UML/MARTE CCSL, Signal and Petri nets

Frédéric Mallet, Charles André

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

UML/MARTE CCSL, Signal and Petri nets

Frédéric Mallet — Charles André

N° 6545

May 2008

Thème COM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--6545--FR+ENG

UML/MARTE CCSL, Signal and Petri nets

Frédéric Mallet*, Charles André*

Thème COM — Systèmes communicants
Projet Aoste

Rapport de recherche n° 6545 — May 2008 — 23 pages

Abstract: UML goal of being a general-purpose modeling language discards the possibility to adopt too precise and strict a semantics. Users are to refine or define the semantics in their domain specific profiles. In the UML Profile for Modeling and Analysis of Real-Time and Embedded (MARTE) systems, we have defined a broadly expressive Time Model to provide a generic timed interpretation for UML models. Our Clock Constraint Specification Language (CCSL) supports the specification of systems with multiple clock domains. Starting with a priori independent clocks, we progressively compose them to get a family of possible time evolutions. Our language supports both synchronous and asynchronous compositions, just like the synchronous language Signal, but also allows explicit non determinism. In this paper, we give a formal semantics to a core subset of MARTE CCSL and we give an equivalent interpretation of this kernel in two other very different formal languages, Signal and Time Petri Nets.

Key-words: UML, MARTE, CCSL, logical time, semantics, Signal, Time Petri Net

This work is part of the platform OpenEmbedd: `http://openembedd.org`

* 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

UML/MARTE CCSL, Signal et réseaux de Petri

Résumé : UML visant à être un langage général il se doit d'adopter une sémantique large pour couvrir un ensemble important de domaines. Ainsi, les points de variation sémantique ouverts par la spécification doivent être exploités par des profils pour raffiner voire même définir une sémantique adaptée à un domaine donné. C'est que nous avons fait dans le profil MARTE (UML Profile for Modeling and Analysis of Real-Time and Embedded systems) et plus spécifiquement son modèle de temps. Ce profil de temps vise à donner une interprétation précise mais suffisamment large pour permettre une interprétation temporelle des modèles UML. Ce modèle de temps comprend un langage de spécification de contraintes (CCSL) qui permet de contraindre progressivement des comportements *a priori* indépendants pour isoler une famille de comportements acceptables. La suite des instants d'activation des comportements contraints est appelée *horloge logique*. Ce langage propose des mécanismes de compositions synchrones et asynchrones des horloges logiques et ressemble en cela beaucoup au langage Signal. Il permet aussi d'exprimer du non-déterminisme explicite et est également inspiré de modèles issus de la théorie de la concurrence, comme les réseaux de Petri.

Ce papier présente la sémantique formelle d'un sous-ensemble représentatif des contraintes de MARTE CCSL en se limitant au temps discret. Pour élargir la communauté et pour faire le lien avec les nombreux travaux existants nous donnons une interprétation de ce sous-ensemble de contraintes en Signal et en réseaux de Petri.

Mots-clés : UML, MARTE, CCSL, temps logique, sémantique, Signal, Réseaux de Petri

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…).

A comprehensive *informal* description of CCSL has previously been presented in [5] and a partial *formal declarative* description is available in [6]. Using a declarative mathematical description allows for being language independent. When constraints are not incompatible they should enforce a causal relationship between UML model elements and thus provide a support to build a real-time UML simulator. To implement CCSL and produce acceptable executions, *i.e.,* compatible with all constraints, it may be interesting to transform it into equivalent formalisms that already benefit from analysis tools. This paper compares a representative selection of CCSL constraints with two very different languages: Petri nets [7] and Signal [8]. Additionally, these two languages being general enough, such a description should help a broad community from the concurrency theory and the synchronous languages understand MARTE clock constraints.

Section 2 starts with a general introduction to CCSL and describe general assumptions made on Petri nets and Signal to allow for a comparison. The following sections gives for each of the selected constraints, its rationale, a mathematical definition, its equivalent in Petri net and/or its equivalent in Signal, if ever. Section 7 illustrates the use of the constraints on an unusual example that process Easter days. This section also introduces the graphical representation of CCSL clock constraints. The graphical representation is not part of MARTE yet and will be proposed to the OMG Revision Task Force for being integrated in next MARTE revision.

2 Formalisms under consideration

2.1 MARTE Time Structure

A *Clock* is a 5-tuple $\langle \mathcal{I}, \prec, \mathcal{D}, \lambda, u \rangle$ where $\mathcal{I}$ is a set of instants (possibly infinite), $\prec$ is a quasi-order relation on $\mathcal{I}$, named *strict precedence*, $\mathcal{D}$ is a set of labels, $\lambda : \mathcal{I} \to \mathcal{D}$ is a labeling function, u is a symbol, standing for a *unit*. In this paper, we only consider the clock temporal structure (or *pure clock*), *i.e.,* the *ordered set* $\langle \mathcal{I}, \prec \rangle$ and the values are never mentionned. $\prec$ 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 way that respects the ordering on $\mathcal{I}$: let $\mathbb{N}^\star = \mathbb{N} \setminus \{0\}$, $\mathrm{idx} : \mathcal{I} \to \mathbb{N}^\star$, $\forall i \in \mathcal{I}$, $\mathrm{idx}(i) = k$ if and only if i is the k^{th} instant in $\mathcal{I}$. We restrict the discussion to discrete-time clocks and do not consider dense time at all. All operators presented are presented with a restricted semantics, assuming that clocks are discrete, whereas these operators may have a more general semantics.

For any discrete-time time structure $c = \langle \mathcal{I}_c, \prec_c \rangle$, $c[k]$ denotes the k^{th} instant in $\mathcal{I}_c$ (*i.e.,* $k = \mathrm{idx}_c\,(c[k])$). For any instant $i \in \mathcal{I}_c$, ${}^\circ i$ is the unique immediate predecessor of i in $\mathcal{I}_c$ and i° is the unique immediate successor of i in $\mathcal{I}_c$, if any. To simplify computations, we assume a *virtual* instant $c[0]$, so that $c[0] \equiv {}^\circ c[1]$.

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 instant relations: *Coincidence* ($\equiv \triangleq \preccurlyeq \cap \succcurlyeq$), *Strict precedence* ($\prec \triangleq \preccurlyeq \setminus \equiv$), *Independence* ($\parallel \triangleq \overline{\preccurlyeq \cup \succcurlyeq}$), and *Exclusion* ($\# \triangleq \prec \cup \succ$).

2.2 Signal

In Signal language, 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* (not to be mistaken with CCSL clocks). As in MARTE, the physical amount of time between two instants is not relevant. The Signal language has two kinds of operators. *Monochronous* operators act only on synchronous signals, *i.e.,* signals that are always present at the same instants, signals that have the same clock. *Polychronous* operators act on signals with any clock and their result may have another clock. In this paper, we only consider the time structure of MARTE and relations on instants, we do not use the labeling function. So CCSL clocks are very similar to signals. CCSL pure clocks compare to Signal clocks (or *pure signals*, type event). CCSL clock relations compare to Signal polychronous operators. In this paper, we never discuss equivalent for Signal monochronous operators that would work on labels associated with instants rather than the time structure itself. The purpose of having a Signal equivalent to CCSL constraints is to use the Signal compiler to perform the clock calculus. An invalid specification , *i.e.,* with clock constraint violations, should result in Signal compilation errors. Ideally, the converse should also be true, *i.e.,* all rejected Signal programs should match an invalid CCSL specification. Unfortunately, depending on assumptions made by the Signal compilers, it may reject valid specifications while never accepting an invalid one.

2.3 Time Petri net

MARTE Time model conceptually differs from Petri's work on concurrency theory [7]. Petri's theory restricts coincidence to single points in space-time in accordance with physical laws. In our model, the foundational relationship *coincidence* gathers *a priori* independent points (instants) to reflect design choices, it is a logical point of view.

Petri nets have well-established mathematical foundations and offer rich analysis capabilities. Petri nets support true concurrency and can be used to specify some of our clock relations. However it is not possible to force two separate transitions to fire "at the same time", *i.e.,* to express coincidence. Thus, we use Merlin's extension of Petri nets [9] that associates a time interval (two times a and b, with $0 \leq a \leq b$ and b possibly unbounded) with each transition: *Time Petri nets.* Times a and b, for transition t, are relative to the moment θ at which the transition was last enabled. t must not fire before time $\theta + a$ and must fire before or at time $\theta + b$ unless it is disabled before then by the firing of another transition. Even with this extension, the specification of CCSL constraints is far from straightforward, as this paper should show.

In our representation, each MARTE discrete-time pure clock $c = \langle \mathcal{I}_c, \prec_c \rangle$ is represented as a single transition c_t (called *clock transition*) of a Time Petri net. Instants of a clock are *firings* of the related transition. For a given initial marking and for a given firing sequence, there is an injective mapping $firingTime : CT \times \mathbb{N}^\star \rightarrow \mathbb{N}$, where CT is the set of clock transitions. $firingTime(c_t, i)$ is the time at which, the clock transition c_t fires for the i^{th} time in the firing sequence. We consider a Time Petri net as equivalent to a CCSL clock constraint, iff for all possible firing sequences and all clock transitions (other transitions do not matter), $firingTime$ preserves the order :

$$(\forall c1, c2 \in C)(\forall k1, k2 \in \mathbb{N}^\star)$$
$$((c1[k1] \preccurlyeq c2[k2]) \iff (firingTime(c1_t, k1) \leq firingTime(c2_t, k2)),$$

where $c1_t$ (resp. $c2_t$) is the clock transition associated with clock $c1$ (resp. $c2$). Even though Time Petri nets can handle continuous time, we restrict our comparison to discrete-time clocks and therefore we consider the transition firing time as a natural number ($\in \mathbb{N}$). Here again, having Time Petri nets equivalents enables the use of some Time Petri net-specific analysis tools. An invalid CCSL specification should be equivalent to a Time Petri net with dead clock transitions.

3 Clock relation alternatesWith

The relation alternatesWith represents alternation between two clocks. A $\boxed{\sim}$ B means that each occurrence of A is followed by an occurrence of B before any other occurrence of A. The *weak form* of this relation allows the i^{th} occurrence of B to be simultaneous (coincident) with the i^{th} occurrence of A, whereas the *strict form* requires A and B to be disjoint.

Typically, an asynchronous communication implies an alternation between sending and receiving. Let A be the sender and B the receiver. The data is received after having been

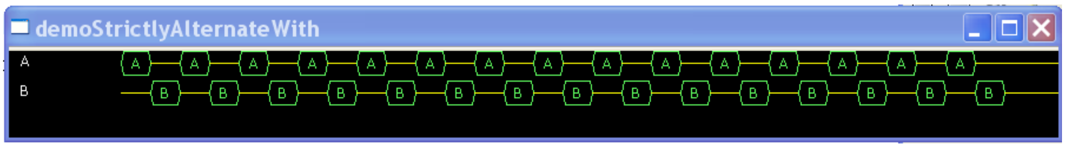


Figure 1: A strictly alternatesWith B

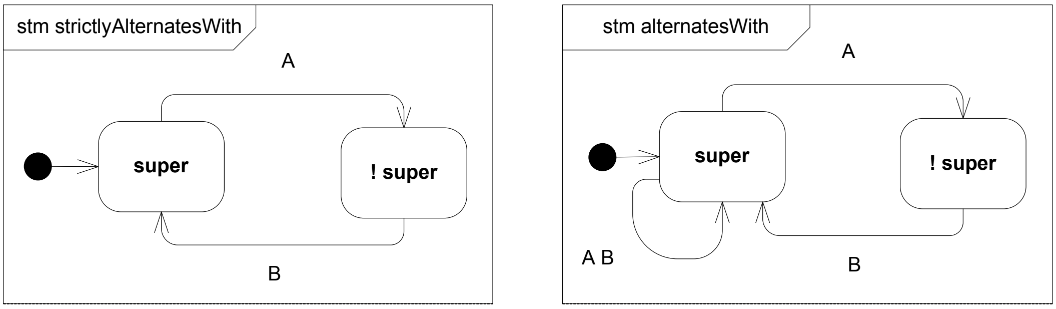


Figure 2: Behavior of clock relations (strictly) alternatesWith

sent. No other communication can start before the previous one completes. The *weak form* allows the sender to receive data synchronously with the emission, but do not force the synchronization. The *strict form* is used to forbid a instantaneous communication.

Figure 1 illustrates the relation strictly alternatesWith and its only possible behavior when ignoring instants where neither A nor B are present (such instants are called *empty* instants). In practice, there can be *arbitrarily* many empty instants between any occurrences of A and B and not necessarily as many between two successive occurrences of A or B. We forbid infinite numbers of empty instants to ensure fairness properties.

Figure 2 shows the equivalent UML StateMachine. This is very similar to the covering step graph of the Time Petri net. Simultaneous events must appear on the same transition. Two different transitions denote independent events. The state machine only shows authorized events. There is no outgoing transition from state *super* with a label B. This does not mean that an event B occurring in this state would be lost but rather that the clock constraint makes it impossible for the event B to occur in this state. If, because of other clock constraints, this condition cannot be enforced, then the specification is rejected. In Signal, the program should be rejected at compilation time. In Time Petri net, a constraint violation should result in dead clock transitions.

3.1 Mathematical definition

Let A and B be two discrete-time clocks.

$$A \text{ strictly alternatesWith } B \iff (\forall k \in \mathbb{N}^\star)(A[k] \prec B[k] \prec A[k+1])$$

$$A \text{ alternatesWith } B \iff (\forall k \in \mathbb{N}^\star)(A[k] \preccurlyeq B[k] \prec A[k+1])$$

The weak form of this clock relation can cause infinitely many different behaviors even if we ignore empty instants. $B[k]$ precedes $A[k+1]$ but can either be coincident with $A[k]$ ($A[k] \equiv B[k]$) or strictly follow it ($A[k] \prec B[k]$). Those are the only two possible situations that matters. When they are disjoint and when no other clocks are involved, the distance between two instants is not relevant.

3.2 Signal equivalent

When two clocks strictly alternate, there exists a super clock, more frequent than both A and B (the relation is *endochronous*). To implement such a relation in Signal, one just need to build explicitely the common super clock. For instance, one can create a local *boolean* signal super that alternatively takes the value *true* and *false*, starting with *true*. A is synchronous with super when super is *true* and B is synchronous with super when super is *false*.

```
process strictlyAlternatesWith = ( ? event A,B )
  (| super := not (super$ init false)
   | A ^= when super
   | B ^= when not super
   |) where boolean super end;
```

The weak form [1] is a bit different because either A and B occurs simultaneously, or A occurs alone and B should occur alone in the future. Note that B cannot occur alone when super is *true*. This is enforced in Signal by the relation $A\ \hat{}= $ when *super*, when A must occur if (and only if) super is *true*, either alone or together with B.

```
process alternatesWith = ( ? event A,B )
  (| nextsuper ^= super ^= A^+ B
   | nextsuper := false when not B default true
   | super := nextsuper $ init true
   | A ^= when super
   | B ^> when not super
   |) where boolean super,nextsuper end;
```

3.3 Time Petri Net equivalent

Figure 3 gives Time Petri nets equivalent to both the strict (lefthand side) and the weak (righthand side) forms of clock relation alternatesWith. They only differ by the time interval

[1] This implementation has been obtained after fruitful discussions with Dumitru Potop-Butucaru

on transition B. The weak form allows the transition B to fire either simultaneously or stricly after the firing of transition A.

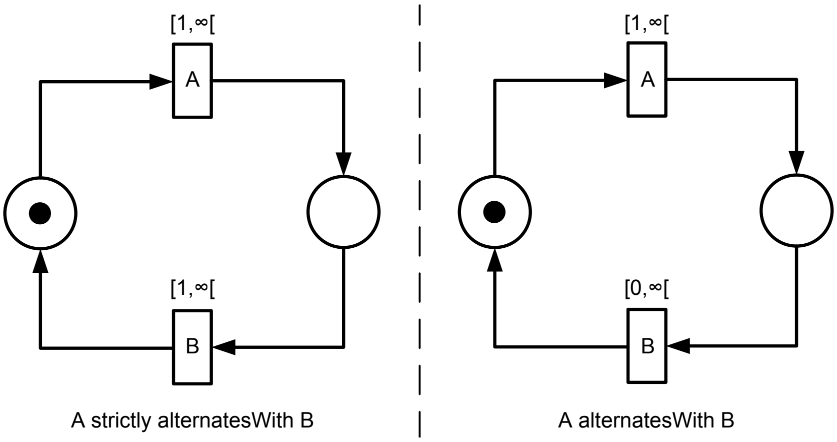


Figure 3: Relation alternatesWith: Time Petri net

To analyze Time Petri nets it is usual to compute the reachability graph [10]. However, even with bounded Time Petri nets the graph can be infinite. Instead, we build the graph of "essential" integer-states as defined by Popova [11]. This graph remains finite while still being sufficient to determine the entire behavior of the net at every time.

Figure 4 shows the graph of essential integer-states for the two forms. The nodes of this graph (states) are pairs $\langle m, J\rangle$, where m is a marking ($m \in P^{\mathbb{N}}$), P is the set of places, $J : T \to \mathbb{N} \cup \{\#\}$ is the timevector, T is the set of transitions. $J(t_i)$ is the time elapsed since transition t_i became most recently enabled or $\#$ if the transition t_i is not enabled.

One "essential" state may represent infinitely many states of the "traditional" reachability graph. State $\langle(1, 0), (i, \#)\rangle$ where $i > 0$ unifies all states where only time increases without any influence on marking or enabling transitions and thus on the relative firing order of clock transitions. Unifying all these states is equivalent to Signal decision of ignoring "empty" events.

4 Clock relation isPeriodicOn

The relation isPeriodicOn builds a periodic clock with respect to a *parent* clock for a given period. Note, that the clocks need not be chronometric. Optionnally, an offset may be specified when the periodic behavior only starts after a given number of occurrences of the parent clock.

A isPeriodicOn B period $=P$ offset $=\delta$ is a strictly periodic behavior preceded by δ occurrences of B alone. A more general clock relation (filteredBy) has been defined in MARTE

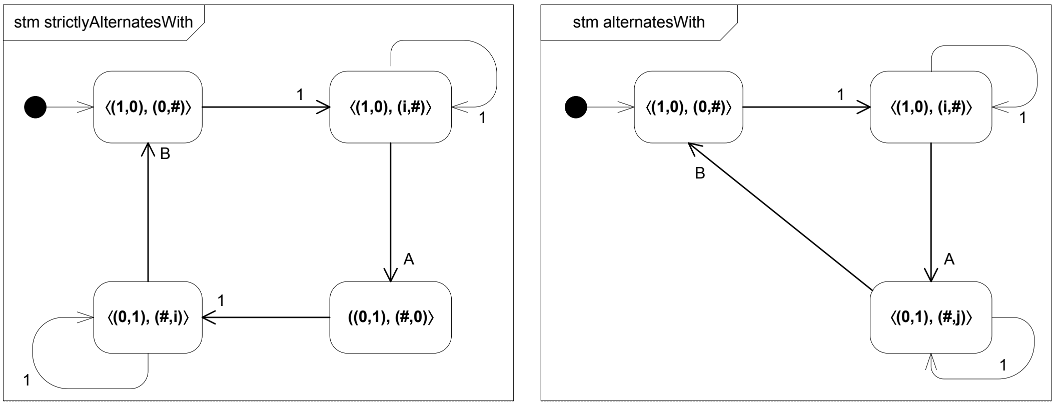


Figure 4: Relation alternatesWith: graph of essential integer-states

to filter a clock and build a subclock (less frequent clock in Signal terminology) by selecting some instants (not necessarily periodically).

Using the clock relation filteredBy (symbolically represented by $\blacktriangledown$) the relation isPeriodicOn can be written $A = B \blacktriangledown 0^{\delta} \bullet (1 \bullet 0^{P-1})$. When $\delta = 0$ and $P = 1$, A and B are synchronous.

Figure 5 gives one possible execution where A isPeriodicOn B period $= 3$ offset $= 5$. Signals A and B are counting their own occurrences. A always first occurs simultaneously with the 6^{th} occurrence of B, whenever it is.

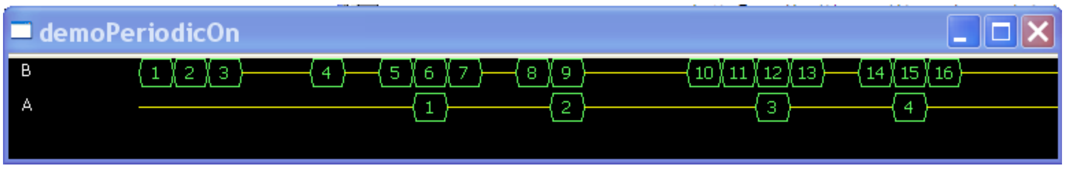


Figure 5: A isPeriodicOn B period=3 offset=5

The weak form does not specify precisely when B should occur. It just specify that B must occur once for each P occurrences of A.

4.1 Mathematical definition

Let A and B be two discrete-time clocks.

$$A \text{ isPeriodicOn } B \text{ period } = P \text{ offset } = \delta \iff (\forall k \in \mathbb{N}^\star)(A[i] \equiv B[(i-1)*P+\delta+1])$$

$$A \text{ isWeaklyPeriodicOn } B \text{ period } = P \text{ offset } = \delta \iff (\forall k \in \mathbb{N}^\star)(B[(i-1)*P+\delta+1] \preccurlyeq A[i] \prec B[i*P+\delta+1])$$

4.2 Signal equivalent

The implementation in Signal of the strict form is straightforward since MARTE filteredBy relation is very close to the operator when. And the periodic case is one simple application. $offset$ and $period$ are parameters, *i.e.,* constant values given at compilation time.

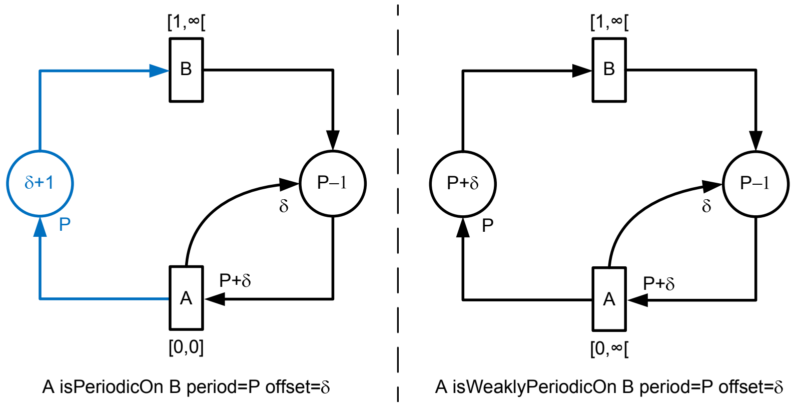
```
process isPeriodicOn = { integer offset,period } ( ? event A, B )
  (| nb ^= B
   | zi := nb$
   | nb := ((zi + 1) when zi/=(period-1)) default 0
   | ^A ^= when zi=0
   |) where integer zi init -offset, nb end;
```

4.3 Time Petri Net equivalent

Figure 6 gives Time Petri net equivalent to both forms of clock relation isPeriodicOn. For the strict form, transition B must fire δ times before anything can happen to transition A. Then every P^{th} firing of transition B, A must fire synchronously because the time interval is [0,0]. Using a time interval [0,0] is very handy to represent instantaneous reactions. However, when several such transitions are enabled at the same time, the use of priorities may become necessary. The left place (in blue) is mandatory. The net has the same behavior with or without this place. However, without this place, the untimed net (skeleton) is unbounded (event though the timed net is bounded). Having unbounded skeletons restricts the kind of analysis that can be performed.

Concerning the weak form (righthand side), without the left place both the timed net and its skeleton would be unbounded because of the unbounded time interval on transition A. This place with its initial marking prevents B from firing more than $2*P-1$ times between two successive firings of A.

Note that as a special case ($P=1$ and $\delta=0$) we retrieve the weak form of clock relation alternatesWith.



A isPeriodicOn B period=P offset=δ

A isWeaklyPeriodicOn B period=P offset=δ

Figure 6: Relation isPeriodicOn: Time Petri net

5 Clock relation sampledOn

The relation sampledOn represents sampling, it can be used to model time-triggered communications or for synchronizing asynchronous inputs. $A = B \lightning C$ defines a subclock of C (less frequent than C in Signal terminology) that occurs only after an occurrence of B. The *strict form* of sampledOn does not instantaneously sample an occurrence of B when it is synchronous with an occurrence of C. In that case, the sampling is postponed.

Figure 7 shows one possible scenario involving the clock relation sampledOn with both forms weak and strict. Signal B counts its occurrences and signal A contains the value actually sampled from B.

With both forms the first sample has the value 1. However, with the weak form the first sample occurs on the first occurrence of C whereas it occurs on the second occurrence of C with the strict form. The second sample has the value 3 and the input 2 has been lost in both cases. The third sample occurs at the same time, whatever the form, but does not carry the same value in both cases.

5.1 Mathematical definition

Let A,B and C be three discrete-time clocks.

$$A = B \text{ strictly sampledOn } C \iff$$
$$(\forall a \in \mathbb{N}^\star)(\exists b, c \in \mathbb{N}^\star)((A[a] \equiv C[c]) \wedge (C[c-1] \preccurlyeq B[b] \prec C[c]))$$

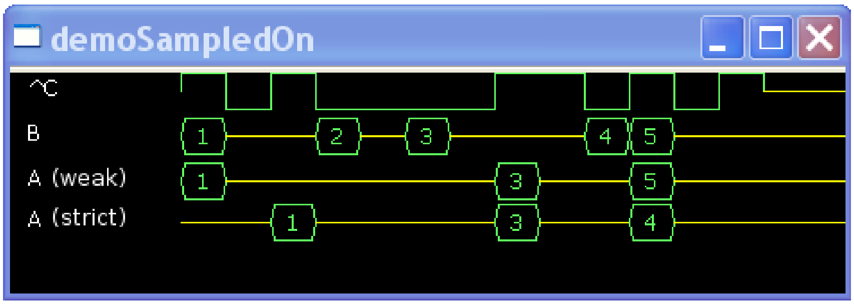


Figure 7: A=B sampledOn C

$$A = B \text{ sampledOn } C \ (A = B \lightning C) \iff$$
$$(\forall a \in \mathbb{N}^\star)(\exists b, c \in \mathbb{N}^\star)((A[a] \equiv C[c]) \wedge (C[c-1] \prec B[b] \preccurlyeq C[c]))$$

5.2 Signal equivalent

As always, the *weak form* of clock relations is more difficult to implement since it implies instantaneous reactions. Synchronous languages are very well-suited to describe such behaviors. The following Signal implementations count the number of occurrences of *inp* between two successive occurrences of *clk*. A sampling occurs where there is at least one occurrence of *inp* (zc/=0).

```
process strictlySampledOn = (? event inp, clk ! event outp )
  (| c ^= zc ^= inp ^+ clk
   | zc := c$ init 0
   | c := 1 when clk when inp default 0 when clk default zc+1 when inp
   | outp := when zc/=0 when clk
   |) where integer c, zc end;
```

The weak form is similar but the input event (*inp*) can occur simultaneously with the sampling clock (*clk*).

```
process sampledOn = ( ? event inp, clk ! event outp )
  (| c ^= inp ^+ clk
   | zzc := 1 when ^inp default zc
   | zc := c$ init 0
   | c := 0 when clk default zc+1
   | outp := when zzc/=0 when clk
   |) where integer c,zc,zzc end;
```

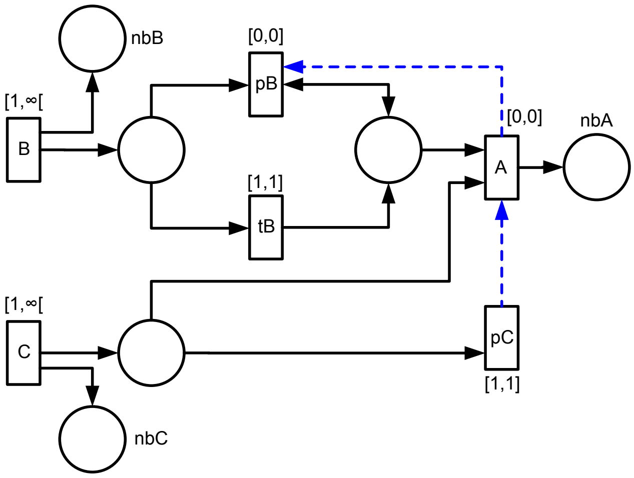


Figure 8: A=B strictly sampledOn C (Time Petri net version)

5.3 Time Petri Net equivalent

Figure 8 shows the Time Petri net implementation of the strict form. This implementation requires priority transitions (dashed/blue arcs between transitions). The arc source has a higher priority than the target. When two transitions are enabled, the transition with highest priority must fire first possibly preventing another transition from firing. When the order does not matter, the transition are said to be independent.

Transition pB empties tokens produced by B when there are multiple occurrences of B between two successive occurrences of C. Transition pC empties tokens produced by C when they are not immediately consumed by transition A. Transition pC must have a higher priority than transition A, an unused clock occurrence is always emptied when possible and not used to sample any input.

The weak form is not easy to implement since it involves instantaneous reactions that leads to races so that the order of firing becomes important. Still, the implementation is possible if we assume that input clock transitions (B and C here) always have a higher priority than other transitions. Because of the lack of room and because it is not possible to formally take this consideration within the model we do not present the implementation here. In a general way, weak forms that involves instantaneous reactions are better represented by synchronous languages.

6 Clock relation delayedFor

The relation delayedFor models a watchdog. A clock start triggers a timer that counts according to a reference clock. A timeout elapsed after delay occurrences of the reference clock. In this retriggerable version, if the clock start occurs again before the time out, then the timer is reinitialized. This is a polychronous operator as the clocks start and timeout are not necessarily synchronous. Note that even though the clock timeout is a subclock of the reference clock, it is not required for the clock start to be a subclock as well.

6.1 Mathematical definition

Let A,B and C be three discrete-time clocks and $\delta \in \mathbb{N}^\star$

$A = B\ delayedFor\ \delta\ on\ C$
$\iff$
$(\forall a \in \mathbb{N}^\star)(\exists c, b \in \mathbb{N}^\star, c > \delta)((A[a] \equiv C[c]) \wedge (C[c-\delta-1] \prec B[b] \preccurlyeq C[c-\delta]))$

6.2 Signal equivalent

The clock relation delayedFor is very different from the Signal operator delay ($) since in Signal the operator delay is monochronous. In the implementation below, signal c counts for the occurrences of *clk* and is reset when *start* occurs. *timeout* occurs when $c = delay$.

```
process delayedFor = { integer delay } ( ? event start,clk ! event timeout )
  (| c ^= start ^+ clk
   | zc := c$ init 0
   | c := 0 when start default zc+1 when clk
   | timeout := when c=delay
   |) where integer c, zc end;
```

6.3 Time Petri net equivalent

Signal operator delay ($) has a simple equivalent in Time Petri net (Figure 9). However, the MARTE operator delayedFor is much more complex and requires the use of inhibitors. In Petri nets, inhibitors are arcs from a place to a transition with a circle as an arrowhead. Contrary to the normal semantics of arcs, the inhibited transition becomes fireable only when no tokens are available in the input place.

Figure 10 shows the MARTE clock relation delayedFor in Time Petri net. As for the relation sampledOn, the transitions pS and tS empty tokens produced by start when more than one token are available. Remember that start and clk are not necessarily synchronous, so start needs to be synchronized.

Transition pC empties the tokens produced by clk when start occurs. The counting should only start on start. The transition inC stops the empty when no tokens from clk are available.

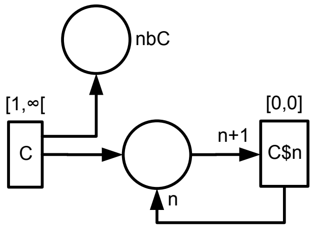


Figure 9: Signal delay: $c\$n$

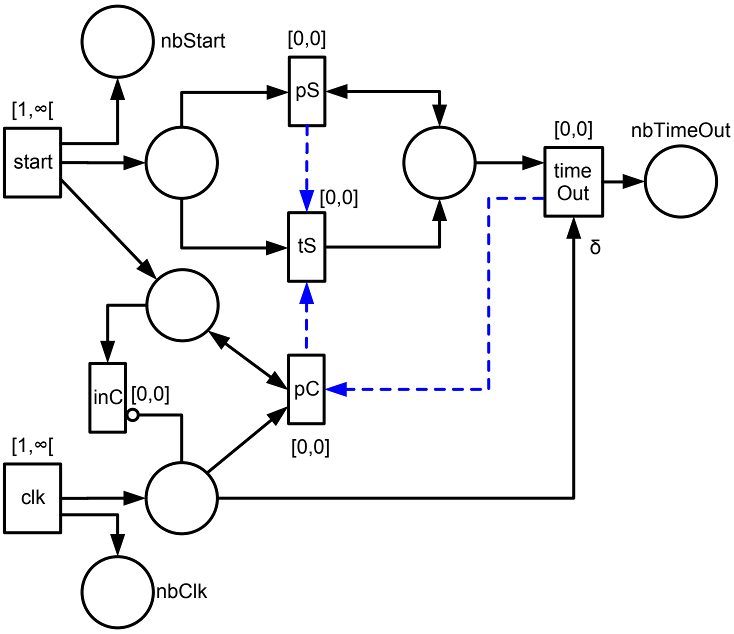


Figure 10: timeOut=start delayedFor δ on clk

7 Illustration: Easter days

7.1 Specification

To illustrate the integration of CCSL clock constraints with UML models we re-use an example that has initially been presented in [6]. This exemple models the canonical rule to process Easter days: "Easter Day is the first Sunday after the 14th day of the lunar month that falls on or after March 21st (nominally the day of the vernal equinox)".

Some requirements refer to events (Sunday, vernal equinox); others involve "temporal" operators (after, on or after). Some events need additional explanations. The *nominal full moon*, refers to an "ecclesiastic" full moon distinct from the astronomic one. It occurs exactly 14 days after the new moon, *i.e.,* , it is the 14th day of the lunar month.

Similarly, the "ecclesiastic" vernal equinox always occurs on March 21st, while the date of the actual spring equinox is the 21st or the 22nd.

7.2 MARTE implementation

Before modeling clock constraints, we need to model clocks. MARTE defines two stereotypes, ClockType and Clock. Clock types gather common informations between similar clocks. The stereotype ClockType extends the UML metaclass Class whereas the stereotype Clock extends the metaclass InstanceSpecification. The clock type we need here will represent days (see Figure 11), it is a discrete clock type. An operation getDate() gives the actual date of the day. A third stereotype, ClockContraint, extending the metaclass Constraint stands for CCSL clock constraints. The specification language of such constraints has to be CCSL.

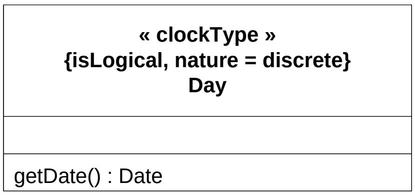


Figure 11: Clock type Day in MARTE

However, a graphical form of the language would be more appropriate to integrate with UML models. We have defined a non-normative representation of clock constraints. This representation is introduced here for the first time. Clock constraints are represented as stereotyped dependencies. The stereotype depends on the kind of clock constraints. For ternary clock constraints (like sampledOn) a first "Clock dependency" is used from the less frequent clock to the more frequent clock. We also use a second auxiliary dependency from the primary dependency to the third clock (the trigger in case of sampledOn).

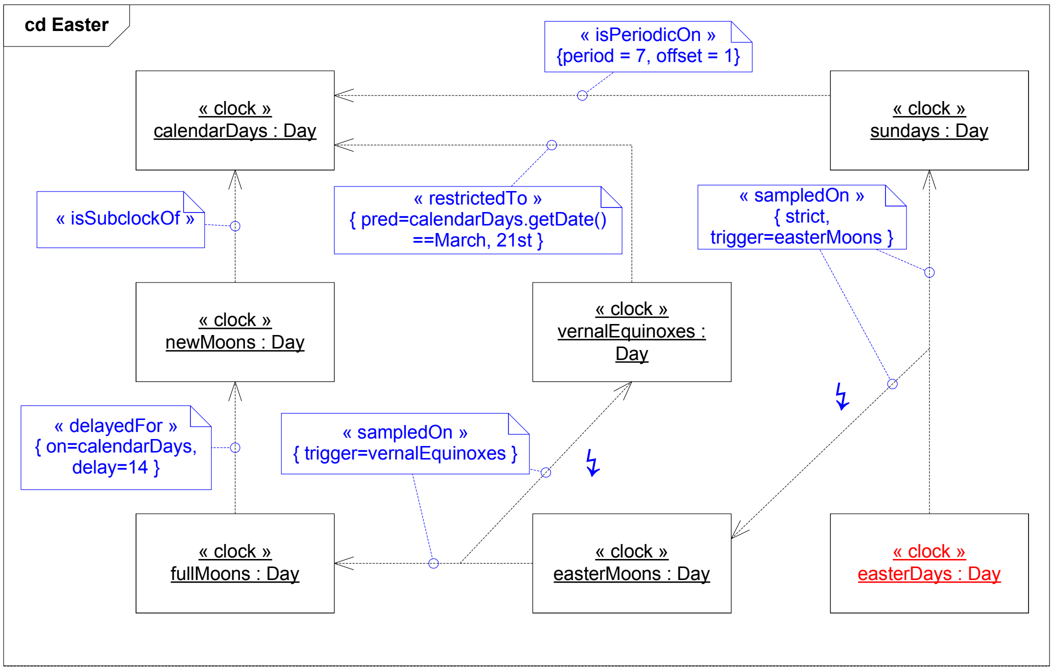


Figure 12: Easter specification with CCSL

Figure 12 shows the specification of Easter applying constraints. Note, that the stereotypes that represent the clock constraints are not normative, as only the time structure itself is normative to let tool vendors make their own graphical choices.

In this specification, the time starts on Saturday, March 1st 2008. This explains the periodic pattern between *sundays* and *calendarDays*. There is one Sunday every *seven*th day and the first Sunday is on March, 2nd. Moreover, to resolve the relation *isFinerThan* from *newMoons* to *calendarDays* we must know the Ephereris, starting of March 1st. The pattern of New Moon occurrences is not completely regular or periodic and cannot be predicted with a simple computation, it has to be given as an input.

Another possible way to represent the same specification using MARTE would be to combine it with SysML parametrics [12]. Each CCSL constraint can be represented as a SysML constraint block whose specification is given as a CCSL expression (see Figure 13). Then, a parametric diagram (see Figure 14) is used to combine constraints together.

7.3 Signal implementation

A full implementation of the Easter specification has been performed in Signal. This implementation has successfully predicted the date of Easter 2008, 2009, and beyond as long

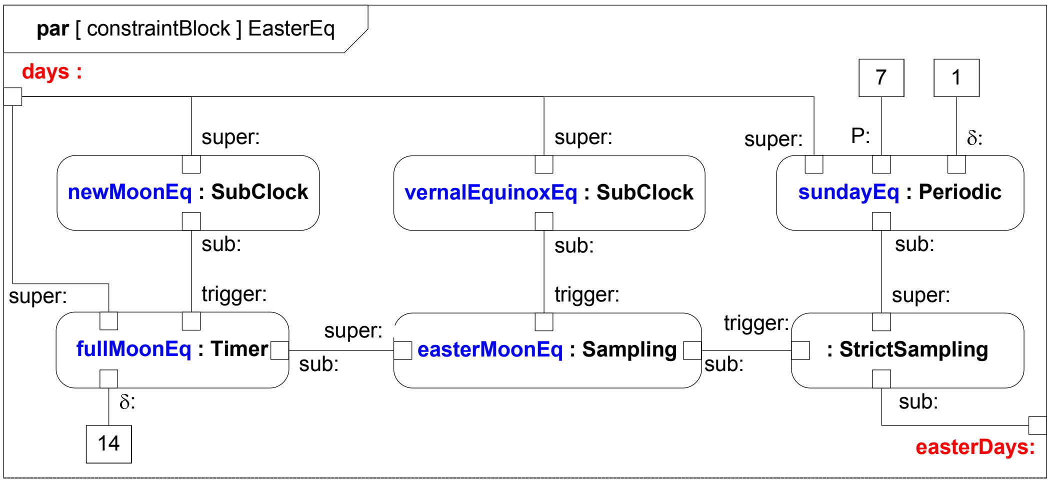


Figure 13: SysML constraint blocks associated with CCSL expressions

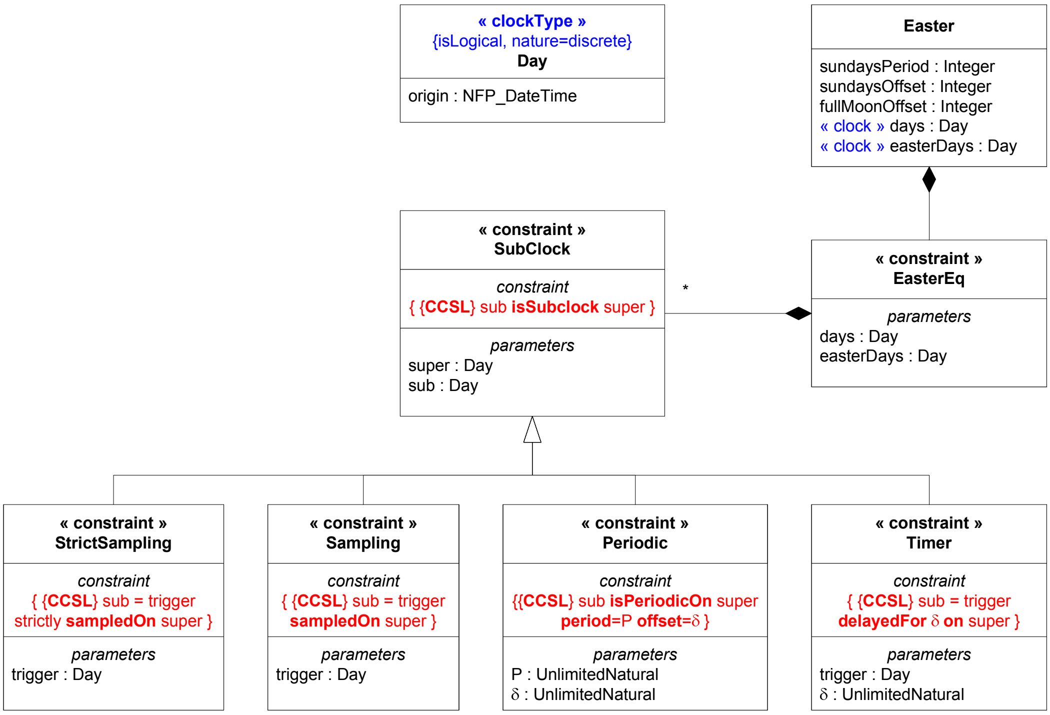


Figure 14: Easter with SysML parametric diagram

as the right Ephemeris is given. This success conforts us in the equivalence of our representation. Even though the date of Easter is known, the specification itself contains several interesting constraints not easy to specify with usual languages.

The formal proof of the equivalence has not yet been performed as the methodology to establish the proof is not clear. Both Signal and Time Petri nets are amenable to formal verifications. We have been using Polychrony [13] for the implementation in Signal and TINA [14] for the implementation in Time Petri nets. Both these environments come with facilities to perform model-checking of properties expressed in a temporal logic. Nevertheless, to establish the equivalence between the properties and our mathematical specification still requires some additional work.

In Signal, the civilian and the lunar months are given as inputs since they cannot be deduced by a simple computation. However, clocks are neither inputs nor outputs but are internal constructs to help constraining a system behavior.

```
process easter =
  ( ? integer month, lunarMonth
    ! integer monthName, newMoon, vernalEquinox, sundays;
       event fullMoon, easterMoon; integer easter )
  (| month08 := month

% oversample month to get days %
   | days2008 := days(month08)

% oversample lunarMonth to get lunar days %
   | lunarDays := days(lunarMonth)
   | lunarDays ^= days2008

% process new moon from input newMoon %
   | newMoon := days2008 when lunarDays = 1

% process full moon = fourteen days later than new moon %
   | fullMoon := delayed{14}(^newMoon, ^days2008)

% process days of the week days in [0,6], first Day of March, 2008
  is saturday=6%
   | weekDays2008 ^= days2008
   | weekDays2008 := cyclingCounter{6,1,6}()

% process month names, starting from March, 3=2+1 %
   | monthName ^= days2008
   | monthName := pre_monthName+1 when pre_monthName<12
                  default 1 when ^month
                  default pre_monthName when ^days2008
```

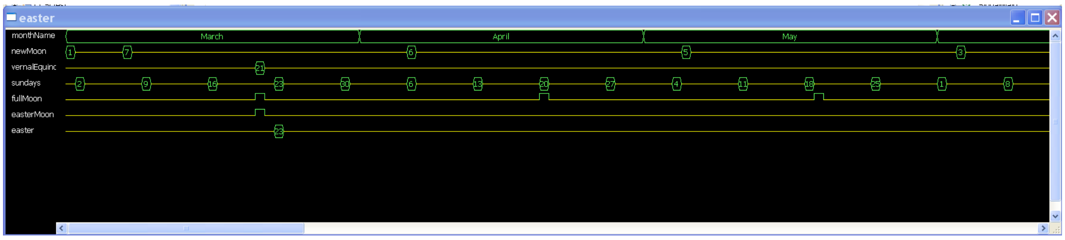


Figure 15: Easter 2008

```
   | pre_monthName := monthName$ init 2

% process vernal equinox: March, 21st%
   | vernalEquinox := days2008 when days2008=21 and monthName=3

% process sundays (=0) knowing than March, 1st 2008 is Saturday (6) %
   | sundays := days2008 when weekDays2008=0

% Easter moon is the first full moon following or equal to the vernal equinox %
   | easterMoon := sampledOn{}(^vernalEquinox, ^fullMoon)

% Easter is the Sunday (stricly) following Easter moon %
   | C_easter := strictlySampledOn{}(easterMoon, ^sundays)
   | easter := days2008 when C_easter

   |) where
      integer month08;
      integer days2008,weekDays2008, pre_monthName;
      integer lunarDays;
      event C_sundays, C_easter;
      use ccsl;
      use counting;
      use calendar
      end
% easter %;
```

Figure 15 shows the result of this Signal process starting on March 1st, 2008. This year, Easter occurs on March, 23rd. Next year, Easter will occur on April 12th, 2009 (Figure 16).

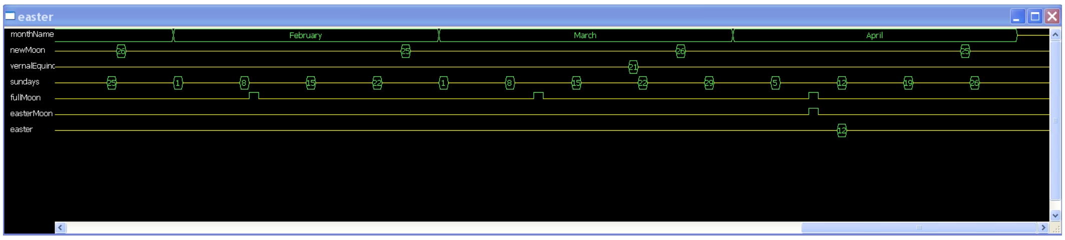


Figure 16: Easter 2009

8 Conclusion

This paper presents several possible equivalent specifications for MARTE clock constraints. Having only a pure mathematical definition as in [6] is not pragmatic since one objective of MARTE Time Model is to provide the support for building executable models for real-time and embedded systems. The transformation of a pure mathematical specification and a simulator is not easy to perform. Nevertheless, we have developed a dedicated simulator that builds a simulation fulfilling all the constraints [6]. Rather than working directly on partial orders there are several formal models available in the community that comes together with analysis and simulation tools. Signal and Time Petri nets are examples.

An OMG Specification is not the right framework to select tools and textual languages. This paper intent is to show that the gap between MARTE-compliant UML models and some existing tools can be filled. To garanty, a full compliance, we still have to prove formally, the equivalence of the proposed Time Petri nets and Signal programs.

The second contribution of this paper is to show the non-normative graphical representation of clock constraints as an instance diagram. Even though, constraints are supposed to apply to a behavior, it is not surprising that we used a structural diagram to represent the constraints. Indeed, MARTE specification introduces a Time Structure. This Time Structure induces a partial order on instants, this partial order is equivalent to the set of all possible execution traces.

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] André, C., Mallet, F.: Clock constraints in UML/MARTE. Research Report 6540, INRIA (2008)

[7] 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

[8] 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

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

[10] Berthomieu, B., Menasche, M.: An enumerative approach for analyzing time petri nets. In: IFIP Congress. (1983) 41–46

[11] Popova-Zeugmann, L., Schlatter, D.: Analyzing paths in time petri nets. Fundamenta Informaticae **37**(3) (1999) 311–327

[12] OMG: Systems Modeling Language (SysML) Specification 1.1. (May 2008) OMG document number: ptc/08-05-17.

[13] Guernic, P.L., Talpin, J.P., Lann, J.C.L.: Polychrony for system design. Journal of Circuits, Systems, and Computers **12**(3) (2003) 261–304

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

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