LABORATOIRE

INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR 6070

MODELING TIME(S) IN UML

Charles ANDRE, Frédéric MALLET, Robert DE SIMONE

Projet AOSTE

Rapport de recherche
ISRN I3S/RR–2007-16–FR

Mai 2007

Laboratoire d'Informatique de Signaux et Systèmes de Sophia Antipolis - UNSA-CNRS
2000, rte.des Lucioles – Les Algorithmes – Bât Euclide B – B.P. 121 – 06903 Sophia-Antipolis Cedex – France
Tél.: 33 (0)4 92 94 27 01 – Fax: 33 (0)4 92 94 28 98 – www.i3s.unice.fr
UMR6070

Modeling Time(s) in UML

Charles ANDRÉ, Frédéric MALLET, Robert DE SIMONE
Projet AOSTE
Laboratoire Informatique Signaux et Systèmes (I3S) / INRIA
Université de Nice Sophia Antipolis — CNRS UMR 6070
06903 Sophia Antipolis, France

E-mail: {andre, fmallet}@unice.fr, Robert.De_Simone@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 heterogeneous 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 an extensive Time metamodel that opens the possibility to cast this approach in a Model-Driven Engineering light. Then, meaningful transformations can extend allocation, as defined in SysML, to timed models. Time modeling also allows for a precise description, in a OCL-like fashion, of timed properties of events and clocks (periodic, sporadic, bounded jitter,...). Finally, Time can be interpreted to control the system dynamics, unlike other proposals based on uninterpreted stereotype annotations.

This report starts with a presentation of the "Time" and "Allocation" sub-profiles of the UML profile for Marte. Their use is illustrated on a communication example borrowed from the AADL standard. This study highlights semantic variations between immediate and delayed communications, and provides a generalization.

Key words *: UML profile, real-time embedded systems, time modeling, time constraints.*

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 designed model can be interpreted and understood completely differently for its behavior 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.

Following some works on dedicated *Models of Computation and Communication* (MoCCs) for real-time embedded systems [8, 3, 7], 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 activation steps in a systems 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 conceptually concurrent model onto an existing, heterogeneous embedded execution platform comprises aspects of spatial distribution and temporal scheduling (between multiple logical times and single physical time). 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 possibly the real-time requirements to be met. While this can be seen as a timed version of a SysML-inspired allocation scheme, it results in a refined time structure which can be traced back to the application for the designer's understanding. In this sense, 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. To understand this, the reader should try to contemplate how to state that an event/signal is *periodic*, or *sporadic*, regarding another clock, using an OCL-like language extended to deal with infinite sequences (of clock ticks) ?

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 full 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 2), Section 3 introduces the Marte subprofiles for time and allocation. Section 4 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.

2.1.1 UML

In UML [12] 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: 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 [14]. 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.

2.1.2 SPT

The UML Profile for Schedulability, Performance, and Time (SPT) [10] 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 *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. But "time" here is only introduced through dedicated stereotype annotations that are *not* interpreted and given meaning as part of UML semantics. Instead, their purpose is to be understood by external analysis tools to perform schedulability or performance evaluation, given that are the right ones, and after translation from the UML model into such tool input format.

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 3.

2.1.3 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 [13]. 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.

The UML profile Omega-RT [6] 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. Note that the *Omega Event* is a stereotype of UML 1.4 Class, while in UML 2, an Event is not even a Classifier. This difference poses a serious compliance issue between Omega and UML 2.

TURTLE-P [2] is a UML profile for the *formal* validation of critical and distributed systems. This profile introduces temporal operators and composition (parallel, sequence, synchronization, and preemption). It deals with temporal nondeterminism, usual in distributed systems. Properties of a TURTLE-P model can be evaluated and/or validated thanks to the formal semantics given in RT-LOTOS (LOTOS extended with temporal operators).

2.1.4 Summary

All the abovementioned profiles introduce relationships between Time and Events or Actions. They annotate the UML model with quantitative information about time. Omega and TURTLE-P introduce temporal operators in behavioral models according to their time semantics. 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).

2.2.1 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 3.2.

2.2.2 SysML allocation

SysML 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 3.2. 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.

2.2.3 AADL binding

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.

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 SAE AADL standard [15].

2.3.1 Architecture and Analysis Description Language (AADL)

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.

3 MARTE

Marte is a response to the OMG RFP to provide a UML profile for real-time and embedded systems [9]. 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 *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.

3.1.1 Concept of time structure

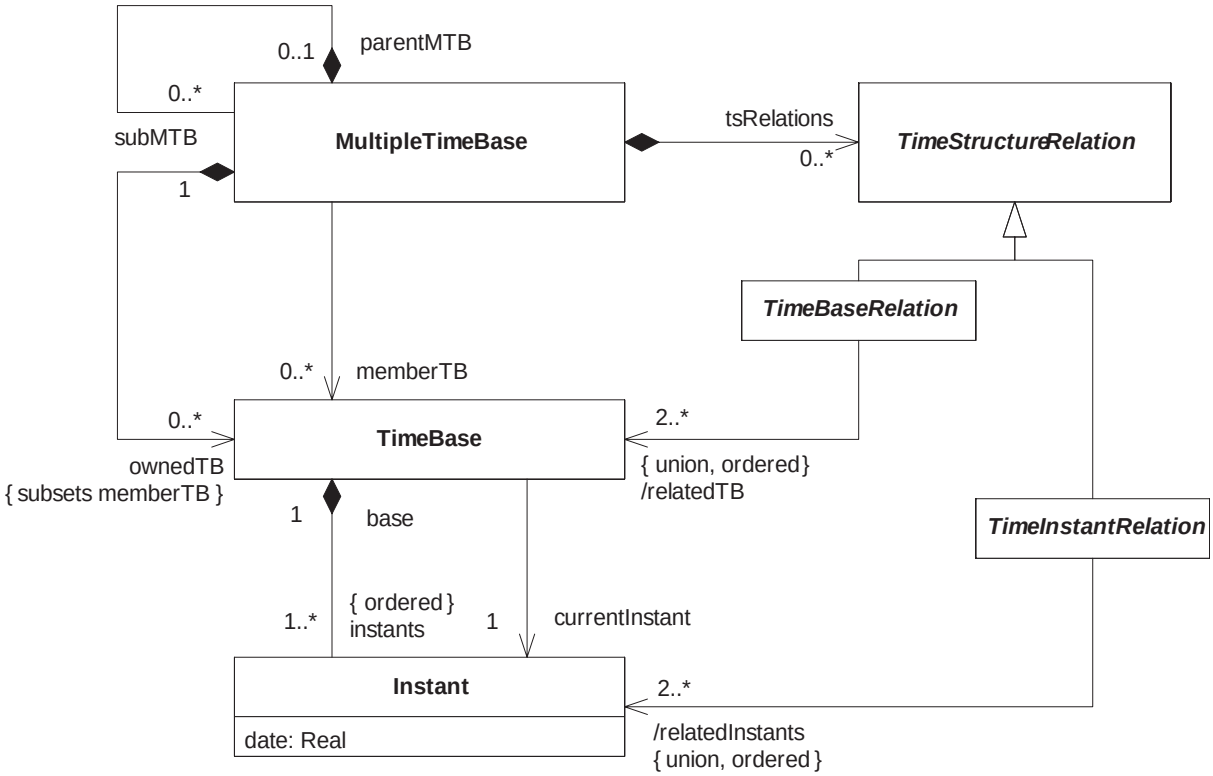


Figure 1. Time structure (Domain view).

Figure 1 shows the main concepts introduced in Marte to model time. This is 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*, even continuous. 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 1 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.

3.1.2 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 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 [1]). Since a clock refers to a TimeBase, the poset $(\mathcal{I}, \preccurlyeq)$ is a linear 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 $\mathcal{R}$. A time structure specifies a poset $(\mathcal{I}_{\mathcal{C}}, \preccurlyeq_{\mathcal{C}})$.

3.1.3 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.

3.1.4 The Marte TimeModeling profile

The time structure presented above constitutes the semantic domain of our time model. The UML view is defined in the "Marte TimeModeling 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.

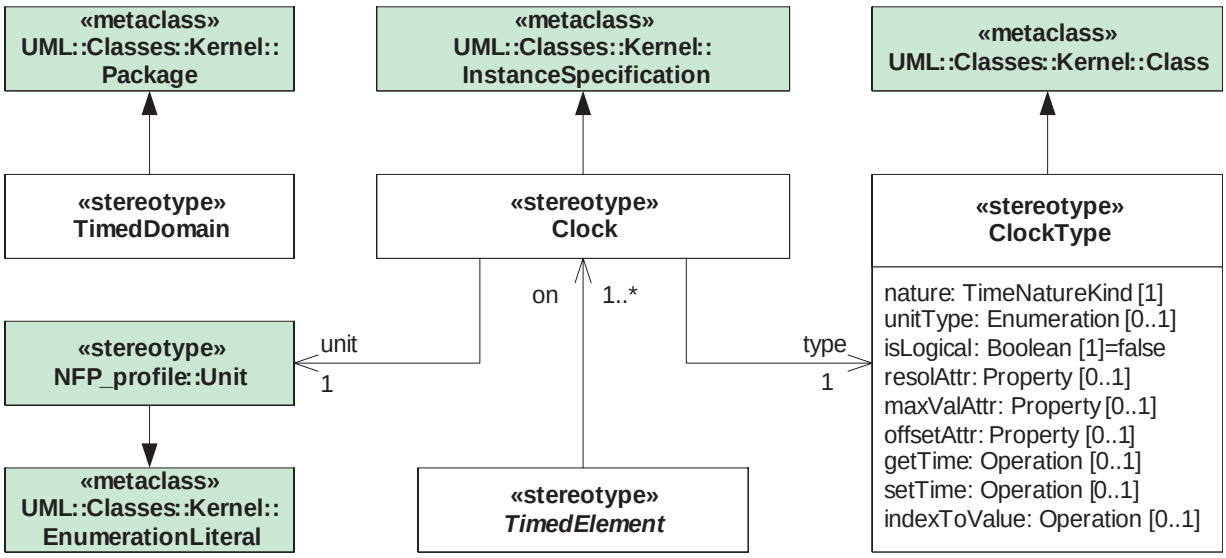


Figure 2. Marte TimeModeling profile: Clock.

The main sterotypes are presented in figures 2 to 4. 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 NFP profile of Marte (NFP is the acronym of Non Functional Property). It is a stereotype of EnumerationLiteral. This sterotype is very convenient since an 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.

3.1.5 Clock constraints

ClockConstraint is a stereotype of the UML Constraint. The clock constraints are used to specify, in UML models, 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 quantifiers. OCL supports recursive definitions and predefined operations on the collection types (e.g., exists, forAll...). However, their usage in complex quantified expressions leads to intricate OCL expressions. The advantage of OCL, which claims to be "a formal language that remains easy to read and write" [11, Chapter 7], is then lost. So, we have chosen to define a simple constraint expression language endowed with a mathematical semantics. For UML, the specification of a clock constraint is an opaque expression. This expression 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.

3.1.6 TimedEvent and TimedProcessing

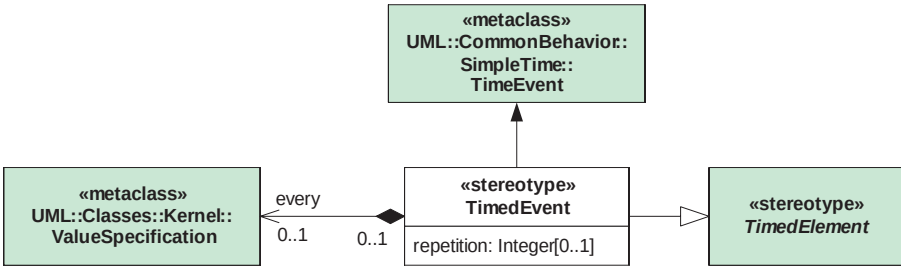


Figure 3. Marte TimeModeling profile: TimedEvent.

In UML, an Event describes a set of possible occurrences; an occurrence may potentially trigger effects in the system. A UML 2 TimeEvent is an Event that defines a point in time (instant) when the event occurs. In Marte, we define TimedEvent a sterotype of TimeEvent (Figure 3) in which the instant specification *explicitly* refers to a clock. Moreover, if the event is recurrent, a repetition period—duration between two successive occurrences of the event—and the number of repetitions may be optionally specified.

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. Figure 4 shows that this concept has been extended: the stereotype TimedProcessing extends the metaclasses Behavior, Action, ans 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. Thus, state machines, activities and interactions can be explicitly 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 relevant 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

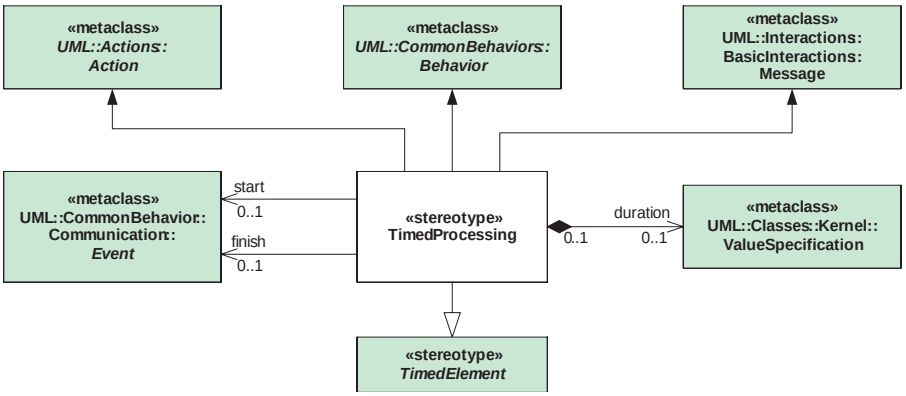


Figure 4. Marte TimeModeling profile: TimedProcessing.

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 3.1). Allocation and refinement provide relations between these timing 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.

3.2.1 The stereotype Allocate

A Marte allocation is materialized by the stereotype Allocate (Figure 5), which is an extension of the UML metaclass Abstraction. A Marte allocation can be associated with non functional property constraints. Allocation can be specified in different kinds: Structural, behavioral, or hybrid. Structural allocation is an association between a group of structural elements and a group of resources. Behavioral allocation is an association between a set of behavioral elements and a service provided by the execution platform. When clear from context, hybrid allocations can also be allowed (for instance 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.

3.2.2 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) or rather one of its specializations. This stereotype (Figure 6) is an extension of the metaclass NamedElement, so almost any model elements can be allocated to or from any other model elements. The stereotype ApplicationAllocationEnd, noted by the keyword «app allocated»,

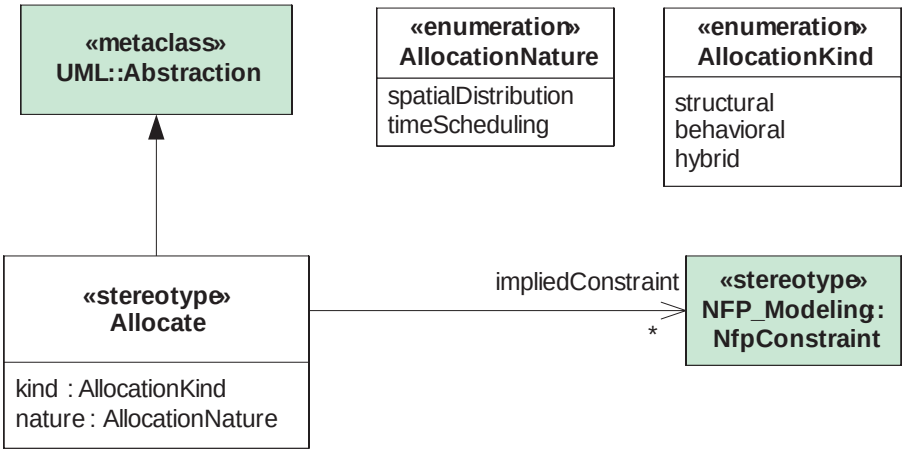


Figure 5. The stereotype «allocate»**.**

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.

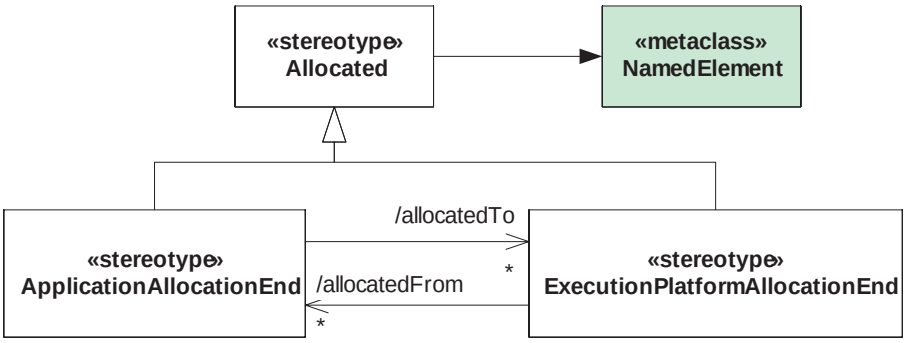


Figure 6. The stereotype «allocated»**.**

4 Illustrative Examples

4.1 Chronometric clocks

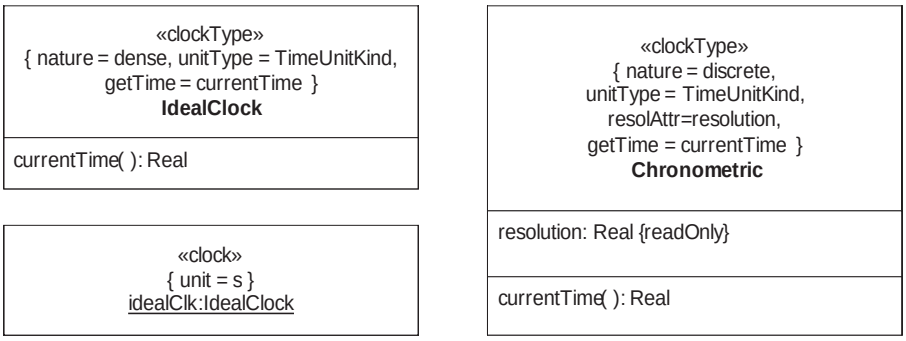


Figure 7. Ideal and Chronometric clocks.

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. 7). idealClk is a dense time clock, its unit is the SI time unit s.

Starting with idealClk, the user can define new discrete chronometric clocks (Fig. 8). 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 the example shown in Fig. 8, only one time domain is considered. It owns 3 clocks: idealClk, cc1 and cc2, which are two instances of Chronometric. cc1 and cc2 use s (second) as a time unit; and they have a resolution of 0.01 s. The three clocks are *a priori* independent. A clock constraint specifies relationships among them.

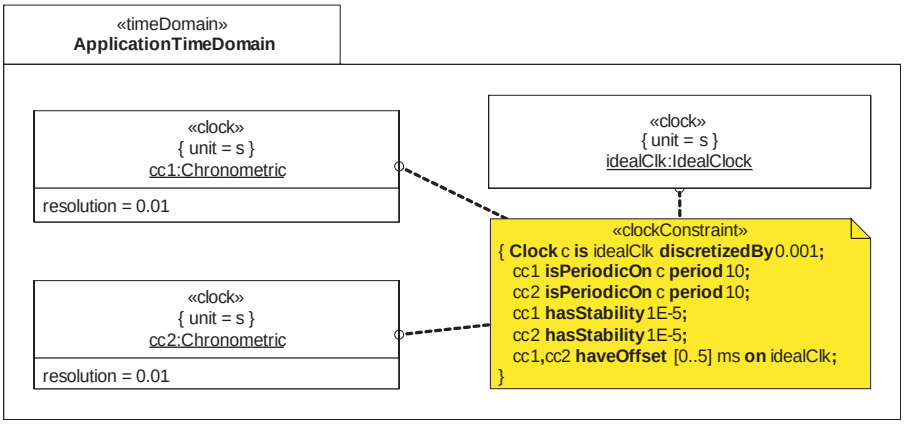


Figure 8. Clock constraints.

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 (see Annex A). 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.

4.2 AADL communication

As a demonstration of the expressiveness of Marte, a forthcoming UML profile for Modeling and Analysis of Real-Time Embedded systems, we take as an example the inter-thread data communication semantics of AADL.

In AADL, the communications can be *immediate* (Fig. 9a) or *delayed* (Fig. 9b). The threads are concurrent schedulable units of sequential executions. Several properties can be assigned to threads, the one of concern here is the *dispatch protocol*. We actually consider only periodic threads, associated with a period and a deadline, specified as chronometric time expressions (*e.g.*, period=50ms or frequency=20Hz). By default, when the deadline is not specified it equals the period.

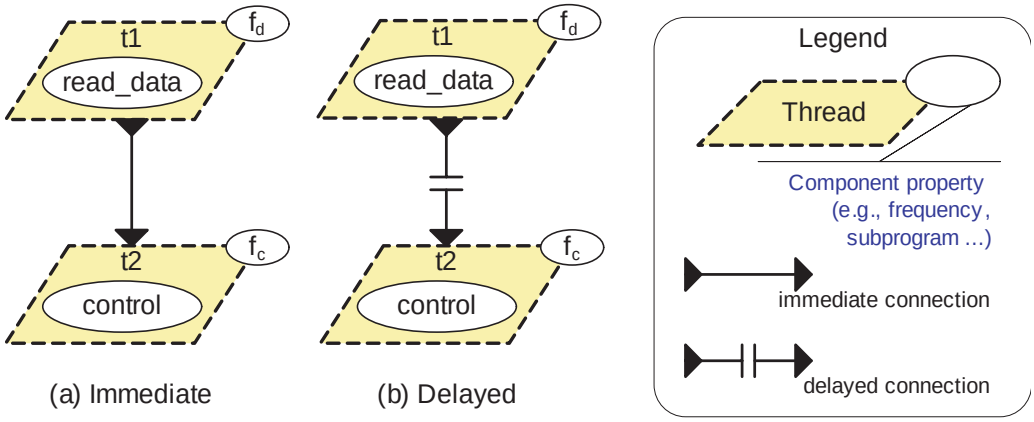


Figure 9. AADL inter-thread data communication.

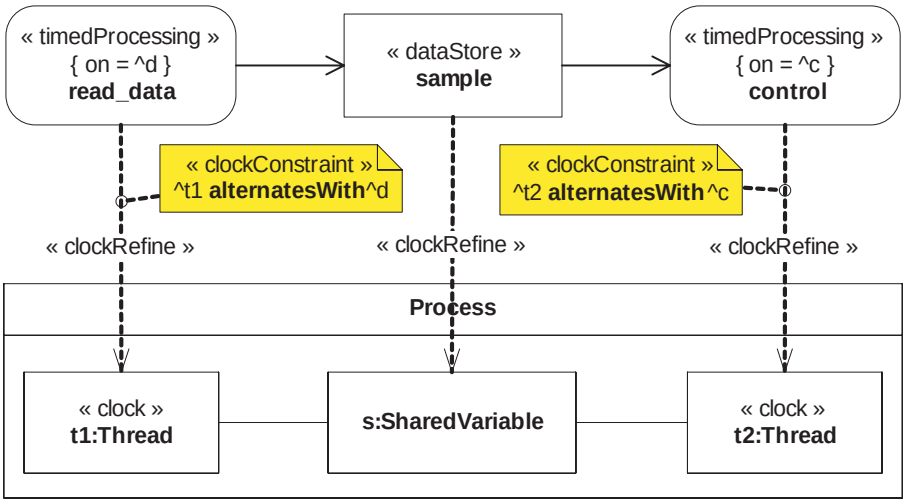


Figure 10. Application/Execution platform in Marte.

4.2.1 Application, execution platform and allocation

A first difference with AADL is that Marte differentiates the algorithm, which can be represented as an activity diagram (Fig. 10, upper part), from the underlying structure, which is modeled here as a composite structure diagram (Fig. 10, lower part), and that implies a logical scheduling. Each part has its own causality constraints. Marte refinement mechanism, and its associated clock constraints, allows for explicit relations amongst the clocks of both parts. In Marte, activation conditions of all application model elements are represented by clocks identified with the appropriate stereotypes, for instance TimedProcessing. As a starting point, we consider the clocks of each element as independent, then the context (dependencies and refinements) constrains these clocks. At last, a timing analysis tool may resolve the constraints to determine a (family of) possible schedules. We strive to avoid overspecification and keep the model as generic as possible, adding only required constraints. From the algorithmic point of view, the actions read_data and control are CallBehaviorAction that execute a given behavior repetitively according to their activation condition (clocks $\widehat{d}$ and $\widehat{c}$ respectively).

4.2.2 Introducing clock constraints

From the execution platform point of view, the threads t1 and t2 are also associated with clocks ($\widehat{t1}$ and $\widehat{t2}$ respectively). In AADL, the period of a thread is expressed as a chronometric time expression and therefore clocks $\widehat{t1}$ and $\widehat{t2}$ are necessarily chronometric.

When allocated to a periodic thread with a given period, an additional constraint must express that these behaviors are executed on the thread dispatch and must complete before the deadline (the next dispatch by default). In Marte, we can differentiate atomic behaviors, for which the execution time is considered negligible as compared to the period, from non-atomic ones. If we consider the behaviors as atomic, the allocation is simply expressed with the constraint given by Eq. 1. Note that this constraint is not symmetrical since t1 may cause d, but not the converse.

$$\widehat{t1}\ \texttt{alternatesWith}\ \widehat{d} \tag{1}$$

If the execution time is not negligible, each action can be represented by two events, the start (*e.g.*, ds for d, cs for c) and the finish (*e.g.*, df for d, cf for c), and a duration. In that latter case, we need three constraints to express that the behavior read_data is repetitively executed on thread t1 (Eqs. 2–4).

$$\widehat{t1}\ \texttt{alternatesWith}\ \widehat{ds} \tag{2}$$

$$\widehat{t1}\ \texttt{alternatesWith}\ \widehat{df} \tag{3}$$

$$\widehat{ds}\ \texttt{isFasterThan}\ \widehat{df} \tag{4}$$

The first two constraints express that the behavior starts and finishes between two consecutive dispatches of thread t1. The last constraint, which reads clock $\widehat{ds}$ is faster than clock $\widehat{df}$, specifies that the action read_data starts before it finishes; it is sufficient to impose that it finishes within the same cycle of execution.

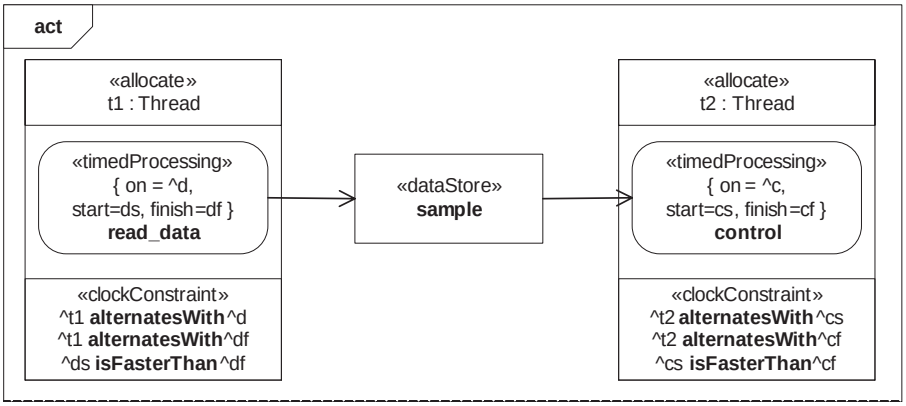


Figure 11. Alternative representation of "Allocation".

An alternative, more compact, representation of the allocation is presented in Figure 11. A new compartment is created in the AllocationActivityGroup.

The next constraint comes from the communication itself, according to the application point of view. We use a UML data store to mean that the action read_data can overwrite the existing value (in the object node) without generating a new token and this very same value can be read several times by the action control (non depleting read). In UML, there must be at least one writing before any reading (Eq. 5).

$$\widehat{d}[1]\ \texttt{precedes}\ \widehat{c}[1] \tag{5}$$

Let $\widehat{wr}$ be the (logical) clock for significant writings in the data store. There could be several consecutive writings in the datastore before one reading. In that case, only the last one is considered significant. Let $\widehat{rd}$ be the corresponding (logical) clock for significant readings from the data store. When the same value is read several times, only the first read is considered to be significant. Furthermore, AADL assumes that communicating threads must have common dispatches. A simple way to achieve that is if all threads start their execution at the same time (they are in phase). The AADL standard considers three cases: *synchronous* threads with the same period, *oversampling* (the period of control is evenly divided by the period of read_data), *undersampling* (the period of read_data is evenly divided by the period of control). Let $q1$ and $q2$ be natural numbers such that $f_d/f_c = q1/q2$. They represent the relative periods of read_data and control. Section 4.2.6 discusses how to compute $q1$ and $q2$ in the general case. When the threads are synchronous (Eq. 6), $q1 = q2 = 1$. When oversampling (Eq. 7), $q1 = 1$ and $q2 > 1$. When undersampling (Eq. 8), $q1 > 1$ and $q2 = 1$. $\max\{q1, q2\}$ is called the hyper-period. In Eq. 7 (resp. Eq. 8), the binary word expresses that each instant of $\widehat{t1}$ (resp. $\widehat{t2}$) is synchronous with every $q2^{th}$ (resp. $q1^{th}$) instant of $\widehat{t2}$ (resp. $\widehat{t1}$).

$$\widehat{t1}\ \boxed{\equiv}\ \widehat{t2} \tag{6}$$

$$\widehat{t1}\ \boxed{\equiv}\ \widehat{t2}\ \texttt{filteredBy}\ (1.0^{q2-1}) \tag{7}$$

$$\widehat{t2}\ \boxed{\equiv}\ \widehat{t1}\ \texttt{filteredBy}\ (1.0^{q1-1}) \tag{8}$$

Selecting the significant writings and readings consists in choosing one event of $\widehat{d}$ every $q1^{\text{th}}$ (Eq. 9) and one event of $\widehat{c}$ every $q2^{\text{th}}$ (Eq. 10). Additionally, Eq. 11 states that each significant writing must precede its related significant reading.

$$\widehat{wr}\ \texttt{isPeriodicOn}\ \widehat{d}\ \texttt{period}\ q1 \tag{9}$$

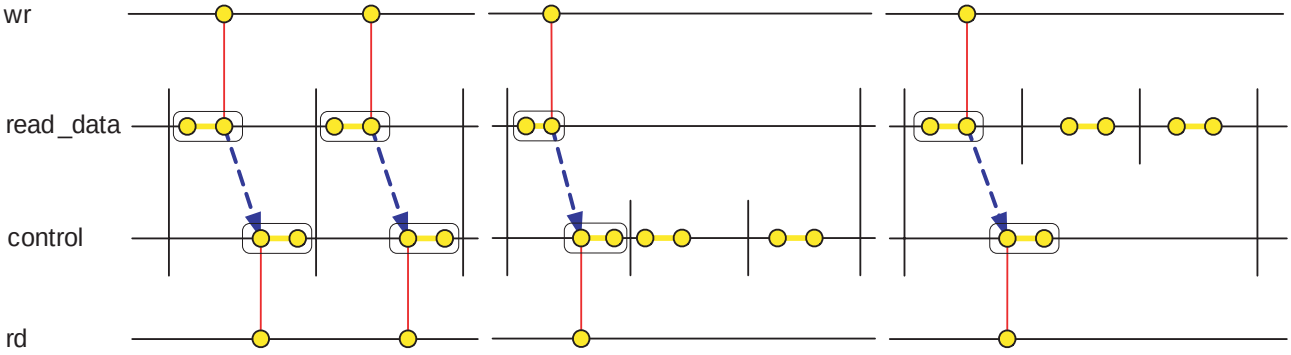
$$\widehat{rd}\ \texttt{isPeriodicOn}\ \widehat{c}\ \texttt{period}\ q2 \tag{10}$$

$$\widehat{wr}\ \texttt{alternatesWith}\ \widehat{rd} \tag{11}$$

We restrict our comparison to the three cases considered by the AADL standard. However, in subsection 4.2.6 we elaborate on the general case.

We have defined all general constraints. In particular, these constraints do not state which instant is considered as a significant writing or reading. This depends on the semantics of the communication. The following two subsections study the three different cases (synchronous, oversampling, undersampling) with both an immediate and a delayed communication, each subsection gives stronger constraints compatible with Eqs. 9–11.

4.2.3 Immediate communication



(a) synchronous (q1=q2=1) (b) oversampling (q1=1, q2=3) (c) undersampling (q1=3, q2=1)

Figure 12. Immediate communications.

An immediate communication means that the result of the sending thread (here read_data) is immediately available to the receiving thread (here control). When threads are synchronous (Fig. 12a), this is denoted by " $\widehat{wr}\ \boxed{\equiv}\ \widehat{d}$ " and " $\widehat{rd}\ \boxed{\equiv}\ \widehat{c}$ ", or more precisely by " $\widehat{wr}\ \boxed{\equiv}\ \widehat{df}$ "and " $\widehat{rd}\ \boxed{\equiv}\ \widehat{cs}$ ". In case of oversampling (Fig. 12b), the result of the action read_data must be written in the object node early enough so that the *first* (for each $q2$-long hyper-cycle) execution of the action control can use it, $q2$ being the quotient of the period of control over the period of read_data. This is denoted by " $\widehat{wr}\ \boxed{\equiv}\ \widehat{d}$ " and " $\widehat{rd}\ \boxed{\equiv}\ \widehat{c}\ \texttt{filteredBy}\ \left(1.0^{q2-1}\right)$ ". In case of undersampling (Fig. 12c), AADL specifies that the execution of the *first* (for each $q1$-long hyper-cycle) execution of the action read_data must complete before the execution of the action control. This is stated by " $\widehat{rd}\ \boxed{\equiv}\ \widehat{c}$ " and " $\widehat{wr}\ \boxed{\equiv}\ \widehat{d}\ \texttt{filteredBy}\ \left(1.0^{q1-1}\right)$ ".

4.2.4 Delayed communication

A delayed communication means the result of the sending thread is made available only at its next dispatch while the receiving thread only reads *after* its own dispatch and *ultimately* when the data is required. The dispatches of the sending and the receiving threads are not necessarily all synchronous, even if there must be a synchronization at some point. When the thread are synchronous (Fig. 13a), the constraint is denoted by Eqs. 12–13. Note that $\delta 4$ offers the possibility to delay the actual execution of read_data. The thread t1 can either be idle or be executing another action before starting to execute read_data. Eq. 12 states that $(\exists \delta 4 \in \mathbb{N})\ (\forall k \in \mathbb{N}^{\star})\ \left(\widehat{wr}[k] \equiv \widehat{t1}[\delta 4 + k]\right)$.

$$(\exists \delta 4 \in \mathbb{N})\ \left(\widehat{wr}\ \boxed{\equiv}\ \widehat{t1}\ \texttt{filteredBy}\ 0^{\delta 4}\ (1)\right) \tag{12}$$

$$\widehat{rd}\ \boxed{\equiv}\ \widehat{c} \tag{13}$$

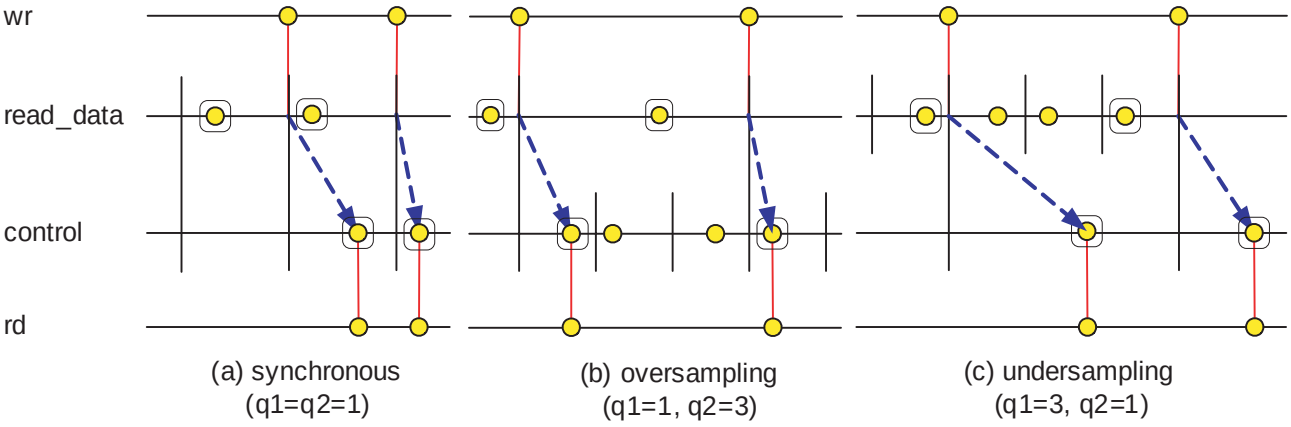


Figure 13. Delayed communications.

In case of oversampling (Fig. 13b), the result will be available for the *first* execution of the action control of the *next* $q2$-long hyper-cycle. This leaves lots of freedom to schedule the action read_data anywhere within the current hyper-cycle. We keep the relation Eq. 12 while Eq. 13 is replaced by Eq. 14.

$$\widehat{rd} \boxed{\equiv} \widehat{c}\ \texttt{filteredBy}\ \left(1.0^{q2-1}\right) \tag{14}$$

In the case of undersampling (Fig. 13c), the result of the *last* execution (for each $q1$-long hyper-cycle) of the action read_data will be available for the action control at the *next* hyper-cycle. This is denoted by combining Eq. 15 with Eq. 13.

$$(\exists \delta 4 \in \mathbb{N}) \left(\widehat{wr} \boxed{\equiv} \widehat{t1}\ \texttt{filteredBy}\ 0^{\delta 4}\ \left(1.0^{q1-1}\right)\right) \tag{15}$$

Note that the relations are not fully symmetrical. This is due to the AADL semantics that changes the rule depending on the kind of communication.

4.2.5 Getting a schedule

In most of the discussed cases, we get in the end a total order amongst events of $\widehat{d}$ and $\widehat{c}$, that is to say we have given a schedule. In few cases, we need additional constraints. Besides, to represent the resulting schedule we have to set up constraints between a chronometric clock and, $t1$ and $t2$. We start by the creation of three chronometric clocks c_{100}, c_{10} and c_{30} of respective frequency 100Hz, 10Hz and 30Hz. The usual way to proceed (see Section. 4.1) is to discretized the ideal clock (idealClk) as in Eqs. 16–18. Note that these are relations, thus the definition of the 30Hz-clock from c_{10}.

$$c_{100} \boxed{\equiv} \text{idealClk}\ \texttt{discretizedBy}\ 0.01 \tag{16}$$

$$c_{10} \boxed{\equiv} c_{100}\ \texttt{filteredBy}\ \left(1.0^{9}\right) \tag{17}$$

$$c_{10} \boxed{\equiv} c_{30}\ \texttt{filteredBy}\ \left(1.0^{2}\right) \tag{18}$$

Now a schedule can be given with the kind of communication and with the three cases: $\widehat{t1} \boxed{\equiv} \widehat{t2} \boxed{\equiv} c_{10}$ (synchronous), $\widehat{t1} \boxed{\equiv} c_{10}$ and $\widehat{t2} \boxed{\equiv} c_{30}$ (oversampling), $\widehat{t1} \boxed{\equiv} c_{30}$ and $\widehat{t2} \boxed{\equiv} c_{10}$ (undersampling),

For an immediate communication, the only case where a partial order remains is for undersampling. From the specified constraints we can only infer that, for each hyper-cycle, the first execution of read_data must complete before the execution of control. However, we cannot decide when to execute control relatively to other executions of read_data. To determine this schedule, we can take as a criterion the actual size of the buffer used for the communication. To get this buffer as small as possible (size=1), we need to schedule control before the second execution of read_data. Were we to schedule according to an earliest deadline first (EDF) policy we would get another schedule. See Fig. 14 for possible schedules.

For a delayed communication, we just have partial orders. Additionnal criterion must be given to get only one schedule. For synchronous threads, the use of an EDF policy is of no help. However, reducing the size of the communication buffer give a schedule (top-most part of Fig. 15). For oversampling, both criterion are compatible and we get the second schedule on Fig. 15. For oversampling, we get two different schedules depending on whether we apply an EDF policy or we attempt to reduce the buffer size.

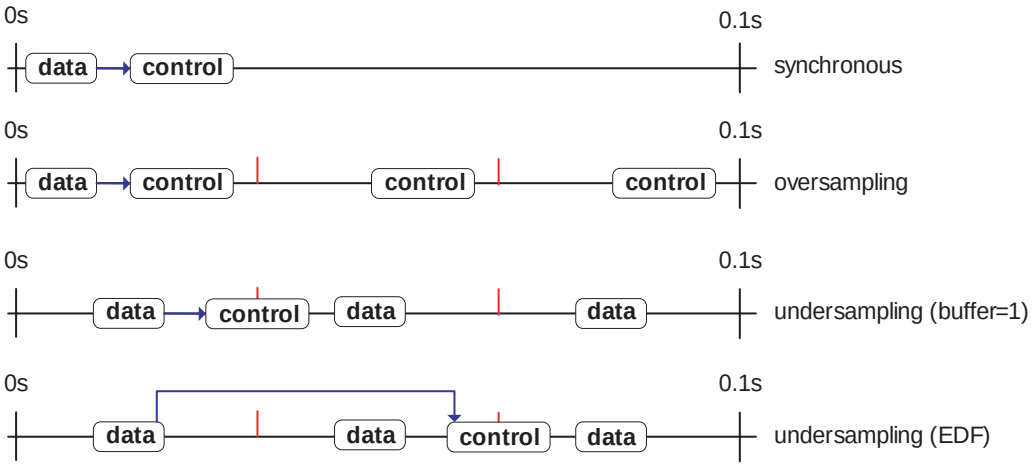


Figure 14. Schedules with immediate communications.

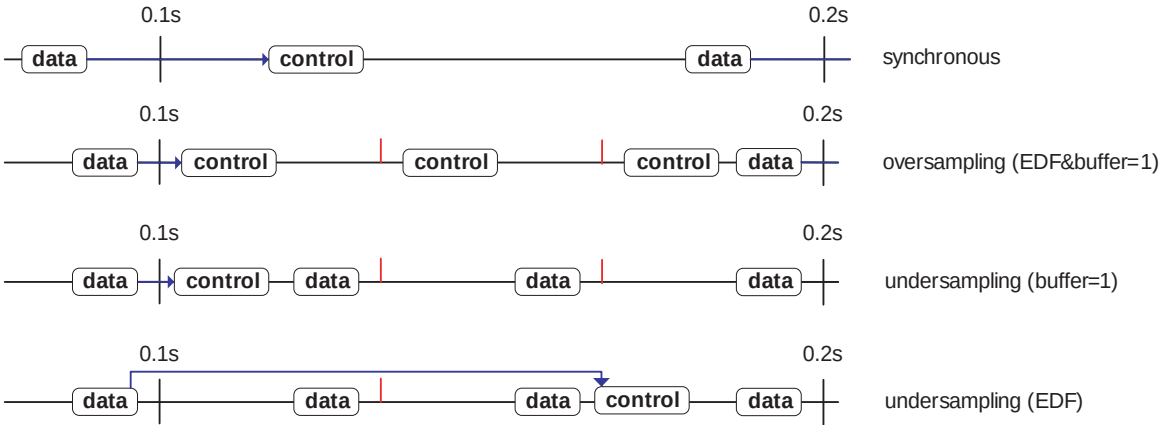


Figure 15. Schedules with delayed communications.

4.2.6 Generalization

We can generalize the constraints to get only two sets of constraints, one for the immediate communication and one for the delayed communication.

In this section we do not restrict to the three special cases addressed in the AADL standard. This generalization does not assume that the frequencies of the threads are natural numbers, it just assumes that they are rational numbers. It also assumes that in the notation of our binary words $Y.x^0 = Y$, for any binary word Y and any bit x.

Let $f_d = n_r/d_r$ and $f_c = n_c/d_c$, $f_d/f_c = (n_r * d_c)\,/\,(n_c * d_r)$ with $n_r, n_c, d_r, d_c \in \mathbb{N}^\star$. Let $r_1 = n_r * d_c$ and $r_2 = n_c * d_r$. We choose $q1$ and $q2$ such as $q1 = r_1/\gcd(r_1, r_2)$ and $q2 = r_2/\gcd(r_1, r_2)$. Note, that we still have $f_d/f_c = q1/q2$. Then, the general constraints are given in Table 1. Note that, in particular cases mentioned in previous sections we get exactly the same constraints.

Again, these constraints are purely logical. In the general case, these constraints are not strong enough to identify deterministically the significant writings and readings. If we take for instance, the case where $q1 = 2$ and $q2 = 5$

	Constraints
immediate	$\widehat{wr} \boxed{\equiv} \widehat{d}\ \texttt{filteredBy}\ \left(1.0^{q1-1}\right)$
delayed	$\widehat{wr} \boxed{\equiv} \widehat{t1}\ \texttt{filteredBy}\ \left(1.0^{q1-1}\right)$
both cases	$\widehat{t1}\ \texttt{filteredBy}\ \left(1.0^{q1-1}\right) \boxed{\equiv} \widehat{t2}\ \texttt{filteredBy}\ \left(1.0^{q2-1}\right)$ $\widehat{rd} \boxed{\equiv} \widehat{c}\ \texttt{filteredBy}\ \left(1.0^{q2-1}\right)$

Table 1. Generalization of constraints

(Fig. 16). If we apply the AADL semantics, we can only say that, within an hyper-cycle (of period lcm$(q1, q2)$), the first execution of read_data produces the sample for the first control, but we cannot know what sample is used by other executions of control. In particular, there is no relation between $t1[2 * n + 1]$ and $t2[5 * n + 2]$.

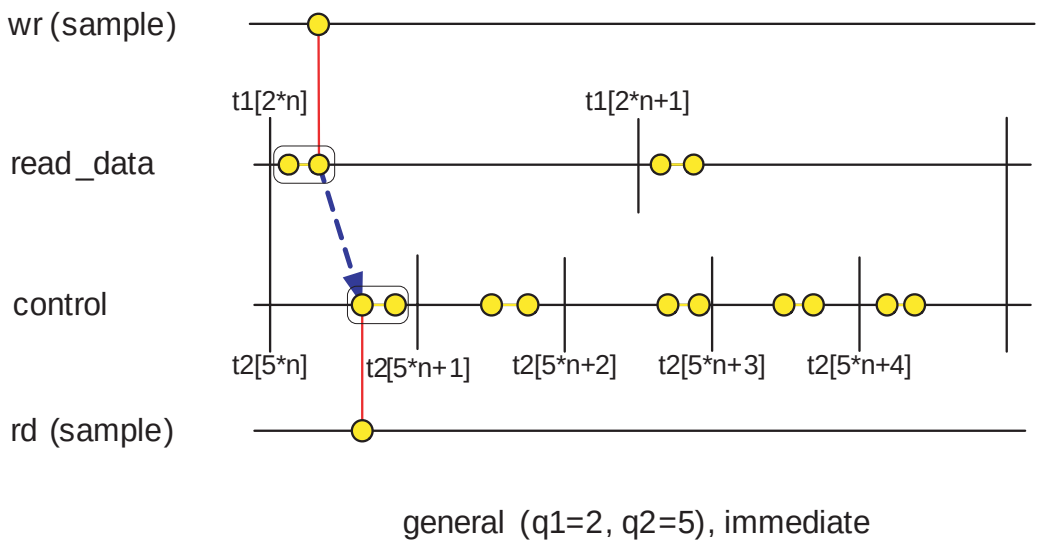


Figure 16. General case with immediate communications and purely logical clocks.

To get a deterministic behavior, we need to give more constraints. A simple solution consists in giving constraints in relation to a chronometric clock. For instance, we can model the cases where $f_d = 10$Hz and $f_c = 25$Hz. We proceed by using the clock c_{100} previously defined (see Eq. 16). Then, we add the two constraints given in Eq. 19 and Eq. 20.

$$\widehat{t1} \boxed{\equiv} \mathsf{c}_{10} \tag{19}$$

$$\widehat{t2} \boxed{\equiv} \mathsf{c}_{100}\ \texttt{filteredBy}\ (1.0^3) \tag{20}$$

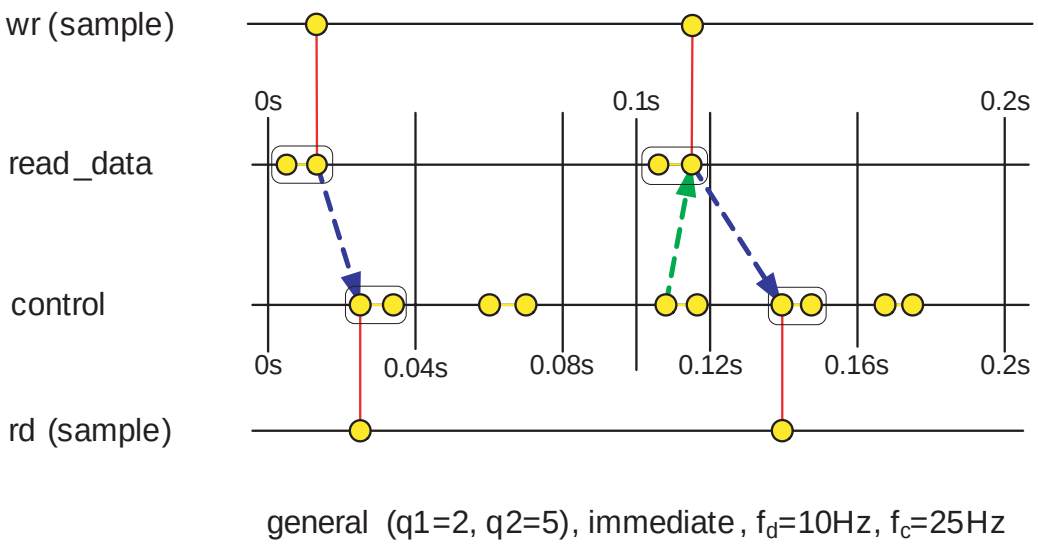
With such constraints, we get a total order (Fig. 17) and then there are two possible cases. The first case appears when $duration(read\_data) + duration(control) \geq 0.02s$. Then, we exactly get the result presented in Fig. 17, where, within an hyper-cycle, the third execution of control uses the sample computed by the first execution of read_data and the fourth execution of control uses the sample computed by the second execution of read_data. In the second case, if $duration(read\_data) + duration(control) < 0.02s$, the third execution of control should use the sample computed by the second execution of read_data. However, note that such systems that very much depend on the exact duration of tasks are not very robust.

If we now take a look at the situation with a delayed communication, with have several possible interpretations of the AADL semantics. The simplest interpretation is that the data is made available (written in the object node) at the first dispatch (of the sending thread) following the execution of the behavior that has produced it (read_data). And the data is read at the first dispatch of the receiving thread following the write (see Fig. 19).

A second interpretation could be that the data is read at the first dispatch of the receiving thread following the actual production of the data (not waiting for the following dispatch of the sending thread). This second interpretation leads to Fig. 20.

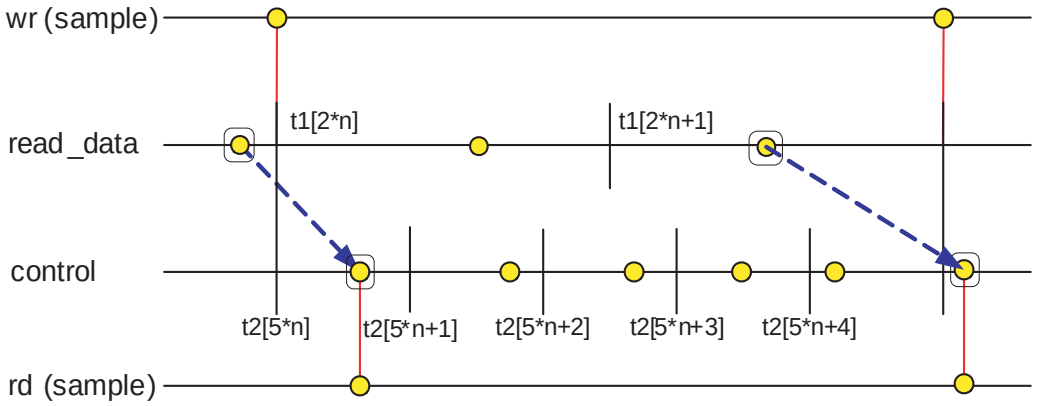
The two interpretations are deterministic. The first one is very simple to implement and the second one requires to be able to control very tighly the communication times.

A UML object node has two interesting attributes: it has an upper bound, possibly unlimited, and it can order events, by default according to a FIFO policy. Thus, there is no reason to assume that the threads are in phase, the sending thread writes (and possibly overwrites) tokens in the object node, while the receiving thread reads them when required. Our definition of the significant writings and readings helps defining when the token is the same—the content must be ovewritten—and when the token is different, which implies that a new token must be created. Actually, the occurrence of $\widehat{wr}$ should create a new token.



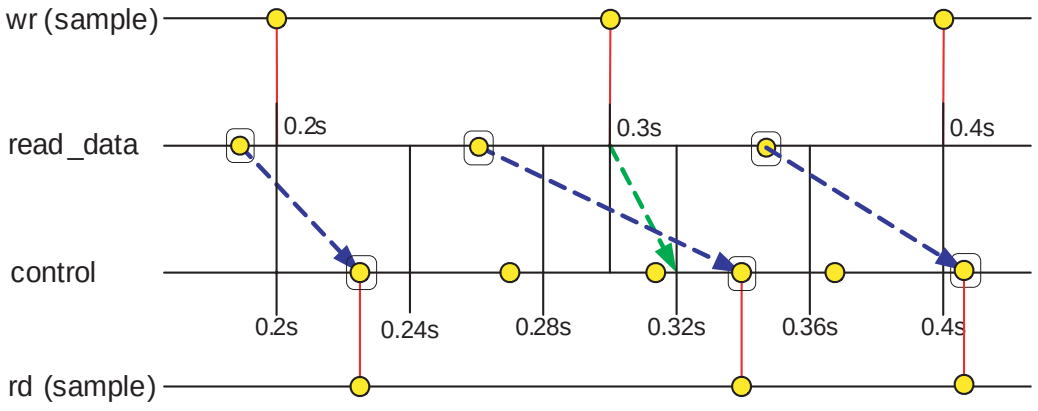
general (q1=2, q2=5), immediate, f_d=10Hz, f_c=25Hz

Figure 17. General case with immediate communications and chronometric clocks.



general (q1=2, q2=5), delayed

Figure 18. General case with delayed communications.



general (q1=2, q2=5), delayed (1)

Figure 19. General case with delayed communications (first interpretation).

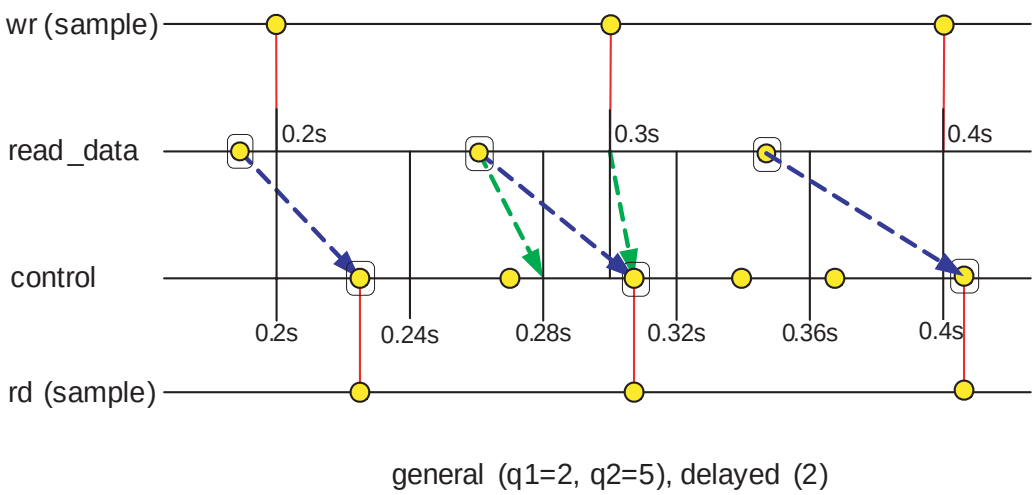


Figure 20. General case with delayed communications (second interpretation).

5 Conclusion

We presented a UML profile for comprehensive Time Modeling. Time here can be of discrete or dense, physical or logical nature. 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 and so on). 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 TimedEvents and TimedProcessing, and to structural elements, leading to clocked Classes and clocked Objets. 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.

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 *in a formal way*. We also showed the intent behind logical time by considering examples with various clocks running at unrelated speeds.

References

[1] C. André, F. Mallet, and M.-A. Peraldi-Frati. 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, April 2007.

[2] L. Apvrille, P. Saqui-Sannes, and F. Khendek. TURTLE-P: a uml profile for the formal validation of critical and distributed systems. *Software and Systems Modeling (SoSyM)*, 5(4):449–466, December 2006.

[3] J. Buck, S. Ha, E. Lee, and D. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *International Journal of Computer Simulation, special issue on "Simulation Software Development"*, 4:155–182, April 1994.

[4] A. Darte, Y. Robert, and F. Vivien. *Scheduling and Automatic Parallelization*. Birkhaüser, 2000.

[5] P. Feautrier. Compiling for massively parallel architectures: a perspective. *Microprogramming and Microprocessors*, (41):425–439, 1995.

[6] S. Graf, I. Ober, and I. Ober. A real-time profile for UML. *STTT, Software Tools for Technology Transfer*, 8(2):113–127, April 2006.

[7] A. Jantsch. *Modeling Embedded Systems and SoCs - Concurrency and Time in Models of Computation*. Morgan Kaufman, 2003.

[8] E. A. Lee and A. L. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12):1217–1229, December 1998.

[9] 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., February 2005. OMG document number: realtime/2005-02-06.

[10] OMG. *UML Profile for Schedulability, Performance, and Time Specification*. Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701., January 2005. OMG document number: formal/05-01-02 (v1.1).

[11] OMG. *Object Constraint Language, version 2.0*. Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701., May 2006. OMG document number: formal/06-05-01.

[12] OMG. *UML 2.1 Superstructure Specification*. Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701., April 2006. OMG document number: ptc/2006-04-02.

[13] I. project. *EAST-ADL: The EAST-EEA Architecture Description Language*, June 2004. ITEA Project Version 1.02.

[14] B. Selic. On the semantic foundations of standard uml 2.0. In *SFM-RT 2004*, volume 3185 of *LNCS*, pages 181–199. Springer-Verlag, 2004.

[15] S. Standards. *SAE Architecture Analysis and Design Language (AADL)*, June 2006. document number: AS5506/1.

A Clock Constraint Specifications

In this annex, a non-exhaustive list of useful clock relations is provided. Remember that these relations are often relational in the sense that they do not lead to a unique solution, and that a specific solution is provided only when a specific deterministic scheduling is given. In the expressions below, a, b, c are clock references.

Clock relations based on the coincidence relation :

Equality "$a \boxed{\equiv} b$" (or $\texttt{equal}(a, b)$) states that the two clocks are identical (all their instants are pairwise coincident).

Disjunction "$a \boxed{\#} b$" (or $\texttt{disjoint}(a, b)$) states that they share no instant (no coincident instant).

Subclocking stands between these two extreme cases. b is a *subclock* of a if for each instant i of b there exists a coincident instant $h(i)$ of a, and the injective mapping h from $\mathcal{I}_b$ to $\mathcal{I}_a$ is order preserving ($(\forall i, j \in \mathcal{I}_b) i \preccurlyeq_b j \Rightarrow h(i) \preccurlyeq_a h(j)$). "a is a *superclock* of b" stands for "b is a *subclock* of a". Several clock relations rely subclocking.

"a `isFinerThan` b" states that b is a subclock of a.

"b `isCoarserThan` a" stands for "a `isFinerThan` b".

"$b = a$ `restrictedTo` p", where p is a predicate on the instant set of clock a. This relation states that b is a subclock of a, and p selects the instants of a that are coincident with the instants of b ($\mathcal{I}_b = h^{-1}(\{i \in \mathcal{I}_a \mid p(i)\})$). For convenience, specializations of this relation have been defined:

"$b = a$ `discretizedBy` r", where a is a dense time clock, and r is a non negative real number. This relation says that b is a discrete time clock equal to the dense clock a discretized with a period r.

"$b = a$ `filteredBy` p", where p is a pattern, states that b is the finest subclock of a such that only the instants of a selected by the pattern p are coincident with instants of b. The patterns are periodic words on $\{0, 1\}$.

Two clock relations :

"$c = a$ `excluding` b", states that c is the finest subclock of a with no instant coinciding with instants of b.

"$c = a$ `followedBy` b", where $a \# b$ and a is finite. This relation states that a and b are subclocks of c; any instant of c is coincident with an instant of either a or b; and any instant of c coincident with an instant of a precedes any instant of c coincident with an instant of b.

Relations involving a third implicit clock c, superclock of the operand clocks

"a `isFasterThan` b" states that the k^{th} instant of a always precedes the k^{th} instant of b, this ordering being observed on c.

"n $\texttt{maxDrift}(a, b)$", where n is a natural number, means that the absolute difference between the number of instants of c coincident with instants of a and the number of instants of c coincident with instants of b, before the k^{th} instant of c is bounded by n for all k. In other words: clocks a and b are not diverging.

Clock relations based on the precedence relation :

Periodicity `isPeriodicOn` is a general relation stating kinds of periodic dependencies.

"b `isPeriodicOn` a", where a and b are discrete time clocks. This relation roughly says that there exists a natural number p such that there is one instant of b every p consecutive instants of a. A more precise formulation is given below, with the extended form of this relation.

"b `isPeriodicOn` a `period` p `jitter` $jmin$, $jmax$", where p, $jmin$, and $jmax$ are natural numbers. This relation means:

$$(\exists d \in \mathbb{N})\ (\forall k \in \mathbb{N})\ (a[d + p * k - jmin] \prec b[k] \preccurlyeq a[d + p * k + jmax])$$

"b `alternatesWith` a" is a particular instance of `isPeriodicOn` where $p = 1, jmin = 1, jmax = 0$, so that instants of b and a alternate.

"b `isSporadicOn` a", where a and b are discrete clocks, means that there exists a natural number n such that there are at least n consecutive instants of a between two successive instants of b.

Weak Subclocking A variant of the *subclocking* relation replaces coincidence by precedence. b is a *weak subclock* of a if for each instant i of b there exists a instant $h(i)$ of a, such that $i \preccurlyeq h(i)$ and $^\circ h(i) \prec i$, where $^\circ j$ denotes the immediate predecessor of instant j in a discrete time clock.

Other clock relations :

"$c = a$ `sampleTo` b", where a, b, and c are discrete time clocks. This relation states that c is a subclock of b such that for each instant $b[k]$ of b coincident with an instant of c, there exists an instant $a[l]$ of a such that $b[k-1] \prec a[l] \preccurlyeq b[k]$.

"$a =$ `when` e", where e is a reference to an event. This is a way to create a clock from an event: there is an instant of the clock for each occurrence of the event.

Clock non functional property relations :

A clock non functional property (clock NFP) relation is a constraint that applies to chronometric clocks, and specifies time related non functional properties for a chronometric clock or a group of chronometric clocks. These relations involve time measurements performed on a reference clock (idealClk, by default). We only describe the relations used in Section 4.

"a `hasStability` r", where a is a chronometric clock and r is a real number. This relation states that the stability of clock a is r. *Stability* is the ability of a clock to report consistent intervals of time; it is measured by derivatives of the clock rate against environmental factors. By default, the stability is against time ($\max\{\partial f/\partial t\}$, where f is the clock rate).

"a , b `haveOffset` r", where a and b are chronometric clocks and r is a real number. The *offset* is the difference between two chronometric clocks at a particular instant in time. This relation states that the offset between the two clocks a and b is r. By default, r is measured on idealClk.

Two functional relations : a projection (filtering) operator can be applied to a clock to define new (sub) clocks. There exist two versions (strong and weak) of this operation. The left-hand side argument is a clock, the right-hand side argument is a binary word, often infinite and periodic.

(strong) filtering "$a \blacktriangledown w$", where a is a discrete time clock, and w a binary word. This relation specifies a discrete clock c such that, c is a subclock of a, and $\forall k \in \mathbb{N}^\star, c[k] \equiv a[w \uparrow k]$.

weak filtering "$a \triangledown w$", where a is a discrete time clock, and w a binary word. This relation specifies a set of discrete clocks such that, these clocks are weak subclocks of a, and for each subclock c the following property holds: $\forall k \in \mathbb{N}^\star, a[(w \uparrow k) - 1] \prec c[k] \preccurlyeq a[w \uparrow k]$.

$w \uparrow k$ is the index of the k^{th} 1 in w. This operation can be recursively defined by: $(1.w) \uparrow k = 1 + w \uparrow (k-1)$, $(0.w) \uparrow k = 1 + w \uparrow k$, and the trivial cases are $w \uparrow 0 = 0$ and $\epsilon \uparrow k = 0$, where ϵ is the neutral element for the binary word concatenation.

Contents