

Clock constraint specification language: specifying clock constraints with UML/MARTE

Frédéric Mallet

Received: 1 July 2008 / Accepted: 26 July 2008 / Published online: 9 August 2008
© Springer-Verlag London Limited 2008

Abstract The Object Management Group (OMG) unified modeling language (UML) profile for modeling and analysis of real-time and embedded systems (MARTE) aims at using the general-purpose modeling language UML in the domain of real-time and embedded (RTE) systems. To achieve this goal, it is absolutely required to introduce inside the mainly untimed UML an unambiguous time structure which MARTE model elements can rely on to build precise models amenable to formal analysis. The MARTE Time model has defined such a structure. We have also defined a non-normative concrete syntax called the clock constraint specification language (CCSL) to demonstrate what can be done based on this structure. This paper gives a brief overview of this syntax and its formal semantics, and shows how existing UML model elements can be used to apply this syntax in a graphical way and benefit from the semantics.

Keywords Unified modeling language · Time model · Constraints · MARTE

1 Introduction

The unified modeling language (UML) [6] aims to be 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) [7], which is currently in the finalization phase. At its foundations, MARTE defines a broadly expressive time model to provide for a generic timed interpretation of UML models. The idea is to define precisely a semantics within the profile rather than allowing tools to provide their own, possibly incompatible with other tools of the same domain.

MARTE Time structure is heavily inspired by the tagged signal model [4], 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 [3] and more generally polychronous/multiclock languages well suited to specify globally asynchronous and locally synchronous (GALS) systems. The concrete syntax of our language, which is called the *clock constraint specification language* (CCSL), is part of the MARTE profile but is not normative and not based on any existing language, to enable tool vendors to choose their own technology. Our goal has been to use explicit keywords that denote usual concepts of the domain (periodic, sporadic, sampling, etc.).

A comprehensive *informal* description of CCSL has been presented previously [2] and a partial *formal declarative* description is available [1]. Using a declarative mathematical description enables language independence. 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. After a brief overview of the semantics of a core CCSL

F. Mallet (✉)
INRIA Sophia Antipolis Méditerranée,
Université de Nice Sophia Antipolis,
2004 rte des Lucioles, BP 93,
06902 Sophia Antipolis Cedex, France
e-mail: Frederic.Mallet@sophia.inria.fr

constraint subset, this paper proposes several possible graphical UML-compatible representations of these constraints. These graphical representations are not yet part of MARTE and may be proposed to the second OMG Finalization Task Force for integration into the next MARTE revision.

Section 2 starts with a brief overview of MARTE Time structure, the official OMG profile. Then, Sect. 3 describes the main aspects of the non-normative constraint language CCSL. Section 4 compares three different visual representations of the constraints.

2 MARTE Time subprofile

2.1 Time structure

A *clock* is a five-tuple $\langle \mathcal{I}, <, \mathcal{D}, \lambda, u \rangle$, where \mathcal{I} is a set of instants (possibly infinite), $<$ 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, and 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* $\langle \mathcal{I}, < \rangle$, and the values are never mentioned. $<$ 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 $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$, $\text{idx} : \mathcal{I} \rightarrow \mathbb{N}^*$, $\forall i \in \mathcal{I}, \text{idx}(i) = k$ 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. For all operators, we always assume that clocks are discrete, whereas these operators may have a more general semantics when applied to dense clocks.

For any discrete-time time structure $c = \langle \mathcal{I}_c, <_c \rangle$, $c[k]$ denotes the k^{th} instant in \mathcal{I}_c (i.e., $k = \text{idx}_c(c[k])$). 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 $c[0] \equiv {}^{\circ}c[1]$. Note, these definitions of predecessor and successor are only possible with a discrete 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 instant relations: *coincidence* ($\equiv \Delta \preccurlyeq \cap \succ$), *strict precedence* ($< \triangleq \preccurlyeq \setminus \equiv$), *independence* ($\parallel \triangleq \preccurlyeq \cup \succ$), and *exclusion* ($\# \triangleq < \cup \succ$).

2.2 UML profile

Figure 1 presents a simplified view of the MARTE Time subprofile. The green elements are defined outside the profile itself but are useful to understand the profile. The stereotype ClockType extends the metaclass Class. It models classes of

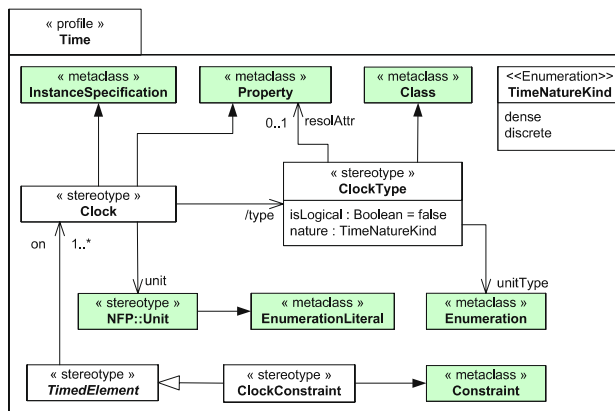


Fig. 1 Excerpt of MARTE Time subprofile

compatible clocks, i.e., clocks of the same nature (discrete or dense) and that use the same set of units. The stereotype Clock extends the metaclasses InstanceSpecification and Property. It models a set of instants as defined in the previous subsection. Clocks can appear either in instance diagrams to represent a snapshot of a system at a given time or in composite structure diagrams to represent a family of possible behaviors. Starting from there, several identified model elements can be associated with one or several clocks using one of the concrete stereotype that specializes the abstract stereotype TimedElement. This association with a clock provides the ability to the model element to embed expressions or value specifications identifying precisely instants or durations. Having different reference clocks is very useful in distributed systems where different elements use different time bases. It is also useful in electronic design with many-core architectures or even mono-core architectures where several time domains are defined (main clock, bus clock, etc). Locally, it is often better to consider these time domains/bases (clocks) as independent so each part can be designed separately. However, during integration, it is required to understand the relations between these clocks to deal with interdomain communications. Clock constraints have been defined with that objective. They make explicit relations amongst clocks. Stereotype ClockConstraint extends metaclass Constraint and the clock constraint specification language (CCSL) gives a possible non-normative concrete syntax to specify constraints. CCSL is briefly introduced in the next section.

3 Clock constraints

The Time structure defines relations between instants. However, as a clock is an infinite set of instants, it is neither realistic nor practical to specify constraints one by one. Based on instant constraints, it is easy to build more powerful relations that define infinitely many instant relations. We call these

relations *clock constraints*. We can classify clock constraints into three families: *coincident-based*, *precedence-based*, and mixed constraints.

3.1 Coincidence-based constraints

Coincidence-based clock constraints define infinitely many coincidence instant relations. Most of the time, this kind of constraints defines a subclock from a given superclock, i.e., a less frequent clock. For instance, we can select every third instant to create a periodic subclock of period 3. However, as most clock constraints are relations, as opposed to functions, they can also do the contrary and define a superclock from one (or a set of) subclock(s). For example, to model a phase-lock loop (PLL) system, one may wish to oversample a given clock four times.

The most frequently used constraint of this family is *isPeriodicOn*, whose semantics is given in a mathematical declarative way by Eq. (1).

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

This family describes *synchronous* relations inspired from operators defined in synchronous languages [3].

3.2 Precedence-based constraints

The *precedence-based* clock constraints define infinitely many precedence instant relations. They characterized asynchronous relations. The most frequently used constraint of this family is *alternatesWith*. Relation *alternatesWith* represents alternation between two clocks. $A \sqsupset B$ means that each occurrence of *A* is followed by an occurrence of *B* before any other occurrences 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 sent. No other communications can start before the previous one completes. The *weak form* allows the sender to receive data simultaneously with the emission, but does not enforce the synchronization. The *strict form* is used to forbid instantaneous communications. The strict form semantics is given by Eq. (2), whereas Eq. (3) gives the semantics of the weak form.

$$\begin{aligned}
 &A \text{ strictly alternatesWith } B \\
 &\iff \\
 &(\forall k \in \mathbb{N}^*)(A[k] < B[k] < A[k + 1])
 \end{aligned} \tag{2}$$

$$\begin{aligned}
 &A \text{ alternatesWith } B \\
 &\iff \\
 &(\forall k \in \mathbb{N}^*)(A[k] \preceq B[k] < A[k + 1])
 \end{aligned} \tag{3}$$

3.3 Mixed constraints

Mixed constraints combine both precedence and coincidence relations. There are useful when modeling communications from an asynchronous part of a design to a synchronous part. The most frequently used constraint of this family is *sampledOn*. The relation *sampledOn* represents sampling; it can be used to model time-triggered communications or for synchronizing asynchronous inputs. $A = B \downarrow C$ defines a subclock of *C* 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 2 shows one possible scenario involving the clock relation *sampledOn* in 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. The strict-form semantics is given by Eq. (4) whereas Eq. (5) gives the semantics of the weak form.

$$\begin{aligned}
 &A = B \text{ strictly sampledOn } C \\
 &\iff \\
 &(\forall a \in \mathbb{N}^*)(\exists b, c \in \mathbb{N}^*) \\
 &((A[a] \equiv C[c]) \wedge (C[c - 1] \preceq B[b] < C[c]))
 \end{aligned} \tag{4}$$

$$\begin{aligned}
 &A = B \text{ sampledOn } C \ (A = B \downarrow C) \\
 &\iff \\
 &(\forall a \in \mathbb{N}^*)(\exists b, c \in \mathbb{N}^*) \\
 &((A[a] \equiv C[c]) \wedge (C[c - 1] < B[b] \preceq C[c]))
 \end{aligned} \tag{5}$$

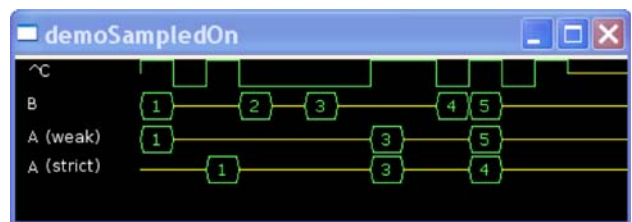


Fig. 2 $A = B \text{ sampledOn } C$

4 Applying constraints to UML models

4.1 Using UML constraints

To apply these constraints, one must first create clocks and therefore clock types. Some very useful clock types are provided in the MARTE library, such as `IdealClock`, which represents a perfect dense chronometric clock. Other clocks, with flaws such as jitter, drift, can be derived from `IdealClock` or rather from one of its instances (a clock). The clock `idealClk` is also provided in the MARTE library and is an instance of `IdealClock`. `idealClk` can be discretized with a given resolution to build, for instance, a chronometric discrete clock whose frequency is 100 Hz (see Eq. (6)).

$$c_100 = \text{idealClk discretizedBy } 0.01 \quad (6)$$

More generally, MARTE also provides support for the use of logical clocks, i.e., clocks not directly related to physical time. Anything that has to be compared (before, after, or coincident) to something else should be considered as a logical clock. For instance, to provide timing information about thread dispatch, a class `Thread` can be defined and the stereotype `ClockType` can be applied to it. Doing so, instances of `Thread` or properties/parts/ports of type `Thread` may become clocks. Note that, being a `ClockType` does not prevent them from being something else, such as a `SchedulableResource`; it only provides support to build clock constraints and express causality relations with other clocks.

Figure 3 illustrates such a case where two periodic threads ($t1$ and $t3$) are mixed with an aperiodic thread ($t2$). The two clock constraints make the two threads (clocks) periodic relative to clock c_100 . The two threads are harmonic since they refer to the same clock with an offset 0 and the ratio of their period is an integer. Thread $t2$ is aperiodic; no relation relates it to a clock. Clock c_100 is a shared clock (for instance, a chronometric discrete clock) and as such appears within a dashed rectangle. c_100 is not owned by class `PeriodicAperiodicPeriodic` and could be used in another composite structure diagram. This diagram has been built using Papyrus, an open-source tool available at <http://www.papyrusuml.org>.

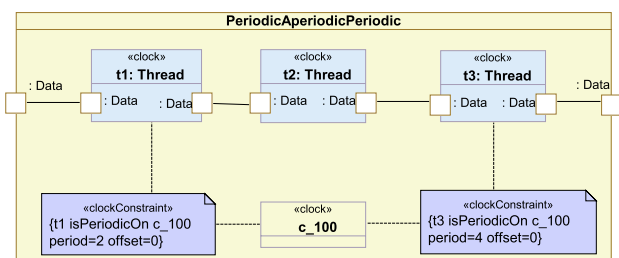


Fig. 3 Periodic threads with CCSL

This notation using constraints requires the user to know CCSL concrete syntax. However, a more obvious visual notation may be more appropriate. Constraints become visually explicit when all constrained elements appear on the same diagram. In this particular diagram, we do not see the subprograms actually executing on these threads. One purpose of the Time model is to apply consistency between different diagrams through clocks. Different timed elements from different diagrams are connected together by clock constraints, which constrain them to behave in a consistent way. The two following subsections propose alternative notations where all clocks are shown on the same diagram.

4.2 Using a CCSL-specific profile

UML profiles can introduce new concepts but can also change the visual notation. Introducing a new visual language for clock constraints within MARTE is not practical because MARTE is very large and there are many other matters to address. A solution is to define aside, a CCSL-specific profile. Each often-used CCSL constraint can have its own stereotype and a specific icon. The most appropriate metaclass to represent CCSL constraint is probably the metaclass `Dependency`.

Figure 4 shows a profile that could ease the use of CCSL constraints. The left-hand side of the figure shows the actual profile. The right-hand side of the figure shows another composite structure diagram for the same three-thread example. In this example, the diagram represents three new clocks (`step1`, `step2`, and `step3`) that are the actual behaviors (subprograms) executed by the threads. These clocks could also have been shown on the previous diagram but this would have resulted in a very heavy construct. Using the defined stereotypes and their icons brings simplicity even though the semantics is exactly the same.

The relation `alternatesWith` from the threads to their assigned subprogram denotes that, every time the thread is dispatched, the subprogram must execute. Additionally, the subprogram cannot be executed again before the completion of the previous execution.

The asynchronous communication from `step1` to `step2` is modeled with a `alternatesWith` relation. Whenever `step1` completes, `step2` takes its output data and executes.

Finally, the ternary relation involving `step2`, `step3`, and `t3` denotes the synchronization of the `step2` output with the clock `t3`. Every time `t3` is dispatched, whatever the current status of `step2` output, the data available is sampled and `step3` executes using this sample. Obviously, this may lead to data loss if the sampling rate is too low, or to the use of the same sample data in multiple executions of `step3` if the sampling rate is too high.

This representation requires the definition of a new profile aside MARTE. In this profile, each CCSL constraint has its own stereotype and its properties are determined according to

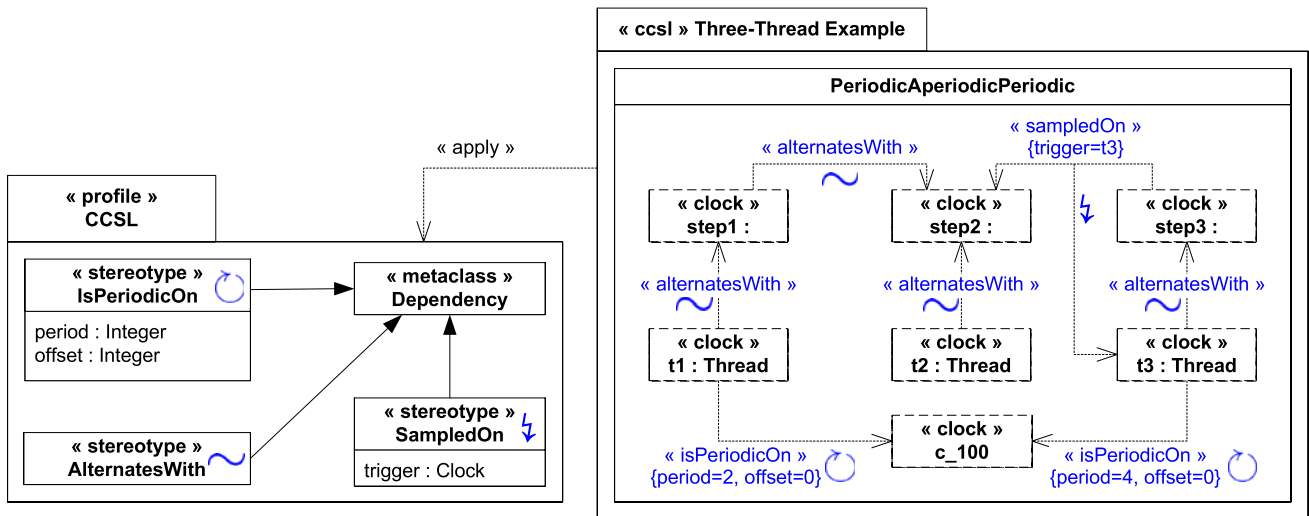


Fig. 4 Using a CCSL-specific profile

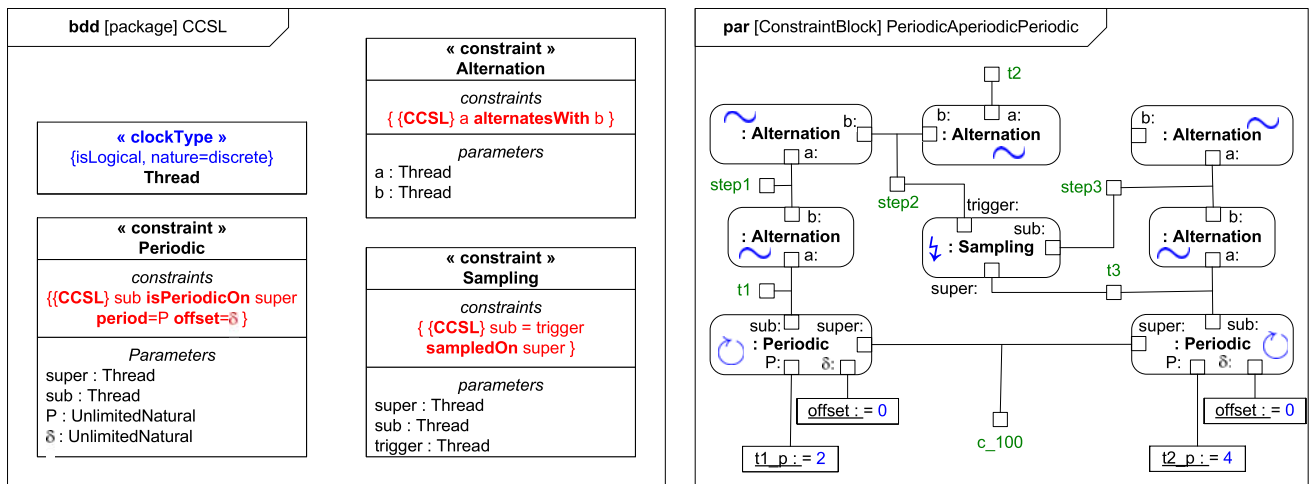


Fig. 5 Using SysML parametric diagrams

the CCSL grammar. An alternative to the creation of another profile is to use parametric diagrams as described in the following subsection.

4.3 Using SysML parametric diagrams

The UML profile for system engineering (SysML) [8] is a specification adopted by the OMG to be used at the system level. It consists of a subset of UML constructs called UML4SysML together with few new extensions, including refinement and parametric diagrams. The former helps to make explicit system-level requirements and trace their proposed implementations. The latter should be used to represent noncausal relations among values of the system and possibly make physical laws required for the design explicit within the model.

So, we can use this SysML construct to represent laws related to time, whether physical or logical. SysML recommends building a new “constraint block” for each new law and then using it in so-called *parametric diagrams* to apply this law to relevant design constraint values. In our case, we have a small number of identified relations among logical clocks. Consequently, we can construct a library of CCSL-specific constraint blocks.

Figure 5 illustrates the same example using SysML constraint blocks and a parametric diagram. The left-hand side shows an excerpt of the CCSL-specific library. Three constraint blocks (Periodic, Alternation, Sampling) have been defined for each of the three CCSL relations previously introduced. Each constraint block has two compartments. The bottom one, called *parameters*, contains typed *formal* parameters. The upper compartment, called *constraints*, contains the constraint itself that applies on the parameters. In our

case, the constraint is defined in CCSL. However, this library is built once and for all, so end-users need not being entirely familiar with the concrete syntax and only need to be familiar with the underlying concepts.

The right-hand side of the figure presents the three-thread example as a SysML parametric diagram. In such a diagram, boxes are properties extracted from the model. Some of the properties are clocks ($t1$, $step1$, etc), while others have integer values ($offset$, $t1_p$, etc.). These properties may come from different diagrams and different blocks. The rounded rectangles are usages of constraint blocks. Their ports, which represent parameters, are connected with properties using noncausal connectors. Being noncausal means that there is no input nor output, whichever value is known causes the other to update. For instance, considering Alternation, if b is known, one can deduce (partially) a but similarly, if a is known, then one can deduce (partially) b .

5 Conclusion

This paper has presented some representative clock constraint relations introduced by the MARTE Time model together with their formal semantics. These clock constraints bring consistency in the timing information of UML models. Introducing formal models in UML models is a problem. OMG specifications are usually large and are not the right place to put formal specifications. Leaving the formal semantics outside the specification leaves the interpretation to tools, possibly resulting in tools for the same domain with different interpretations. These different interpretations are a major impediment that reduces interoperability and is one of the reasons that prevents the acceptance of UML in a wider community.

With CCSL, we rely on UML constraints. However, constraints are not the best visual representation. We propose herein two visual alternative notations that result in more compact diagrams. These two solutions are not specific to CCSL and could be easily extended to other constraint-based languages such as the Object Constraint Language (OCL). The limitation lies in finding the right icon or shape that

immediately makes the link with the constraint. Our first proposition implies the creation of a CCSL-specific profile. The second relies on SysML parametric diagrams. It is interesting to see that all the proposed notations are composite structure diagrams or derived notations.

These two notations attempt to replace CCSL concrete syntax by a visual representation and to allow a representation of otherwise scattered timing information on a single diagram.

The next step is to provide analysis support for such diagrams. We already have implemented an Eclipse-based simulator for CCSL constraints [1] and we are trying to integrate it with parametric diagrams to obtain direct visual feedback of a simulation run. We have also made some progress in transforming CCSL constraints into other formalisms, such as Signal or Petri nets [5], which have their own analysis techniques and tools.

Acknowledgments This work elaborates on previous works [1,5] with Charles André who is striving to define sound foundations for the MARTE Time structure and CCSL.

References

1. André C, Mallet F (2008) Clock constraints in UML/MARTE. Research Report 6540, INRIA. <http://hal.inria.fr/inria-00280941/en/>
2. André C, Mallet F, de Simone R (2007) Modeling time(s). In: Engels G, Opdyke B, Schmidt DC, Weil F (eds) MoDELS, Lecture Notes Comput Sci, vol 4735. Springer, Berlin, pp 559–573
3. Benveniste A, Caspi P, Edwards SA, Halbwachs N, Le Guernic P, de Simone R (2003) The synchronous languages twelve years later. Proc IEEE 91(1)
4. Lee EA, Sangiovanni-Vincentelli AL (1998) A framework for comparing models of computation. IEEE Trans Comput Aided Des Integr Circuits Syst 17(12):1217–1229
5. Mallet F, André C (2008) UML/MARTE CCSL, signal and petri nets. Research Report 6545, INRIA. <https://hal.inria.fr/inria-00283077>
6. Object Management Group (2007) Unified Modeling Language, Superstructure. Version 2.1.2 formal/2007-11-02
7. Object Management Group (2008) UML Profile for MARTE, beta 2. OMG document number: ptc/08-07-09
8. OMG (2008) Systems Modeling Language (SysML) Specification 1.1. OMG document number: ptc/08-05-17