# INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *Modeling with logical time in UML for real-time embedded system design*

Charles André — Arnaud Cuccuru — Robert de Simone — Jean-Pierre Talpin

## N° 5895

Avril 2006

—————— Thème COM ——————

*Rapport de recherche*

# Modeling with logical time in UML for real-time embedded system design

Charles André , Arnaud Cuccuru , Robert de Simone , Jean-Pierre Talpin

**Abstract:** Design of real-time embedded systems requires particular attention to the careful scheduling of application onto execution platform. Precise cycle allocation is often requested to obtain full communication and computation throughput.

Our objective is to provide a UML profile where events, actions, and objects can be annotated by "logical" clocks. Initially, clocks are not necessarily related. The goal of the scheduling process (and algorithms) is to regulate the data and control flows within predictable bounds. To this end it extracts clock relations that best map the application onto a desired execution platform. "Clocks-as-schedules" then act as activation conditions, driving these internal events and actions according to the desired activation patterns. Extra communication and buffering latencies can be introduced in the process.

In the paper we describe the domain view of multiple time and logical clocks. We introduce a range of useful operations on them, and their use in various UML views.

**Key-words:**  UML, Logical time, Real-time embedded systems

# Modélisation UML temps logique pour la conception des systèmes embarqués temps réel

**Résumé :** La conception des systèmes embarqués requiert une attention particulière quant à l'expression du placement/ordonnancement des applications sur les plateformes d'exécution. Une allocation au cycle près est souvent requise afin d'obtenir un débit maximal en terme de calcul et de communication.

Notre objectif est de fournir un profil UML où les évènements, les action et les objets peuvent être annotés par des horloges "logiques". A la base, il n'existe pas nécessairement de relations entre ces horloges. Le but du processus d'ordonnancement et des différents algorithmes associés est de réguler les flots de donnée et de contrôle à l'intérieur de bornes prédictibles. Pour cela, ces algorithmes extraient des relations entre horloges qui permettent une allocation optimale de l'application sur la plateforme d'exécution ciblée. Les "horloges d'ordonnancement" agissent ainsi en tant que conditions d'action, pilotant les évènements et actions internes selon les motifs d'activation désirés. Au cours du processus, des latences suplémentaires liées aux communications et bufferisations peuvent être introduites.

Dans le document, nous décrivons la vue domaine du temps multiple et des horloges logiques. Nous introduisons plusieurs opérateurs de manipulation d'horloges, et leur utilisation dans différentes vues UML.

**Mots-clés :** UML, Temps logique, Systèmes temps-réel embarqués

# Contents

# 1   Introduction

As embedded real-time systems have become pervasive and ubiquitous in contemporary technologies, their development requires highly reliable approaches. To meet these safety and performance requirements, design has to be supported by a trustable mathematical basis to provide required formal concepts. A noteworthy example is the attention required to the careful scheduling of functions and operations to be performed by the application on the targeted execution platform, which demands precise cycle allocation to obtain full communication and computation throughput.

However, precise cycle allocation is error-prone and tedious. A corpus of methods has been proposed in recent years, where these precise relations are obtained by analysis and optimization techniques from more relaxed high-level modeling of systems. In other words, the final scheduling is synthetized algorithmically from constraints on the application, the target HW/SW execution platform, and the allocation mapping of functions to resources.

Our current objective is to provide a UML modeling framework (profile) in which to represent the ingredients of such an approach. Indeed, UML provides broad ways to specify the different modeling views involved. But its largely "untimed" basis needs to be augmented with proper semantical annotations on temporal aspects.

While akin to many previous attempts at time modeling in UML (UML-RT [17], RT-UML [8], ACCORD/UML [10], SPT [14],...), our proposal still differs from them in several ways. It is based on "logical" time bases (or *clocks*) that are introduced to count/tick/trigger successive behaviors of signals, actions, objects (and so on). Clocks that are mutually independent (or only loosely coupled) provide for multiple time models. Thus clocks should act as *activation conditions* driving the internal events and actions according to the desired activation patterns. Relations between clocks can be provided by users, or infered by analysis from the system description. In the latter case specific optimization algorithms can take advantage from some usual limitations of RTE systems (predictability, determinism, static computation structure) to propose spatial and temporal allocations that will best fit functional applications to execution platforms. This has the effect of strenghtening the relations between clocks, constraining them to a more rigid interdependency.

We describe operators that are used to combine and compare them, and their introduction to annotate behaviors and objects. Clocks can be hierarchically organized, fully or partially. A clock can be a periodic subclock of another (amongst other). Ultimately, a clock which upsamples all others in the system can be thought of as "discrete physical time", but the existence of a link to physical time is not mandatory in our modeling framework.

In a given methodological flow, the approach should be used to adjust the various rates involved in different parts of the application under design. Communications can be introduced by the spatial mapping of functions onto concurrent resources. Extra latencies and buffering objects may also be requested at places to regulate the data and control flows within predictable bounds.

Logical clocks associated with successive object or action behaviors are a convenient way to represent explicit schedules as first-class citizens of the model. Such schedules can then be named, visualized, and computed upon. Results can be displayed back to the designer as a clock schemes refinement. It provides effective information on the mapping decisions taken towards implementation.

The paper is organized as follows: section 2 describes the time model and its logical clocks, their insertion in the application model and their allocation to an execution platform model. Section 3 provides more technical definitions on clocks and schedules, focusing on the discrete time case. Section 4 describes a case study which highlights the approach. We conclude with perspectives.

**Related works** Our approach should appeal to readers familiar with model-driven design based on mathematical *Models of Computation* [3, 11]. The notion of logical clocks providing the schedule framework owes to the theory of tagged systems [13]. *Synchronous reactive formalisms* [2] are examples of languages that handle (logical) time as explicit mechanisms at the very heart of their semantic foundations. Schedule computations from data-flow based application descriptions were developed in the theory of *Timed Event Graphs* [4, 1]. It has found many usages, with some important developments for software pipelining [7] and hardware circuit timing [15]. It found some early incarnation in the context of synchronous programming with the theory of affine clocks [18]. The explicit representation of schedules in the discrete/periodic/regular case was introduced in the theory of *N-synchronous* processes [5].
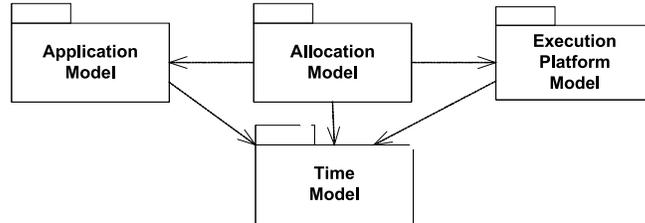
# 2   Modeling framework



Figure 1: The various models and their package architecture

Our approach relies on the modeling framework displayed in figure 1. A *functional application* is to be *allocated* onto a candidate architectural *execution platform*. The application may exhibit potential concurrency and relative asynchrony between various treatments, as well as subsystems running at various related speeds. The platform may consist of mixed hardware and basic software parts. The *allocation mapping* consists of both *spatial distribution* and *temporal scheduling* of functional operations onto platform resources and services. Note that the words "functional" and "architectural" here do not exactly match "behavioral" and "structural" notions. Functional applications have a strong behavioral impact, but contain also hierarchical structure descriptions; execution platforms, while providing the architectural block-diagram of resources and connections, also describe some basic behavioral "service" aspects of the platform.

The allocation mapping between the application and the platform can be entirely specified by the designer, or computed by analysis from a number of characterization figures. It will result in a refinement of the time model of the application by the architecture constraints. This time refinement will be reflected in a tighter set of relations between logical clock schedules. For instance relatively independent clocks can be interleaved when functions are mapped to the same resource. The importance of schedule descriptions as explicit model elements should fully appear here.

## 2.1   Time Model

Our objective here is to introduce conceptual definitions related to our vision of *Time*. The essential ones are represented in their domain view in figures 2 and 3.

The TimeBaseModels package (Fig. 2) introduces a structural view of Time. Time can be *simple* (totally ordered), or *multiple* (partially ordered in the form of several, loosely coupled time bases). Precedence of instants can thus be defined as a partial relation.

A clock attaches quantitative informations (time values) to instants or set of instants (Fig. 3). A clock can be *logical* or *chronometric* (the second indicating that time is supposed to be measured from physical devices, external to the model by definition). In the paper we shall concentrate on (more novel) logical time.
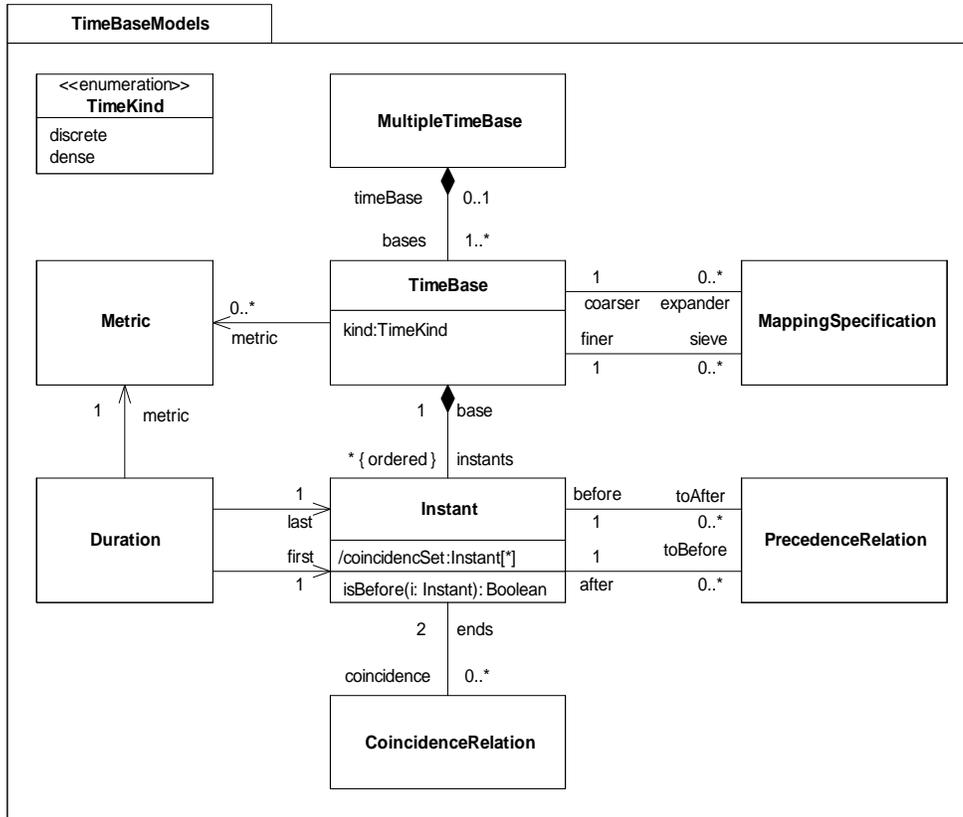
Figure 2: The time base models package

Generally, an instant value on a given clock, may denote many instants. When the optional clock attribute `maximalAttribute` is not defined, an instant value denotes one instant at most, so that a clock provides an unambiguous access to instants. This is the case for the logical clocks we consider in this paper. So, even if time base and clock are two different concepts, the word clock will be often used in place of time base, this in accordance with standard usages.

Time can be *discrete* or *dense*. We shall stick here to discrete time, where Time Bases can be seen as generated from clock ticking events. When modeling repetitive tasks with discrete time it becomes possible to express explicit schedules relating the respective paces of clocks rather easily (which does not mean that it is always impossible in the dense case).

Establishing tighter relations between clocks shall be the main scheduling purpose in the methodological system design activity associated with our modeling approach. Coincidence of instants
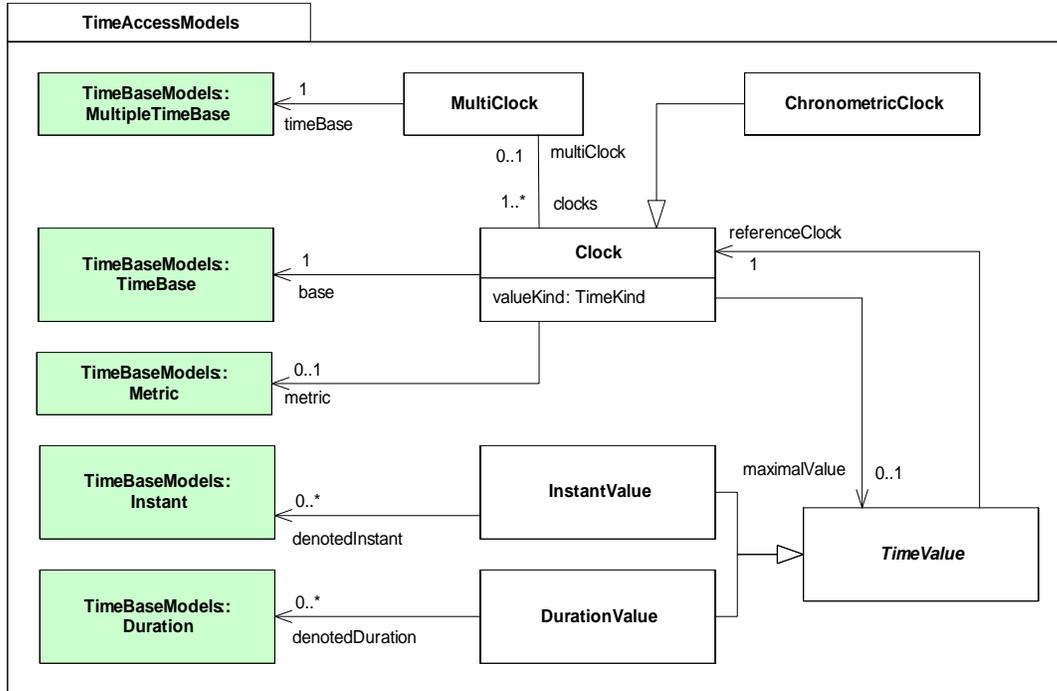
Figure 3: The time access models package

ticked by distinct clocks is allowed (simultaneity), so that synchronous activities can be modeled. Special time value expressions are provided to express coincidences as mappings between clocks.

Providing a schedule relation between two clocks will often consist in providing a third one, faster than both, on which the instants of each will be projected. Then the rate and phase relations will explicitly show. Operators for defining schedules in the case of discrete logical time will be presented in section 3.

## 2.2   Application Modeling

Applications should be represented using the familiar UML modeling views for structural and be-havioral aspects: state, activity and interaction diagrams, structured classes and components. In the field of RTE systems it is often the case that activity and "component blocks" diagrams play a particular role, as noticed through their emphasis in SysML for instance.

The primary extension to these models to build our application model shall be the introduction of the logical clocking information available (or requested) as annotations. The objective is to provide schedules for elementary behavioral elements such as *actions* and *events*. When a collection of
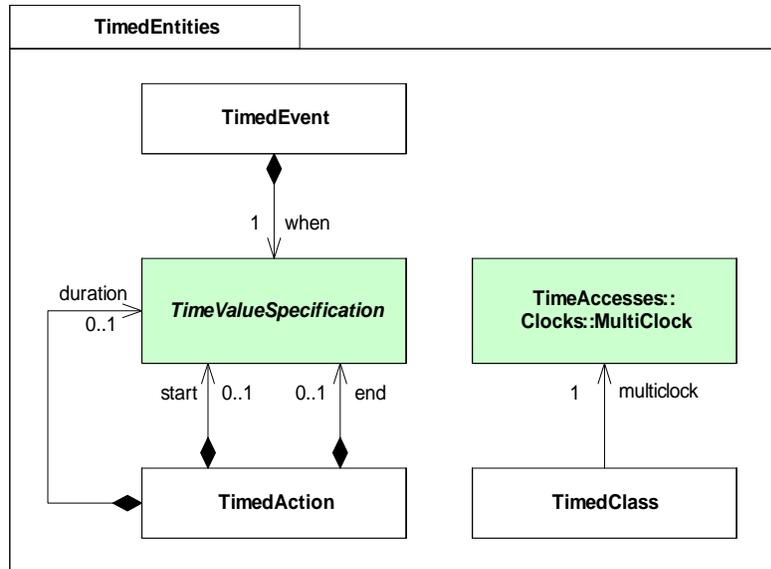
Figure 4: The time entities package

behaviors is "on the same clock", the clock information can be put on the structural container they all belong to, when it exists (classifier, structured class, component, . . . ). The activation instants of behaviors on a given clock can be provided as *absolute values*, but much more often as *relative values* introducing latencies and durations. Such values are provided usually as TimeValueExpressions. The extension of relevant modeling elements to their Timed versions is displayed in figure 4.

### 2.2.1 Timed semantics

The operational semantics of UML is basically untimed (some would called it *asynchronous*). When timing considerations are introduced, they are often added as external "non-functional" constraints, and their satisfaction is not demanded explicitly in the "official semantics". This is certainly fine to define a computational model that works when these timing annotations are not provided, and we certainly do not want to question this model in such case. But when explicit timing (logical or physical) is specified in the model, the semantics should of course take it into account (or else the UML semantics be discarded for modeling intentions that would only rely in its diagrammatic views and discard its meaning). Following Bran Selic [16], we see two places where this is important:

**inter-object communications.** Example questions here are: What is the time relation between the *send* and *receive* actions on a signal event? How could one model rendez-vous synchronization when modeling demands? The actual arriving time can have a drastic impact on real-time

scheduling techniques (and then on the ordering of behaviors in a faithful operational seman-
tics).

**intra-object communications** Actions are triggered by object/data flows and control flows. But in
the current state of UML 2.0 there is not much differences betwen the two kinds of flows.
*Data-flow* models are found very useful for abstract modeling, but hardly used for real-life
semantics. Instead they are usually transformed (by classical scheduling) in a model where all
information on cycle time activation is carried by the control flow (it "provides" control), and
the data-flow is weakened to a *data-path*; indeed, control-flow design is supposed to ensure
that the proper data have arrived in the proper locations (here, at the input pins of the action),
when it is activated. Setting up the control flow right amounts in our framework to a clock
calculus, setting up the clock rates right to design this schedule.
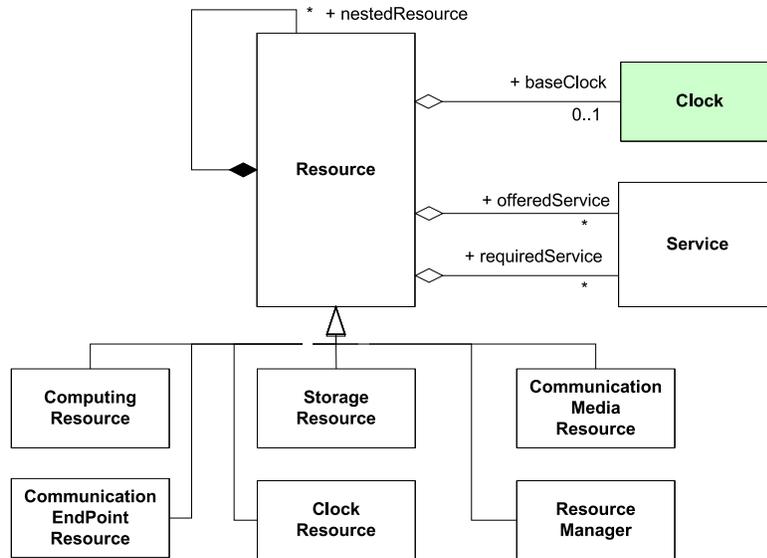
## 2.3   Execution Platform Modeling



Figure 5: Execution platform metamodel, with examples of hardware specific resources refinement
(simplified diagram)

The purpose of the Execution Platform Model is to enable embedded systems designers to spec-
ify and dimension the architectures meant to support the applications. The actual allocation (of
application to platform) can be entirely specified by the designer, or sometimes computed by anal-
ysis tools [12]. Such tools require basic information on the computation and communication costs

for basic functions, from which they attempt to minimize the overall cost of well-chosen allocations. Details are out of the scope of this paper. The result can be displayed as extra clock relations. Typically a new ground clock is introduced, on which existing clocks are synchrononized to the desired effect. For instance functions that share a resource must be interleaved.

As illustrated in Fig. 5, the model mainly introduces two concepts: *resources* and *services*. Structurally, an execution platform is a block-diagram of resources of several natures: *computing, storage, communication endpoints*, and *interconnect media*. Platform services use a predefined number of resources in a way described as an interaction scenario. *Generic* services describe natural aspects of the platform and can be used to introduce *costs* at a higher description level. *Specific* services can be used to provide descriptions that come to the level of application functions. This is used to bridge the possible gap in atomicity level: one groups the necessary platform behaviors and resources so that it can realize directly the function.

No assumptions are made on the granularity of the resources considered in the specification of an execution platform. According to nature of the application, an execution platform can be a coarse grain description of a basic software operating system offering services for thread and memory management, or very fine grain view point of a hardware execution platform (down to ASICs). Note that the hierarchical aspect of the metamodel (composition relation between resources in Fig. 5) enables all kinds of layering specifications. For example, the bottom layer of an execution platform (typically specified as an assembly of fine grain hardware resources) can be abstracted by a higher level software execution layer.

Resources timing and clocking information are often of a more physical nature as those of the application. Time bases provide a specification of computing speeds for hardware elements. Durations provide an account of computational complexity. For example, in a hardware execution platform, physical relations typically exist between the operating frequencies (clocks) of the various resources (e.g., a bus clock ticking one time every ten ticks of the processor clock, i.e., the bus clock is ten time slower than the processor clock but is synchronized with it).

## 2.4 Allocation modeling

The Allocation metamodel provides mechanisms that enable to express relationships between the elements describing the structure and behavior of an application, and the resources describing the structure and behavior of the targeted execution platform. The purpose of these relationships is to describe how the functionality expressed in the application will be eventually realized by the execution platform.

The Allocation metamodel offers different kinds of relationships (structure to structure, behavior to structure, behavior to behavior), that will not be described in details in this paper. But ultimately, an action considered as atomic in the application must be related (directly or indirectly through an analysis process) to the (possibly multiple) execution of one or more services supported by one or several resources of the targeted execution platform. This property is illustrated in Fig. 6, where Collaboration and Behavior concepts have a have similar semantics to homonymous UML 2 concepts (respectively the set of structural elements, i.e., resources, that will collaborate to "realize" the

action, and the description of the interactions, i.e., services executions, between the different parts of the collaboration).
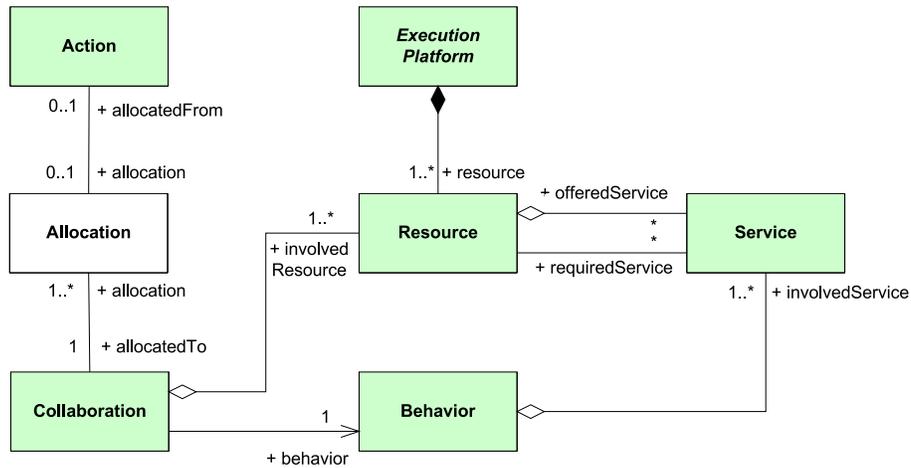


Figure 6: Allocation metamodel (partial)

According to the various costs associated with the "local" services executions and according to the description of the interactions between the resources involved, concrete temporal informations can be extracted and used to refine the temporal description of the application. A use case example of this process is given the case study section.

# 3 Clocks and schedules

This section provides operations on time bases, mostly meant to combine them. The usual way to combine two clocks is to project them both on a third one that "contains" them. This can be non-deterministic, or reflect a given scheduling strategy. When a clock $A$ is *finer* than a clock $B$ (meaning that $A$ ticks at least each time $B$ does), a schedule *filter* can be provided to retrieve $B$ from $A$. In effect it selects the instants where $B$ ticks out of the ones in $A$. In the discrete case it can be represented as a strictly increasing sequence of integers, or even more concretely by a sequence of bits. A "1" at a given index position $i$ in the sequence indicates that $B$ ticks at $A$'s $i^{th}$ tick.

We first introduce some notations.

**Binary words**  Let $\mathbb{N}$ denote the set of natural numbers including $0$, $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$, and $\mathbb{B} = \{0,1\}$ the set of bit values. The elements of $\mathbb{B}$ are ordered: $0 < 1$. A *binary word* is a possibly infinite sequence of Bits $w : \mathbb{N}^* \to \mathbb{B}$. Let Binary be the data type defined on binary words, equipped with the concatenation ($\bullet$). For $w :$ Binary, $k \in \mathbb{N}^*$, $w[k]$ denotes the $k^{th}$ bit in $w$. The index of the $k^{th}$ 1 in $w$ is denoted by $[w]_k$. If the $k^{th}$ 1 does not exist, then $[w]_k$ returns $\infty$, and by convention $[w]_0 = 0$.

We extend (bit-wise extension) Boolean operators on bits (not, or, and, ...) to Binary. We also define two relations on Binary:

**Subsetting**  ($w_1$ subsets $w_2$): $w_1 \subseteq w_2 \Leftrightarrow \forall k \geq 1, w_1[k] \leq w_2[k]$

**Lead**  ($w_1$ has lead over $w_2$): $w_1 \preceq w_2 \Leftrightarrow \forall k \geq 1, [w_1]_k \leq [w_2]_k$

**Periodic binary words**  A binary word $w$ is called periodic if of the form $u.v^\star$, with $u, v \in \{0,1\}^\star$. Periodic words are remarkable in that all the following constructions will be algorithmically computable on them. They are useful in modeling as they correspond to regular iterative and pipelined applications as often found in RTE domain. $u$ corresponds to a transitory initialization phase, and $v$ to a stationary cyclic behavior. The theory of periodic words as schedules is developed in [5].

**Discrete Time bases**  As shown in the time metamodel (Fig. 2), two time bases can be related by a mapping specification. We note $A = w_A^R$ *on* $R$, where $w_A^R \in$ Binary, to indicate that $A$ is obtained from $R$ by selecting only the instants $i$ where $w_A^R[i] = 1$. We omit the reference to $R$ in $w_A$ when clear from context. $R$ is called a *superclock* of $A$ given $w_A$, and $w_A$ is the filter for $A$ relative to $R$. We say that $R$ *upsamples* $A$ when $\exists w, A = w$ *on* $R$. We say that $A$ downsamples $R$ when $R$ upsamples $A$, and call $A$ a subclock of $R$. A set of clocks with a common superclock $R$ will be said to define a *Time Domain* $\mathcal{D}_C \subseteq \{A : \mathsf{TimeBase} \mid C \sqsubseteq A\}$.

Even though it may seem in our definitions that all computations are based on a common superclock, in practice the scheduling activities will consist in *choosing* such superclocks and schedule filters so that the various sister-clocks are properly articulated.

## 3.1 Relations on Discrete Time Bases

In the rest of the section we shall always suppose that $A$ and $B$ are clocks expressed on the same superclock $R$ (in other words $A = w_A$ *on* $R$ and $B = w_B$ *on* $R$).

**Disjunction relation** ($A$ and $B$ are disjoint): $A \mid B$ iff $\forall i, \neg(w_A[i] \text{ and } w_B[i])$.

**Finer/Coarser relations** ($A$ *is finer than* $B$): $A \sqsubseteq B$ iff $\forall i, w_A[i] \geq w_B[i]$

Disjoint clocks never tick simultaneously. A clock is finer than another one if it always ticks whenever the other does. We say that $B$ is coarser than $A$ when $A$ is finer than $B$.

It should be noticed than when $A$ is finer than $B$, then it can be seen as a superclock for $B$, with a proper (different) filter, defined as follows: $w_B^A[k] = w_B^R \left[ [w_A^R]_k \right]$. This says that the indexes of the new filters are those instants when $w_A^R$ holds as 1, and that the corresponding value is that of $w_B^R$ in these selected instants.

## 3.2 Operations on time bases

### 3.2.1 Point-wise operations on a Time Domain

The *complement* operation is a unary operation: complement : TimeBase $\rightarrow$ TimeBase and is defined by: $C = R \setminus A \iff w_C^R = \text{not } w_A^R$.
Tab. 1 contains some operations whose signature is TimeBase $\times$ TimeBase $\rightarrow$ TimeBase. Their semantics is given in terms of operations on Filters.

| Time base operation | Binary word operation |
|---|---|
| A union B | $w_A^R$ or $w_B^R$ |
| A intersection B | $w_A^R$ and $w_B^R$ |
| A minus B | $w_A^R$ and not $w_B^R$ |

Table 1: Some point-wise operations on Time Bases of a Time Domain

### 3.2.2 Global operations on a Time Domain.

**Downsampling** : TimeBase $\times$ Binary $\rightarrow$ TimeBase. $B = A|_w$ iff $(A \sqsubseteq B) \wedge \left( w_B^A = w \right)$.

**Upsampling** : TimeBase $\times$ Binary $\rightarrow$ TimeBase. $B = A|^w$ iff $(B \sqsubseteq A) \wedge \left( w_A^B = w \right)$.

**Delay** $\gg$: TimeBase $\times$ $\mathbb{N}$ $\rightarrow$ TimeBase. ($n$ is a duration). $w_{A \gg n}^A = 0^n \bullet 1^\star$

**Latest** $\sqcup$ : TimeBase $\times$ TimeBase $\rightarrow$ TimeBase.
$$\forall k \geq 1, \left[ w_{A \sqcup B}^R \right]_k = \max \left( \left[ w_A^R \right]_k, \left[ w_B^R \right]_k \right)$$

**Earliest** $\sqcap$ : TimeBase $\times$ TimeBase $\rightarrow$ TimeBase.
$$\forall k \geq 1, \left[ w_{A \sqcap B}^R \right]_k = \min \left( \left[ w_A^R \right]_k, \left[ w_B^R \right]_k \right)$$

**Sample** $\downarrow$: TimeBase $\times$ TimeBase $\rightarrow$ TimeBase (provides a subclock on $B$).

$$\forall k \geq 1, w^B_{A\downarrow B}(k) = \bigvee_{i=[w^R_B]_{k-1}+1}^{[w^R_B]_k} w^R_A(i)$$

**Delay, Earliest** and **Latest** operators are convenient ways to use clocks to regulate traffic and synchronize various flows. Some words of explanations might be in order here for the **Sample** operator. It is based on the pace of $B$ (as superclock), but only ticks whenever there was a tick on $A$ between the previous tick of $B$ (excluded) and the current tick of $B$ (included). So it sets the very instants where $B$ can be notified (on its clock) of some new event that occurred from a behavior based on $A$ clock.

Other relations may be derived to figure whether the **Sample** can be lossy (two successive events occurred on $A$ between two $B$'s ticks, or duplicating (no event on $A$ between two $B$'s ticks, so that if an activity based on $B$ needs a value produced at $A$'s rate, it will use the same twice). Such predicates are useful in AADL[9], a modeling framework used in automotive and aeronautics design.

We are currently working on ways to introduce appealing graphical UML views of these operators, so that clock adjustments for smooth scheduling of applications can be inserted by designers or, even better, notified by analysis tools to these designers for acceptance.

# 4   An illustrative example

As an illustrative example, we consider the downscaling of a high definition (HD) video image into a standard definition (SD) image. This example is primarily meant to show the use of *Clocks-as-schedules*. In that sense, it hardly uses Execution Platform and Allocation modeling, and focuses on Timed models inside application instead.

The downscaling application has been modeled with Multi-Periodic Process Networks [6], a model influenced by Petri nets, data-flow graphs, and Kahn Process Networks. A HD image consists of 1080 lines, each made of 1920 pixels. A SD image has 720 lines with 480 pixels each. The transformation reduces the number of pixels per line (ratio of 8:3), and the number of lines (ratio of 9:4). So, altogether the output pixel rate is $\frac{3}{8} \times \frac{4}{9} = \frac{1}{6}$ of the input rate. Functionally, the transformation can be decomposed in the horizontal filtering of each HD lines, followed by the vertical filtering of the resulting lines. Filtering includes smoothing (each new pixel results from a weighted sum of a neighborhood) and a decimation (discarding pixels). This transformation must be done in real-time: the pixels of input HD image are received at a rate imposed by the inClk clock, and the pixels of the output SD image have to be delivered at a rate imposed by the outClk clock. The two clock frequencies are specified in the standards. At the implementation level, horizontal and vertical filterings are performed in a pipe-line mode, which calls for a precise schedule of elementary operations. For simplicity we describe only the horizontal filtering.

Given the specification of the application and a target execution platform, we proceed as follows:

1. A static model of the main data structures handled by the application is designed. It brings out dimensional data-flow aspects, closely related to repetitive processings.

2. Functionalities of the application are expressed by activity diagrams or pseudo algorithmic representations. Local logical clocks should be introduced at this step to support scheduling.

3. Functionalities are allocated to services supported by an Execution Platform. Information about the duration of the service executions is collected from the platform model and integrated in the logical clocks.

4. A clock calculus determines the schedule of all actions. If the external timing constraints are not satisfied, then the activities or the local clocks have to be modified, and the procedure is applied again from step 3.

**Step 1**   The static model is given in Fig. 7. A HD line consists of 240 HD Horizontal Blocks (HDHoB), each made of 8 pixels. On the SD side, a line is also made of 240 SD Horizontal Blocks (SDHoB) of 3 pixels each. The filtering of a HD line can be refined into 240 horizontal block filterings. The horizontal block filtering reduces the line length in a ratio of 8:3.

**Step 2**   The logical clocks introduced for the HD line filtering are justified below and their equations are gathered in Tab. 2.
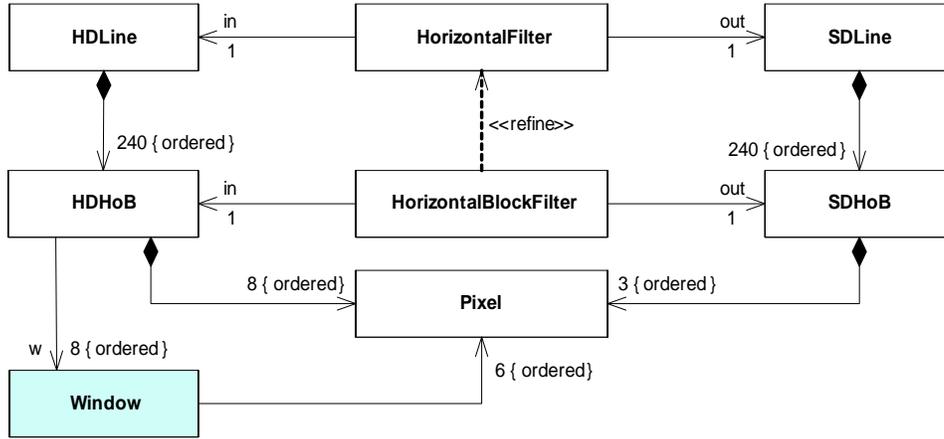
Figure 7: Static model

| # | clock name | equation |
|---|---|---|
| 1 | pxInClk | inClk and pxInClk alternate |
| 2 | HDHoBClk | $\text{HDHoBClk} = \text{pxInClk}|_{(1.0^7)^*}$ |
| 3 | HDLineClk | $\text{HDLineClk} = \text{HDHoBClk}|_{(1.0^{239})^*}$ |
| 4 | smoothClk | $\text{smoothClk} = \text{HDHoBClk}|_{(10100100)^*}$ |

Table 2: Clocks after step 2

The pxInClk clock samples the pixels received at the rate of inClk without loss or repetition. This holds whenever ticks of the two clocks alternate.

Since a HD horizontal block consists of 8 received pixels, the clock attached to the start of a HDHoB filtering (HDHoBClk) must be 8 times slower than pxInClk.

Since a HD line contains 240 blocks, the clock attached to the start of a HDLine filtering (HDLineClk) must be 240 times slower than HDHoBClk.

A block filtering should consist of 8 smoothing operations on a window. But because of the pixel decimation, only 3 out of the 8 are necessary. So, the 8 iterations making a block filtering are not identical. The body of the loop does or does not compute a smoothing function and generate a new pixel. This kind of variable iterations are elegantly expressed by a clock expression. Let smoothClk be the clock starting a body execution with smoothing and pixel generation. The binary pattern 10100100 preserves 3 results out of 8 computations. It reflects a design decision that fixes which pixels must be discarded, in an evenly spread way.

The WindowFiltering activity triggered by smoothClk is

1. Get the incoming pixel.

2. Push it in the (sliding) window w (the oldest pixel is lost).

3. Compute a dot product: $s = \sum_{k=0}^{k=5} w[k] * a[k]$, where $w[k]$ is the value of the $k^{th}$ pixel in the window, and $a$ is an array of weighting coefficients.

4. Create a pixel with value $s$ and insert it in the SDHoB.

The complementary clock of smoothClk (i.e., $\mathsf{HDHoBClk}|_{(01011011)^*}$) triggers only the first two actions of the above activity.

**Step 3** In order to select an Execution Platform, we have to consider required performance. 25 full images are sent every second. Each image has 1125 lines (1080 visible lines + 45 service lines in the HDTV NHK standard). Each line has 1920 pixels. Thus, the inClk has a frequency of $25 \times 1125 \times 1920 = 54.10^6$ Hz, hence a period of 18.5 ns. So small an amount of time forbids the use of general purpose execution platform. An ASIC solution is chosen. Coefficients for the dot product are negative power of 2 so that the dot product consists of simple bit-shift followed by additions.

The resources are 2 registers, 1 shift-register, and a tailored ALU. Tab. 3 contains pertinent information about the execution platform. prClk is the clock of the circuit. Duration is expressed in number of cycles of this clock.

| Resource | Type | Service | Duration | Clock |
|---|---|---|---|---|
| pxInBuf | Register | get():Pixel | 1 | prClk |
| slidingWindow | Shift-Register | push(p:Pixel) | 1 | prClk |
| | | get(i:integer): Pixel | 1 | prClk |
| filter | ALU | dotProduct():Pixel | 7 | prClk |
| pxOutBuf | Register | put(p:Pixel) | 1 | prClk |

Table 3: Execution Platform

From Tab. 3 we extract new clock relations gathered in Tab. 4. The duration for the execution of the WindowFiltering activity is $1 + 1 + 7 + 1 = 10$ cycles of clock prClk. So, this clock should be 10 times faster than clock pxInClk (Eq. 5). On the other hand, the sliding window has to be filled-in before it can operate at its full speed. This needs a delay of 5 instants on the HDHoBClk clock (Eq. 6). Now, a fine schedule of actions can be derived from the WindowFiltering activity specification and the known durations of the actions. For instance, the get action on pxInBuf has to be scheduled one instant of the prClk after the sampling of a pixel. This is expressed by Eq. 7. The push action in the sliding window has to be scheduled two instants after the sampling of a pixel (Eq. 8), and so on.

| # | clock name | equation |
|---|---|---|
| 5 | prClk | $\text{prClk} = \text{pxInClk}|^{(1.0^9)^*}$ |
| 6 | HDHoBClk′ | $\text{HDHoBClk}' = \text{HDHoBClk} \gg 5$ |
| 7 | pxInBufGetClk | $\text{pxInBufGetClk} = (\text{prClk} \gg 1)|_{(1.0^9)^*}$ |
| 8 | slidingWindowPushClk | $\text{slidingWindowPushClk} = (\text{prClk} \gg 2)|_{(1.0^9)^*}$ |

Table 4: Clocks after step 3

## 5  Conclusions and Future Directions

We have provided a modeling framework to represent time schedule informations in Real-Time Embedded applications. The approach is model-based and relies fully on existing UML modeling paradigms. Explicit schedules are represented as logical clocks and clock relations. In the case of predictible periodic behaviors they can be computed upon, but the modeling scope is not limited to this case; in the larger spectrum they can be kept as relations. We focused on discrete clocks, but rational clocks could be defined as well.

The approach relies on time refinement, by which a set of loosely related clocks can be inter-scheduled to one that encompasses them all. Execution platform models provide mandatory duration values for computations, and allocation mappings allow to transport these constraints to the application demands. Defining ad-hoc platform *services* allows to bring it closer to the application.

Further work should be conducted to demonstrate the use of explicit logical clock scheduling in broader scope. Real case studies on execution platform models would also lead to a better understanding of the approach. We are currently implementing these concepts into UML profile modelers, and presenting them at the OMG in the framework of the MARTE profile proposal (*Modeling and Analysis of Real-Time Embedded systems*).

# References

[1] F. Baccelli, G. Cohen, G.J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity*. Wiley, 1992.

[2] Benveniste, Caspi, Edwards, Hallbwachs, Le Guernic, and de Simone. The synchronous languages twelve years later. *Proceedings of the IEEE*, 91(1), 2003.

[3] J.T. Buck, S. Ha, E.A. Lee, and D.G. 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] J. Carlier Ph. Chrétienne. *Problème d'ordonnancement: modélisation, complexité, algorithmes*. Masson, Paris, 1988.

[5] Albert Cohen, Marc Duranton, Christine Eisenbeis, Claire Pagetti, Florence Plateau, and Marc Pouzet. N-synchronous kahn networks. In *POPL 2006 Proceedings*, January 2006.

[6] Albert Cohen, Daniela Genius, Abdesselem Kortebi, Zbigniew Chamski, Marc Duranton, and Paul Feautrier. Multi-periodic process networks: Prototyping and verifying stream-processing systems. In *Euro-Par '02: Proceedings of the 8th International Euro-Par Conference on Parallel Processing*, pages 137–146, London, UK, 2002. Springer-Verlag.

[7] Vincent H. Van Dongen, Guang R. Gao, and Qi Ning. A polynomial time method for optimal software pipelining. In *Proceedings of the Second Joint International Conference on Vector and Parallel Processing: Parallel Processing*, volume 634 of *LNCS*, pages p613–624, 1992.

[8] Bruce Powell Douglass. *Doing Hard Time: Developing Real Time Systems with UML, Objects, Frameworks, and Patterns*. Addison-Wesley, 1999.

[9] Peter H. Feiler, Bruce Lewis, and Steve Vestal. The SAE avionics architecture description language (AADL) standard : A basis for model-based architecture-driven embedded systems engineering. In *RTAS 2003 Workshop on Model-Driven Embedded Systems*, May 2003.

[10] S. Gérard, F. Terrier, and Y. Tanguy. Using the model paradigm for real-time systems development: Accord/uml. In *OOIS'02-MDSD*, volume 2426 of *LNCS*, Montpellier (F), 2002. Springer-Verlag.

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

[12] C. Lavarenne, O. Seghrouchni, Y. Sorel, and M. Sorine. The SynDEx software environment for real-time distributed systems, design and implementation. In *Proceedings of European Control Conference, ECC'91*, Grenoble, France, July 1991.

[13] 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.

[14] 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).

[15] François R.Boyer, El Mostapha Aboulhamid, Yvon Savaria, and Michel Boyer. Optimal design of synchronous circuits using software pipelining. In *Proceedings of the ICCD'98*, 1998.

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

[17] Bran Selic and James Rumbaugh. Using UML for Modeling Complex Real Time Systems. Technical report, ObjecTime, 1998. http://www.objectime.com.

[18] Irina Smarandache. Transformations affines d'horloges: application au codesign de systèmes temps-réel en utilisant les langages signal et alpha. In *Thèse de l'Université de Rennes*, 1998.