# On the semantics of UML/Marte Clock Constraints

Frédéric Mallet, Charles André

Int. Symp. on Object/component/service-oriented Real-time distributed Computing (ISORC) (2009) 301-312

# On the semantics of UML/MARTE Clock Constraints

Frédéric Mallet and Charles André
Université de Nice Sophia Antipolis
INRIA Sophia Antipolis Méditerranée
{fmallet,candre}@sophia.inria.fr

## Abstract

*The 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 MARTE, we devised a broadly expressive Time Model to provide a generic timed interpretation for UML models. Our clock constraint specification language supports the specification of systems with multiple clock domains. Starting with a priori independent clocks, we progressively constrain them to get a family of possible executions. Our language supports both synchronous and asynchronous constraints, 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 clock constraint language and we give an equivalent interpretation of this kernel in two other very different formal languages, Signal and Time Petri nets.*

## 1 Introduction

The Unified Modeling Language (UML) [9] aims at being a unified general-purpose modeling language. Its semantics is purposely loose to cover a large domain. So-called *semantic variation points* 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) [10], which is currently in the finalization phase. In its foundations, MARTE defines a broadly expressive Time Model that should provide a generic timed interpretation for UML models. The idea is to define a precise 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 [6], which sets up a common framework for comparing Models of Computation and Communication in the RTE domain. Its origins also lay in works around synchronous languages [3] and more generally polychronous/multiclock languages well-suited to specify Globally Asynchronous and Locally Synchronous (GALS) systems. MARTE comes with a specification language, called *Clock Constraint Specification Language* (CCSL), which is not based on any existing language to let tool vendors choose their own technology. Our goal has been to define a formal language whose keywords denote usual concepts of the domain (periodic, sporadic, sampling...).

CCSL is to be a pivot language to define various but compatible models of computations for *timed* UML models. In our view, UML tools would be used for the graphical editing, while formal languages would provide the execution semantics of separate sub-models. Different languages may be used for different aspects. CCSL would bring interoperability amongst the different models and languages.

A comprehensive *informal* description of CCSL has been presented in [2] and a partial *formal declarative* language-independent description is available in [7]. However, to implement CCSL and build a real-time UML simulator, an operational semantics is more suitable. Instead of redefining an operational semantics for yet another language we rather give equivalent notations for CCSL operators in other executable languages that already have their own analysis tools.

This paper selects a subset of CCSL constraints and studies the suitability of two very different languages (Petri nets [11] and Signal [4]) to model the same concepts. These two languages being general enough, such a description should help a broad community understand (and use) UML/MARTE clock constraints.

Section 2 starts with a general introduction to CCSL and describes general assumptions made on Petri nets and Signal to allow for a comparison. The following sections give for each of the selected constraints, its rationale, a mathematical definition and equivalent Petri net and Signal implementations, when possible.

## 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 total, irreflexive, and transitive binary relation (a quasi-order) 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*. Here, we only consider *pure clock*, *i.e.,* the *ordered set* $\langle \mathcal{I}, \prec \rangle$ and not the labels.

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 $\mathbb{N}^\star = \mathbb{N} \setminus \{0\}$, $\mathrm{idx} : \mathcal{I} \rightarrow \mathbb{N}^\star$, $\forall i \in \mathcal{I}$, $\mathrm{idx}(i) = k$ if and only if $i$ is the $k^{\mathrm{th}}$ instant in $\mathcal{I}$. We restrict the discussion to discrete-time clocks and do not consider dense time at all. A restricted semantics of operators is introduced, assuming that clocks are discrete, whereas these operators may have a more general semantics. For any discrete time clock $c = \langle \mathcal{I}_c, \prec_c \rangle$, $c[k]$ denotes the $k^{\mathrm{th}}$ instant in $\mathcal{I}_c$ (*i.e.,* $k = \mathrm{idx}_c(c[k])$). To simplify computations, we assume a *virtual* instant $c[0]$, so that $c[0] \prec c[1]$.

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 two new instant relations: *Coincidence* ($\equiv \triangleq \preccurlyeq \cap \succcurlyeq$), *Strict precedence* ($\prec \triangleq \preccurlyeq \setminus \equiv$).

*Clock relations* are pre-defined patterns to apply (infinitely) many instant relations to infinite clocks. Clock relations relying on the coincidence instant relation are similar to synchronous operators. Those relying on the precedence relation are close to asynchronous operators. Some mixed relations allows for building asynchronous compositions of multiple synchronous sub-systems. These subsystems may refer to different clocks. In the following, we give examples from all the three categories to give a good overview of CCSL expressiveness.

### 2.2 Signal

In the 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 values 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 functions. So CCSL clocks are very similar to signals and 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.

### 2.3 Time Petri net

MARTE Time model conceptually differs from Petri's work on concurrency theory [11]. Petri's theory restricts coincidence to single points in space-time. In our model, the foundational relationship *coincidence* gathers *a priori* independent points (instants) to reflect design choices.

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 [8] 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 $\theta$ 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. We also associate a place $c_p$ (called *clock place*) with each clock, this place accumulates one token at each occurrence of the clock and thus represents the actual *local time* of the clock, its index. For a given initial marking and for a given firing sequence, there is an injective function $firing : CT \times \mathbb{N}^\star \rightarrow \mathbb{N}$, where $CT$ is the set of clock transitions. $firing(c_t, i)$ is the time at which, the clock transition $c_t$ fires for the $i^{\mathrm{th}}$ time in the firing sequence. We consider a Time Petri net as equivalent to a CCSL clock constraint, iif for all possible firing sequences and all clock transitions (other transitions do not matter), $firing$ preserves the ordering (Eq. 1).

$$\begin{aligned} &(\forall c1, c2 \in C)(\forall k1, k2 \in \mathbb{N}^\star) \\ &\qquad ((c1[k1] \preccurlyeq c2[k2]) \\ &\Leftrightarrow (firing(c1_t, k1) \leq firing(c2_t, k2)) \end{aligned} \tag{1}$$

where $c1_t$ (resp. $c2_t$) is the clock transition associated with clock $c1$ (resp. $c2$). Note that, 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 ($\in \mathbb{N}$).

## 3 Asynchronous alternation

Instant relation precedence ($\prec$) can be extended to clocks to build precedence-based relations. The simplest extensions appears when a clock is *faster than* (or precedes) another one. $A$ is (strictly) faster than $B$ if the $i^{th}$ occurrence of $A$ (strictly) precedes the $i^{th}$ occurrence of $B$, for all $i \in \mathbb{N}^\star$. Clock relation precedes ($\boxed{\prec}$) has been defined in CCSL even though its use in this pure form is not very frequent. A more frequently used relation is alternatesWith, which is a kind of mutual precedence. It composes two precedes relations and represents an alternation between two clocks. A $\boxed{\sim}$ B means that $A$ precedes $B$ and $A$ cannot be more than one tick ahead of $B$, *i.e.,* the advance is bounded by 1. Each occurrence of $A$ is followed by one 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 $A$ (not stricly future), whereas the *strict form* requires $A$ and $B$ to be disjoint.

Typically, an asynchronous communication implies an alternation between sending and receiving. The data is received after having been sent. No other communication can start before the previous one completes. The *weak form* allows the sender to receive the data synchronously with the emission, but does not force the synchronization. The *strict form* is used to forbid a synchronous 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, or empty events in Signal). In practice, there can be (infinitely) many empty instants between any occurrences of $A$ and $B$ and not necessarily as many between two successive occurrences of $A$ or $B$.

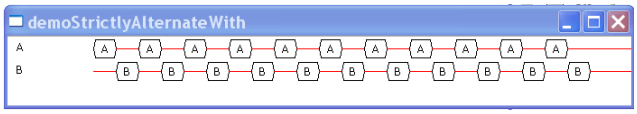


**Figure 1. A strictly alternatesWith B**

Figure 2 shows the equivalent UML StateMachine for both forms of the relation alternatesWith. 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 is rejected at compilation time. In Time Petri net, a violation of constraints results in clock transitions being dead. A liveness analysis indicates constraint inconsistencies.

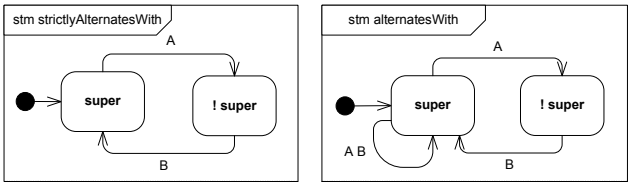


**Figure 2. Behavior of alternatesWith in UML**

### 3.1 Mathematical definition

Let $A$ and $B$ be two discrete-time clocks. Eq. 2 denotes the semantics of the strict form, whereas Eq. 3 denotes the semantics of the (left) weak form. A variant authorizes the relation to be weak on the right side. However, if it is weak on the right side, it is so for all instants. It is fordidden to be alternatively weak on right and left sides because this would lead to causality problems (*e.g.,* $A[k] \equiv A[k+1]$).

$$A \text{ strictly alternatesWith } B \iff (\forall k \in \mathbb{N}^\star)(A[k] \prec B[k] \prec A[k+1]) \quad (2)$$

$$A \text{ alternatesWith } B \iff (\forall k \in \mathbb{N}^\star)(A[k] \preccurlyeq B[k] \prec A[k+1]) \quad (3)$$

The weak form of this clock relation can cause infinitely many different executions even if we ignore empty instants. $B[k]$ precedes $A[k+1]$ but can either be synchronous 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 is 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 the common super clock explicitly.

In the following Signal implementation, line 1 declares a concurrent process and line 2 declares its two pure input signals. Line 3 builds a two-state automaton (see Fig. 2, left part) that alternates between the two states. The boolean signal super is local (see line 6) and alternatively takes

the value $true$ and $false$, starting with $true$. Signal $A$ is present when and only when super is $true$ (line 4). Signal $B$ is present when and only when super is $false$ (line 5).

```
1. process strictlyAlternatesWith =
2.  ( ? event A,B )
3.  (| super := not (super$ init false)
4.   | A ^= when super
5.   | B ^= when not super
6.   |) where boolean super end;
```

The weak form is more complex because either $A$ and $B$ simultaneously occurs, or $A$ occurs alone and $B$ should occur alone in the future. Note that B cannot occur alone when super is $true$. The implementation below directly implements the state machine shown on the righthand side of Figure 2. There are still two states encoded with the local boolean signal super. The state can also change when either $A$ or $B$ occurs. The signal union is denoted by the operator $\hat{}+$ in Signal (line 3). When $B$ occurs, then the next state (nextsuper) is necessarily $true$ (line 5), whatever the current state and whether or not $A$ occurs. When $B$ does not occur the next state is $false$ (line 4). Conversely, $A$ must and can only occur when super is true (line 7). When super is $false$, $B$ must occur but $B$ can also occur when $super$ is true. Line 8 reads that $B$ is more frequent than when super is $false$. The only other possible case is when super is $true$ because of the signal union in line 3.

```
1. process alternatesWith =
2.  ( ? event A,B )
3.  (| nextsuper ^= super ^= A^+ B
4.   | nextsuper := false when not B
5.                       default true
6.   | super := nextsuper $1 init true
7.   | A ^= when super
8.   | B ^> when not super
9.   |) where boolean super,nextsuper end;
```

### 3.3 Time Petri Net equivalent

Figure 3 gives Time Petri nets equivalent to both the strict (left side) and the weak (righthand side) forms of clock relation alternatesWith. They only differ by the time interval on transition $B$. The weak form allows the transition $B$ to fire either simultaneously or stricly after the firing of transition $A$.

In the initial state (time=0) no transitions are enabled, only time can evolve. Then, transition $A$ is enabled but there is no upper bound for its firing. Transition $B$ will not become enabled before $A$ fires. When $A$ eventually fires, $B$ becomes immediately enabled for the weak form and can fire "synchronously" with $A$. In the strict form, because of the time interval $[1, \infty[$, $B$ must wait one instant before being enabled. Still, there is no upperbound. When $B$ fires in turn, the initial state is reached (apart from the counting places) and everything starts again.

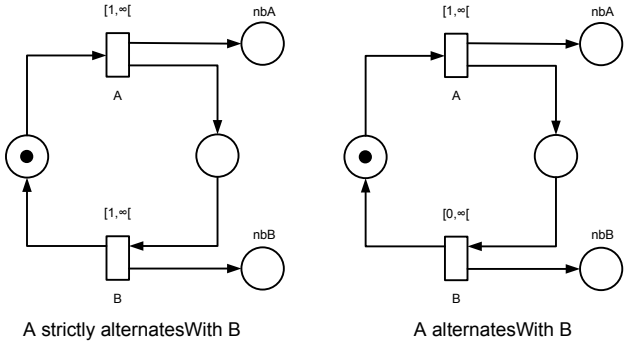


**Figure 3. alternatesWith in Time Petri net**

## 4 Periodicity

Instant relation coincidence ($\equiv$) can be extended to clocks to build coincidence-based relations. The simplest extension consists in making synchronous clocks ($\equiv AB$), *i.e.,* the $i^{th}$ occurrence of $A$ is coincident with the $i^{th}$ occurrence of $B$, for all $i \in \mathbb{N}^{\star}$.

Another extension consists in making subclocks or super clocks, *i.e.,* each and every instant of the sub clock $A$ is coincident with one instant of the super clock $B$ ($A$ isSubclockOf $B$).

Relation isPeriodicOn is a refinement of relation isSubclockOf that builds a periodic clock with respect to a *super clock* for a given period. Note, that the clocks need not be chronometric. Optionally, an offset may be specified when the periodic behavior only starts after a given number of occurrences of the parent clock. More general than the periodic relation is the k-periodic pattern, which can be modeled in CCSL with the clock relation filteredBy (symbolically represented by ▼). This relation uses an infinite periodic binary word to define the coincident instant. Each one in the binary one represents a coincidence. $A$ isPeriodicOn $B$ period $=P$ offset $=\delta$ is equivalent to a synchronous filtering starting from the $\delta^{th}$ occurrence of $B$ where $A$ occurs every $P^{th}$ occurrence of $B$. This can also be written $A = B \blacktriangledown (O^{\delta} \bullet (1 \bullet O^{P-1})^{\omega})$, where $\bullet$ stands for the binary word concatenation. When $\delta = 0$ and $P = 1$, $A$ and $B$ are synchronous. The use of infinite binary words for filtering clocks has been inspired by some work on n-synchronous kahn networks [5].

Figure 4 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.

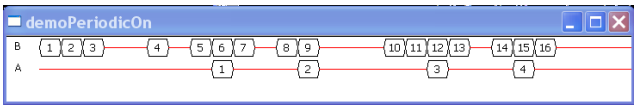


**Figure 4. A isPeriodicOn B period=3 offset=5**

### 4.1 Mathematical definition

Let $A$ and $B$ be two discrete-time clocks. Eq 4 gives the semantics of clock relation isPeriodicOn.

$$A \text{ isPeriodicOn } B \text{ period } =P \text{ offset } =\delta \iff (\forall k \in \mathbb{N}^{\star})(A[i] \equiv B[(i-1) * P + \delta + 1]) \quad (4)$$

It is easy to show that if $period = 1$ and $offset = 0$ then $A$ and $B$ are synchronous. By definition, $AisSubclockOfB$. Note that the definition does assume any direction. If $B$ is already defined, then $A$ will be uniquely defined. However, the relation also works in the other direction. If $A$ is already defined, then $B$ will be partially defined by applying such a constraint.

### 4.2 Signal equivalent

The implementation in Signal is straightforward since CCSL filteredBy relation is very close to the Signal operator when. And the periodic case is one simple application.

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

Not surprisingly, all coincident-based operators have almost direct equivalent in synchronous languages in general and in Signal. It is not always the case for precedence-based operators for which it may be tedious. Even when the languages can be twisted to model such constraints, the compiler is not always able to find a solution. Consequently, it is really useful to have a specification language that simply supports various concepts even if several implementation languages must be combined to find possible solutions.

### 4.3 Time Petri Net equivalent

Figure 5 gives a Time Petri net equivalent to clock relation isPeriodicOn. Transition $B$ must fire $\delta + 1$ 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. Indeed, with classical Petri nets, it is not possible at all to express coincidence. In Time Petri net, we use such a trick to do it but it becomes difficult to use the same trick when more than two transitions are to be synchronized. In such cases, applying priorities may become necessary (see Section 5).

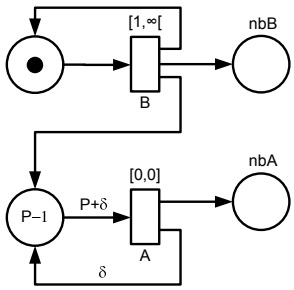


**Figure 5. A isPeriodicOn B period=P offset=$\delta$ (Time Petri net version)**

## 5 Sampling

Upto here, we have described pure synchronous (filteredBy, isSubclockOf, isPeriodicOn) or pure asynchronous (isFasterThan, alternatesWith) operators. To asynchronously combine synchronous domains another kind of operators is required: mixed operators.

The relation sampledOn is used to synchronize on a given clock external events, *a priori* asynchronous. This sampling operator is often used for synchronizing asynchronous inputs, but also to model time-triggered communications.

$A = B \downarrow C$ defines a subclock of $C$ (less frequent than $C$ in Signal terminology) that occurs only after an occurrence of $B$. Obviously, $B$ is not assumed to be a subclock of $C$. The *strict form* of sampledOn does not sample an occurrence of $B$ when it is synchronous with an occurrence of $C$, this instant will only be sampled on the next occurrence of $C$.

Figure 6 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 $B$.

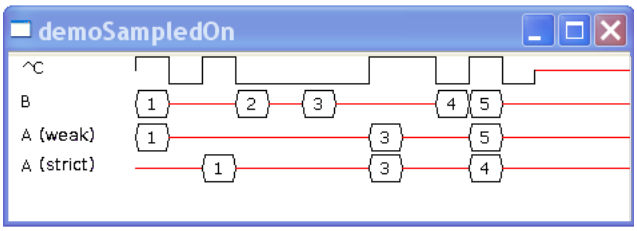


**Figure 6. A=B (strictly) sampledOn C**

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. Eq. 5 gives the semantics of the strict form, whereas Eq. 6 gives the semantics of the weak form.

$$A = B \text{ strictly sampledOn } C \iff (\forall a \in \mathbb{N}^\star)(\exists b, c \in \mathbb{N}^\star) ((A[a] \equiv C[c]) \wedge (C[c-1] \preccurlyeq B[b] \prec C[c])) \quad (5)$$

$$A = B \text{ sampledOn } C \ (A = B \not\downarrow C) \iff (\forall a \in \mathbb{N}^\star)(\exists b, c \in \mathbb{N}^\star) ((A[a] \equiv C[c]) \wedge (C[c-1] \prec B[b] \preccurlyeq C[c])) \quad (6)$$

### 5.2 Signal equivalent

The *weak form* of clock relations is more difficult to implement since it implies instantaneous reactions. Synchronous languages are well-suited to describe such behaviors, even though sampling is not a primitive operator. 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 if the input event ($inp$) occurs simultaneously with the sampling clock ($clk$), *i.e.,* it is not strictly future, then it must be sampled. This requires to be one more step ahead ($zzc$).

```
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;
```

### 5.3 Time Petri Net equivalent

Figure 7 shows the Time Petri net implementation of the strict form. This implementation requires priority transitions (dashed/blue arcs between transitions). The arc source has an higher priority than the target. When two transitions are enabled, the transition with the highest priority must fire first possibly preventing another transition from firing (if the system is not conflict-free). When the order does not matter, the transition are said to be independent.

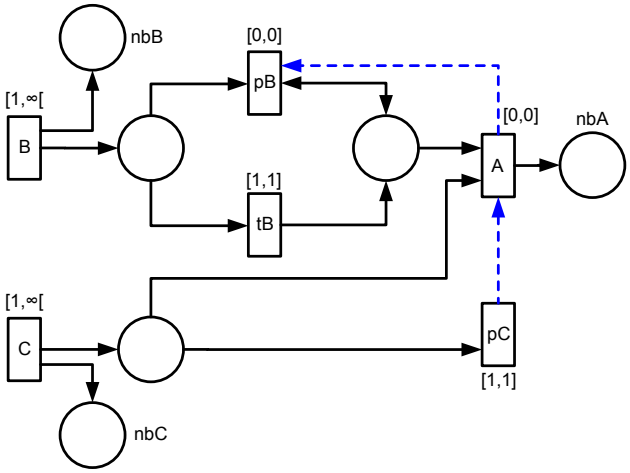


**Figure 7. A=B strictly sampledOn C (Time Petri net version)**

Transition pB purges tokens produced by $B$ when there are multiple occurrences of $B$ between two successive occurrences of $C$. Transition pC purges tokens produces by $C$ when they are not immediately consumed by transition $A$.

Transition pC must have a higher priority than transition A, an old clock concurrence is always purged when possible and not used to sample any input.

Figure 8 shows the Time Petri net implementation of the weak form. It is mainly a matter of changing the priority between transitions. Nevertheless, one big difference appears on the time interval of transition $tB$. Since there can be an instantaneous sampling of input $B$ (if $C$ is coincident) the path from transition $B$ to transition $A$ via $tB$ cannot be longer than 0.

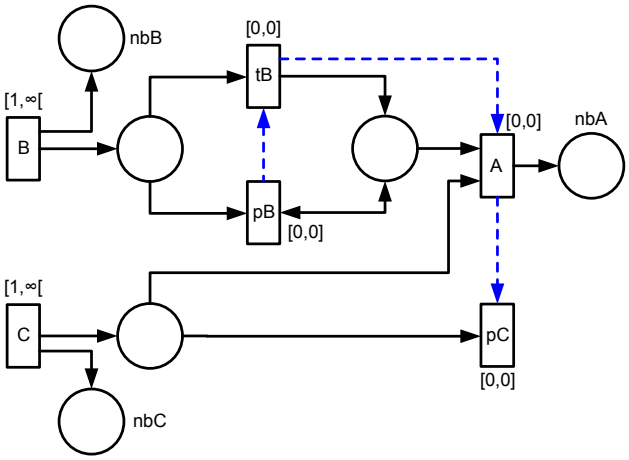


**Figure 8. A=B sampledOn C (Time Petri net version)**

With these two implementations, there can still be inconsistent firing sequences unless a mechanism is added to force all inputs that must must in a given instant to fire before any other transition is fired. Such a mechanism mimics the synchronous languages where all inputs are captured (present or absent) in a first step and outputs are processed in a second step. Due to the lack of room, this simple mechanism is not detailed any further here.

## 6 Clock relation delayedFor

The relation delayedFor models a watchdog and is also a mixed operator. A clock start triggers a timer that counts according to a reference clock. A timeout elapsed after delay occurrences of the reference clock. It 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 xsynchronous. 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. Figure 9 illustrates a possible execution of the clock constraint delayedFor.

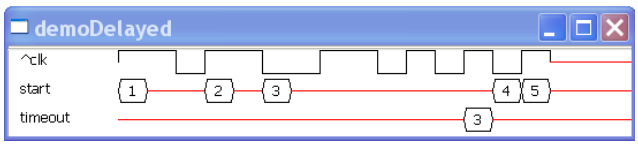


**Figure 9. timeout=start delayedFor 4 on clk**

### 6.1 Mathematical definition

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

$$A = B \; delayedFor \; \delta \; on \; C \iff (\forall a \in \mathbb{N}^\star)(\exists c, b \in \mathbb{N}^\star, c > \delta) ((A[a] \equiv C[c]) \wedge (C[c-\delta-1] \prec B[b] \preccurlyeq C[c-\delta])) \quad (7)$$

### 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 very simple equivalent in Time Petri net (Figure 10). 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 token are available in the place.

Figure 11 shows the MARTE clock relation delayedFor in Time Petri net. As for the relation sampledOn, the transitions pS and tS purge 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 purges the tokens produced by clk when start occurs. The counting should only start on start. The transition inC stops the purge when no tokens from clk are

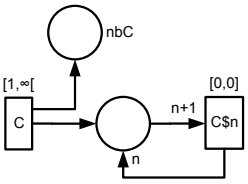


**Figure 10. Signal delay: C$n**

available. It purges on clk ticks that occurred before the occurrence of $start$. This implementation uses an inhibitor arc (denoted with an empty circle as an arrowhead). A transition with an incoming inhibitor arc (*e.g.,* $inC$) can only fire if there is no token in the associated place.

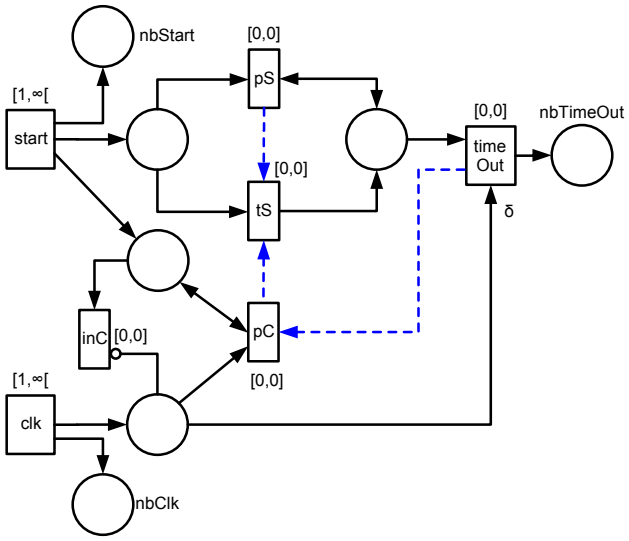


**Figure 11. timeOut=start delayedFor $\delta$ on clk**

## 7 Conclusion

This paper presents several possible equivalent specifications for MARTE clock constraints. Having only a pure mathematical definition as in [1] 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. Instead of building a MARTE-specific analysis tools, we show here how we can rely on existing languages that come with analysis and simulation tools. We also exhibit some typical examples where Petri nets are well-adapted to express CCSL constraints and others where synchronous languages are more suitable. Fortunately, both Time Petri nets and synchronous languages propose facilities to run model-checking tools. Therefore, we can express temporal logic properties and prove that these properties hold with the two proposed implementations. Proving that the properties are also compatible with our denotational semantics is less easy and remains to be done.

An OMG Specification is not the right framework to select tools and textual languages. This paper intent is to show that MARTE contains powerful time primitives, which can be used to exploit UML semantics variation points so as to make the UML models semantically closer to formalisms amenable to formal analysis.

## Acknowledgments

The Aoste project is a joint project between I3S Laboratory (UMR 6070 CNRS) and INRIA Sophia Antipolis Méditerranée.

This work has also been partially supported by the project OpenEmBeDD (http://openembedd.org).

Many thanks to Dumitru Potop-Butucaru and to the Espresso team for the fruitful discussions on the internals of the Signal compiler.

## References

[1] C. André and F. Mallet. Clock constraints in UML/MARTE. Research Report RR-6540, INRIA, 2008.

[2] C. André, F. Mallet, and R. de Simone. Modeling time(s). In G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, editors, *MoDELS*, volume 4735 of *Lecture Notes in Computer Science*, pages 559–573. Springer, 2007.

[3] A. Benveniste, P. Caspi, S. Edwards, N. Hallbwachs, P. le Guernic, and R. de Simone. The synchronous languages twelve years later. *Proceedings of the IEEE*, 91(1), 2003.

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

[5] A. Cohen, M. Duranton, C. Eisenbeis, C. Pagetti, F. Plateau, and M. Pouzet. N-synchronous kahn networks: a relaxed model of synchrony for real-time systems. In J. G. Morrisett and S. L. P. Jones, editors, *POPL*, pages 180–193. ACM, 2006.

[6] 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.

[7] F. Mallet and C. André. CCSL: specifying clock constraints with UML/Marte. *ISSE*, 4(3):309–314, 2008.

[8] P. Merlin. *A Study of the Recoverability of Computer Systems*. PhD, University of California, Irvine, 1974.

[9] Object Management Group. Unified Modeling Language, Superstructure 2.2, Nov. 2007. OMG: formal/07-11-02.

[10] Object Management Group. *UML Profile for MARTE, beta 2*, June 2008. OMG document number: ptc/08-06-08.

[11] C. Petri. Concurrency theory. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and their properties*, volume 254 of *Lecture Notes in Computer Science*, pages 4–24. Springer-Verlag, 1987.