Modeling Time(s)

Charles André, Frédéric Mallet, Robert de Simone

Modeling Time(s)

Charles André, Frédéric Mallet, and Robert de Simone

I3S, Université de Nice-Sophia Antipolis, CNRS, F-06903 Sophia Antipolis
INRIA, F-06902 Sophia Antipolis
{candre,fmallet,rs}@sophia.inria.fr

Abstract. Time and timing features are an important aspect of modern electronic systems, often of embedded nature. We argue here that in early design phases, time is often of logical (rather than physical) nature, even possibly multiform. The compilation/synthesis of heterogeneous applications onto architecture platforms then largely amounts to adjusting the former logical time(s) demands onto the latter physical time abilities. Many distributed scheduling techniques pertain to this approach of "time refinement".
We provide extensive Time and Allocation metamodels that open the possibility to cast this approach in a Model-Driven Engineering light. We give a UML representation of these concepts through two subprofiles, parts of the foundations of the forthcoming OMG UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE). Time modeling also allows for a precise description of time-related entities and their associated timed properties.

Key words: UML profile, real-time embedded.

The original publication is available at www.springerlink.com[1].

1 Introduction

Modeling of Time should be a central concern in Model-Driven Engineering for Real-Time Embedded systems. Nevertheless, (too?) many modeling frameworks consider Time annotations as to be considered in timing/schedulability/performance, and accordingly build uninterpreted stereotypes and label locations with insightful names only for the future analysis tool (and no meaning at all for the time augmented profile). Given that the default operational semantics of the UML is inherently *untimed*, and rightfully so since there is no Time information in the ground metamodel, one can reach the situation where the same model can be understood differently depending on whether it is considered from the UML causality model or the intended timed analysis viewpoint. Our primary goal here is to lay the foundation for a Time model which could be *deeply* embedded in UML as a profile allowing the subsequent clean and precise definition of a *timed causality* model enforcing timed operational semantics of events and actions.

[1] http://dx.doi.org/10.1007/978-3-540-75209-7_38

Following some works on dedicated *Models of Computation and Communication* (MoCCs) for real-time embedded systems [1–3], we view *Time* in a very broad sense. It can be *physical*, and considered as *continuous* or *discretized*, but it can also be *logical*, and related to user-defined clocks. For instance, durations could be counted in terms of numbers of execution steps, or clock cycles on a processor or even more abstract time bases, without a strong relation to the actual physical duration (which may not be known at design time, or fluctuate, or be a parameter that allows the same model to be instantiated under different contexts and speeds). With modern embedded designs where, for low-power reasons, the actual processor clock can be shut down and altered at times, such usage of logical time in the application design will certainly become customary. In our approach, time can even be *multiform*, allowing different time threads to progress in a non-uniform fashion.

This approach looks certainly non-standard, but is getting increasing interest from a number of directions. A mostly untimed concurrent application can be considered as comprising several unrelated (or loosely coupled) time threads (thereafter called "clocks", not to be confused with the physical measurement device which we will never consider). The process of allocating the various operations/functions/actions of such a concurrent model onto an embedded execution platform comprises aspects of spatial distribution and temporal scheduling. This is accomplished by resolving the mutual sets of timing constraints imposed by the designer of the time scales of the application, the target architecture, and the real-time requirements to be met. We call the approach one of *Time refinement*.

A number of existing transformation techniques can be cast in this framework. Nested loop scheduling and parallelization [4, 5] in high-performance computing, software pipe-lining, SoC synthesis phases from so-called *transactional level (TLM)* down to cycle-accurate *RTL* level, to mention a few. In all cases, the purpose is to progressively refine the temporal structure, which starts with a number of degrees of freedom, to attain a fully scheduled and precisely cycle-allocated version, with predictable timing. In that sense our model allows, and it is in fact its primary aim, to describe formal clock relations in a simple mathematical way.

We provide a UML model for Time in its different guises, physical/logical, dense/discrete, single/multiple, and some useful basic operators and relations to combine timed events or clocks. From this set of primitives, we hope to build explicit representation of MoCCs, and to provide a Timed causality model to endow the timed models with a timed semantics, according to the one that would be considered by analysis tools. When the relation are simple enough (periodic or regular), the system of contraints imposed by these relations can be solved, and the schedule itself becomes an explicit modeling element, traceable to the designer. In other, more complex cases, the constraints embody a given scheduling policy, which can be analyzed with corresponding schedulability analysis techniques when applicable.

After describing some existing time and allocation models (Section **??**), Section **??** introduces our contribution, the MARTE[2] subprofiles for time and allocation. Section **??** briefly illustrates their use.

2 Existing time and allocation models

2.1 Time modeling

This subsection focuses on time models and time-related concepts in use in the UML and some of its profiles.

UML In UML [6] Time is seldom part of the behavioral modeling, which is essentially untimed (by default, events are handled in the same order as they arrive in event handlers). UML describes two kinds of behaviors [7]: the intra-object behavior—the behavior occurring within structural entities—and the inter-object behavior, which deals with how structural entities communicate with each other. The CommonBehaviors package defines the relationship between structure and behavior and the general properties of the behavior concept. A subpackage called SimpleTime adds metaclasses to represent *time* and *duration*, as well as actions to observe the passing of time. This is a very simple time model, not taking account of problems induced by distribution or by clock imperfections. In particular the UML *causality model*, which prescribes the dynamic evaluation mechanisms, does never refer to time (stamps). Instead, the UML specification document explicitly states that *"It is assumed that applications for which such characteristics are relevant will use a more sophisticated model of time provided by an appropriate profile"*. Our contribution can be seen as providing the means for building such sophisticated time models.

SPT The UML Profile for Schedulability, Performance, and Time (SPT) [8] aimed at filling the lacks of UML 1.4 in some key areas that are of particular concern to real-time system designers and developers. SPT introduces a *quantifiable* notion of time and resources. It annotates model elements with quantitative information related to time, information used for timeliness, performance, and schedulability analyses.

SPT only considers *(chrono)metric* time, which makes implicit reference to physical time. It provides time-related concepts: concepts of instant and duration, concepts for modeling events in time and time-related stimuli. SPT also addresses modeling of timing mechanisms (clocks, timers), and timing services. SPT, which relies on UML 1.4, had to be aligned with UML 2.1. This is one of the objectives of the MARTE profile, presented in Section **??**.

[2] a preliminary version is available at www.promarte.org

Non OMG profiles Several "unofficial" UML profiles are also considering time modeling. We mention a few, developed for different purposes, as work related to ours.

EAST-EEA is an ITEA project on Embedded Electronic Architecture [9]. It provides a development process and automotive-specific constructs for the design of embedded electronic applications. Temporal aspects in EAST are handled by requirement entities. The concepts of Triggers, Period, Events, End to End Delay, physical Unit, Timing restriction, can be applied to any behavioral EAST elements. It is compliant with UML2.0, the intent is to deliver a UML2 profile.

The UML profile Omega-RT [10] focuses on analysis and verification of time and scheduling related properties. It is a refinement of the SPT profile. The profile is based on a specific concept of event making it easy to express duration constraints between occurrences of events. The concept of *observer*, which is a stereotype of state machine, is a convenient way for expressing complex time constraints. It would have to be aligned with UML2.0.

Summary The abovementioned profiles introduce relationships between Time and Events or Actions. They annotate the UML model with quantitative information about time. None consider logical and multiform time.

2.2 Allocation models

These are concerned with the mapping of application elements onto architectural platform resources and services. The following frameworks are currently untimed. It is in fact our main goal that a Time Model can be used to select and optimize such mapping according to the timing demands of both sides (and possibly additional real-time requirements).

UML deployments UML deployments consist in assigning concrete software elements of the physical world (artifacts) to nodes. Nodes can represent either hardware devices or software execution environments. Artifacts are physical piece of information—a file or a database entry—and model elements are stored in resources specified by artifacts. The MARTE allocation mechanism is complementary to the UML deployment mechanism, the differences are described in section **??**.

SysML allocation SysML[11] provides a mechanism to represent, at an abstract level, cross-associations among model elements with the broadest meaning. A SysML allocation is expected to be the precursor of more concrete relationships. It differentiates three of the many possible and not exclusive categories: behavior, flow and structure allocations. Behavior allocations separate the functions from the structure; they provide a way to allocate a behavior to a behavioral feature. Flow allocations have many usages; they include allocations of activity transitions (SysML flows) to connectors of structured activities (SysML blocks).

Structure allocations acknowledge the needs for a mapping relation of logical parts to more physical ones. The MARTE allocation is inspired from the SysML allocation and the differences are described in section **??**. One reason for this choice is that we want to be able to define, in the most convenient way, how various durations and clock streams are connected in the course of the allocation. This can easily fit some of SysML *constraints/parametrics* and *requirements* modeling features, which were originally used to model physical constraints or uninterpreted requirement engineering information respectively.

2.3 Timed allocation models

We believe that suitable models for real-time and embedded systems design and analysis should support both time and allocation. We give here a brief insight of the Society of Automotive Engineer(SAE)'s Architecture Analysis & Design Language(AADL) standard [12].

The temporal semantics of AADL concepts is defined using "hybrid automata". These automata are hierarchical finite state machines with real-valued variables that denote the time. Temporal constraints, expressed as state invariants and guards over transitions, define when the discrete transitions occur. Concurrent executions are modeled using threads managed by a scheduler. The dispatch protocol (periodic, aperiodic, sporadic and background) determines when an active thread executes its computation. AADL supports multiform time models. However, it lacks model elements to describe the application itself, independently of the resources. UML activities allow for a description of the application, actions executed sequentially or concurrently, without knowing, at first, whether actions are executed by a periodic thread or a subprogram. This important information is brought by an orthogonal process, the allocation. After several iterations, analysis, the threads are eventually deployed (or bound) to the execution platform.

AADL offers a binding mechanism to assign software components (data, thread, process, etc.) to execution platform components (memory, processor, buses, etc.). Each software component can define several possible bindings and properties may have different values depending on the actual binding. This binding mechanism encompasses both the UML deployment and the MARTE allocation, while sometimes it is useful to separate the two concepts.

3 MARTE

MARTE is a response to the OMG RFP to provide a UML profile for real-time and embedded systems [13]. MARTE is a successor of SPT, aligned with UML 2, and with a wider scope. MARTE introduces a number of new concepts, including time and allocation concepts, which are central to this paper.

3.1 MARTE time model

Time in SPT is a *metric* time with implicit reference to physical time. As a successor of SPT, MARTE supports this model of time. However, MARTE goes beyond this quantitative model of time and adopts more general time models suitable for system design. In MARTE, Time can be *physical*, and considered as *dense* 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.

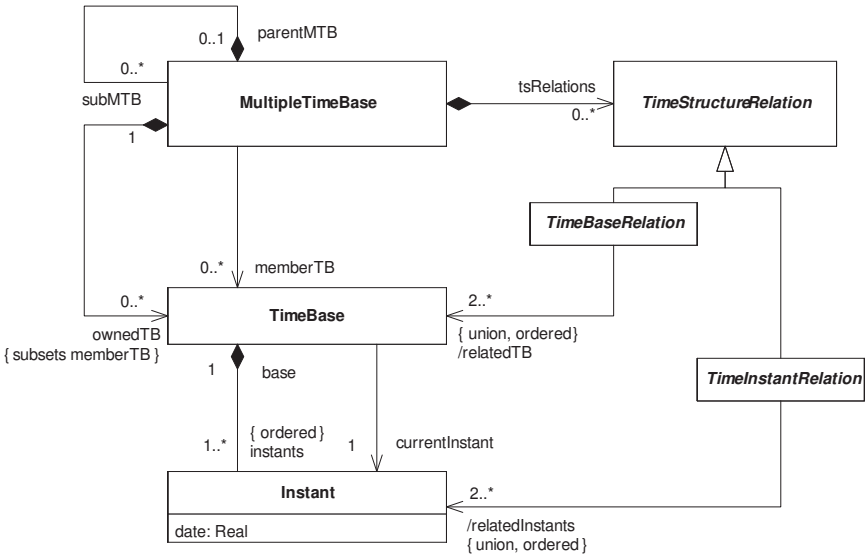

Fig. 1. Time structure (Domain view).

Concept of time structure Figure **??** shows the main concepts introduced in MARTE to model time. This is a conceptual view, or in the UML profile terminology, a *domain view*. The corresponding UML representations will be presented later. The building element in a time structure is the *TimeBase*. A time base is a totally ordered set of instants. A set of instants can be *discrete* or *dense*. The linear vision of time represented by a single time base is not sufficient for most of the applications, especially in the case of multithreaded or distributed applications. Multiple time bases are then used. A MultipleTimeBase consists of one or many time bases. A time structure contains a tree of multiple time bases.

Time bases are *a priori* independent. They become dependent when instants from different time bases are linked by relationships (coincidence or precedence). The abstract class TimeInstantRelation in Figure **??** has CoincidenceRelation and

PrecedenceRelation as concrete subclasses. Instead of imposing local dependencies between instants, dependencies can be directly imposed between time bases. A TimeBaseRelation (or more precisely one of its concrete subclasses) specifies many (possibly an infinity of) individual time instant relations. This will be illustrated later on some time base relations. TimeBaseRelation and TimeInstantRelation have a common generalization: the abstract class TimeStructureRelation. As a result of adding time structure relations to multiple time bases, time bases are no longer independent and the instants are partially ordered. This partial ordering of instants characterizes the *time structure* of the application.

This model of time is sufficient to check the logical correctness of the application. Quantitative information, attached to the instants, can be added to this structure when quantitative analyses become necessary.

Clock In real world technical systems, special devices, called clocks, are used to measure the progress of physical time. In MARTE, we adopt a more general point of view: a clock is a *model* giving access to the time structure. Time may be logical or physical or both. MARTE qualifies a clock refering to physical time as a *chronometric* clock, emphasizing on the quantitative information attached to this model. A Clock makes reference to a TimeBase. Clocks and time structures have mathematical definitions introduced below. This formal modeling is transparent to the user of the profile.

The mathematical model for a clock is a 5-tuple $(\mathcal{I}, \preccurlyeq, \mathcal{D}, \lambda, u)$ where $\mathcal{I}$ is a set of instants, $\preccurlyeq$ is an order relation on $\mathcal{I}$, $\mathcal{D}$ is a set of labels, $\lambda : \mathcal{I} \rightarrow \mathcal{D}$ is a labeling function, u is a symbol, standing for a *unit*. For a chronometric clock, the unit can be the SI time unit s (second) or one of its derived units (ms, us...). The usual unit for logical clocks is tick, but user-defined units like clockCycle, executionStep ... may be chosen as well. To address multiform time, it is even possible to consider other physical units like angle degrees (this is illustrated in an application of our time model to an automotive application [14]). Since a clock refers to a TimeBase, $(\mathcal{I}, \prec)$ is an ordered set.

A *Time Structure* is a 4-tuple $(\mathcal{C}, \mathcal{R}, \mathcal{D}, \lambda)$ where $\mathcal{C}$ is a set of clocks, $\mathcal{R}$ is a relation on $\bigcup_{a,b \in \mathcal{C}, a \neq b} (\mathcal{I}_a \times \mathcal{I}_b)$, $\mathcal{D}$ is a set of labels, $\lambda : \mathcal{I}_{\mathcal{C}} \rightarrow \mathcal{D}$ is a labeling function. $\mathcal{I}_{\mathcal{C}}$ is the set of the instants of a time structure. $\mathcal{I}_{\mathcal{C}}$ is not simply the union of the sets of instants of all the clocks; it is the quotient of this set by the coincidence relation induced by the time structure relations represented by $\mathcal{R}$.

Time-related concepts Events and behaviors can be directly bound to time. The occurrences of a (timed) event refer to points of time (instants). The executions of a (timed) behavior refer to points of time (start and finish instants) or to segments of time (duration of the execution). In MARTE, Instant and Duration are two distinct concepts, specializations of the abstract concept of Time. TimedEvent (TimedBehavior, resp.) is a concept representing an event (a behavior, resp.) *explicitly* bound to time through a clock. In this way, time is not a mere annotation: it changes the semantics of the timed model elements.

MARTE Time profile The time structure presented above constitutes the semantic domain of our time model. The UML view is defined in the "MARTE Time profile". This profile introduces a limited number of powerful stereotypes. We have striven to avoid the multiplication of too specialized stereotypes. Thanks to the sound semantic grounds of our styereotypes, modeling environments may propose patterns for more specific uses.

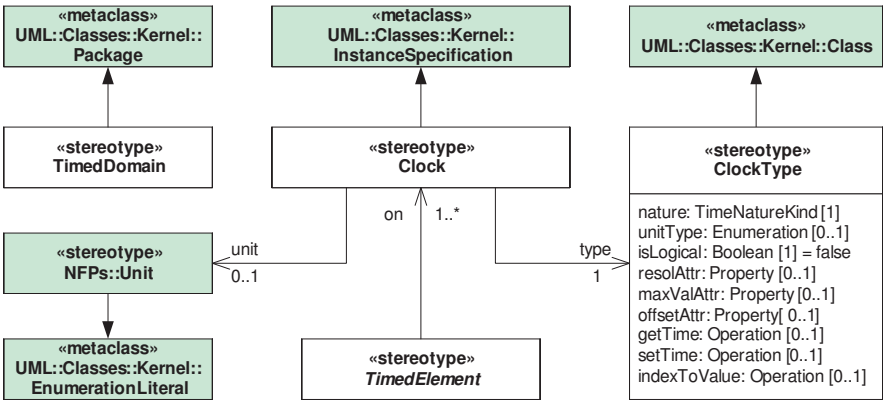

Fig. 2. MARTE TimeModeling profile: Clock.

The main sterotypes are presented in figures **??** to **??**. ClockType is a stereotype of the UML Class. Its properties specifies the kind (chronometric or logical) of clock, the nature (dense or discrete) of the represented time, a set of clock properties (*e.g.*, resolution, maximal value...), and a set of accepted time units. Clock is a sterotype of InstanceSpecification. An OCL rule imposes to apply the Clock stereotype only to instance specifications of a class stereotyped by ClockType. The unit of the clock is given when the stereotype is applied. Unit is defined in the Non Fonctional Property modeling (NFPs) subprofile of MARTE, it extends EnumerationLiteral. It is very convenient since a unit can be used like any user-defined enumeration literal, and conversion factors between units can be specified (*e.g.*, $1ms = 10^{-3}s$). TimedElement is an abstract stereotype with no defined metaclass. It stands for model elements which reference clocks. All other *timed* stereotypes specialize TimedElement.

Clock constraints ClockConstraint is a stereotype of the UML Constraint. The clock constraints are used to specify the time structure relations of a time domain. In turn, these relations characterize the $\mathcal{R}$ relation of the underlying mathematical model of the time structure.

The *context* of the constraint must be a TimedDomain. The *constrained elements* are clocks of this timed domain and possibly other objects. The *specification* of a clock constraint is a set of declarative statements. This raises the

question of choosing a language for expressing the clock constraints. A natural language is not sufficiently precise to be a good candidate. UML encourages the use of OCL. However, our clocks usually deal with infinite sets of instants, the relations may use many *mathematical* quantifiers, which are not supported by OCL. Additionnally, OCL [15] is made to be evaluatable, while our constraints often have to be processed altogether to get a set of possible solutions. So, we have chosen to define a simple constraint expression language endowed with a mathematical semantics. The specification of a clock constraint is a UML::OpaqueExpression that makes use of *pre-defined* (clock) relations, the meaning of which is given in mathematical terms, outside the UML. Our *Constraint Specification Language* is not normative. Other languages can be used, so long as the semantics of clocks and clock constraints is respected.

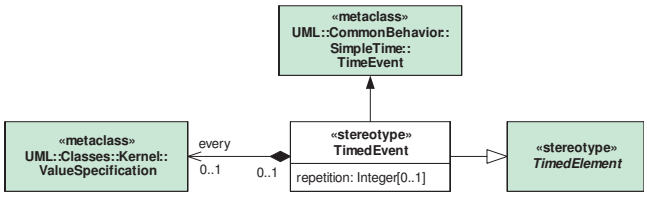

Fig. 3. MARTE TimeModeling profile: TimedEvent.

TimedEvent and TimedProcessing In UML, an Event describes a set of possible occurrences; an occurrence may trigger effects in the system. A UML2 TimeEvent is an Event that defines a point in time (instant) when the event occurs. The MARTE stereotype TimedEvent extends TimeEvent (Figure **??**). Its instant specification *explicitly* refers to a clock. If the event is recurrent, a repetition period—duration between two successive occurrences of the event—and the number of repetitions may be specified.

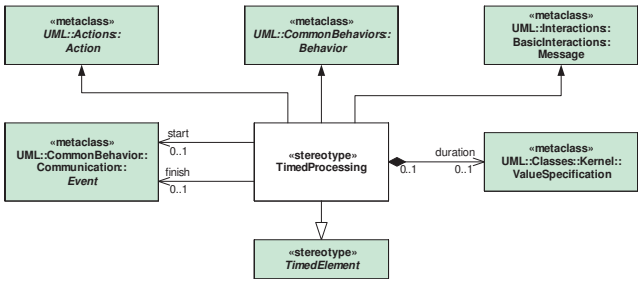

Fig. 4. MARTE TimeModeling profile: TimedProcessing.

In UML, a Behavior describes a set of possible executions; an execution is the performance of an algorithm according to a set of rules. MARTE associates a duration, an instant of start, an instant of termination with an execution, these times being read on a clock. The stereotype TimedProcessing (Figure **??**) extends the metaclasses Behavior, Action, and also Message. The latter extension assimilates a message tranfer to a *communication* action.

Note that, StateMachine, Activity, Interaction being Behavior, they can be stereotyped by TimedProcessing, and thus, can be bound to clocks.

3.2 MARTE allocation model

Allocation of functional application elements onto the available resources (the execution platform) is main concern of real-time embedded system design. This comprises both spatial distribution and temporal scheduling aspects, in order to map various algorithmic operations onto available computing and communication resources and services.

The MARTE profile defines *application* and *execution platform* models. A MARTE allocation is an association between a MARTE application and a MARTE execution platform. Application elements may be any UML element suitable for modeling an application, with structural and behavioral aspects. An execution platform is represented as a set of connected resources, where each resource provides services to support the execution of the application. So, resources are basically structural elements, while services are rather behavioral elements. Application and Execution platform models are built separately, before they are paired through the Allocation process. Often this requires prior adjustment (inside each model) to abstract/refine its components so as to allow a direct match. Allocation can be viewed as a "horizontal" association, and abstraction/refinement layering as a "vertical" one, with the abstract version relying on constructs introduced in the more refined model. While different in role, allocation and refinement share a lot of formal aspects, and so both are described here.

Application and Execution platform elements can be annotated with time information based on logical or chronometric clocks (Section **??**). Allocation and refinement provide relations between these timings under the form of constraints between the clocks and their instants. Other similar non-functional properties such as space requirement, cost, or power consumption are also considered.

In MARTE, we use the word allocation rather than deployment (as in UML) since allocation does not necessarily imply a physical distribution and could simply represent a logical distribution or scheduling. Execution platform models can be abstract at some points and not necessarily seen as concretization models. For instance, two pieces of an algorithm could be allocated to two different processor cores, while the executable file containing both pieces would be deployed on the memory of the processor and the source file containing the specification of the algorithm would be deployed on a hard disk. This dual function was recognized in SPT, where allocation was called realization, while refinement was used as such. MARTE allocation and refinement are complementary to the UML deployment; we prefer to keep the three concepts separated. This is not

the case of AADL that provides a single mechanism—the binding—for all three concepts. The allocation mechanism proposed by MARTE is actually very close to the structure allocations of SysML because it allocates logical parts to more physical ones. However, MARTE makes it explicit that both the logical and physical parts could be either of a behavioral or structural nature. Contrary to SysML, MARTE makes a difference between allocation—from application model elements to execution platform model elements—and refinement of an abstract model elements (logical or physical) into more specific elements.

The stereotype Allocate A MARTE allocation is materialized by the stereotype Allocate (Figure **??**), which extends the UML metaclass Abstraction, and can be associated with NFP constraints. Allocation can be structural, behavioral, or hybrid. Structural allocation associates a group of structural elements and a group of resources. Behavioral allocation associates a set of behavioral elements and a service provided by the execution platform. When clear from context, hybrid allocations are allowed (*e.g.*, when an implicit service is uniquely defined for a resource). At the finer level of detail, behavioral allocation deals with the mapping of UML actions to resources and services.

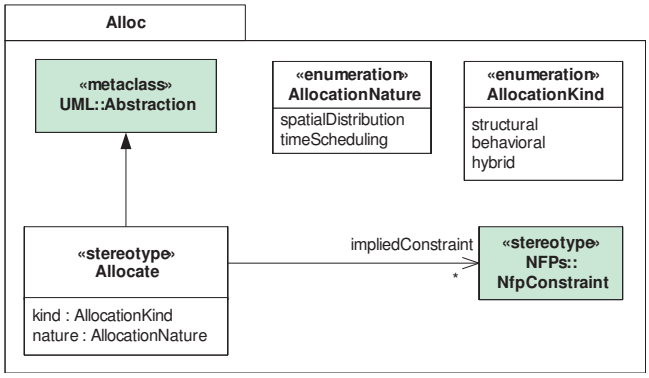

Fig. 5. The stereotype «allocate».

The stereotype Allocated MARTE advocates the need to differentiate the potential sources of an allocation from the targets. Each model element involved in an allocation is annotated with the stereotype Allocated (as in SysML), which extends the metaclass NamedElement or rather one of its specializations (Figure **??**). The stereotype ApplicationAllocationEnd, noted by the keyword «ap_allocated», denotes a source of an allocation. The stereotype ExecutionPlatformAllocationEnd, noted by the keyword «ep_allocated», represents the target of an allocation. The stereotype Allocated is not abstract to ensure compatibility with SysML, but one of its specializations should be preferred. The property allocatedTo, respectively

allocatedFrom, is a derived property resulting from the process of creating the abstraction (allocation); they facilitate the identification of the targets, respectively the sources, of the allocation when all model elements cannot be drawn on the same diagram.

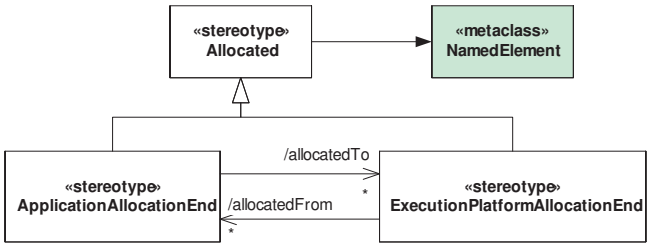

Fig. 6. The stereotype «allocated».

4 Illustrative Examples

4.1 Chronometric clocks

The MARTE TimeLibrary provides a model for the *ideal time* used in physical laws: idealClk, which is an instance of the class IdealClock, stereotyped by ClockType (Fig. **??**). idealClk is a dense time clock, its unit is the SI time unit s.

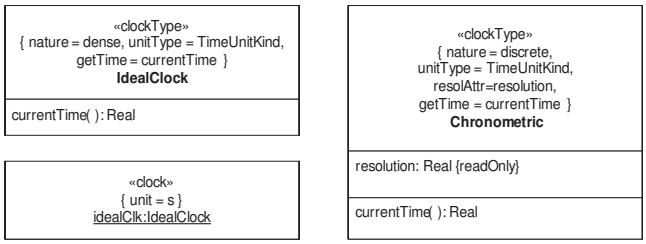

Fig. 7. Ideal and Chronometric clocks.

Starting with idealClk, the user can define new discrete chronometric clocks (Fig. **??**). First, the user specifies Chronometric—a class stereotyped by ClockType—which is discrete, not logical (therefore chronometric), and with a read only attribute (resolution). Clocks belong to timed domains. In Fig. **??**, a single time domain is considered. It owns 3 clocks: idealClk, cc1 and cc2, two instances of Chronometric that both use s (second) as a time unit; and whose resolution is 0.01 s. The three clocks are *a priori* independent. A clock constraint specifies relationships among them.

The first statement of the constraint defines a clock c local to the constraint. c is a discrete time clock derived from idealClk by a *discretization* relation. The resolution of this clock is 1 ms. The next two statements specify that cc1 and cc2 are subclocks of c with a rate 10 times slower than c. The fourth and fifth statements indicate that cc1 and cc2 are not perfect clocks. Flaws are characterized by *non functional properties* like stability and offset. Their rate may have small variations (a stability of 10^{-5} implicitly measured on idealClk). The last statement claims that the two clocks are out of phase, with an offset value between 0 and 5 ms measured on idealClk. Note that even if cc1 and cc2 look alike, they are not identical because relations are not necessarily functional.

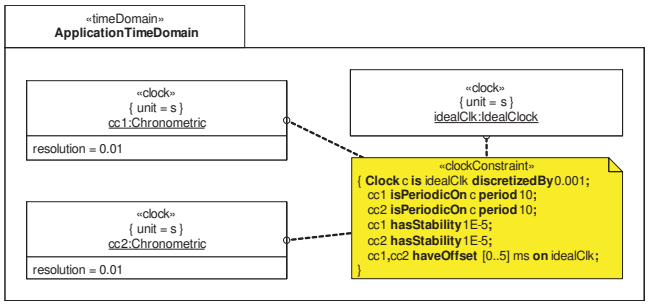

Fig. 8. Clock constraints.

4.2 AADL communication

To explain its *port-based* communication semantics, the AADL specification takes the example of a thread Read that captures a sample and sends it to a second thread Control. The two threads are assumed to be dispatched at the same time. Several cases are studied, the case where the two threads are dispatched with the same period, the case where Read is dispatched twice faster than Control (undersampling), and the case where Read is dispatched twice slower than Control (oversampling). These cases are studied first with an immediate communication (the output value is available as soon as the thread Read completes) and then with a delayed communication (the output value is available only at the next dispatch of the thread Read).

To compare our approach to AADL, we take the case of an immediate communication, which is the more challenging, with undersampling. As said before, the main difference of our approach is that we separate the application models from the execution platform models. The application is described with a UML activity diagram (Fig. **??**) using purely logical clocks and stereotyped by «TimedProcessing». The behavior of Read and Control are executed repetitively, they communicate through a datastore object node that allows for multiple readings of the same sample.

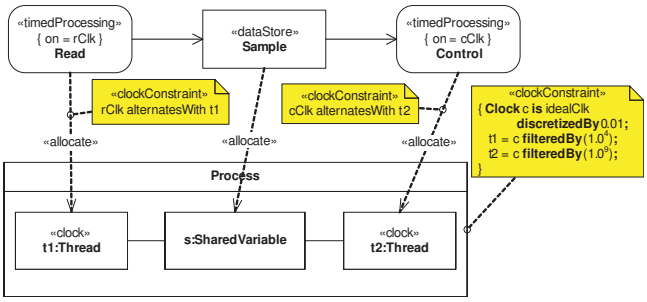

Fig. 9. Clock constraints.

In a second step, the application is allocated to the model of an execution platform, a process containing two threads that communicate through a shared variable. In our model, t1 and t2, or rather their dispatch time, are considered as clocks. These two harmonic clocks are defined using a local 100Hz-clock c. Then t1 and t2 are derived from c with respective frequency 20Hz and 10Hz.

Additional clock constraints are associated with the allocate dependency to map the application clocks to the platform clocks. All these constraints define a partial order. In the case of a delayed communication, these would have been enough to have an equivalence of all possible schedulings. With an immediate communication, an additional constraint is required to guarantee the same behavior than AADL. This constraint would follow a greedy scheduling in order to execute Control as soon as possible. Our constraint model allows for delaying subjective choices as much as possible in order to avoid overspecification.

5 Conclusion

We presented a UML profile for comprehensive Time Modeling. Time here can be of discrete or dense, physical or logical. Logical time allows to model various time threads sustaining asynchronous or loosely time-related concurrent processes. This philosophy (of assigning logical clocks in order to explicitly handle time rates) borrows to foundational notions in embedded MoCC design. To this we add a kernel language of clock constraint relations, as well as timed events constraint relations. This constraint language, while currently simple, allows to define most useful clock relations (such as being periodic). While the profile can be considered as a "creative" translation of existing ideas on tagged systems to a UML setting (with all the alignments it required that were far from trivial), the clock constraint language and its use as a formal specification of classical time relation notions is original, to the best of our knowledge.

Time annotation can then be applied to behavioral elements, leading to TimedEvent and TimedProcessing, and to structural elements, leading to clocked Classes and clocked Objects. This can be performed on application models and architecture models of the embedded design. Then the system dynamics should

run according to (partial) timing constraints, if possible, according to a timed operational semantics. Providing timed constructs in UML behavioral models (state diagrams and activity diagrams mostly) would be the next step here. Numerous examples exist (outside the UML) of timed languages and calculi under the form of MoCC constructors inside the proper time domain. Model transformation tools could extract the time properties, feed them into timing analysis tools and bring the result back into UML within a UML simulator. In that sense, our profile would give the time semantics of UML models.

Clock constraints provide partial scheduling information, and an actual schedule can be obtained by solving such a set of constraints, some of which originate from the application model, some from the execution platform model, and some from the system's real-time requirements. The same formalisms of clock relations can also be used in some case to represent the *result* of the scheduling decisions, and display them to the designer.

We provided modeling instances and case studies to illustrate and motivate the modeling framework. We showed how it allows to introduce a number of useful time predicates on events. We also showed the intent behind logical time by considering examples with various clocks running at unrelated speeds.

References

1. Lee, E.A., Sangiovanni-Vincentelli, A.L.: A framework for comparing models of computation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **17**(12) (1998) 1217–1229
2. Buck, J., Ha, S., Lee, E., Messerschmitt, D.: Ptolemy: A framework for simulating and prototyping heterogeneous systems. International Journal of Computer Simulation, special issue on "Simulation Software Development" **4** (1994) 155–182
3. Jantsch, A.: Modeling Embedded Systems and SoCs - Concurrency and Time in Models of Computation. Morgan Kaufman (2003)
4. Darte, A., Robert, Y., Vivien, F.: Scheduling and Automatic Parallelization. Birkhaüser (2000)
5. Feautrier, P.: Compiling for massively parallel architectures: a perspective. Microprogramming and Microprocessors (41) (1995) 425–439
6. OMG: UML 2.1 Superstructure Specification, Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701. (2006) OMG document number: ptc/2006-04-02.
7. Selic, B.: On the semantic foundations of standard uml 2.0. In: SFM-RT 2004. Volume 3185 of LNCS., Springer-Verlag (2004) 181–199
8. OMG: UML Profile for Schedulability, Performance, and Time Specification, Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701. (2005) OMG document number: formal/05-01-02 (v1.1).
9. ITEA: EAST-ADL — The EAST-EEA Architecture Description Language. (2004) ITEA Project Version 1.02.
10. Graf, S., Ober, I., Ober, I.: A real-time profile for UML. STTT, Software Tools for Technology Transfer **8**(2) (2006) 113–127
11. OMG: Systems Modeling Language (SysML) Specification. (2006) OMG document number: ad/2006-03-01.

12. SAE: Architecture Analysis and Design Language (AADL). (2006) document number: AS5506/1.
13. OMG: UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), Request for proposals, Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701. (2005) OMG document number: realtime/2005-02-06.
14. André, C., Mallet, F., Peraldi-Frati, M.A.: A multiform time approach to real-time system modeling: Application to an automotive system. Technical Report ISRN I3S/RR–2007–14–FR, I3S laboratory, Sophia-Antipolis, France (2007)
15. OMG: Object Constraint Language, version 2.0, Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701. (2006) OMG document number: formal/06-05-01.