Clock Constraints in UML MARTE CCSL

Charles André, Frédéric Mallet

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Clock Constraints in UML/MARTE CCSL

Charles André — Frédéric Mallet

N° ????

May 2008

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--????--FR+ENG

Clock Constraints in UML/MARTE CCSL

Charles André*, Frédéric Mallet *

Thème COM — Systèmes communicants
Projet Aoste

Rapport de recherche n° ???? — May 2008 — 22 pages

Abstract: The UML Profile for Modeling and Analysis of Real-Time and Embedded (RTE) systems has recently been adopted by the OMG. Its Time Model extends the informal and simplistic Simple Time package proposed by UML2 and offers a broad range of capabilities required to model RTE systems including both discrete/dense and chronometric/logical time. MARTE OMG specification introduces a Time Structure inspired from Time models of the concurrency theory and proposes a new clock constraint specification language (CCSL) to specify, within the context of UML, usual logical and chronometric time constraints.

This paper presents, for the first time, the formal semantics of some representative CCSL clock constraints concerning logical discrete time. Considering the Time Structure as a concurrent system, we propose a dynamic interpretation to build acceptable solutions that fully respect the constraints. An unusual example about processing Easter days illustrates the use of CCSL and the construction of solutions.

Key-words: Time Model, UML/MARTE, logical time

* Université de Nice Sophia Antipolis

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Constraintes d'Horloges de UML/MARTE CCSL

Résumé : L'OMG a récemment adopté le profil UML MARTE pour la Modélisation et l'Analyse de systèmes Temps Réel et Embarqués (TRE). Son modèle de temps étend le paquetage *Simple Time* de UML2 qui avait l'inconvénient d'être à la fois informel et simpliste. Le modèle de MARTE propose un large spectre de nouvelles possibilités nécessaires pour la modélisation de systèmes temps réel et notamment il prend en compte le temps discret et dense, le temps logique et chronométrique. La spécification OMG de MARTE est très volumineuse et n'offrait pas le cadre idéal pour une spécification formelle. Elle définit une structure de temps inspirée de modèles issus de la théorie de la concurrence. Elle propose également un langage de spécification de contraintes d'horloges (CCSL) pour spécifier les contraintes de temps usuelles du domaine, contraintes qui utilisent à la fois le temps logique et chronométrique. La spécification montre aussi comment intégrer ces contraintes dans un modèle UML existant.

Ce papier présente pour la première fois la sémantique formelle d'une sélection de contraintes d'horloge offertes par CCSL. Cette sélection ne concerne que du temps discret et logique, mais est toutefois représentative de la diversité des contraintes proposées. Le papier considère la structure de temps de MARTE comme un système concurrent et propose une interprétation dynamique de ce système pour calculer des solutions acceptables, c'est-à-dire qui respectent toutes les contraintes. Un exemple original, qui calcule la date de Pâques, est utilisé pour illustrer la puissance du langage CCSL et la construction dynamique de solutions acceptables. illustrates the use of CCSL and the construction of solutions.

Mots-clés : Modèle de Temps, UML/MARTE, temps logique

1 Introduction

The Unified Modeling Language (UML) [1] aims at being a unified and general-purpose modeling language. Its semantics is purposely loose to cover a large domain and introduces so-called *semantic variation points* that provide for extensions to refine (or even define) a semantics when required for a specific domain. These extensions are to be defined in the context of a UML Profile. In the domain of real-time and embedded (RTE) systems, the Object Management Group (OMG) has recently adopted the UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [2], which is currently in the finalization phase. In its foundations, MARTE defines a broadly expressive Time Model to provide for a generic timed interpretation of UML models. The idea is to precisely define a semantics within the Profile rather than allowing tools for giving their own, possibly incompatible with other tools of the same domain.

MARTE Time Structure is heavily inspired by the Tagged Signal Model [3], which intends to define a common framework for comparing several Models of Computation and Communication in the RTE domain, and from various works around synchronous languages [4] and more generally polychronous/multiclock languages well-suited to specify Globally Asynchronous and Locally Synchronous (GALS) systems. The concrete syntax of our language, called *Clock Constraint Specification Language* (CCSL), is part of MARTE Profile but is not normative and not based on any existing language to let tool vendors choose their own technology. Our goal has been to use explicit keywords that denote usual concepts of the domain (periodic, sporadic, sampling...).

This paper presents a precise semantics of some frequently used CCSL statements (constraint expressions) but is not a comprehensive description of the language, which has already been *informally* introduced in a previous work [5]. Section 3 defines MARTE Time Structure. Then, the specification of an academic example, simple and yet representative of the wide variety of proposed clock constraints, is given in Section 4. Section 5 continues on the semantics of MARTE clock constraints. Section 6 shows how clock constraints can be composed to obtain one (of the possibly many) behavior that satisfies the clock constraints, or reject the specification when the system is inconsistent (incompatible clock constraints). This dynamic clock calculus lays the path to a UML simulator for real-time systems. Before concluding, we briefly compare CCSL with other models related to the RTE domain.

2 Related Work

In UML, Time is seldom part of the behavioral modeling, which is essentially untimed. By default, events are handled in their arrival order. In UML2, the subpackage Simple Time introduces metaclasses to represent time and duration. This very simple model of Time explicitely calls for extensions (by an appropriate Profile) to provide both a more sophisticated model and a precise semantics. Several models of Time and Concurrency (outside the scope of UML) have been defined and have inspired our work. We briefly describe them in this section.

There has been several attempts to give a formal semantics to UML constructs. Lots of the existing work focus on behavioral models (activities and interactions) [6, 7, 8] and attempt to give a semantics to UML metaclasses in a transformational way. MARTE Time model includes both structural and behavioral aspects (not the behavior only) and does not focus on one specific diagram. However, we do not consider the whole behavior nor the whole metamodel. Our intent is rather to give a global consistency to timing aspects of a UML model.

Our time model is based on partial ordering of instants. This is close to Petri's work on concurrency theory [9]. This model restricts coincidence to single points in space-time. In our model, the coincidence relationship "melts" *a priori* independent points (instants) to reflect design choices and thus is a foundational relation for us. General Net theory has also influenced our work. The precedence-based relations and especially the clock constraint *synchronization* are inspired from the synchronic distance concept [10].

Petri nets have well-established mathematical foundations and offer rich analysis capabilities. Petri nets support true concurrency and could be used to specify our clock relations (at least some of them). However, it is very difficult to express simultaneity. It is not possible to force two transitions to fire "at the same time". An extension, Time Petri Net [11] adds time intervals to transitions, thus providing a support for simultaneity. Even with that extension, the specification of CCSL constraints is far from straight forward.

Our logical time model is also akin to synchronous language time model. Coincidence-based CCSL clock constraints are easily expressed with the language Signal [12]. Signal is a relational language that supports multiclock (polychronous) specifications. A signal is a sequence of values of the same type, which are present at some instants. The set of instants where a signal is present is the clock of the signal. There are two kinds of operators. *Monochronous* operators act only on synchronous signals, *i.e.,* signals that are always present at the same instants. These operators mainly operate on values rather than clocks. *Polychronous* operators act on signals with any clock and their result may have another clock. In MARTE, the Time Structure only refers to the instants and a labeling function can be given to associate values with instants. In this paper, we have focused on the Time Structure ignoring the labeling function (the values). CCSL clocks then compare to pure signals (type event in Signal). Considered CCSL constraints compare to Signal polychronous operators.

3 Logical Time

MARTE Time model deals with both discrete and dense time. In MARTE, a *clock* gives access to a *time structure*. A clock can be either *chronometric* or *logical*. The former is related to "physical time" while the latter is not. This paper focuses on discrete-time logical clocks—referred to as logical clocks in this paper—and time is qualified as logical. Logical time is the time model in use in synchronous languages [4]. Logical clocks as originally defined by Lamport [13] are a special case where the labeling function is an increasing monotonic function.

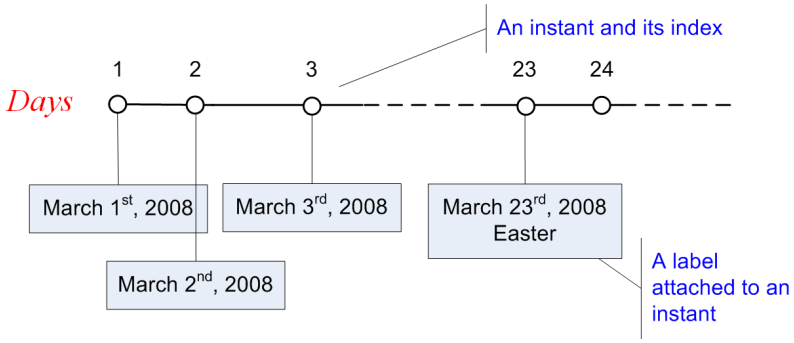

Figure 1: An example of discrete time Clock

3.1 Clock

A *Clock* is a 5-tuple $\langle \mathcal{I}, \prec, \mathcal{D}, \lambda, u \rangle$ where $\mathcal{I}$ is a set of instants, $\prec$ is a quasi-order relation on $\mathcal{I}$, named *strict precedence*, $\mathcal{D}$ is a set of labels, $\lambda : \mathcal{I} \rightarrow \mathcal{D}$ is a labeling function, u is a symbol, standing for a *unit*. For logical clocks, u is often called tick, it can be processorCycle (or a busCycle) as well or any other logical activation of a behavior. The *ordered set* $\langle \mathcal{I}, \prec \rangle$ is the temporal structure associated with the clock. $\prec$ is a total, irreflexive, and transitive binary relation on $\mathcal{I}$.

A *discrete-time clock* is a clock with a discrete set of instants $\mathcal{I}$. Since $\mathcal{I}$ is discrete, it can be indexed by natural numbers in a fashion that respects the ordering on $\mathcal{I}$: let $\mathbb{N}^\star = \mathbb{N} \setminus \{0\}$, $\mathrm{idx} : \mathcal{I} \rightarrow \mathbb{N}^\star$, $\forall i \in \mathcal{I}$, $\mathrm{idx}(i) = k$ if and only if i is the k^{th} instant in $\mathcal{I}$. In MARTE, a logical clock can be associated with any Event: this clock "ticks" at each event occurrence.

For any discrete time clock $c = \langle \mathcal{I}_c, \prec_c, \mathcal{D}_c, \lambda_c, u_c \rangle$, $c[k]$ denotes the k^{th} instant in $\mathcal{I}_c$ (*i.e.*, $k = \mathrm{idx}_c\,(c[k])$). For any instant $i \in \mathcal{I}_c$ of a discrete time clock, ${}^\circ i$ is the unique immediate predecessor of i in $\mathcal{I}_c$. For simplicity, we assume a virtual instant the index of which is 0, and which is the (virtual) immediate predecessor of the first instant. i° is the unique immediate successor of i in $\mathcal{I}_c$, if any.

Example of Clock :
The Clock *Days* (Figure 1) is a discrete-time clock. It is a logical representation of calendar days starting on the 1st of March 2008. Note that instants are strictly ordered, but the interval between two consecutive instants is meaningless in this logical model. In this example, the labels are dates, with optional additional text (*e.g.*, "Easter" in the label attached to the 23rd instant).

Often $\mathcal{D}$ is a data type with associated operations (*e.g.*, a scalar domain like integer or real numbers). In this paper we are not concerned with the labels.

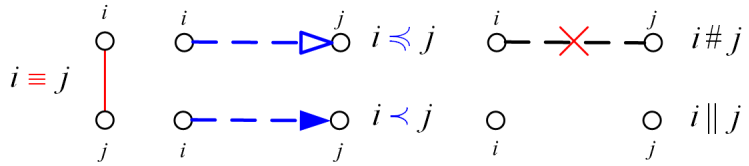

Figure 2: Graphical representation of instant relations

3.2 Time structure

A *Time Structure* is a pair $\langle C, \preccurlyeq \rangle$ where C is a set of clocks, $\preccurlyeq$ is a binary relation on $\bigcup_{c \in C} \mathcal{I}_c$, named *precedence*. $\preccurlyeq$ is reflexive and transitive. From $\preccurlyeq$ we derive four new relations: *Coincidence* ($\equiv \triangleq \preccurlyeq \cap \succcurlyeq$), *Strict precedence* ($\prec \triangleq \preccurlyeq \setminus \equiv$), *Independence* ($\| \triangleq \overline{\preccurlyeq \cup \succcurlyeq}$), and *Exclusion* ($\# \triangleq \prec \cup \succ$). The graphical representation of instant relations is given in Figure 2.

Let $I = \left(\bigcup_{c \in C} \mathcal{I}_c\right) / \equiv$. The Time Structure $T = \langle C, \preccurlyeq \rangle$ is *well-structured* if $\langle I, \preccurlyeq \rangle$ is a partially ordered set (POset).

4 Example: Easter date

As a running example, we choose to specify the clock *EasterDays*, the instants of which stand for Easter days.

4.1 Specification

"The canonical rule is that Easter Day is the first Sunday after the 14$^{\text{th}}$ day of the lunar month that falls on or after March 21$^{\text{st}}$(nominally the day of the vernal equinox)"

We reformulate this specification, emphasizing and numbering the various requirements: Easter Day is the *first Sunday* ① *after* ② the 14$^{\text{th}}$ *day of the lunar month* ③ that falls *on or after* ④ the *vernal equinox* ⑤.

Some requirements refer to events (Sunday ①, vernal equinox ⑤); others envolve "temporal" operators (after ②, on or after ④). Some event names are domain-specific and need additional explanations: the *nominal full moon* ③, which is defined as the 14$^{\text{th}}$ day of the lunar month, is an "ecclesiastic" full moon, distinct from the astronomic one. It occurs exactly 14 days after the new moon. Similarly, the "ecclesiastic" vernal equinox always occurs on March 21$^{\text{st}}$, while the date of the actual spring equinox is the 21$^{\text{st}}$ or the 22$^{\text{nd}}$.

4.2 Modeling in terms of logical clocks

A logical clock is associated with each event Day, Sunday, VernalEquinoxDay, NewMoonDay, FullMoonDay. Figure 3 represents these clocks respectively named *Days*, *Sundays*, *VEquinoxDays*, *NewMoonDays*, *FullMoonDays*, and the constraints between instants imposed

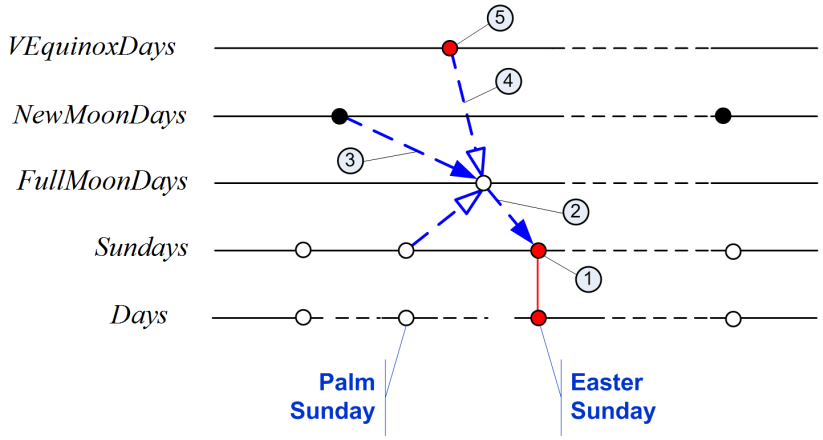

Figure 3: Easter temporal constraints

by the canonical rule for Easter. The numbers on the picture refer to the requirements abovementioned.

Of course, the relations on instants represented in Figure 3 hold for all Easter days. So, instead of expressing many times the same constraints on an instant-pair base, it is more convenient to express constraints directly between clocks. This is what *clock constraints* are all about. Clock contraints are described in Section 5. The questions are now:

1. How to express the clock constraints (syntax)?
2. What is the semantics of the clock constraints?
3. How to compute/analyze/simulate the behavior of a system that satisfies the set of clock constraints?

Question 1 is addressed by dedicated languages. In MARTE we have proposed a language to specify clock constraints: the Clock Constraint Specification Language (CCSL). We have also developed a GUI that allows capture of clock constraints. To answer question 2 we adopt a mathematical characterization of the clock constraints. As for the behavior, we propose in this paper a state-based model and rules for transition firings. All these points are discussed in the following sections.

5 Expression of clock constraints

The relationships introduced in Section 3.2 are binary relations relating pairs of instants. Specifying a full time structure using only these elementary relationships is not realistic. Moreover clocks are usually infinite, therefore forbidding an enumerative specification of

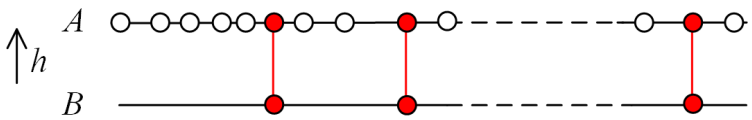

Figure 4: Coincidence-base clock constraint

the instant constraints. Hence the idea to introduce *clock constraints*. Instead of defining individual instant pairings, a clock constraint specifies generic associations between instants of the constrained clocks.

In this section we define the more general clock constraints and we introduce some usual constraints, derived from the basic ones. The clock constraints can be classified in three main categories: 1) coincidence-based constraints, 2) precedence-based constraints, and 3) mixed constraints.

5.1 Coincidence-based clock constraint

Coincidence-based clock constraints are very classical in synchronous languages and can then be very easily specified with such languages.

5.1.1 Sub-Clocking

is the basic coincidence-based clock constraint relationship.
B isSubClockOf A, where A and B are clocks, B being a discrete-time clock ($\mathcal{I}_B$ is countable). Intuitively, this means that each instant in B is coincident with one instant in A, and this without introducing causality loop (Figure 4). More formally:

$\exists h : \mathcal{I}_B \rightarrow \mathcal{I}_A$ such that

(1) h is injective
(2) h is order preserving: $(\forall i, j \in \mathcal{I}_B)(i \prec_B j) \implies (h(i) \prec_A h(j))$
(3) an instant and its image are coincident: $(\forall i \in \mathcal{I}_B) i \equiv h(i)$

In what follows, this constraint is denoted as $B \boxed{\subseteq} A$ or equivalently $A \boxed{\supseteq} B$ that reads "A is a super-clock of B". We also say that A isFinerThan B and B isCoarserThan A.

5.1.2 Derived coincidence-based clock constraints

specify h in different ways. Four instances are given hereafter:

Equality $A = B$ means that the two clocks are "synchronous": h is a bijection and the instants of the two clocks are pair-wise coincident.
The next three constraints create a new clock from an existing one.

Restriction A restrictedTo P where A is a clock and P is a predicate, creates a new clock (sub-clock of A), say B, such that P is a predicate on $\mathcal{I}_A \times \mathcal{I}_B$, and

$$(\forall i \in \mathcal{I}_B, \forall j \in \mathcal{I}_A) h(i) = j \Longleftrightarrow P(j,i) = \mathsf{true}$$

Discretization A discretizedBy r, where A is a dense-time clock and r is a real number, creates a discrete-time clock (sub-clock of A), say B. The definition of the predicate P takes account of the labeling function $\lambda_A : \mathcal{I}_A \to \mathbb{R}$:

$$(\forall i \in \mathcal{I}_B)(\forall j \in \mathcal{I}_A)(\exists d \in \mathbb{R}) P(j,i) = \mathsf{true} \Longleftrightarrow \lambda_A(j) = d + (\mathrm{idx}_B(i) - 1) * r$$

Filtering A filteredBy w, where A is a discrete-time clock and w is a binary word, creates a discrete-time clock (sub-clock of A), say B, such that

$$(\forall i \in \mathcal{I}_B, \forall j \in \mathcal{I}_A) h(i) = j \Longleftrightarrow \mathrm{idx}_A(j) = w \uparrow \mathrm{idx}_B(i).$$

$w \uparrow k$ is the index of the k^{th} 1 in w. The use of infinite k-periodic binary words in this kind of context has previously been made in N-Synchronous Kahn networks [14]. More about binary words can be found in our previous work [5]. This constraint is frequently used and denoted as $A \blacktriangledown w$. It allows the specifier to select a subset of instants, and then to enforce other contraints on these instants. Note that, stating $B = A \blacktriangledown w$ constrains both A and B and is equivalent to $A \blacktriangledown w = B$.

5.2 Precedence-based clock constraint

Precedence-based clock constraints are easy to specify with concurrent models like Petri nets but are not usual in synchronous languages.

5.2.1 Precedence

is the basic precedence-based clock constraint. It is twofold: A precedes B, and A strictly precedes B, where A and B are clocks, B being a discrete-time clock. Intuitively, this means that each instant in B (immediately) follows one instant in A (Figure 5). More formally:

For A precedes B, $\exists h : \mathcal{I}_B \to \mathcal{I}_A$ such that

(1) h is injective
(2) h is order preserving: $(\forall i, j \in \mathcal{I}_B)(i \prec_B j) \implies (h(i) \prec_A h(j))$
(3) an instant and its image are ordered: $(\forall i \in \mathcal{I}_B) h(i) \preccurlyeq i$
(3′) if stricly precedes : $(\forall i \in \mathcal{I}_B) h(i) \prec i$

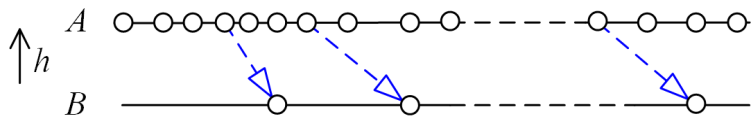

Figure 5: Precedence-base clock constraint

5.2.2 Derived precedence-based clock constraints

are illustrated by three often used clock constraints.

Speed Let A and B be two discrete-time clocks. A isFasterThan B (denoted by $A \boxed{\preccurlyeq} B$) if $(\forall i \in \mathcal{I}_B)(k = \mathrm{idx}_B(i)) \implies A[k] \preccurlyeq B[k]$.
There also exists a strict form of this constraint: A isStrictlyFasterThan B (denoted by $A \boxed{\prec} B$) if $(\forall i \in \mathcal{I}_B)(k = \mathrm{idx}_B(i)) \implies A[k] \prec B[k]$.
Of course, B isSlowerThan A (B isStrictlySlowerThan A, resp.) iff A isFasterThan B (A isStrictlyFasterThan B, resp.).

Alternation A alternatesWith B, where A and B are discrete-time clocks, is defined by: Let $A' = A \blacktriangledown 0.1^\omega, (A \boxed{\preccurlyeq} B) \wedge (B \boxed{\prec} A')$.
The following expression is equivalent and uses instant relations instead of clock relations, $(\forall i \in \mathcal{I}_A)(k = \mathrm{idx}_A(i)) \implies (A[k] \preccurlyeq B[k] \prec A[k+1])$.

Synchronization A sync$[\alpha, \beta]$ B, where A and B are discrete-time clocks, α and β two integers greater than 0. This constraint is more complex. Its characterization in terms of the strictly precedes constraints needs auxiliary clocks:

$$A_f = A \blacktriangledown \left(0^{\alpha-1}.1\right)^\omega \qquad B_f = B \blacktriangledown \left(0^{\beta-1}.1\right)^\omega \qquad A_f \boxed{\prec} B_s$$
$$A_s = A \blacktriangledown 0^\alpha.\left(1.0^{\alpha-1}\right)^\omega \qquad B_s = B \blacktriangledown 0^\beta.\left(1.0^{\beta-1}\right)^\omega \qquad B_f \boxed{\prec} A_s$$

This constraint can also be expressed using instant relations:

$$(\forall k \in \mathbb{N}^\star)(A[k\alpha] \prec B[k\beta+1]) \wedge (B[k\beta] \prec A[k\alpha+1]).$$

5.3 Mixed constraints

5.3.1 Sampling

$C = A$ sampledOn B, where A, B, and C are discrete-time clocks, defines C as a sub-clock of B that ticks only after a tick of A (Figure 6).

$$(\forall c \in \mathcal{I}_C)(\exists b \in \mathcal{I}_B)(\exists a \in \mathcal{I}_A)(c \equiv b) \wedge (a \preccurlyeq b) \wedge (^\circ b \prec a)$$

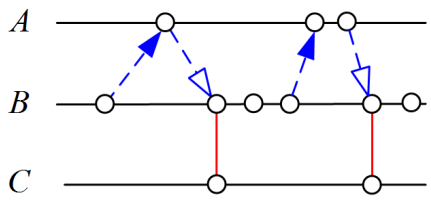

Figure 6: $C = A$ sampledOn B

The strict form $C = A$ strictly sampledOn B has the following characterization:

$$(\forall c \in \mathcal{I}_C)(\exists b \in \mathcal{I}_B)(\exists a \in \mathcal{I}_A)(c \equiv b) \wedge (a \prec b) \wedge (^\circ b \preccurlyeq a)$$

6 Time Structure as concurrent system

A Time Structure can be represented—like in Figures 4 to 6—by a graph showing instants and temporal relations between them. Instead of this explicit representation, which is generally infinite, we consider finite graphs made of clocks and clock constraints. This graphical representation is of course equivalent to the textual form expressed in CCSL. Remind that the semantics of a well-structured Time Structure is a POset that we do not intend to build.

6.1 Clock Dependency Graph (CDG)

A *CDG* is a graph whose vertices are clocks, and whose edges represent clock constraints. There are two kinds of edges: solid-line edges, standing for coincidence-based constraints, and dashed-line edges, standing for precedence-based constraints. Edges can be annotated with information about the constraint itself (*e.g.*, filter, delay ...). When the relation `speed` holds between two clocks, the edge is directed from the faster to the slower. See Figure 8 for an instance of CDG related to Easter.

A CDG is not a simple graphical alternative to a CCSL specification. It offers several advantages. It is a synoptic view of the clock constraints suitable for a UML static representation of the Time Structure. On this graph some *causality cycles* (inconsistency in clock constraints leading to incorrect Time Structure) can easily be detected. CDGs are also useful in modular specifications of Time Structure (see example in Section 7.2).

6.2 Behavior of a Time Structure

A Time Structure can be viewed as a *concurrent system*, where agents are the clocks and the state of the system is given by the *current instant* of each clock. Here, agent actions reduce to a single one: *clock ticking*. A clock ticks when its current instant moves to its next instant. A change-of-state in the system results from a set of "concurrent" clock tickings, called a

step. A step must respect all the clock constraints. Of course, the adjective "concurrent" has to be revisited to capture our notion of coincidence.

In what follows, we propose a way to determine a possible *run* of the system, a run being a sequence of steps. This solution relies on graph redrawing, which is in fact a form of structural operational semantics.

6.2.1 Dynamic Constraint Graph

(DCG) This graph represents the constraints between clocks at a step of a run. The set of nodes of this graph is static, while its set of arcs is dynamic. More precisely: a *DCG* is a labeled directed graph $\langle \mathcal{C}, \mathcal{A} \rangle$ where $\mathcal{C}$ is a set of clocks (nodes of the graph), and $\mathcal{A} \subseteq \mathcal{C} \times \mathcal{C} \times \mathbb{N} \times DepKind$ a set of labeled arcs, where $DepKind = \{\texttt{after}, \texttt{synchro}\}$ (dependency kind).

Each arc a connects a *constrained* clock ($a.src$) to a *constraining* clock ($a.tgt$). A natural number $a.ei \in \mathbb{N}$ (expected instant) is associated with the arc. An arc has also an attribute *a.d:DepKind*. $a.d$ precises the dependency kind between the source and target clocks (either precedence, drawn as a dashed-line or implication, drawn as a solid-line). Let $\mathcal{A}_A$ ($\mathcal{A}_S$, resp.) be the subset of arcs the dependency kind of which is `after` (`synchro`, resp.). The *exclusion* between clocks is not directly memorized in the DCG, but in an auxiliary Boolean matrix (XM exclusion matrix) instead.

Each node c has an attribute $c.ci \in \mathbb{N}$ standing for the current instant of c.

6.3 Construction of the DCG

Before each new step of a run, a new DCG has to be built. Two auxiliary Boolean matrices (the exclusion matrix—XM—and the implication matrix—IM) are also created. A clock constraint induces constraints between pairs of clocks: the constrained clock (cd) and the constraining clock (cg). For each clock constraint, and for each pair of clocks (cd, cg) involved in the constraint, create a candidate arc $a = \langle cd, cg, ei, d \rangle$ in the DCG. The expected instant ei and the dependency kind d depend on the constraint. Set the entry $XM(cd, cg)$ if the constraint is an exclusion. Set the entries $IM(cd, cg)$ and possibly the entry $IM(cg, cd)$ when required by the constraint. A pair of clocks (cd, cg) may appear in several constraints, so that the constraints have to be *accumulated*. A *resolution operation* $\oplus$, which is commutative and associative, is then applied. This operation is described in Table 1. Of course, trying to introduce a `synchro` arc a when the clocks are exclusive ($XM(a.cd, a.cg) = true$) leads to inconsistency and causes the abortion of the DCG construction. The integration of the candidate arc into the DCG is then defined by the $\uplus : 2^{\mathcal{T}} \times \mathcal{T} \rightarrow 2^{\mathcal{T}}$ operation where $\mathcal{T} = \mathcal{C} \times \mathcal{C} \times \mathbb{N} \times DepKind$. $\uplus$ is such that for any a, $\mathcal{A} \uplus a = \mathsf{if}\ \exists a' \in \mathcal{A}, a'.src = a.src \wedge a'.tgt = a.tgt\ \mathsf{then}\ (\mathcal{A} \setminus \{a'\}) \cup \{a \oplus a'\}\ \mathsf{else}\ \mathcal{A} \cup \{a\}$.

$a'.d \backslash a.d$	after	synchro
after	$\langle a.src, a.tgt, \max\{a.ei, a'.ei\}, \mathsf{after}\rangle$	if $a'.ei < a.ei$ then a else abort
synchro	if $a.ei < a'.ei$ then a' else abort	if $a.ei = a'.ei$ then a' else abort
where $a.src = a'.src$ and $a.tgt = a'.tgt$		

Table 1: Resolution operation (returns $a' \oplus a$).

We illustrate this structural transformation from clock constraints to arcs on only two representative examples, due to place limitation.

Filtering: $B = A$ filteredBy w induces an implication dependency from A to B and sometimes from B to A (when instants of A and B are coincident). More precisely, if $w[A.ci+1] = 1$ then $\mathcal{A}_S \leftarrow \mathcal{A}_S \uplus \langle B, A, A.ci+1, \mathsf{synchro}\rangle$, $\mathcal{A}_S \leftarrow \mathcal{A}_S \uplus \langle A, B, B.ci+1, \mathsf{synchro}\rangle$ else $\mathcal{A}_S \leftarrow \mathcal{A}_S \uplus \langle B, A, w \uparrow (B.ci+1), \mathsf{synchro}\rangle$. Note that if A and B were previously connected by a precedence arc, this operation causes the deletion of this arc in $\mathcal{A}_A$.

Alternation: A strictly alternatesWith B induces mutual precedence dependencies between A and B. $\mathcal{A}_A \leftarrow \mathcal{A}_A \uplus \langle A, B, B.ci+1, \mathsf{after}\rangle$, $\mathcal{A}_A \leftarrow \mathcal{A}_A \uplus \langle B, A, A.ci+1, \mathsf{after}\rangle$.

6.4 Determination of a Step

When the DCG is constructed and the matrices XM and IM are filled, compute the transitive closure of the implication relation (on matrix IM) and add possible new `synchro` arcs.

6.4.1 Enabled clocks

Build the set of clocks satisfying the `after` constraints:

$$EnabledSet = \{c \in \mathcal{C} | (\forall a \in \mathcal{A}_A) a.src = c \implies a.tgt.ci \geq a.ei\}.$$

6.4.2 Fireable clocks

Now check the `synchro` constraints and determine the set of fireable clocks, which is a subset of $EnabledSet$, such that:

$$FireableSet = \{c \in EnabledSet | (\forall a \in \mathcal{A}_S) \\ a.src = c \implies (a.tgt \in EnabledSet) \wedge (a.tgt.ci = a.ei - 1)\}$$

6.4.3 Step

Select a subset of fireable clocks to compose the step. The selection must respect the following rules:

1. (compulsory) Clocks whose instants coincide must either fire all or none;

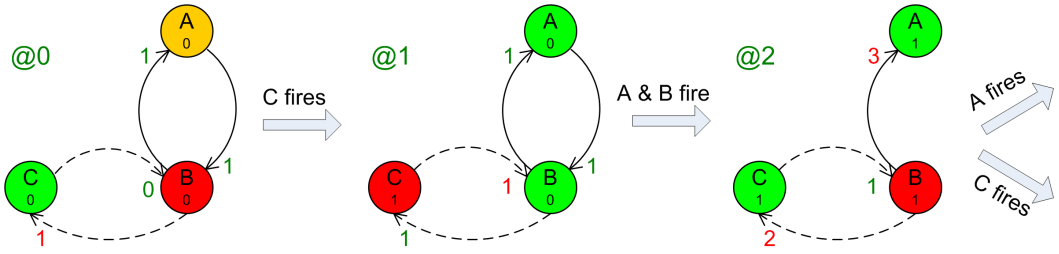

Figure 7: DCGs at the first three steps

2. (compulsory) A step cannot fire exclusive clocks;
3. (strategy dependent) Independent fireable clocks may fire in a step.

The possible (simulation) strategies offered by our tool are:
• *minimal step:* fire only a coincident set (such a set can be reduced to a singleton) among possible fireable clocks (random choice).
• *maximal step:* fire a maximal subset of fireable clocks.
• *interactive choice:* fire one or several independently fireable coincident sets.

6.4.4 Example

consider a simple three-clock constraint time structure:

$$B = A \blacktriangledown (1.0)^{\omega}; \quad C \boxed{\sim} B$$

Figure 7 represents DCGs at the first three steps. Disabled clocks are red, enabled clocks amber, and fireable clocks are green. Note that at step 2, clocks A and C can fire independently.

7 Easter model

This section illustrates the use of clock constraints for determining Easter day (specification given in Section 4). The origin of time has arbitrarily been set to Saturday March 1st, 2008. The first new moon is then on March 7th.

7.1 Simplified clock model

A simplified CCSL specification (see below) considers that *NewMoonDays* and *VEquinoxDays* are strictly periodic on *Days* (period of 30 and 365, respectively). We introduce an auxiliary clock *EasterMoonDays* that represents the *nominal full Moon* days. Circled numbers after clock constraints are not part of CCSL code; There are just comments referring to the requirement numbers in Figure 3.

$$
\begin{aligned}
Sundays &= Days \text{ filteredBy } 0.(1.0^{6})^{\omega} && ① \\
VEquinoxDays &= Days \text{ filteredBy } 0^{20}.(1.0^{364})^{\omega} && ⑤ \\
NewMoonDays &= Days \text{ filteredBy } 0^{6}.(1.0^{29})^{\omega} \\
FullMoonDays &= NewMoonDays \text{ delayedFor } 14 \text{ on } Days && ③ \\
EasterMoonDays &= VEquinoxDays \text{ sampledOn } FullMoonDays && ④ \\
EasterDays &= EasterMoonDays \text{ strictly sampledOn } Sundays && ②
\end{aligned}
$$

Figure 8 shows the corresponding clock dependency graph. This time structure gives the correct date for Easter 2008 but fails to predict Easter 2009. A more accurate specification of the new Moon must be given.

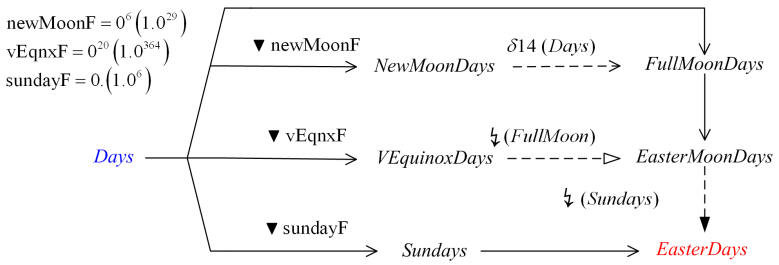

Figure 8: Clock dependency graph for Easter

7.2 Clock module

NewMoonDays and *VEquinoxDays* are not strictly periodic. On the calendar, the duration between two consecutive new moons is either 29 days or 30 days. The exact duration depends on the actual motion of the Moon. The mean *synodic period* of the moon is 29.53 days. The exact date of the new moon is given by the ephemeris. For instance March 7$^{\text{th}}$ new moon occurred at 18:14 *i.e.*, 0.76 day after March 7$^{\text{th}}$ 00:00.

A better approximation for *NewMoonDays* uses local clocks. Let *HDays* be a clock that represents hundredths of day, and *ENM* a clock for the (approximative) Ephemeris New Moon. The relationship between *HDays* and *Days* is a simple periodic filtering: there is a *Days* instant every 100$^{\text{th}}$ *HDays* instant. The relationship between *ENM* and *HDays* is also periodic but more complex: Starting from March 1$^{\text{st}}$ 00:00, the first ephemeris new moon occurs 6.76 days after and then (almost) periodically every 29.53 days. Now, the calendar new moon is obtained by a sampling of the ephemeris new moon on *Days*. Hence:

$$
\left\{
\begin{array}{c}
Days = HDays\blacktriangledown \left(0^{99}.1\right)^{\omega} \\
ENM = HDays\blacktriangledown 0^{675} \left(1.0^{2952}\right)^{\omega} \\
NewMoonDays = ENM \lightning Days
\end{array}
\right\} \backslash HDays, ENM
$$

The $\frac{1}{2}$ operator is the sampling operator. *HDays* and *RNM* being local clocks, are hidden (\ operator). The corresponding CDG is shown in Figure 9.

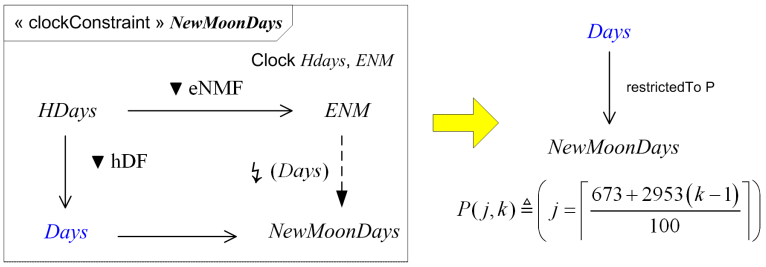

Figure 9: Clock Dependency Graph for NewMoonDays

Since ENM isSporadicOn $Days$ gap $= 29$ (*i.e.,* two consecutive instants of ENM are separated by at least 29 instants of $Days$), we can deduce that $Days$ and $NewMoonDays$ are constrained by $NewMoonDays = Days$ restrictedTo P, where $P(j,k) \triangleq (j = \lceil (673 + 2953\,(k-1))/100 \rceil)$. Thus, four clock constraints reduce to a single clock constraint. With these new constraints, the date of Easter 2009 is correctly predicted. A longer term prediction can be obtained by more accurate predicates, for instance by implementing Oudin's algorithm (1940).

8 Conclusion

This paper introduces a formal semantics for MARTE Clock Constraint Specification Language. A simple (but not trivial) example illustrates the use of CCSL in modeling time constraints and gives the opportunity to explain how to build valid solutions dynamically.

MARTE OMG Specification introduces a conceptual view of the Time Model with an informal (natural language) semantics and the adequate UML syntax to refer to this Time Model in UML user models. To avoid divergent interpretations a formal semantics was needed, especially for real-time critical systems.

The initial intent of MARTE being to cover both design and analysis, many relations (may be too many) have been introduced for convenience. This paper provides a classification of the constraints based on two foundamental relationships (coincidence-based and precedence-based).

MARTE Time Model has also introduced in UML, the notion of logical time, missing in the standard and very useful to digital circuit design. Logical time is a common concept in synchronous languages and Petri nets. The relationship between CCSL and other languages like Petri nets and Signal deserves to be explored so that our models could benefit from existing analysis tools like TINA [15] (for Time Petri Net) and Polychrony(for Signal). This is still on-going work. For now, we have implemented a simulator that captures clock

constraints associated with a UML model, processes them and yields graphical execution traces. This simulator is part of OpenEmbeDD platform (`http://openembedd.org`).

References

[1] Object Management Group: Unified Modeling Language, Superstructure (November 2007) Version 2.1.2 formal/2007-11-02.

[2] Object Management Group: UML Profile for MARTE, beta 1. (August 2007) OMG document number: ptc/07-08-04.

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

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

[5] André, C., Mallet, F., de Simone, R.: Modeling time(s). In Engels, G., Opdyke, B., Schmidt, D.C., Weil, F., eds.: MoDELS. Volume 4735 of Lecture Notes in Computer Science., Springer (2007) 559–573

[6] Li, X., Meng, C., Yu, P., Zhao, J., Zheng, G.: Timing analysis of UML activity diagrams. In Gogolla, M., Kobryn, C., eds.: UML. Volume 2185 of Lecture Notes in Computer Science., Springer (2001) 62–75

[7] Damm, W., Josko, B., Pnueli, A., Votintseva, A.: A discrete-time UML semantics for concurrency and communication in safety-critical applications. Sci. Comput. Program. **55**(1-3) (2005) 81–115

[8] Störrle, H.: Semantics and verification of data flow in UML 2.0 activities. Electr. Notes Theor. Comput. Sci. **127**(4) (2005) 35–52

[9] Petri, C.: Concurrency theory. In Brauer, W., Reisig, W., Rozenberg, G., eds.: Petri Nets: Central Models and their properties. Volume 254 of Lecture Notes in Computer Science. Springer-Verlag (1987) 4–24

[10] Reisig, W.: Petri nets: an introduction. Monograph on Theoretical Computer Science. Springer-Verlag, Berlin (1985)

[11] Merlin, P.: A Study of the Recoverability of Computer Systems. PhD, University of California, Irvine (1974)

[12] Benveniste, A., Guernic, P.L., Jacquemot, C.: Synchronous programming with events and relations: the signal language and its semantics. Sci. Comput. Program. **16**(2) (1991) 103–149

[13] Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7) (1978) 558–565

[14] Cohen, A., Duranton, M., Eisenbeis, C., Pagetti, C., Plateau, F., Pouzet, M.: N-synchronous Kahn networks. In: POPL 2006 Proceedings. (January 2006)

[15] Berthomieu, B., Vernadat, F.: Time petri nets analysis with TINA. In: QEST, IEEE Computer Society (2006) 123–124

A Binary Words

B Macros for this paper

C Instant relations

Syntax	Notation	LaTeX
`predecessor`	$\overset{\circ}{.}$	`\predSuc .`
`successor`	$.\overset{\circ}{}$	`. \predSuc`
`coincidence`	$\equiv$	`\coinc`
`precedence`	$\preccurlyeq$	`\pred`
`strict precedence`	$\prec$	`\spred`

Table 2: Operators for Instants.

Syntax	Notation	LaTeX
`predecessor of` i	$\overset{\circ}{i}$	`\ipred{i}`
`successor of` i	$i\overset{\circ}{}$	`\isucc{i}`
i `precedes` j	$i \preccurlyeq j$	`\iprec{i}{j}`
i `strictly precedes` j	$i \prec j$	`\isprec{i}{j}`
i `coincidesWith` j	$i \equiv j$	`\icoinc{i}{j}`

Table 3: Instant relations.

D Clock relations

Syntax	Notation	LaTeX
B `isSubClockOf` A	$B \boxed{\subseteq} A$	`\csubeqclock{B}{A}`
A `isSuperClockOf` B	$A \boxed{\supseteq} B$	`\csupeqclock{A}{B}`
A `isFasterThan` B	$A \boxed{\preccurlyeq} B$	`\cprec{A}{B}`
A `isStrictlyFasterThan` B	$A \boxed{\prec} B$	`\csprec{A}{B}`
A `alternatesWith` B	$A \boxed{\sim} B$	`\caltern{A}{B}`
A `sync`(α, β) B	$A \boxed{\bowtie} (\alpha, \beta) B$	`\csynchronized{A}{B}{`α`}{`β`}`

Table 4: Clock relations.

E Functional clock relations

Syntax	Notation	LaTeX
A `filteredBy` w	$A \blacktriangledown w$	`\sproj{A}{w}`
A `followedBy` B	$A \bullet B$	`\cconcat{A}{B}`
A `sampledOn` B	$A \lightning B$	`\csampledOn{A}{B}`

Table 5: Clock functional relations.

Notation	LaTeX	Semantics
$w\,[k]$	`\kthbit{w}{k}`	k^{th} bit of w
$w[k..l]$		sub-binary word from k to l
$w[k..]$		binary word starting from k. Possibly infinite
$w1 \bullet w2$	`\wcat{w1}{w2}`	binary word concatenation
$\|w\|$	`\numberOf{w}{}`	length of w
$\|w\|_b$	`\numberOf{w}{b}`	number of bits set to b in w
$w \uparrow k$	`\kthone{w}{b}`	index of the k^{th} 1 in w
$w \downarrow k$	`\oneuptok{w}{b}`	number of 1 upto k in w
$w1 \circ w2$	`\wcomp{w1}{w2}`	binary word composition

Table 6: Binary word notations.

Contents

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399