MARTE/CCSL for Modeling Cyber-Physical Systems

Frédéric Mallet[1,2,3]

[1] Univ. Nice Sophia Antipolis, I3S UMR 7271 CNRS, 06900 Sophia Antipolis, France
[2] INRIA Sophia Antipolis Méditerranée, 06900 Sophia Antipolis, France
[3] East China Normal University, Software Engineering Institute, Shanghai, China
`Frederic.Mallet@unice.fr`

Abstract. Cyber Physical Systems (CPS) combine digital computational systems with surrounding physical processes. Computations are meant to control and monitor the physical environment, which in turn affects the computations. The intrinsic heterogeneity of CPS demands the integration of diverse models to cover the different aspects of systems. The UML proposes a great variety of models and is very commonly used in industry even though it does not prescribe a particular way of using those models together. The MARTE profile proposes a set of extensions to UML in a bid to allow for the modeling of real-time and embedded systems (RTES). Yet CPS are a wider class of systems than mere RTES. Hence a legitimate question arises as whether MARTE can be used for CPS as well. This paper illustrates some possible uses of MARTE to model CPS and uses logical clocks as a way to bring together the different models.

Keywords: Heterogeneous models, Logical clocks, Fuel management system

1 Introduction

Cyber-Physical Systems combine digital computational systems with surrounding physical processes. Computations are meant to control and monitor the physical environment, which in turn affects the computations. The intrinsic heterogeneity of CPS demands the integration of diverse models to cover the different aspects of systems. The Unified Modeling Language (UML) proposes a great variety of models and is very commonly used in industry even though it does not prescribe a particular way of using those models together. The profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE [25]) proposes a set of extensions to UML in a bid to allow for the modeling of real-time and embedded systems. Yet CPS are a wider class of systems than mere RTES. Hence a legitimate question arises as whether MARTE can be used for CPS as well. This paper illustrates some possible uses of MARTE to model CPS. It deliberately leaves aside the traditional aspects well covered in the literature on traditional discrete embedded systems. The central concepts put forward is the notion of logical clock introduced by Lamport and used ever since in many applications. The logical clocks have been brought into the UML world through MARTE time subprofile. They are used in this chapter as handles attached to various modeling elements (whether structural or behavioral, related to hardware or software). The Clock Constraint Specification Language (CCSL), initially defined in an annex of MARTE, is used to constrain/animate these handles (and the associated modeling elements) to specify the expected behavior along with some expected interactions and synchronizations amongst the diagrams.

Section 2 starts by enumerating the main, generally accepted, characteristics of cyber-physical systems. Then Section 3 introduces the UML profile for MARTE by insisting on aspects previously highlighted as important for modeling cyber-physical systems. In particular, after a general overview it elaborates on the time and allocation subprofiles before introducing CCSL. Section 4 illustrates each aspect highlighted using a common example of fuel management system borrowed from previous works on CPS.

2 Main Characteristics of CPS

The main characteristics of Cyber Physical Systems and main design challenges have been identified some years ago [17, 18]. We focus here only on proposing a modeling framework based on standard notations to capture some most important aspects. We present our view focusing on what is most relevant to understand our approach. The following characteristics are further discussed in this section and illustrated in Section 4.

CPS are:

- heterogeneous, in the sense that they combine various models of computations relying on both discrete and continuous time abstractions;
- platform-aware and resource-constrained, and thus the software depends on various non-functional properties imposed by the platform;
- time-sensitive and often safety-critical;
- widely distributed with heterogeneous interconnects.

CPS are first and foremost complex systems and as such designing them requires several models, usually hierarchical, to fully capture the different aspects and views, whether structural or behavioral. Structural models include a description of the components or blocks of the systems and of the communication media involved. Behavioral models include hierarchical state machines and data-flow streaming diagrams. Expected or faulty interactions with the surrounding environment can be captured as a set of use cases or requirements that correspond to positive or negative scenarios. Such models are usually called **heterogeneous** in the sense that they combine different models, each of which may follow a different *model of computation*.

CPS also have the main characteristics of embedded systems, which are usually **platform-aware**. Contrary to standard software engineering, embedded system design depends a lot on the execution platform on which the system should execute, be it a system-on-a-chip (SoC), with multiple computing resources and a complex memory hierarchy, or a wide scale distributed system, with potentially all the variety of interconnects and communication media. This awareness of the platform makes it important to account for how and when the available resources are accessed or ‘consumed’, considering together both the spatial and temporal dimensions. The spatial dimension is not only about how much resource is available but also about where the resources are physically located in the system relative to each other. How much resource is available is indeed easy since it is usually given by the technology used and the targeted selling price. However, how the resources are used makes all the difference between two a priori equivalent products. The spatial dimension encompasses the interconnect topology, *i.e.,* physical parallelism available, but also and more importantly where the data and programs are allocated. Indeed, the distance between the data memory and the computing resource that executes the program largely impact the fetching time that may potentially largely exceed the computing time. Then, this spatial distribution requires to perform the temporal scheduling of both the execution of programs and the routing of data from memory to computing resources, forth and back. This leads to *logical concurrency* coming both from the physical parallelism and the inherent data and control dependencies of the application. CPS are therefore **resource-constrained** and **time-sensitive** systems. Even though the resources (memory size, computing power) are not necessarily as scarce as they used to be, nevertheless finding the right trade-off between the resource usage, the computation speed and the cost makes it a multi-criteria optimization problem difficult to solve. The cost is not only measured in terms of money, but this includes all kinds of additional extra functional properties (like power, energy, thermal dissipation), also called **non-functional properties**.

More than being mere embedded systems, CPS are usually made of multiple interconnected embedded subsystems, some of which are computing devices and some other being physical devices. This requires some abilities to describe **heterogeneous interconnects**, while simpler embedded systems usually only rely on homogeneous communication structures. Being made of several computing devices also constitute a big step since it requires to model the whole system as a closed model, with software, devices but also with the environment and the expected **continuous interactions** with this environment. In standard software development, the environment is by definition outside the system to be developed. Close loop systems have been modeled for several years with tools and techniques to find approximate solutions to differential equations are well established. However, the integration with discrete models still causes problems in many tools and each of them proposes ad-hoc solutions. A seamless integration with UML models is still to be proposed.

Finally, cyber-physical systems are often big and often interacts directly with users that are not even aware of the computer. The size is an aggravating factor since a single system concerns potentially millions of people (smart cities, intelligent transportation systems. . .). It means some CPS are **safety-critical**, just like embedded systems but at a larger scale. It then increases the demand to have sound models along with verification tools. Sometimes, they also require certification tools to be accredited and allowed to be use in public environments. However, we do not address at all the certification issue here.

To summarize, CPS demand an integration between continuous models, classical state-based or data-flow models, hardware descriptions, non-functional constraints. UML offers a tool-neutral non-proprietary solution that already contains most of the required notations. However, those notations need to be tailored to capture specific aspects of CPS (time, non-functional properties, continuous models). Both MARTE and SysML offer some extensions dedicated to these goals, and we discuss here some examples of useful features of either MARTE or SysML to model CPS. These notations also need to come with adequate, not tool-specific, **explicit semantics** if we are to address safety-critical issues. In this paper, the semantics is given using the Clock Constraint Specification Language [21], a formal specification language defined as a companion of MARTE in an annex.

3 The UML Profile for MARTE

This section gives a basic introduction to the MARTE profile. It gives a general overview and focus on some specific aspects that are further developed in the following sections, like the time and allocation subprofile. It also gives an introduction to the Clock Constraint Specification Language defined in an annex of MARTE since CCSL is used in this chapter to describe the interactions between the models. This is done by specifying some causal relationships between the models and the way they synchronize. This section does not cover the whole MARTE specification and a comprehensive review of MARTE along with some complementary information on how to use it for modeling cyber physical systems can be found in a dedicated book [28].

3.1 Overview

UML and its extensions. The Unified Modeling Language [26] is a general-purpose modeling language specified by the Object Management Group (OMG). It proposes graphical notations to represent all aspects of a system from the early requirements to the deployment of software components, including design and analysis phases, structural and behavioral aspects. As a general-purpose language, it does not focus on a specific domain and maintains a weak, informal semantics to widen its application field. However, when targeting a specific application domain and especially when building trustworthy software components or for critical systems where life may be at stake, it becomes necessary to extend the UML and attach a formal semantics to its model elements. The simplest and most efficient extension mechanism provided by the UML is through the definition of profiles. A UML profile adapts the UML to a specific domain by adding new concepts, modifying existing ones and defining a new visual representation for others. Each modification is done through the definition of annotations (called stereotypes) that introduce domain-specific terminology and provide additional semantics. However, the semantics of stereotypes must be compatible with the original semantics (if any) of the modified or extended concepts, *i.e.,* the base metaclass.

The UML profile for MARTE [25] extends the UML with concepts related to the domain of real-time and embedded systems. It supersedes the UML profile for Schedulability, Performance and Time (SPT [24]) that was extending the UML 1.x and that had limited capabilities. UML 2.0 has introduced a simple (or even simplistic) model of time and has proposed several new extensions that made SPT unusable. Therefore MARTE has been defined to be compatible with UML Simple Time model and now supersedes SPT as the official OMG specification. SysML [10] is another extension dedicated to systems engineering. We use some notations coming from SysML and we introduce those notations when required. The task forces of MARTE and SysML have synchronized their effort to allow for a joint use of both profiles.

The remainder of this subsection gives an overview of the MARTE, which is made up of three parts: *Foundations*, *Design* and *Analysis*.

MARTE Foundations. The foundation part of MARTE is itself divided into five chapters: CoreElements, NonFunctionalProperties (NFP), Time, Generic Resource Modeling (GRM) and Allocation.

CoreElements define configurations and modes, which are key parameters for analysis.

In real-time systems, preserving the non-functional (or extra-functional) properties (power consumption, area, financial cost, time budget. . .) is often as important as preserving the functional ones. The UML proposes no mechanism at all to deal with non-functional properties and relies on mere string for that purpose. The NFP subprofile offers mechanisms to describe the quantitative as well as the qualitative aspects of properties and to attach a unit and a dimension to quantities. It defines a set of predefined quantities, units and dimensions and supports customization. NFP comes with a companion language called Value Specification Language (VSL) that defines the concrete syntax to be used in expressions of non-functional properties. VSL also recommends syntax for user-defined properties.

Time is often considered as an extra-functional property that comes as a mere annotation after the design. These annotations are fed into analysis tools that check the conformity without any actual impact on the functional model: *e.g.,* whether a deadline is met, whether the end-to-end latency is within the expected range. Sometimes though, time can also be of a functional nature and has a direct impact on what is done and not only when it is done. All these aspects are addressed in the time chapter of MARTE. The next section elaborates on the time subprofile.

GRM chapter provides annotations to capture the available resources on which the applicative part shall be deployed.

The allocation chapter gives a SysML-compatible way to make this deployment. In MARTE, we use the wording allocation since the UML deployment usually implies (in people's minds) a physical distribution of a software artifact onto a physical node. Allocation in MARTE goes further. It encompasses the physical distribution of software onto hardware, but also of tasks onto operating system processes, and, more importantly, it covers the temporal distribution (or scheduling) of operating parts that need to share a common resource (*e.g.,* several tasks executing on a single core processor, distributed computations communicating through an interconnect). This subprofile is further described in subsection 3.3.

MARTE for design. The design part has four chapters: High Level application modeling, Generic component modeling, Software Resource Modeling, and Hardware Resource Modeling.

The first chapter describes real-time units and active objects. Active objects depart from passive ones by their ability to send spontaneous messages or signals, and react to event occurrences. Normal objects, the passive ones, can only answer to the messages they receive or react on event occurrences. The three other chapters provide a support to describe resources used and in particular execution platforms on which applications may run. A generic description of resources is provided, including stereotypes to describe communication media, storages and computing resources. Then this generic model is refined to describe software and hardware resources along with their non-functional properties.

MARTE for analysis. The analysis part also has a chapter that defines generic elements to perform model-driven analysis on real-time and embedded systems.

This generic chapter is specialized to address schedulability analysis and performance analysis.

The chapter on schedulability analysis is not specific to a given technique and addresses various formalisms like the classic and generalized Rate Monotonic Analysis (RMA), holistic techniques, or extended timed automata. This chapter provides all the keywords usually required for such analyzes.

Finally, the chapter on performance analysis, even if somewhat independent of a specific analysis technique, emphasizes on concepts supported by the queueing theory and its extensions.

MARTE usages and extensions. MARTE extends the UML for real-time and embedded systems but should be refined by more specific profiles to address specific domains (avionics, automotive, silicon) or specific analysis techniques (simulation, schedulability, static analysis). Because MARTE targets different domains and/or different analysis techniques, the time model of MARTE is rich and combines physical and logical clocks. This is further explained in the next subsection.

We have briefly reviewed here the whole specification of MARTE. However, MARTE is not expected to be used as a whole on a single specification, as his usage chapter states. It is expected to be the base of several complementary methodologies that cover different aspects of a system. This chapter covers one aspect and proposes a partial usage to capture important features. It is not intended to offer a comprehensive cover of all aspects.

3.2 Time in MARTE

This subsection only gives a very brief introduction to the time model of MARTE. See [3, 2, 20] for more explanations and examples.

Time in SPT is a metric time with implicit reference to physical time. As a successor of SPT, MARTE supports this model of time. UML 2, issued after SPT, has introduced a model of time called *SimpleTime*. This model also makes implicit references to physical time, but is too simple for use in real-time applications, and was initially devised to be extended in dedicated profiles.

MARTE goes beyond SPT and UML 2. It adopts a more general time model suitable for system design. In MARTE, Time can be physical, and considered as continuous or discretized, but it can also be logical, and related to user-defined clocks. Time may even be multiform, allowing different times to progress in a non-uniform fashion, and possibly independently to any (direct) reference to physical time. In MARTE, time is represented by a collection of *Clocks*. The use of word *Clock* comes from vocabulary used in the synchronous languages. They may be understood as a specific kind of events on which constraints (temporal, hence the name, but also logical ones) can be applied. Each clock specifies a totally ordered set of instants, *i.e.*,, a sequence of event occurrences. There may be dependence relationships between the various occurrences of different events. Thus this model, called the MARTE time structure, is akin to the Tagged Systems [16]. To cover continuous and discrete times, the set of instants associated with a clock can either be dense or discrete.

Figure 1 shows the main stereotypes introduced by MARTE Time subprofile.

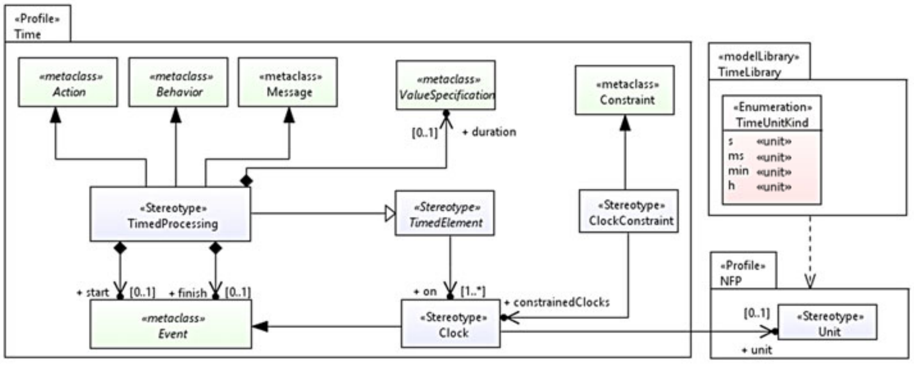

Fig. 1. Excerpt of MARTE Time subprofile

Stereotype Clock is one foundational stereotype that extends UML metaclass Event. A Clock carries specific information such as its actual unit, and values of quantitative (resolution, offset...) or qualitative (time standard) properties, if relevant.

TimedElement is another stereotype introduced in MARTE. A timed element is an abstract stereotype that associates at least one clock with a modeling element. TimedProcessing is a specialization of TimedElement, which extends the UML metaclasses Action, Behavior and Message. It defines a start and a finish event for a given action/behavior/message. These events (which are usually clocks) specify when the action starts or when it finishes. TimedProcessing also specifies the duration of an action. Duration is also measured on a given logical or physical clock. In a MARTE model of a system, stereotype TimedElement or one of its specializations is applied to model elements which have an influence on the specification of the temporal behavior of this system. The expected behavior of such TimedElements is controlled by a set of ClockConstraints. Those constraints specify dependencies between the various occurrences of events. CCSL, which is further described in the following subsection, can be used to specify those constraints formally.

The MARTE Time subprofile also provides a model library named TimeLibrary. This model library defines the enumeration TimeUnitKind which is the standard type of time units for chronometric clocks. This enumeration contains units like s (second), its submultiples, and other related units (*e.g.,* minute, hour). The library also predefines a clock called IdealClock, which is a dense chronometric clock with the second as time unit. This clock is assumed to be an ideal clock, perfectly reflecting the evolutions of physical time. It should be imported in user's models with references to physical time concepts (*e.g.,* frequency, physical duration).

3.3 Allocation in MARTE

Since embedded systems are platform-aware, one need a way to map the elements of the application onto the execution platform. This aspect is specifically addressed by the allocation subprofile of MARTE, which is further described in this subsection.

The wording *Allocation* has been retained to distinguish this notion from UML Deployment diagrams. Deployments are reserved to deploy artifacts (*e.g.,* source code, documents, executable, database table) onto deployment targets (*e.g.,* processor, server, database system). The MARTE allocation is much more general than that. For instance, it is meant to represent the allocation of a program onto a system thread, or of a process onto a processor core. More generally, it is used to represent the association of an element (action, message, algorithm) that consumes a resource onto the consumed resource (processing unit, communication media, memory). Wordings 'mapping' or 'map' have also been discarded since they very often refer to a function and then map one input from a domain to one single output in the co-domain. The allocation process, however, is an n-to-m association. Take for instance, a bunch of tasks that need to be scheduled on several cores.

Note that the wording *execution platform* has been preferred to 'architecture' or 'hardware'. Indeed, architecture is a way to describe the structure of a system, while an execution platform contains both structural and behavioral parts. On the other hand, the execution platform is not necessarily a piece of hardware. It can be a piece of software, a virtual machine, a middleware, an operating system or a mixed platform that combines software and hardware intellectual properties (IPs).

Finally, it is also important to note that this notion of allocation is common between MARTE and SysML in a bid to ease the combination of the two profiles. In particular, for CPS both profiles must be used jointly [28].

Figure 2 shows the two main stereotypes of the subprofile, Allocate and Allocated. Allocate represents the allocation itself, while Allocated may be used on both sides to mark either the element that is allocated or the resource onto which an element is allocated. The property nature is meant to distinguish two kinds

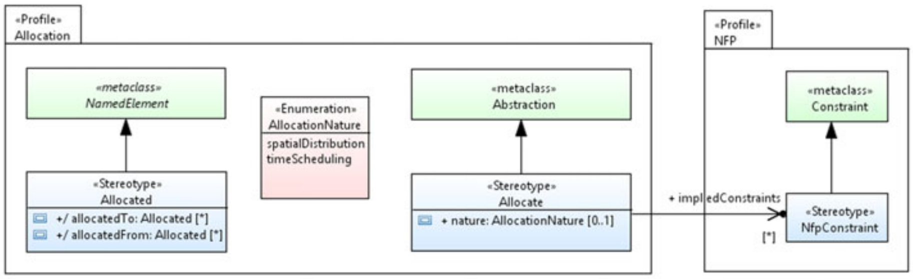

Fig. 2. Exerpt of MARTE Allocation subprofile

of possible allocations: *spatial* and *temporal*. Typically, when messages are allocated onto a buffer or a memory, this is a spatial allocation. Indeed, the message will consume/use some cells of the memory. However, when two tasks are allocated onto a processing unit, this is a temporal allocation (scheduling); It means the two tasks must be scheduled to avoid resource conflicts. When a program is allocated onto a processor, it can be seen both as spatial and temporal; Spatial because the program consumes disk and memory resources, temporal because while this program executes, another one cannot execute simultaneously. The allocation usually implies constraints that describe precisely the impact (or cost) of the allocation on the non-functional properties. This is why there is an association to a specific MARTE stereotype called NfpConstraint, *i.e.,* to capture the constraints implied by the allocation in terms of memory consumption, power consumption, execution time, for instance.

Figure 3 shows an example of allocation borrowed from [23]. The upper part depicts an algorithm as an UML activity diagram. Two input values are captured as $in1$ and $in2$. Those values are processed respectively by actions $step1$ and $step2$ producing two intermediate values. Those two intermediate values are used by $step3$ to process the final result. The bottom part shows a possible execution platform as a composite structure diagram. It contains two tasks (or threads, or processes) that communicate through a shared memory. Each task is scheduled at a different frequency.

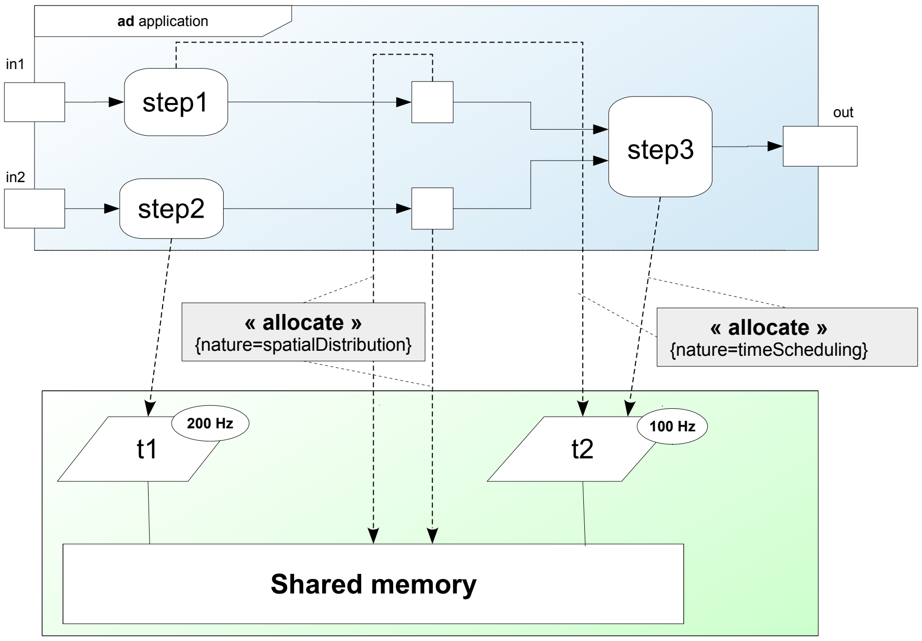

Fig. 3. Example of Allocation: spatial allocation vs. temporal scheduling

MARTE allocation is used in two different ways. The first way corresponds to a temporal scheduling operation. Indeed, the three steps from the application part are scheduled on the tasks of the execution platform. In this example, $step1$ and $step3$ are allocated to task $t2$, while $step1$ is allocated to task $t1$. The second kind of allocation is more of a spatial nature. It allocates the intermediate values to the shared memory. Even though there are different in nature, both kinds of allocations imply some constraints (temporal and non-functional) that should be checked and captured as $NfpConstraints$. The spatial distribution actually implies that one must check whether there is enough space available in the shared memory for the two communications to overlap. The temporal scheduling implies that the two actions $step1$ and $step2$ must not execute simultaneously. Both kinds of constraints can be captured and analyzed with CCSL [23].