling

As explained in the section describing abstract layering, model-based development process must support abstraction refinement. This also introduces the need to trace and propagate changes up and down the layer hierarchy. Hence, a modeling language must support explicit modeling of the refinement relationships between models at different abstraction layers, and should also allow attaching non-functional properties to such relationships.

6.3.2 Slicing and Needs for RTES

As might be expected, a real-time and embedded modeling language must also support different kinds of domain-specific phenomena in a suitable manner. In particular, it needs modeling concepts dedicated to specifying quantitative characteristics such as deadlines, periods, bandwidths, processing capacities, etc. as well as qualitative features that are related to behavior, such as communication methods and concurrency. This results in a number of concrete language requirements described below.

Time

The temporal (behavioral) models of real-time and embedded systems can be grouped into three main categories as follows [1]:

(a) *Asynchronous/Causal models* are merely concerned with the proper ordering of activities (instructions, actions, so on), due to some control or data flow prescription. Some amount of scheduling may be needed if the specified flow is partial. Therefore, in such cases, time is viewed in terms of causal dependencies rather than specific quantities or durations. This model is used widely at the algorithmic software level (and in software models of hardware at the transaction level). In the presence of concurrency, the varying speeds of asynchronous components (with synchronous or asynchronous communications) generally lead to non-deterministic behavior.

(b) *Synchronous/Clocked models* add the notion of simultaneity of events and activities. Time is modeled as a discrete set of instants, and need not be connected to any physical reality, in the sense that the corresponding clock need not be regular. Henceforth we shall call this time "logical". This type of time model is used in (discrete step) simulation formalisms such as Simulink/Stateflow, in synchronous languages and Statecharts, as well as in hardware description languages at the register transaction level (e.g., VHDL, Verilog, SystemC, etc.).

(1) *Real/Continuous time models* take physical durations into account. These are important for doing various time-related analyses (e.g., deadline matches) and, in particular, for real-time scheduling as in RMA approaches [8]. They are also used for modeling the temporal characteristics of the physical environment or system with which the embedded system is interacting (usually before discretization).

A real-time and embedded modeling language needs concepts for dealing with different models of time, an, in particular, the three models of time described above, since they represent most of the common cases.

Quantitative Aspects

This concerns the use of mathematical techniques to identify or predict certain quality attributes of a system. They include, for example stress, thermal, or fluid analyses in mechanical engineering, as well as performance or reliability analyses in software engineering. One challenging problem in model-based engineering is to integrate models that are commonly used for system production or software code generation with the information that is relevant to perform these kinds of analyses. An important goal is to reduce the time required to prepare a design model for performing analysis and ensuring that the analysis model accurately represents the system. A related challenge is to hide, as much as possible, the underlying complexity of the formal mathematical model underlying the analysis methods. Both goals may be achieved by deriving the analysis model more or less directly from a suitably annotated system model using automated or semi-automated support.

To this end, it is critical to be able to capture the non-functional characteristics (e.g., performance, reliability, power consumption) in system models.

Furthermore, it should be possible to do this with precision and with maximum flexibility [9]. Thus, rather than fix in advance the set of non-functional properties that can be expressed with the language, modelers should be allowed to define the desired information in the form that is the most suitable for their specific analysis technique.. Such annotations should be interpretable by different editing or analysis tools and should not be dependent on any specific tool configuration. However, it would be impractical if modelers would have to specify all this semantic information in every design. Hence, a special requirement is a trade-off between usability and flexibility. Usability concerns favor declaring a set of fully interpretable non-functional properties for a given modeling subdomain, which are easily referred to and preserve the same meaning for every usage, whereas flexibility requires a capability for users to define their own types of non-functional properties, provided, of course, that they are semantically well-formed.

Qualitative Concerns
By qualitative concerns we refer firstly to aspects related to parallelism and related communication issues. By their very essence, real-time and embedded systems are indeed closely coupled to the real world which is inherently concurrent. Consequently, the modeling language must firstly provide the ability to specify concurrent entities capable of interacting and communicating with each other.

Underlying this preliminary concern is the more complex need to support various specific models of communication and computation. Behind this basic need is a more complex need to deal with heterogeneity of different models of communication and computation. Indeed, because of the growing complexity of systems, their development is more and more based on the possibility to consider a system as being made of a set of smaller parts. These latter can be developed using different approaches, and then different technologies, involving different models of computation and communication.. A useful modeling language must, therefore, provide a means for composing sub-systems relying on various heterogeneous models of computation and communication. This requirement may be met, for example, either by providing a means of composing different models of communication and computation, or by providing a generic model of computation and communication that can be specialized for different categories of real-time and embedded systems. In the latter case, since the model of computation and communication of the different sub-components of an application are based on the same generic model, it may be easier to compose them.

6.4 MARTE, a Standard Real-Time and Embedded Modeling Language

The Object Management Group (OMG, www.omg.org) is one of the principal international organizations promoting standards supporting the usage of model-based software and systems development. The Unified Modeling Language

standard (UML, [5]) is probably the most representative of these and has had significant success in the software industry as well as in other domains such as IT and financial systems. UML is now the most widespread language used for modeling in both industry and academia. It was designed as a general-purpose modeling language as well as a foundation for generating different domain-specific languages, mainly through its profile mechanism. The latter capability allows the general concepts of UML to be specialized for a specific domain or application.

In this section, we first introduce the UML profile concept, which is a very powerful means for defining domain-specific modeling languages (DSMLs). Next, we present MARTE, which is a UML profile for modeling real-time and embedded systems and is, in effect, a domain-specific modeling language for the real-time and embedded domain.

6.4.1 UML Profiling Capabilities

Because of the diverse nature of the disciplines needed for designing real-time and embedded system, it is clear that a single modeling language will not be enough to cover all the various concerns involved in this specific area. Consequently, there has been much discussion about the suitability of UML for such domains relative to custom domain-specific modeling language designed from scratch [10]. A custom language has the obvious advantage that it can be defined in a way that is optimally suited to a specific problem. At first glance, this may seem the ideal approach, but closer examination reveals that there it can have serious drawbacks. If each individual sub-domain of a complex system uses a different modeling language, the problem will be how to interface the various sub-models into a consistent integrated whole that can be verified, tested, or simply unambiguously understood. Furthermore, there is also the problem of designing, implementing, and maintaining suitable industrial-strength tools and training for a each custom language, which can result in significant and recurring expenses.

Conversely, although UML was designed to eliminate the accidental complexity stemming from gratuitous diversity [11], it still provides a built-in mechanism, the *profile* concept, for creating domain-specific modeling languages that can take advantage of existing UML tools and widely available UML expertise. Note that we are not saying that UML profiles completely avoid DSML integration problems. However, many of the fragmentation issues[3] [12] stemming from diversity can be mitigated because all domain-specific modeling languages derived from UML share a common semantic and syntactic foundation. There is typically a lot of commonality between the various disciplines in real-time

[3] This is used to refer to the situation that occurs when different domain-specific languages are used to describe different aspects of a complex system. For example, one language might be used to describe the user interface function while a different one for the database management and access functions. The individual languages involved could have very different models of computation, which raises the question of how to meld the different specifications into a coherent and consistent whole.

and embedded system design. For instance, the concepts of package, composition, property and connector, which are provided by UML, are common to many disciplines, as are the basic notions of object, class, and interface.

The basic premise of profiles is that all domain-specific concepts are derived as extensions or refinements of existing UML concepts. These extensions are called *stereotypes*. A stereotype definition must be consistent with the abstract syntax and semantics of standard UML, which means that a profile-based model can be created and manipulated by any tool that supports standard UML. Moreover, because of the underlying UML foundations of a profile, it is more easily learned by anyone with a knowledge of UML.

A stereotype may have attributes and be associated with other stereotypes or existing UML concepts. From a notational viewpoint, stereotypes can also be used to adapt the concrete syntax of UML in order to provide a more domain oriented concrete syntax if needed. For instance, a class model element stereotyped as ≪clock≫ might use a picture of a clock symbol instead of the generic UML class symbol.

We can distinguish two main categories of UML profiles [13]: specification and annotation profile. *Specification profiles* are fully-fledged domain-specific modeling languages and are used to model systems from the viewpoint of a particular domain. SysML [14] is an example of this kind of profile. *Annotation profiles* are used to add supplementary information to various kinds of UML elements that can then be interpreted by specialized tools or domain experts for different purposes, such as model analyzers or code generators. Note that annotation profiles are particularly useful for defining domain-specific modeling languages that support abstract layering and slicing. As we shall describe later, some parts of the MARTE profile, namely the sub-profiles that support various analyses, are examples of this latter category.

While specification profiles is generally well understood, some explanation may be necessary to understand the second category, annotation profiles. Specifically, in case of MARTE's analysis sub-profiles, a given analysis concept may be manifested in a number of different ways in a particular model. For example, a real-time clock may be manifested as a lifeline in a UML sequence diagram or as a role in a UML collaboration diagram. From the analysis viewpoint, all of these different manifestations represent the same thing. This means that it is necessary to be able to apply the same analysis stereotype to different kinds of UML concepts, and conversely, different stereotypes (possibly from different analysis viewpoints) may be applied in the same model element.

Concepts defined in the MARTE annotation profiles that support quantitative analysis can be applied to make a standard UML model look like an analysis model (e.g., a performance model). This is achieved by tagging appropriate elements of the original UML model to represent concepts from the analysis viewpoint. These can then be used by an automated performance analysis tool to determine the fundamental performance properties of a software design. At the same time (and independently of the performance modeler) a reliability engineer might overlay a reliability-specific view on the same UML model to determine its

overall reliability characteristics, and so on. Annotation profiles allow the same model to be viewed from different viewpoints (e.g., schedulability, performance, security, availability or timing). Finally, it should be noted that UML profiles have the very useful feature to be dynamically applied to a user model (e.g., to produce a domain-specific viewpoint) without changing the underlying model in any way. Subsequently, the profile can be removed to reveal the original model unchanged. As described in section 6.4.3, this feature is crucial to this type of profile usage.

6.4.2 MARTE Basics

As noted previously, UML is a general-purpose modeling language that can be specialized or extended for dealing with specific domains or concerns. The field of real-time and embedded software systems is one such domain for which extensions to UML are required to provide more precise expression of domain-specific phenomena (e.g., mutual exclusion mechanisms, concurrency, deadline specifications, and the like). The OMG had already adopted a UML profile for this purpose, called the "UML Profile for Schedulability, Performance and Time" (SPT, [6]). It provided concepts for dealing with model-based schedulability analysis, focused primarily on rate monotonic analysis, and also concepts for model-based performance analysis based on queuing theory. In addition, SPT also provided a framework for representing time and time-related mechanisms. However, practical experience with SPT revealed shortcomings within the profile in terms of its expressive power and flexibility. For example, it was necessary to support the design of both hardware and software aspects of embedded systems and more extensive support for schedulability and performance analysis, encompassing additional techniques such as hierarchical scheduling. Furthermore, when the new significantly revised version of UML, UML2, was adopted by the OMG, it became necessary to upgrade the SPT profile. Consequently, a new Request For Proposals (RFP) was issued by the OMG seeking a new UML profile for real-time and embedded systems. This profile was named MARTE (an abbreviated form of "Modeling and Analysis of Real-Time and Embedded systems," [1]). The intent was to address the above issues as well as to provide alignment with another standard OMG profile, the UML profile for Quality of Service and Fault Tolerance (QoS & FT, [15]). The latter enables specification of not only real-time constraints but also other embedded systems characteristics, such as memory capacity and power consumption. MARTE was also required to support modeling and analysis of component-based architectures, as well as a variety of different computational paradigms (asynchronous, synchronous, and timed).

In response to this request for proposal, a number of OMG member organizations collaborated on a single joint submission. This group, called the ProMARTE consortium, consisted of the following enterprises: Alcatel, ARTiSAN Software Tools, Carleton University, CEA LIST, ESEO, ENSIETA, France Telecom, International Business Machines, INRIA, INSA from Lyon, Lockheed Martin, MathWorks, Mentor Graphics Corporation, NoMagic, the Software Engineering Institute (Carnegie-Mellon University), Softeam, Telelogic AB, Thales, Tri-Pacific

Software, and Universidad de Cantabria. The resulting submission was voted on and accepted by the OMG in June 2007 [16] as a "Beta Specification".

As prescribed by the OMG's Policies and Procedures manual (P&P, [17]), following adoption, a Finalization Task Force (FTF) was instituted to prepare the new specification for its formal adoption as an official OMG technology recommendation. The working period of a finalization task force is about 18 months and its first phase (around 6 months) comprises a comments-gathering phase during which feedback from the broader user and vendor communities is collected. Of particular significance is input from commercial and other tool vendors intending to support the new specifications in their products. The second phase is then dedicated to solving the key issues identified in the initial phase resulting in a Beta 2 version of the specification. This version is first screened by the OMG's Architecture Board and, if deemed acceptable, is submitted to the OMG's Board of Directors for final approval. At that point, the resulting specification becomes formally available as version 1.0. In the case of MARTE, it is expected that this will be available by the first quarter of 2009.

6.4.3 Architecture and Some Details of MARTE

As noted, MARTE is a UML profile intended for model-based engineering of real-time and embedded systems. It consists of a set of extensions (i.e., specializations) of appropriate general UML concepts providing real-time and embedded designers and developers with first-class language constructs from their domain. Many of these extensions pertain to so-called non-functional aspects of real-time and embedded applications. Non-functional concerns of an application can be classified into two categories, quantitative and qualitative aspects. Furthermore, these extensions may be available at different levels of abstraction and, finally, they have been defined to support modeling, analysis, or both. In order to satisfy all these requirements, MARTE is structured as a hierarchy of (sub-)profiles, as depicted in the UML package diagram in figure 6.1. It has four main parts.

The topmost package, which is the foundation on which the rest of MARTE is based, consists of four basics sets of UML extensions, also called MARTE sub-profiles:

- *Non-Functional Properties Modeling (NFP)* This sub-profile provides modeling constructs for declaring, qualifying, and applying semantically well-formed non-functional aspects of UML models. The non-functional properties sub-profile supports the declaration of non-functional properties as UML data types, whereas the value specification language is used to specify the values of those types and any potential functional relationships between them. It is complemented by the Value Specification Language (VSL), which is a textual language for specifying algebraic expressions. The Value-Specification Language sub-profile is separated out in the annexes package because it was intended to be reused in other OMG profiles.
- *Time Modeling* This consists of concepts for defining time in applications, and also for manipulating the underlying time representation. The Time

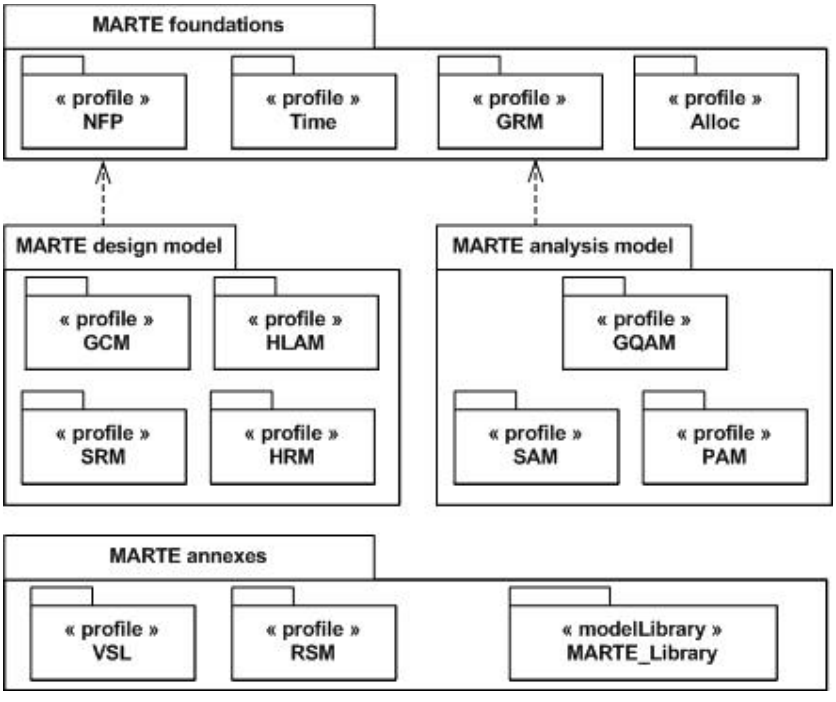

Fig. 6.1. MARTE's Architecture View

extension defined in MARTE provides support for three qualitatively different models of time: chronometric, logical, and synchronous.

- *Resource Modeling (GRM)* One important requirement with regards to real-time and embedded system modeling is to represent the set of resources underlying an application and also how the system uses them. The Generic Resource Modeling (GRM) sub-profile consists of an ontology of resources enabling modeling of common computing platforms (i.e., a set of resources on top of which an application may be allocated to be computed), and high-level concepts for specifying resource usage. The level of abstraction used here is at a general system level.
- *Allocation Modeling* This sub-profile of the foundational layer provides a set of general concepts pertaining to allocation of functionality to entities responsible for its realization. It may be either time-related allocation (i.e., scheduling) or space allocation. It also tackles the more abstract issue of refinement between models at different levels of abstraction. Note that non-functional characteristics may be attached to an allocation description (e.g., when specifying the allocation of a function to a given execution engine, it is possible to specify its worst case execution time).

Starting from these foundational concepts, MARTE is then split into two different categories of extensions: One (denoted in figure 6.1 as the "MARTE design

model") is dedicated to supporting model-based design, that is to say modeling activities related to the left branch of the classical "V" development cycle[4], whereas the other, denoted by the "MARTE analysis model" package, provides support for model-based analyses of real-time and embedded applications (i.e., more devoted to validation and optimization).

Model-based design of real-time and embedded systems with MARTE proceeds mostly in a declarative way. Hence, MARTE users may annotate their models with real-time or embedded concerns using the extensions defined within the High-Level Application Modeling sub-profile (refer to the following section that illustrates this using extracts of the MARTE specification and an example). For instance, concurrent computing units with real-time features may be denoted using an extension called «rtUnit» and, by giving specific values to its properties, they can also indicate what is the model of computation for the concurrent unit. Note also that MARTE enables component-based system engineering (either software or hardware) through its specific sub-profile called the Generic Component Model (GCM). This component model supports both message- and data-based communication schemes between components. In addition, MARTE also defines very refined concepts that enable users to describe its computing platforms, either software or hardware, in a very detailed and precise manner [18, 19, 20]. These features are supported by two sub-profiles, Software Resource Modeling (SRM) and Hardware Resource Modeling (HRM). In addition, based on its Software Resource Modeling sub-profile, MARTE includes in its annexes facilities for modeling OSEK-, ARINC-, or POSIX-compliant software computing platforms. Finally, to deal effectively with the increasing degrees of parallelism available on chips, one of the MARTE annexes includes the Repetitive Structure Modeling sub-profile, which enables compact representations of multidimensional regular structures encountered in chip design.

Model-based analysis using MARTE is enabled primarily through the extensions defined either in the Generic Quantitative Analysis Modeling profile (GQAM), or using one of its two refinements, which are dedicated to schedulability analysis [21] and performance analysis [22] respectively. The annotation mechanism used in MARTE to support model-based analyses uses UML stereotypes. These typically map the UML elements of the application or platform into corresponding analysis domain concepts, including specifications of values for properties which are needed to carry out the analyses. One of the typical use cases of MARTE described in the following section provides more detail.

In summary, MARTE was designed to cover all five categories of real-time and embedded systems listed earlier. The table below summarizes how MARTE covers the requirements identified in section 6.2 of this chapter. The left-most column denotes the different parts of MARTE: a part being either a sub-profile or a specific model library.

[4] Note that we are not advocating the "V"development cycle as the reference process model for MARTE. We are simply using it to help orient the reader.

6.4.4 An Extract of the MARTE Specification

In this section we illustrate in practical terms some of the ideas described in preceding sections. However, due to space limitations it is not possible to provide examples covering the full specification. Therefore, we will focus on the MARTE part dedicated to high-level modeling of real-time and embedded systems design. In particular, the fragment of the MARTE profile focusing on the definition of a real-time unit and a protected passive unit.

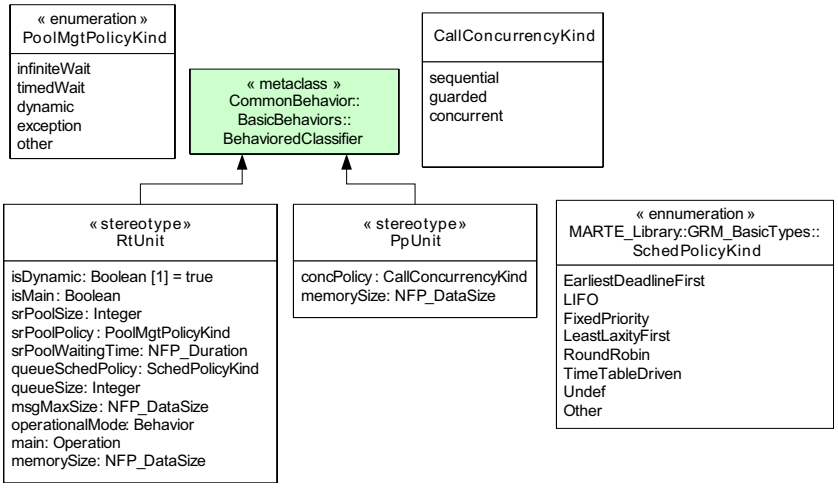

Fig. 6.2. Extract from the MARTE specification: the real-time unit and the passive protected unitmetamodels

Figure 6.2 illustrates the graphical definition of two main concepts of the MARTE specification: RtUnit and PpUnit. An RtUnit (Real-time Unit) is a unit of concurency that encapsulates in a single concept both the object and the process paradigms. This allows concurrency control to be encapsulated within a single unit. Any real-time unit can invoke services of other real-time units, or send and receive data flows to and from those units. It owns one or more schedulable resources (i.e., threads or tasks in operating system terminology). A PpUnit (Protected passive Unit), on the other hand, is used to represent data containers that can be shared between real-time units but with some form of concurrent access protection. Therefore, a PpUnit specifies its concurrency policy, via its concPolicy attribute. It does not own any schedulable resource.

The next figure describes a UML class diagram of a very simple automotive cruise control system annotated with the two of the aforementioned stereotypes. Both classes, CruiseController and ObstacleDetector, are stereotyped as real-time units. The first of these creates dynamically schedulable resources (e.g., threads) to handle the execution of its services, while the second has a pool of ten (10) schedulable resources. Both real-time units are sharing data handled by

Table 6.1. MARTE's coverages summary in terms of RTE-specific modeling language (a box with the symbol ⊂ [23] means the MARTE part provides some support for the requirement, else it is marked as ∅)

	Slicing			Layering		
	Quantitative Concerns	Qualitative Concerns	Time	Allocation	Resource	Refinement
NFP	⊂	⊂	∅	⊂	∅	⊂
Time	⊂	⊂	⊂	∅	∅	∅
GRM	∅	∅	∅	∅	⊂	∅
Alloc	∅	∅	∅	⊂	∅	⊂
GCM	∅	⊂	∅	∅	∅	∅
HLAM	⊂	⊂	⊂	∅	∅	∅
SRM	∅	∅	∅	∅	⊂	∅
HRM	∅	∅	∅	∅	⊂	∅
GQAM	⊂	∅	∅	∅	⊂	∅
SAM	⊂	∅	∅	∅	⊂	∅
PAM	⊂	∅	∅	∅	⊂	∅
VSL	⊂	⊂	∅	∅	∅	∅
CCSL	∅	∅	⊂	∅	∅	∅
RSM	∅	∅	∅	∅	⊂	∅
MARTE Library	⊂	⊂	⊂	∅	⊂	∅

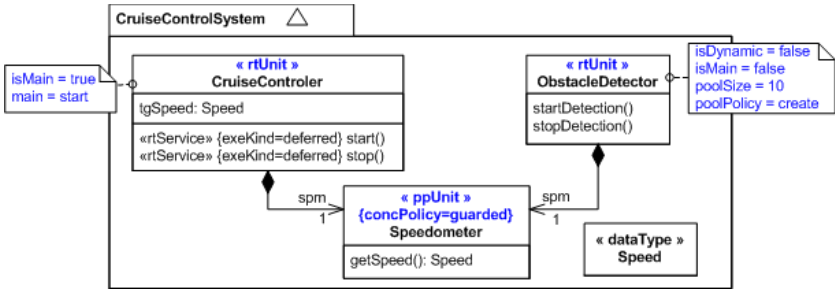

Fig. 6.3. An example of MARTE model using both stereotypes, ≪rtUnit≫ and ≪ppUnit≫

the class Speedometer. Because real-time units execute concurrently, the access to the data encapsulated within the class Speedometer needs to be protected. To do that, the class Speedometer is tagged with the ≪ppUnit≫ stereotype. Its concPolicy property is set to guarded, meaning that only one real-time unit at a time can access a feature of Speedometer, while subsequent ones arriving later are suspended until the first user releases it.

6.4.5 Typical MARTE Usage Scenarios

The modeling capabilities of MARTE are rich enough for a wide range of design approaches. This yields the flexibility to support and integrate multiple design perspectives, but also to deal with the problem of understanding and choosing among a variety of language alternatives. In both cases, there is no standard prescriptive way of using the language constructs across the development cycle. This means that individual projects or enterprises can define their own specific modeling framework and methodology that suits their needs best. We identify below a

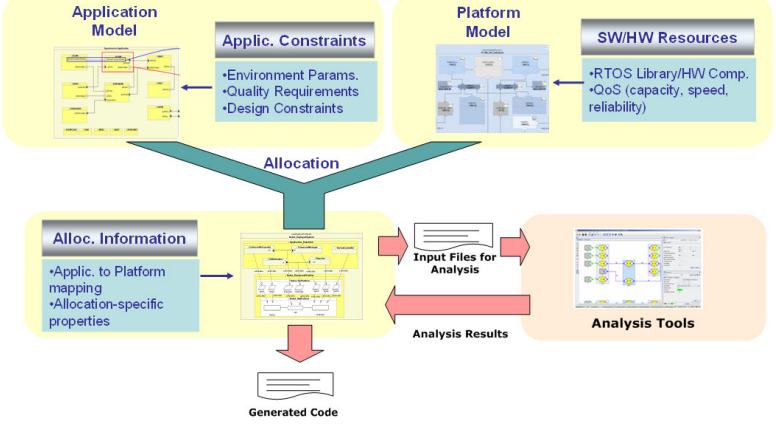

Fig. 6.4. Some typical scenarios of MARTE usage

set of representative scenarios in which using MARTE provides significant benefits. Although these scenarios certainly do not cover all possibilities, they allow us to illustrate the application of MARTE in a more focused and concrete manner.

Figure 6.4 illustrates some of example scenarios following the Y-Chart scheme [7]. For the sake of simplicity, we limit the discussion of MARTE usage here to just three simple use cases: (1) an application-oriented use case that illustrates the modeling of non-functional features of a system, (2) a platform-oriented use case that corresponds to the definition of the hardware and software resources of the system, and (3) an allocation-oriented use case that denotes how to model the deployment of an application to a platform.

Application-Oriented Use Case

The first MARTE usage scenario, depicted in the upper left-hand corner of Figure 6.4, deals with the application side and describing non-functional requirements. UML has traditionally been used to document user requirements by means of use case diagrams. Use cases follow a graphical, scenario-based approach. Although use cases may be formalized to certain degree, for example by using sequence diagrams in order to detail such usage histories, they are often criticized for a number of limitations. For instance, they are applied mainly to model functional requirements, but are not very helpful for modeling non-functional ones. One possible way of using MARTE is to add annotations that characterize non-functional constraints in use case diagrams and their corresponding sequence diagrams. This provides two important capabilities leading toward more formal requirements specification.

First, note that these non-functional requirements are specified jointly with their corresponding functional requirements. While specifying non-functional aspects is possible with UML comments, this would make their semantic relationship to the concrete functional elements hard to capture. In particular, the verification of requirements satisfaction in real-time systems is strongly dependent on the coupling between system function and timing. In MARTE, timing annotations provide semantic definitions closely related to the system behavior. For instance, one may define a jitter constraint in the arrival of an event and specify whether such an event relates either to a send, receive, or consume occurrence within a sequence diagram. Second, non-functional annotations follow a well-defined textual syntax, which is supported by the Value Specification Language of MARTE. The main advantages of this level of formalization are the ability to support automated validation, verification, and traceability, while being easily understood by stakeholders.

To model the application structure and behavior, MARTE adds key semantics to UML model elements. In particular, a common model of computation provides semantic support for the real-time object paradigm (the MARTE's High-Level Application Modeling sub-profile). This paradigm allows specifying applications at a high level of abstraction, by delegating concurrency, communication, and time-constraint aspects to a modular unit called RtUnit (see Section 6.4.4). Such units can be encapsulated in structural units (*structured components*) specifying interfaces and interactions with other structural elements. MARTE adopted the notions of port and flow from SysML [14] and extended them with the notion of message-based communications.

Platform-Oriented Use Case
MARTE can also be used to explicitly model resource patterns such as processing resources, communication buses, or power supply devices along with a set of predefined quality attributes (illustrated in the upper right-hand corner of Figure 6.4). Furthermore, the operating system (e.g., RTOS) and other software library layers can be modeled and reused for multiple application models. In this chapter we want to particularly focus on the usage of the software resource modeling capabilities of MARTE (the MARTE's Software Resource Modeling sub-profile), which deal with one of the more important open issues in Model-based Engineering: making platform models explicit.

Historically, model-driven approaches for real-time and embedded systems have focused on improving dedicated real-time and embedded platform models [6, 7] (i.e., platform models used as meta-models). At the same time, the Model-based Engineering community has proposed generic transformation languages (e.g., ATL [4] or SmartQVT [24]) which facilitate the description of dedicated bridges or transformations between meta-models. Current model-driven approaches for real-time and embedded therefore entail specific model transformations from a set of source platforms to a set of target platforms. In many case, however, the platforms are not described as explicit input models to be used in the transformation. This is a serious limitation as software platforms (e.g., RTOS, programming languages) are continuously evolving and the resulting dearth of customizable transformations hampers description of reusable generative processes.

In [25], the authors propose a model-based framework enabling explicit platform description considering the latter as an input parameter to the model transformation. The proposed approach uses the MARTE's Software Resource Modeling sub-profile for describing the specific platforms. Its principal innovation is that it avoids hard-coding the platform model in the model transformation. The main benefit is a cleaner separation of concerns within the design flow enabling easy porting to different platforms without requiring a new transformation for each case. Real-time and embedded decisions are made explicitly in the input models and not implicitly in the transformation description itself. This also improves the maintainability of the transformations. In this way, the generation process becomes more flexible, more adaptable, and more reusable for a variety of different real-time and embedded platforms. In [25], the authors also focused on the transformations dedicated to porting multitasking applications to heterogeneous computing platforms (e.g., multitasking operating systems).

Allocation-Oriented Use Case
An allocation view represents the system as a hierarchy of an application (at the top) and the software/hardware platform layers (at the bottom), as shown in the bottom of Figure 6.4. A set of MARTE stereotypes allow representing such hierarchies either using allocation/deployment or using composition relationships.

In order to generate code for a given software platform, the system model built on the real-time concepts (e.g., «rtUnit») must be allocated to a specific software platform model, as described in the preceding use case. In [26], the

authors implement a real-time framework that gives meaning to the features defined in the MARTE's High-Level Application Modeling sub-profile. This is achieved by providing execution support for realizing the behavior, communications, and message management associated with this kind of objects. The code generation facility provided by this execution framework, called ACCORD, consists of a set of transformations used to apply implementation patterns on real-time concepts, and a standard C++ code generator. A methodology (ACCORD|UML) constraints the usage of MARTE concepts and parameters to a subset that are semantically meaningful in the ACCORD execution framework.

An allocation view should incorporate the non-functional annotations that results from the running of the application on a particular hardware/software platform. Some of these annotations are directly mapped to platform properties, while others require special techniques to determine their values through either computation or simulation. For instance, fine-grained timing analyzers can help to determine the worst case execution times of relevant pieces of code, which are then used in scheduling analysis to predict end-to-end response times.

In [27], MARTE is used in practical system integration scenarios (modeled as "analysis contexts"), where multiple candidate configurations are analyzed from a timing perspective, potentially by multiple techniques. In this paper, the authors use the Value-Specification Language to specify non-functional variables that are further evaluated to make efficient design decisions. One of the benefits is that multi-objective analyses can be performed and trade-off decisions can be taken on the basis of a smart binding of exposed parameters, which are used in different analysis contexts. Each analysis technique may involve specific parameters to be evaluated. Furthermore, *sensitivity analysis* can be used at the system design level to understand the degree to which the overall results are sensitive to a given parameter.

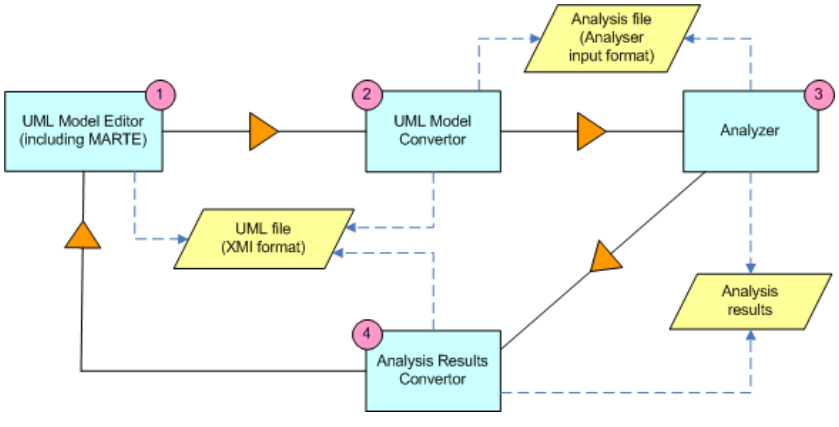

Fig. 6.5. Schema of the MARTE model-processing framework

Finally, MARTE fosters model processing in the way that it enables adding semantics to a given UML model (e.g., for code production or quantitative analysis purposes). In this context, one generic usage of MARTE enables the model-processing schema described in Figure 6.5. Note that this process can be highly automated, which, in some cases, can even eliminate the need for analysis domain experts which are often difficult to find.

6.5 Related Work

Both academia and industry have already proposed languages to support model-based development of embedded systems.

SysML [14] is an OMG standard language "for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities". The so-called Block concept of SysML is the common conceptual entity that can represent many different kinds of system elements such as electronic or software components, mechanical parts, information units, and almost any other type of structural entity in the system under interest. Blocks articulate a set of modeling perspectives enabling separation of concerns during systems design. Among these perspectives, requirement diagrams provide constructs for specifying text-based requirements and their relationships, including requirements hierarchies, as well as derivation, satisfaction, verification, and refinement relationships between requirements. Block description diagrams and internal block diagrams enable the specification of more generic interactions and phenomena than those existing just in software systems. This includes physical flows such as liquids, energy, or electrical flows. The dimension and measurement units of the flowing physical quantities can be explicitly defined. Although most behavior constructs in SysML are similar to those of UML (interactions, state machines, activities, and use cases), SysML refines some of them for modeling continuous systems and probabilities in activity diagrams. A perspective called parametric diagram allows SysML users to describe, in a graphical manner, analytical relationships and constraints, such as those described by mathematical equations. Formally, SysML is defined as a UML profile.

MARTE and SysML are complementary in many ways [28]. The MARTE component model shares the same notions of ports and flows, and additionally extends them with the concept of client/server port. This is intended to support the request-reply and publish-consume communication paradigms. In addition, there are some actions under way at the OMG to align the semantics of data and event flows, to define a common framework to specify quantities, units, dimensions and values, and to improve some aspects such as allocation and timing modelling. This will be reflected in future versions of MARTE and SysML.

AADL (Architecture Analysis and Design Language) [29] is an architecture description language standardized by SAE (Society of Automotive Engineers). AADL has been designed for the specification, analysis, and automated integration of real-time performance-critical (timing, safety, schedulability, fault

tolerant, security) distributed systems. A system modeled in AADL consists of application software components bound to execution platform components. AADL application software components are made of data, threads, and process components. AADL thread components model units of concurrent execution. A scheduler manages the execution of a thread. AADL execution platform components include processors, memory, buses and devices. Although AADL was defined as a domain-specific language from scratch, there is a MARTE rendering of AADL, as stated in Annex F of the AADL specification. This has been formalized as a subset of MARTE with some specific guidelines defined in Annex A of the MARTE specification.

In the automotive domain, AUTOSAR [30] is unquestionably the standard to specify component-based software infrastructure and it includes a standardized API. AUTOSAR's goal is to support the exchange of parts of embedded system implementation artifacts that have already been pre-designed or designed independently by different teams (e.g., by OEM's, software suppliers), without the time-consuming and costly need to re-configure, port, and re-build their code. In AUTOSAR, application models are organized in units called software components. Such components hide the implementation of the functionality and behavior they provide and simply expose well-defined connection points called ports. In particular, atomic software components are entities that support an implementation and hold behavioral entities called runnables. A runnable is an entity that can be executed and scheduled independently from other runnable entities. There is an action project funded by the European Union's Seventh Framework Program called ADAMS [31] especially dedicated to promote and show the complementarity of MARTE with other automotive and avionics standards, among which AUTOSAR is of main interest.

In addition to AUTOSAR, some of the European automotive actors have defined an architecture description language, EAST-ADL [32]. This complements AUTOSAR to cover the system level that lies outside the scope of AUTOSAR. This includes requirements modeling, feature content at the level of a vehicle description, architecture variability, functional structure of applications, middleware, plant (environment), abstract hardware architecture, and preliminary functional allocation. The ADAMS project provided some important results on the alignment of MARTE and EAST-ADL [33]. This was reflected as a set of guidelines to describe EAST-ADL-like models with a subset of MARTE concepts oriented to the design of automotive applications. Finally, note that these guidelines are part of the MARTE standard specification.

In addition to the aforementioned standards, two other non-standard approaches provide similar facilities as MARTE: MIC and Ptolemy.

Vanderbilt University's Model Integrated Computing (MIC, [34]) tool suite consists of meta-programmable model-builder (GME), model-transformation engine (UDM/GReAT), tool-integration framework (OTIF), and design space exploration tool (DESERT). This tool suite is based on specific languages for metamodeling and provides the ability to build domain-specific modeling languages. This framework is described in depth in a separate chapter of this book.

Ptolemy is a model-based tool dedicated to real-time and embedded systems [35]. This project provides support for heterogeneous modeling, simulation, and design of concurrent system. The modeling principle fostered in Ptolemy is called actor-oriented design. This relies on the concepts of models, actors, ports, parameters and channels. Actors (the core Ptolemy concept for supporting component based development) communicate with other concurrent computing actors only via their ports. Ports of two communicating actors needs to be linked via a channel. A set of communicating actors belong to a given model which may have parameters. Each model specifies a director that define its model of computation and each of the actors owned by the model will conform to the model of communication defined by the director. The key concept of actor as defined in Ptolemy was inspired by the work introduced first in 1970's by Carl Hewitt of MIT, and later formalized by Agha in [36]. The MARTE, concept of a real-time unit used for modeling real-time and embedded systems shares the same origins. More specifically, the MARTE concept was inspired by the active object concept of UML in one hand, as well as ACCORD concept of real-time object [37].

6.6 Conclusions and Perspectives

The complexity of modern real-time and embedded systems is starting to exceed the capabilities of traditional code-centered technologies. Fortunately, new model-based engineering methods have proven themselves capable of overcoming many of these limitations. These modern methods rely on intensive use of computer-based automation and take advantage of computer languages with higher-level constructs that abstract away much of the underlying implementation technology as compared to programming languages. The benefits gained from using this approach increase the closer the language is to the problem domain, which is why there is much interest in defining so-called domain-specific modeling languages (DSML). One such language is MARTE, which was specifically designed for modeling systems and phenomena in the real-time and embedded domain. It allows direct expression of domain phenomena such as time and timing mechanisms, concurrency control mechanisms, software and hardware platforms and resources, as well as precise specification of their quantitative characteristics (e.g., latency, capacity, speed and execution times).

MARTE is a profile-based language, which means that it was derived by refinement and extension of the industry-standard UML language. This allows it to reuse many existing UML tools as well as widely available UML expertise, while still retaining the expressive power and other advantages of a specialized computer language. Furthermore, MARTE itself is an industry standard, adopted and endorsed by the OMG as one of its official technology recommendations.

The domain-specific nature of MARTE enables not only more straightforward specification of real-time and embedded applications but also automated

and semi-automated engineering analyses of MARTE-based models. This important capability allows candidate designs to be objectively evaluated for key performance and quality indicators early in the development cycle, before committing full development resources. Consequently, potentially expensive design flaws and shortcomings can be detected and eliminated earlier and at far less cost compared to traditional code-based methods.

At the time of this writing, MARTE is available in its version 1.0 on the OMG web site (www.omg.org). In June 2009, a revision task force was launched by the OMG. This task force is scheduled to complete its work within one year leading to a minor revision that will incorporate minor fixes for specification issues received by the OMG in the meantime.

MARTE has already been applied extensively in practice by industry and is supported by numerous tool vendors as indicated by the list of ongoing projects that identify MARTE as central to their concerns (cf. the OMG web site dedicate to MARTE, www.omgmarte.org). But MARTE, is also an interesting research subject being explored by academia and other research institutions. The expectation is that all of these research activities will lead to new proposals for using MARTE for designing and validating real-time and embedded systems based on standards. And, of course, it will also lead to proposed enhancements and extensions to the standard itself.

References

[1] Object Management Group: UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP (2005-02-06) (February 2005)
[2] Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, Reading (1990)
[3] The Mathworks, `http://www.mathworks.fr/`
[4] Eclipse-atl, `http://www.eclipse.org/m2m/atl/`
[5] Object Management Group: UML Version v2.1.2 (2007-02-05) (February 2007), `http://www.omg.org/spec/UML/2.1.2/`
[6] Object Management Group: UML Profile for Schedulability, Performance, and Time, v1.1 (2005-01-02) (January 2005), `http://www.omg.org/technology/documents/formal/schedulability.htm`
[7] Chen, R., Sgroi, M., Martin, G., Lavagno, L., Sangiovanni-Vincentelli, A.L., Rabaey, J.: UML for Real: Design of Embedded Real-Time Systems. In: Selic, B., Lavagno, L., Martin, G. (eds.), pp. 189–270. Kluwer Academic Publishers, Dordrecht (2003)
[8] Klein, M., Ralya, T., Pollak, B., Obenza, R.: A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems. LNCS. Kluwer Academic Publishers, Dordrecht (1993)
[9] Espinoza, H.: An Integrated Model-Driven Framework for Specifying and Analyzing Non-Functional Properties of Real-Time Systems. Information Processing Letters (2007)
[10] Gray, J., Tolvanen, J.P., Kelly, S., Gokhale, A., Neema, S., Sprinkle, J.: Domain-Specific Modeling (in CRC Handbook of Dynamic System Modeling). CRC Press, Boca Raton (2007)

[11] Selic, B.: On the semantic foundations of standard UML 2.0. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 181–199. Springer, Heidelberg (2004)
[12] Shonle, M., Lieberherr, K., Shah, A.: XAspects: An Extensible System for Domain-Specific Aspect Languages. In: Object-Oriented Programming. LNCS. Springer, Heidelberg (2003)
[13] Selic, B.: A Systematic Approach to Domain-Specific Language Design Using UML. In: ISORC (2007)
[14] Object Management Group: Systems Modeling Language, Version 1.1(2008-11-01) (November 2008), http://www.omg.org/cgi-bin/doc?formal
[15] Object Management Group: UML Profile for Modeling QoS and FT Characteristics and Mechanisms, v1.1 (2006-05-02) (Mai 2006), http://www.omg.org/spec/QFTP/1.1/
[16] Object Management Group: UML Profile for MARTE, Beta 2 (2008-06-09) (Juni 2008), http://www.omg.org/cgi-bin/doc?ptc/
[17] Object Management Group: Policies and Procedures, Version 2.7 (2008-06-01) (Juni 2008), http://www.omg.org/cgi-bin/doc?pp
[18] Thomas, F., Gérard, S., Delatour, J., Terrier, F.: Software Real-Time Resource Modeling. In: Proceedings of the International Conference Forum on Specification and Design Languages (FDL). Information Processing Letters (2007)
[19] Taha, S., Radermacher, A., Gerard, S., Dekeyzer, J.L.: An Open Framework for Hardware Detailed Modeling. In: IEEE Proceedings of SIES. Information Processing Letters (2007)
[20] Taha, S., Radermacher, A., Gerard, S., Dekeyzer, J.L.: Marte: Uml-based hardware design from modeling to simulation. In: Proceedings of the international conference forum on specification and design languages (fdl). Information Processing Letters (2007)
[21] Tawhid, R., Petriu, D.C.: Integrating Performance Analysis in the Model Driven Development of Software Product Lines. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 490–504. Springer, Heidelberg (2008)
[22] Espinoza, H., Medina, H.J., Dubois, H., Gerard, S., Terrier, F.: Towards a UML-based, Modeling Standard for Schedulability Analysis of Real-time Systems. In: International Workshop MARTES, MoDELS/UML 2006 (2006)
[23] Selic, B.: From Model-Driven Development to Model-Driven Engineering. LNCS. Springer, Heidelberg (2007)
[24] (Smartqvt), http://smartqvt.elibel.tm.fr/
[25] Thomas, F., Delatour, J., Gérard, S., Terrier, F.: Toward a Framework for Explicit Platform Based Transformations. In: 11th IEEE International Symposium on Object-oriented Real-time distributed Computing. LNCS. Springer, Heidelberg (2008)
[26] Mraidha, C., Tanguy, Y., Jouvray, C., Terrier, F.: Gerard: Presented in Workshop UML&AADL 2008 and Published in Proceeding of the 13th IEEE International Conference on Engineering of Complex Computer Systems. LNCS. Springer, Heidelberg (2008)
[27] Espinoza, H., Servat, D., Gérard, S.: Leveraging Analysis-Aided Design Decision Knowledge in UML-Based Development of Embedded Systems. LNCS. Springer, Heidelberg (2008)

[28] Espinoza, H., Selic, B., Cancila, D., Gérard, S.: Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems. In: ECMDA 2009, Published in Proceeding of the Conference (Model Driven Architecture- Foundations and Applications). LNCS, pp. 98–113. Springer, Heidelberg (2009)
[29] SAE: Architecture Analysis and Design Language (AADL) Annex Volume 1: Annex A: Graphical AADL Notation, Annex C: AADL Meta-Model and Interchange Formats, Annex D: Language Compliance and Application Program Interface Annex E. LNCS. Springer, Heidelberg (2006)
[30] Autosar, http://www.autosar.org/
[31] Adams-Project, http://www.adams-project.org/
[32] East-Adl, http://www.east-adl.org/
[33] Espinoza, H., Gérard, S., Lönn, H., Kolagari, R.T.: Harmonizing MARTE, EAST-ADL2, and AUTOSAR to Improve the Modelling of Automotive Systems. In: Presented in the Workshop STANDRT, Autosar (2009)
[34] (ISIS,MIC Tool Distribution), http://www.isis.vanderbilt.edu/Projects/gme/
[35] Lee, E.A.: Overview of the Ptolemy Project, Technical Memorandum No. UCB/ERL M03/25 (2003)
[36] Agha, G.: Actors: a model of concurrent computation in distributed system. MIT Press, Cambridge (1986)
[37] Terrier, F., Fouquier, G., Bras, D., Rioux, L., Vanuxeem, P., Lanusse, A.: A real time object model. In: TOOLS Europe 1996 (1996)