Combining CCSL and Esterel to specify and verify time requirements

Charles André, Frédéric Mallet

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Combining CCSL and Esterel to specify and verify time requirements

Charles André — Frédéric Mallet

N° 6839

February 2009

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6839--FR+ENG

Combining CCSL and Esterel to specify and verify time requirements

Charles André* , Frédéric Mallet*

Thème COM — Systèmes communicants
Projet Aoste

Rapport de recherche n° 6839 — February 2009 — 30 pages

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

This paper introduces the formal semantics of CCSL clock constraints and proposes a process to use CCSL both as a high-level specification language for UML models and as a golden model to verify the conformance of implementations.

A digital filtering video application is used as a running example to support the discussion. The application is first formally specified with CCSL and the specification is refined based on feedback from the CCSL-dedicated simulator. In a second phase, an Esterel program of the application is considered. This program is instrumented with observers derived from the CCSL specification. Esterel Studio formal verification facilities are then used to check the conformity of the Esterel implementation with the CCSL specification. A specific library of Esterel observers has been built for this purpose.

Key-words: Time Model, MARTE, Synchronous languages, Esterel, SyncCharts

Submitted to LCTES'09

TimeSquare is released as part of the platform OpenEmbeDD, a Model Driven Engineering open-source platform for Real-Time and Embedded systems (`http://www.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

CCSL et Esterel pour combiner et vérifier des propriétés de temps

Résumé : Le profil UML pour la modélisation et l'analyse de systèmes temps réel et embarqués (MARTE) a été récemment adopté par l'OMG. Son modèle de temps étend le paquetage SimpleTime de UML2 qui est à la fois simple et informel. MARTE ajoute une grande variété d'éléments requis pour modéliser les systèmes temps réel et embarqués, et en particulier la prise en compte du temps logique et chronométrique, discret ou dense. La spécifiaction OMG de MARTE propose un modèle de causalité temporel inspiré des modèles de temps de la théorie de la concurrence et propose un nouveau langage de spécification de contraintes temporelles appelé specification introduces a Time Structure inspired (CCSL - clock constraint specification language)

Ce rapport présente la sémantique formelle des contraintes d'horloge de CCSL et propose un processus pour utiliser CCSL à la fois comme langage abstrait de spécification de modèles UML et comme modèle de référence pour vérifier la conformité d'implantations candidates.

Un application de filtrage numérique d'un flux vidéo est utilisée tout au long du rapport pour illustrer le propos. L'application est d'abord spécifiée avec CCSL, puis rafinée par retro-ingénierie en utilisant les retours fournis par un simulateur dédié à CCSL. Dans une deuxième phase, un programme Esterel est considéré comme implantation possible de la spécification. Ce programme est instrumenté avec des observateurs dérivés de la spécification CCSL. L'environnement de vérification formelle Esterel Studio permet alors de garantir la conformité de l'implantation Esterel avec la spécification CCSL. Une bibliothèque d'observateurs spécifiques à CCSL a été construite à cette fin.

Mots-clés : modèle de temps, MARTE, langages synchrones, Esterel, SyncCharts

1 Introduction

Synchronous languages [12, 5] are well-suited to formal specification and analysis of reactive system behavior. They are even more relevant with safety-critical applications where lives may be at stake. However, to cover a complete design flow from system-level specification to implementation, synchronous languages need to interoperate with other, more general, specification languages. One of the candidates is the Unified Modeling Language (UML) [20] associated with SysML, the UML profile for systems engineering [28, 21]. This is very tempting since synchronous languages internal formats rely on state machines or data flow diagrams both very close to UML constructs state machines and activities. Moreover, SyncCharts [1] are a synchronous, formally well-founded, extension of UML state machines and are mathematically equivalent to Esterel [7], one of the three major synchronous languages. As for SysML, it adds two constructs most important for specification: requirements and constraint blocks.

There have been attempts to bridge the gap between UML and synchronous languages. Some [14] choose to import UML diagrams into Scade, a synchronous environment that combines safe state machines (a restriction of SyncCharts) together with block diagrams, the semantics of which is based on Lustre. Others [4] prefer to define an operational semantics of UML constructs with a synchronous language, like Signal [6]. In both cases, the semantics remains outside the UML and within proprietary tools. Other tools, from the same domain, would interpret the same models with a completely different semantics, not necessarily compatible. Therefore, it is impossible to exchange diagrams between tools, not only because of syntactical matters but also for semantic reasons. Different environments are then competing rather than being complementary. To provide full interoperability between tools of the embedded domain, the UML absolutely requires a timed causality model. The UML Profile for Modeling and Analysis of Real-Time and Embedded systems [27] (MARTE), which is currently in its finalization process, has introduced a time model [3] with that purpose. This time model comes with a companion language, called Clock Constraint Specification Language (CCSL) and defined in one annex of the MARTE specification. CCSL is advertised as a pivot language to make explicit the interactions between different models of computations, like Ptolemy directors [11]. It offers a rich set of constructs to specify time requirements and constraints.

This paper introduces the formal semantics of a fundamental subset of CCSL and focuses on possible connections with Esterel programs. MARTE Time model is briefly introduced in Section 2. In Section 3, we then describe the system used as a running example, a digital filtering video application. Section 4 gives the semantics of the selected CCSL clock constraints, which is our first contribution. Section 5 uses CCSL to specify the behavior of the example. The specification is simulated with TimeSquare, an environment dedicated to MARTE Time Specification and CCSL analysis. We then rely on Esterel Studio formal verification facilities to check the conformance of a candidate Esterel/SyncChart implementation with its specification. This is our second contribution. Finally, we further comment on some related works in Section 6.

2 Logical Time

MARTE Time model deals with both discrete and dense time. In MARTE, a *clock* gives access to a *time structure*. A clock can be either *chronometric* or *logical*. The former is related to "physical time" while the latter is not. This paper focuses on the structural relations between clocks and these relations do not differentiate between chronometric and logical clocks. However, some relations only apply to discrete clocks (logical or chronometric) and others apply to both discrete and dense clocks. Dense clocks considered in MARTE are necessarily chronometric. Logical clocks refer to discrete-time logical clocks and represent *logical time*. Logical time is the time model in use in synchronous languages. Logical clocks as originally defined by Lamport [13] are a special case where the labeling function is an increasing monotonic function.

On the example, we mainly exhibit causal/logical problems and we do not discuss much the chronometric aspects.

2.1 Clock

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

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

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

2.2 Time structure

A *Time Structure* is a pair $\langle C, \preccurlyeq \rangle$ where C is a set of clocks, $\preccurlyeq$ is a binary relation on $\bigcup_{c \in C} \mathcal{I}_c$, named *precedence*. $\preccurlyeq$ is reflexive and transitive. From $\preccurlyeq$ we derive four new relations: *Coincidence* ($\equiv \triangleq \preccurlyeq \cap \succcurlyeq$), *Strict precedence* ($\prec \triangleq \preccurlyeq \setminus \equiv$), *Independence* ($\| \triangleq \overline{\preccurlyeq \cup \succcurlyeq}$), and *Exclusion* ($\# \triangleq \prec \cup \succ$). The graphical representation of instant relations is given in Figure 1.

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

CCSL defines a concrete syntax to specify instant relations or more generally clock relations, which represent infinitely many instant relations. The next section introduces our

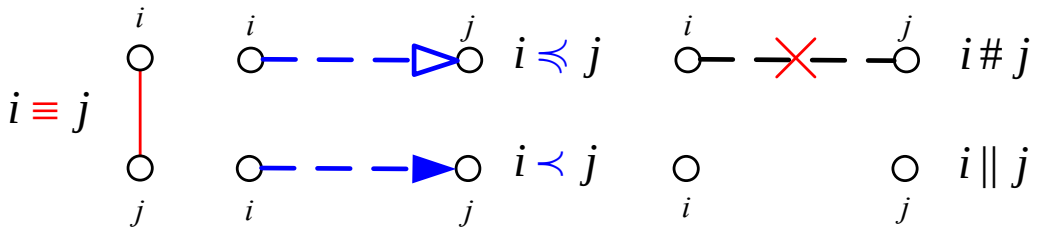

Figure 1: Graphical representation of instant relations

example and highlight clock relations required by the specification. Section 4 defines the semantics of some CCSL clock relations/constraints.

3 Example: Digital Filter

This section introduces the example selected to illustrate our proposal: a simple digital image filtering (DF) application. This example is borrowed from the "*Getting Started Manual*" of Esterel Studio and was designed as a tutorial on its modeling capabilities.

3.1 Informal Specification

DF is used in a video system. It reads image pixels from a memory, filters them and sends output pixels out to a display device.

One image is composed of LPI lines per image, each line consists of PPL pixels per line. The pixels are stored in words. A word contains PPW pixels per word, a line WPL words per line $\left(WPL = \lceil PPL/PPW \rceil\right)$. The pixel transformation (digital filtering) is defined by a dot product:

$$y[k] = \sum_{j=-L}^{j=+L} c[j] * x[k-j] \tag{1}$$

where k is a natural number, index of pixels in a line, y is an array of output pixels, x is an array of input pixels, c is an array of $2L+1$ constant coefficients.

Figure 2 shows DF as a component with four signal ports. The input port InWord conveys WORD, the output port OutPixel conveys PIXEL. The two other output ports (Ready and EndOfLine) are *pure signals*, that is, they do not carry values and are used for signaling some event occurrences.

A rough specification of the behavior of DF is as follows. DF requests a new incoming word by asserting Ready ①. In response, an external memory sends back the next word of the image (signal InWord). OutPixel are sequentially issued after receiving InWord ② and performing the filtering. EndOfLine is asserted each time the last pixel of a line is emitted

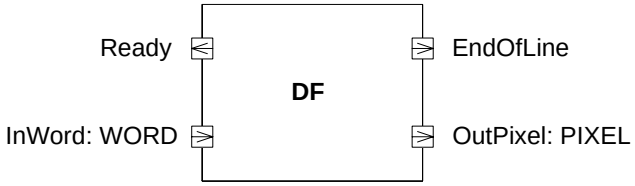

Figure 2: DF component

③. The circled numbers (①, ②, ③), in Figure 3, refer to instant relations and are discussed in the next subsections.

3.2 Modeling with logical clocks

Each *event* of the system is modeled as a logical clock and the specification is imposed by applying constraints to these clocks. An event can be a signal receipt (e.g., $inWord$), a signal emission (e.g., $outPixel$), or the presence of a pure signal (e.g., $ready$, $endOfLine$). A logical clock ticks each time the associated event occurs. For convenience, we denote the clock associated with a signal by the name of the signal in italic and with an initial lower case letter.

Precedence arrows and coincidence edges in Figure 3 represent some instant relations implied by the specification. Precedence relation ① states that for each request (each tick of $ready$) a new word must be released ($inWord$ must tick). Precedence ② expresses that each received word produces four output pixels. The rounded-corner rectangle makes it explicit that a word gives rise to four output pixels exactly. Coincidence ③ says (for the unlikely case of a 8-pixel line) that the first tick of $endOfLine$ is coincident with the 8^{th} outpixel.

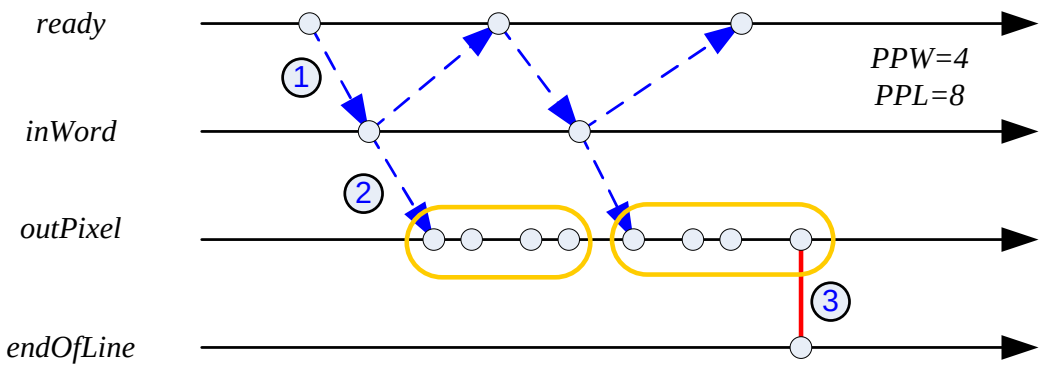

Figure 3: Some time constraints on the DF behavior

Of course, the instant relations represented in Figure 3 hold for all lines of the image and all parameter settings. Instead of expressing instant relations on an instant-pair base, it is more convenient to apply constraints on clocks directly. Adequate clock constraints that implement the specification correctly are informally described in the following subsection.

3.3 Clock constraints

Clock constraints are a generic way to define various aspects of a specification. They may derive from the algorithm itself or from performance requirements, and also from the data structure used or the operating mode. We have identified here four primary constraints covering several of these aspects.

(Cstr 1) The specified *protocol* implies that each request (*ready*) is followed by a new word ($inWord$) and that no new request is sent before the preceding request has been acknowledged. This is an alternation constraint where, *ready* alternates with $inWord$. In terms of clocks, an instant of *ready* precedes an instant of $inWord$, which precedes the next instant of *ready*, and so on.

(Cstr 2) Because of the chosen *data structure*, input pixels are packed within words of length PPW, whereas output pixels are released individually. The algorithm imposes that the number of pixels is preserved. A by-packet precedence relation denotes such a fact. Each tick of $inWord$ precedes a group of PPW consecutive ticks of $outPixel$.

(Cstr 3) $endOfLine$ ticks every PPL ticks of $outPixel$. This constraint directly reflects the semantics of the end of line.

(Cstr 4) Additional non-functional constraints must be set to impose readiness and reduce communication buffers. Such a constraint should avoid to delay unnecessarily the processing of received input words and gives rise to further precedence constraints between $outPixel$ and $inWord$ (see subsection 5.1).

The pixel transformation being a dot product (Eq. 1), each output pixel depends on $2L + 1$ consecutive input pixels. CCSL only deals with the structural relations, therefore the actual transformation is not relevant and the data dependencies (between inputs and ouputs of the pixel transformation) are implemented by several precedence constraints that are surely much stronger than (Cstr 2). Figure 4 shows these precedence constraints for one single image row in the simplistic case where $PPL = 8$, and $L = 2$. A more general characterization is given in Annex B.

As always in pipelined specifications, three phases must be considered for each line. The prolog, when filling the pipeline, the kernel, when the pipeline is in a steady state, the epilog, when draining the pipeline.

Figure 4 (A) shows the beginning of the line processing (prolog) where padding pixels must be assumed to apply the dot product. OutPixel[0] depends on InPixel[-2] ... InPixel[2]. A default value is given to padding pixels InPixel[-2] and InPixel[-1].

Figure 4 (B) illustrates the steady phase (kernel) where the computation of each output pixel depends on five inputs pixels. OutPixel[2] depends on InPixel[0] ... InPixel[4]. Because of the implicit ordering on input pixels (InPixel[j] precedes InPixel[k], for any $j < k$), only one precedence is required: InPixel[4] must precede OutPixel[2].

Figure 4 (C) represents the ending of the 8-pixel line processing (epilog). OutPixel[7] depends on InPixel[5] ... InPixel[9]. InPixel[8] and InPixel[9] are also padding pixels for which a default value is assumed.

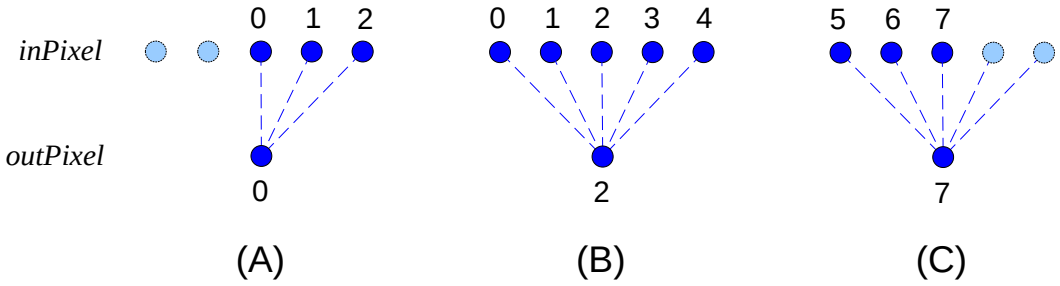

Figure 4: Pixel dependency

Since signal InPixel is not part of the interface, the precedence relations between InPixel and OutPixel have to be expressed as precedence relations between InWord and OutPixel. This is done in subsection 5.1.

Overall, the specification mixes synchronous (Cstr 3) and asynchronous (Cstr 1, 2, 4) constraints and involves functional and non-functional aspects. CCSL has been designed to address such specifications. It is heavily inspired by the Tagged Signal Model [15], 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 and more generally polychronous/multiclock languages well-suited to specify Globally Asynchronous and Locally Synchronous (GALS) systems. TimeSquare is our analysis framework that provides a support for expressing, simulating and formally analyzing CCSL constraints.

4 Clock constraints in CCSL

Relationships introduced in Section 2.2 are binary relations relating pairs of instants. Specifying a full time structure using only these elementary relationships is not realistic, all the more so since a clock usually has an infinite number of instants therefore forbidding an enumerative specification of instant constraints. Instead of defining individual instant pairings, a clock constraint specifies generic associations between (infinitely) many instants of the constrained clocks.

In this section we define the more general clock constraints and we introduce some usual constraints, derived from the basic ones. The clock constraints are classified in three main

categories: 1) coincidence-based constraints, 2) precedence-based constraints, and 3) mixed constraints.

4.1 Coincidence-based clock constraints

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

Sub-Clocking is the most basic coincidence-based clock constraint relationship. B isSubClockOf A, where A and B are clocks, imposes B to be a sub-clock of A. Intuitively, this means that each instant in B is coincident with one instant in A, and this without introducing causality loop (Figure 5). More formally:

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

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

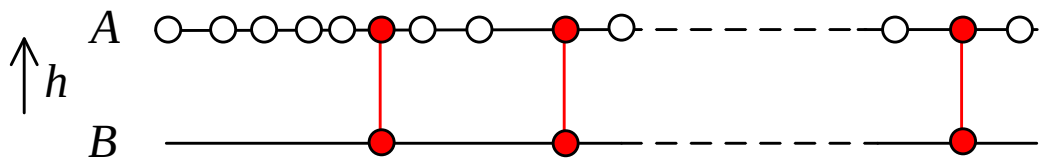

Figure 5: Coincidence-based clock constraint

In what follows, this constraint is denoted as $B \boxed{\subset} A$ that reads "B is a sub-clock of A" or equivalently "A is a super-clock of B".

4.2 Derived coincidence-based clock constraints

h can be specified in many different ways.

Equality $A \boxed{=} B$ is the symmetric relation that makes the two clock A and B "synchronous": h is a bijection and the instants of the two clocks are pair-wise coincident. It is strictly equivalent to $B \boxed{=} A$.

Other coincidence-based *clock expressions* allow the creation of a new clock, sub-clock of a given clock (denoted *new clock* $\triangleq$ *defining expression*). Four such clock expressions are presented hereafter.

Restriction $B \triangleq A$ restrictedTo P where A is a given clock, B is a new clock, and P is a predicate on $\mathcal{I}_A \times \mathcal{I}_B$, such that

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

Discretization $B \triangleq A$ discretizedBy r, where A is a dense-time clock, r is a real number, and B a discrete-time clock. Discretization is a special restriction the predicate of which takes account of the labeling function $\lambda_A : \mathcal{I}_A \rightarrow \mathbb{R}$ such that

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

This is the main operator to discretize dense clocks and therefore to discretize chronometric clocks.

Filtering $B \triangleq A$ filteredBy w, where A and B are discrete-time clocks, and w is a binary word. For filtering, the associated predicate is such that

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

where $w \uparrow k$ is the index of the k^{th} 1 in w. The use of infinite k-periodic binary words in this kind of context has previously been studied in N-Synchronous Kahn networks [9]. This constraint is frequently used in clock constraint specifications and is denoted $A \blacktriangledown w$ in this paper. It allows the selection of a subset of instants, on which other constraints can then be enforced.

In what follows, a (periodic) binary word is denoted as $w = u(v)^\omega$, where u is called the *transient* part of w and v its *periodic* part. The power ω means that the periodic part is repeated an unbounded number of times. So, $u(v)^\omega$ denotes $u.v.v.\cdots.v.\cdots$.

Periodicity Defining the periodicity of discrete clocks consists in using a binary word with a single 1 in the periodic part. A isPeriodicOn B period p offset d defines a periodic clock A. The same clock can be built with a filtering $A \triangleq B \blacktriangledown 0^d.(1.0^{p-1})^\omega$. In this expression, for any bit b, b^0 stands for the empty binary word. Note that this is a very general definition of periodicity that does not require B to be chronometric contrary to the usual definition.

4.3 Precedence-based clock constraint

Precedence-based clock constraints are easy to specify with concurrent models like Petri nets but are not usual in synchronous languages. A discussion on main differences with Time Petri nets can be found in some of our previous work [17].

Precedence The clock constraint Precedence consists in applying infinitely many precedence instant relations. Two forms can be distinguished: the strict precedence A strictly precedes B, and the non strict precedence A precedes B. A and B are clocks. Intuitively, this means that each instant in B (immediately) follows one instant in A (Fig. 6). More formally:

For A (strictly) precedes B, $\exists h : \mathcal{I}_B \rightarrow \mathcal{I}_A$ such that

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

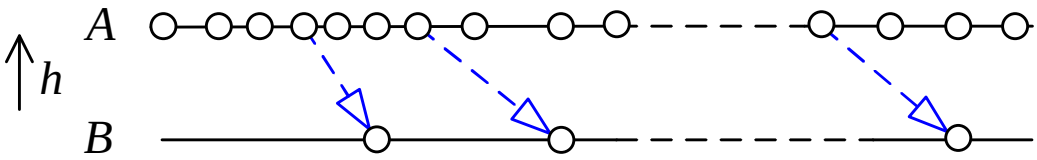

Figure 6: Precedence-based clock constraint

4.4 Derived precedence-based clock constraints

When A and B are discrete-time clocks, the precedence relationship gives rise to more specific constraints. Three often used precedence constraints are discussed here.

Discrete precedence A precedes B (denoted $A \boxed{\preccurlyeq} B$) if $(\forall i \in \mathcal{I}_B)(k = \mathrm{idx}_B(i)) \implies A[k] \preccurlyeq B[k]$. There also exists a strict form (denoted $A \boxed{\prec} B$) of this constraint: A strictly precedes B if $(\forall i \in \mathcal{I}_B)(k = \mathrm{idx}_B(i)) \implies A[k] \prec B[k]$.

Alternation A alternatesWith B (denoted $A \boxed{\sim} B$) is defined by:

$$\left(A \boxed{\prec} B\right) \wedge \left(B \boxed{\prec} A'\right), \text{where } A' \triangleq A \blacktriangledown 0.1^{\omega}$$

The following specification is equivalent and uses instant relations instead of clock relations, $(\forall i \in \mathcal{I}_A)(k = \text{idx}_A(i)) \implies (A[k] \prec B[k] \prec A[k+1])$.

Synchronization A synchronizesWith B (denoted $A \boxed{\bowtie} B$) is such that

$$\left(A \boxed{\prec} B'\right) \wedge \left(B \boxed{\prec} A'\right)$$
$$\text{where } A' \triangleq A \blacktriangledown 0.1^{\omega}, \text{and } B' \triangleq B \blacktriangledown 0.1^{\omega}$$

This constraint can also be expressed using instant relations:

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

Precedences used in the definition of Alternation and Synchronization can be non-strict precedences, thus there exist four different variants of these clock relations. Another extension considers instants by "packets". For instance, A by α strictly precedes B by β (denoted $A/\alpha \boxed{\prec} B/\beta$) is a short notation for

$$\left(A_f \boxed{\prec} B_s\right)$$
$$\text{where } A_f \triangleq A \blacktriangledown \left(0^{\alpha-1}.1\right)^{\omega}, \text{and } B_s \triangleq B \blacktriangledown \left(1.0^{\beta-1}\right)^{\omega}$$

4.5 Mixed constraints

Mixed constraints combine coincidences and precedences. They are used to synchronize clock domains in globally asynchronous and locally synchronous models.

Sampling The commonest constraint of this kind is the *Sampling* constraint. $C \triangleq A$ sampledOn B, where B and C are discrete-time clocks, defines C as a sub-clock of B that ticks only after a tick of A (Fig. 7).

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

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

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

5 Design and analysis of the example

This section illustrates the use of CCSL constraints to specify and analyze the digital image filtering example. The first subsection focuses on the specification. The second one briefly

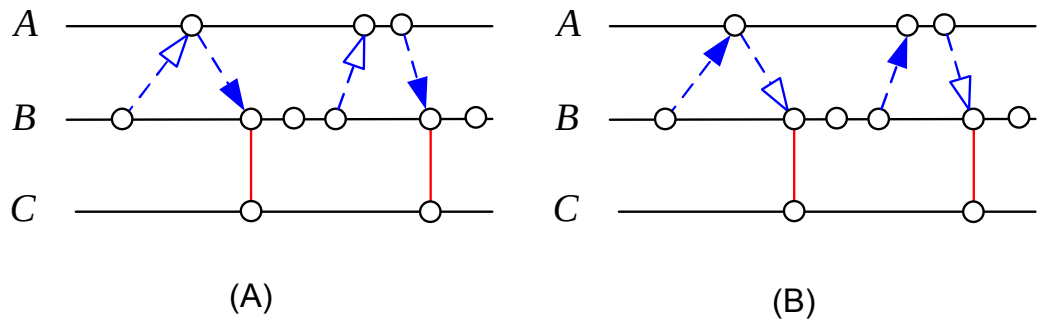

Figure 7: Sampling constraints

introduces the TimeSquare simulation engine. TimeSquare is the analysis environment specifically built to specify and analyze CCSL constraints. The third one contains a short commented description of the Esterel/SyncCharts programs of the Digital Image Filtering. The last subsection elaborates on the combined usage of TimeSquare and Esterel Studio to support formal verification of an Esterel implementation against our CCSL specification.

5.1 CCSL specification

CCSL constraints build on UML semantic variation points to extend UML models with a specification of time requirements. The following five CCSL relations are the formal counterpart to the informal specification given in Section 3. Circled numbers after clock constraints are not part of CCSL code; There are just comments referring to the requirement numbers in Figure 3 and introduced in subsection 3.3. CCSL relations can either be fully textual or be symbolically represented. The textual representation, although very verbose attempts to be close to some plain English specification. The symbolic representation is much more compact, this is why we have selected the latter here.

$$ready \boxed{\sim} inWord \quad ①$$

$$inWord \boxed{\prec} \left(outPixel/4\right) \quad ②$$

$$endOfLine \boxed{=} outPixel \blacktriangledown (0^7.1)^\omega \quad ③$$

$$\left(inWord \blacktriangledown (0.1)^\omega\right) \boxed{\prec} \left(outPixel \blacktriangledown 0^2.\left(1.0^7\right)^\omega\right) \quad ④$$

$$\left(outPixel \blacktriangledown 0.\left(1.0^7\right)^\omega\right) \boxed{\prec} \left(inWord \blacktriangledown (0.1)^\omega\right) \quad ⑤$$

The first three relations ①–③ are immediate representations of Cstr 1, 2 and 3, described in subsection 3.3, when there are four pixels per words ($PPW = 4$) and eight pixels per line ($PPL = 8$). Alternation ($\boxed{\sim}$) is often used to model asynchronous protocols, where a request alternates with an acknowledgement. Here *ready* is the request and *inWord* the

acknowledgement. Packet-based precedence (as in ②) denotes the way pixels are gathered within words. In ③, end of line periodically occurs when eight output pixels have been emitted. The periodic pattern (in ③-⑤) models regular data flows. Here, each pixel line has the same length, and the same transformation periodically applies to each line. Eq. ③ is a good illustration of the clock relation filteredBy (▼). With the binary word $(0^7.1)^\omega = (00000001)^\omega$, the seven first output pixels are ignored and the first instant of $endOfLine$ is synchronous with the 8^{th} output pixel (Fig. 3). The following seven output pixels are also ignored and so on.

The last two relations are a bit trickier. They rely on the periodic pattern combined with a precedence relation. Relation ④ is directly implied by the dot product. As discussed in subsection 3.3, it relates the input words (clock $inWord$) to the output pixels (clock $outPixel$) since these two clocks are part of the interface. Figure 8 illustrates this constraint and emphasizes on the relation between the input pixels and the input words by circling pixels belonging to the same word. The general case for whatever pixels per word and per line, is further discussed in Annex B. Relation ⑤ is a back-pressure constraint to guarantee that output pixels are delivered fast enough so the communication buffer contains at most two words.

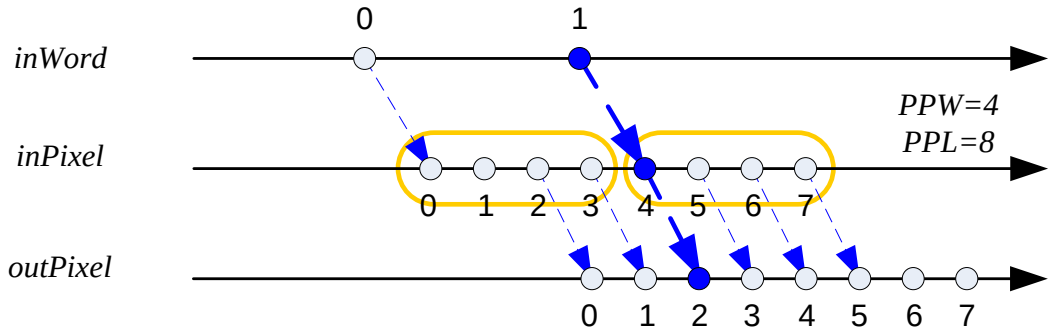

Figure 8: Additional precedences

5.2 Running simulations

From this specification, TimeSquare can run simulations. For each constraint, a set of Boolean equations determines disabled clocks, which must not tick, and enabled clocks that may tick. From there, the simulation policy selects clocks that are actually fired. At each step, depending on the fired clocks, CCSL expressions are rewritten according to the operational semantics. Some SOS rewriting rules are given in Annex A.3.

Several simulation policies are offered. The *random policy* randomly chooses one possible solution amongst the set of solutions. The *minimum policy* chooses a consistent solution where the number of firing clocks is minimal. The *maximum policy* chooses a consistent solution where the number of firing clocks is maximal. Boolean equations associated with

each kernel CCSL constraint are given in Annex A.2. In TimeSquare, Boolean equations are encoded using JavaBDD, a generic Java API that supports implementation of various Binary Decision Diagrams (BDD) libraries. The simulation produces an output in the Value Change Dump (VCD) format, which is a normative trace format released as part of the IEEE standard for Verilog-HDL and which is often used in Electronic Design Automation (EDA) tools. The VCD file is annotated with proprietary comments (pragmas) so the TimeSquare VCD viewer can display feedback on causality relations that leads to this specific run. Tracability is indeed very important.

Figure 9 illustrates a correct run for the given specification generated by TimeSquare simulation engine. Note that alternative runs may also be correct since the simulation engine generates one possible solution.

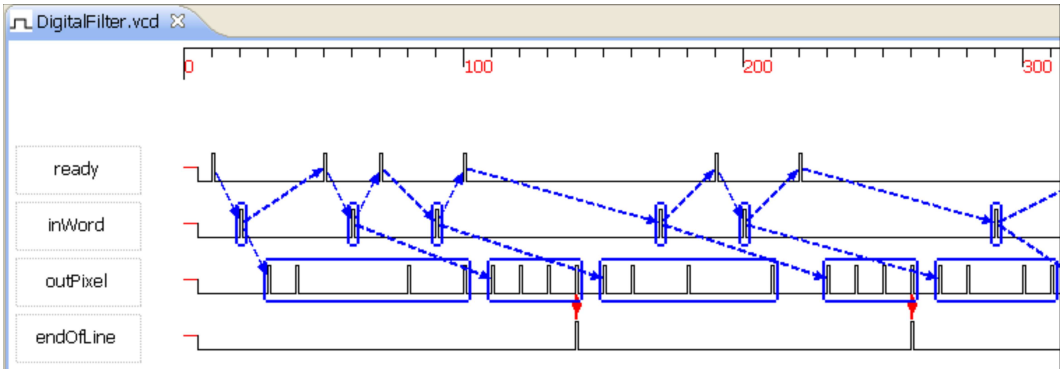

Figure 9: One acceptable solution generated by TimeSquare

TimeSquare VCD viewer displays instant relations when requested. Precedence relations are displayed as dashed arrows. Coincidence relations are shown as vertical lines with a diamond on the side of the super clock. When packet-based constraints (as in ①) are used, the packets are depicted as rounded-corner rectangles surrounding the related clock ticks.

Even though simulation can help to discover some specification inconsistencies, it only considers one possible solution at a time. It must be combined with exhaustive analysis for corner bug detection and formal verification of safety requirements. This is addressed in the next two subsections.

5.3 Esterel/SyncCharts modeling

The full syntax and semantics of these programming languages are beyond the scope of this paper. The combined usage of Esterel and SyncCharts is richly illustrated in a text-book [29]. The foundations of the Esterel language are given in a paper [7], SyncCharts are introduced in another paper [1], and their compilation is explained in a book [23]. Esterel and SyncCharts semantics are fully compatible, and any SyncChart can be translated into

a semantically equivalent Esterel code. As synchronous reactive languages, they have a cycle-based semantics: a run is a sequence of non-overlapping reactions consisting of input reading, computation, and output writing.

In this presentation we just comment some design choices present in the solution proposed in the previously mentioned Esterel-Studio's Getting Started manual. The DF program consists of two parts: the Feeder, written in SyncCharts, and the Filter, written in Esterel. The Feeder specifies the protocol (request/acknowledge) for getting input words, the unpacking of input words into input pixels, the feeding of the Filter with input and padding pixels, and the detection of the end of line. The Filter is a pipelined version of a sliding-window filtering, not detailed here. The behavior of the Feeder is specified by the SyncChart in Figure 10. SyncCharts are akin to StateCharts (hierarchical and concurrent composition of state machines) with a richer variety of transitions that express different kinds of *preemptions* (strong, weak, and normal termination).

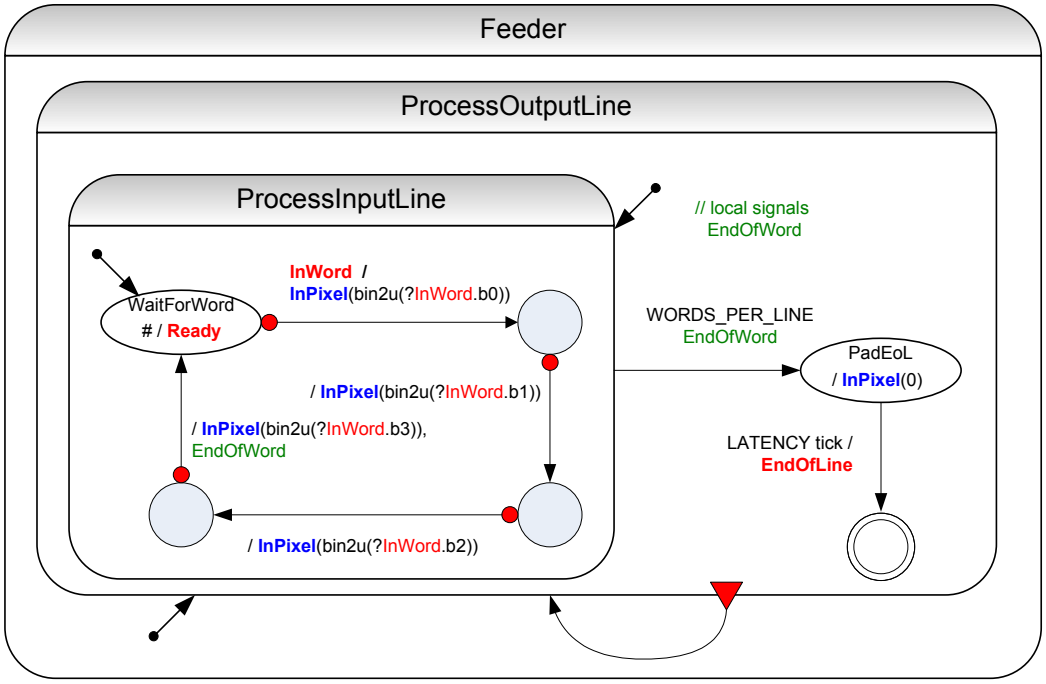

Figure 10: Behavior of the Feeder

Macrostate ProcessInputLine manages the request/acknowledge protocol and the decomposition of an input word. The request (signal Ready) is sustained up to the receipt of the acknowledge (signal InWord). This behavior is specified by state WaitForWord, which emits signal Ready at each instant, until signal InWord occurs. This occurrence triggers the outgo-

ing transition from state WaitForWord (strong preemption). When an input word has been completely decomposed, a local signal (EndOfWord) is emitted. The process is cyclically repeated until the last word of the line has been processed (transition exiting macrostate ProcessInputLine). This is a weak preemption, letting macrostate ProcessInputLine react before its effective preemption. State PadEoL provides $LATENCY$ padding pixels to the Filter to drain its pipeline. After emitting the last padding pixel, a normal termination occurs (self transition to ProcessOutputLine) and the same process starts again with a new input line processing.

5.4 Checking an Esterel program against a CCSL specification

5.4.1 Formal verification tools in Esterel Studio

Since any SyncChart can be transformed into an equivalent Esterel code [2, 29], from now on the expression "Esterel or SyncCharts program" is abbreviated as "Esterel program". The Esterel compiler is part of a comprehensive development environment named Esterel Studio. This environment provides compilation, simulation, coverage, verification and code generation facilities. In this subsection we consider only the fourth one. Formal verification of Esterel programs relies on two complementary technologies: 1) Symbolic model checking based on a BDD technology, 2) Bounded and Full model checking based on SAT-technology. *Bounded Model Checking* (BMC) is efficient in searching for bugs in design and property specifications. Since BMC can only *falsify* properties, it cannot be used to prove a property correct. On the contrary, *Full Model Checking* (FMC) can prove that a property holds, but the process may take a great amount of time. The FMC makes its best to combine SAT-solver with induction [25] and improved strategy combining interpolation and SAT-based model checking [18]. Symbolic Model Checking (SMC) can be used both to falsify and to prove properties. The drawback of this BDD-based model checking is the possibility to run out of memory and thus being inconclusive.

A property to check is directly expressed in Esterel either as an *assertion* or as an *observer*. An assertion may represent an *assumption* about the execution environment of the program to check. An assertion also allows implementing parts of its intended behavior as executable and verifiable predicates, into the design code. An observer is a special program unit, not part of the design, and used in property checking. It continuously observes input and output signals of the program and detects possible property violations. Used in combination with model checking, observers are a powerful means to find bugs and formally establish properties. If a violation occurs, the model checker generates a simulation trace leading to this violation, thus exhibiting a counter-example of the checked property. Note that the observers are non-intrusive: they do not alter the behavior of the tested Esterel program.

5.4.2 CCSL clock constraints in Esterel

Clock constraints are predicates that can be expressed as Esterel assertions or observers. The mathematical semantics of CCSL, partially given in Annex A, enables these translations. Note that, because an observer detects possible violations, an Esterel observer for a given clock relation actually programs the *negation* of the logical expression associated with this relation. To ease clock constraint checking, we have developed a library of generic Esterel modules called CcslStrlLib. This library provides one observer for each clock relation, one Esterel module for each clock expression, and special modules called *Adaptors* that bind program signals to CCSL clocks.

The verification of the clock constraints specifications is performed without any change in the Esterel program. The user has only to build one Esterel observer for each CCSL clock relation, and put these observers in the Observer directory of the Esterel Studio project.

Figure 11 shows the observer of Cstr 3. The V3 signal is emitted if and only if Cstr 3 is violated. All the blocks in the observer module are instantiations of generic modules from CcslStrlLib. The upper Adaptor takes the program valued signal OutPixel and generates the pure signal S_OutPixel which represents the CCSL clock *outPixel*. This clock ticks whenever OutPixel is present. The lower adaptor is a simple repeater: signal S_EndOfLine is emitted whenever the program pure signal EndOfLine is present. S_EndOfLine represents CCSL clock *endOfLine*. In Cstr 3, the clock relation is Equality ($\boxed{=}$), so we instantiate a CCSL Equality observer, which may emit the violation signal V3. The left-hand side of the relation is a clock expression involving a filtering. This expression is represented in the observer by an instantiation of the generic module for periodic expressions, with the actual parameters (period=8, offset=7). The two dashed rounded-corner rectangles are just annotations that highlight the observer structuration.

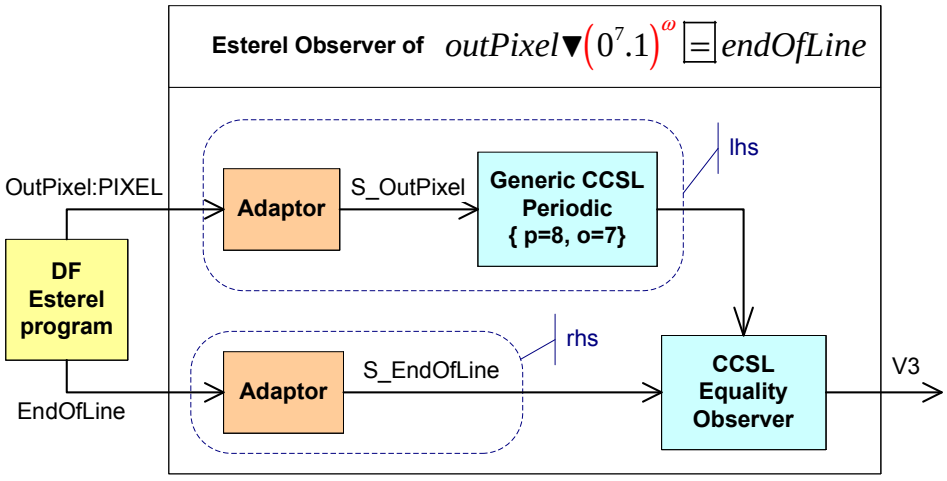

Figure 11: Esterel Observer of Cstr 3

The corresponding Esterel code is trivial for the adaptors:

```
sustain {
  S_OutPixel if OutPixel,
  S_EndOfLine if EndOfLine }
```

The CCSL equality observer is also a simple Esterel code:

```
module CR_equal:
  input A,B;
  output Violation;
  sustain Violation if (A xor B)
end module
```

The signal Violation is emitted whenever the presence status of signal A differs from the one of signal B. This is merely the negation of the logical specification given in Annex A.2, clock relation (equal).

The module "Generic CCSL Periodic" is a bit more complex:

```
1  module CE_periodic:
2    input Super; // the superclock
3    output Output; // the generated subclock
4    constant OFFSET: unsigned; // parameters
5    constant PERIOD: unsigned; // assumed > 0
6    await immediate Super;
7    if static OFFSET > 0 then
8      await OFFSET Super // transient
9    else
10     emit Sub;
11     await PERIOD Super
12   end if;
13   emit Sub;
14   every PERIOD Super do  // period
15     emit Sub
16   end every
17 end module
```

The values of the two constants OFFSET and PERIOD are set at the *compile-time* module instantiation. The **static** keyword in line 7 is exploited by the compiler to instantiate either the then-statement if OFFSET is positive or the else-statement if OFFSET is 0. The reactive behavior relies on the language capability to wait for a given number of occurrences of a signal (**await** OFFSET Super) and possibly in an infinite loop (**every** PERIOD Super **do** ... **end**).

5.4.3 Design verification

Generally, verification starts with a fast search for bugs or property violations. This is done by BMC. In the application at hand, a violation is detected when checking Cstr 1 (Figure 12).

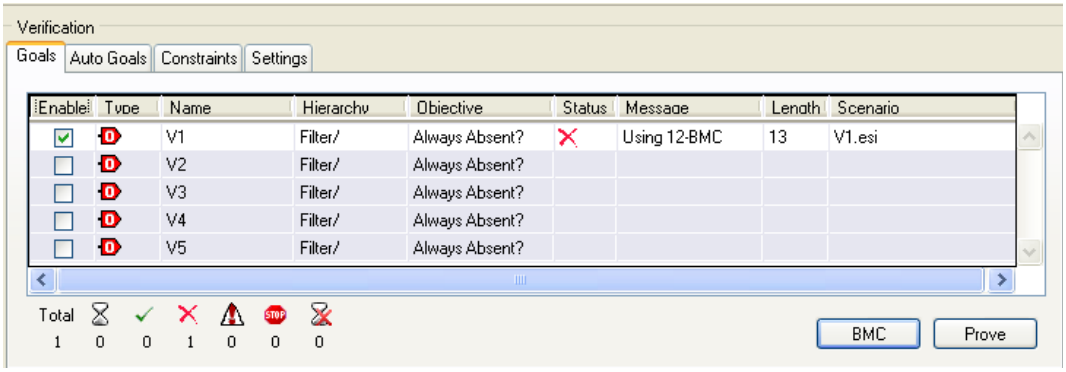

Figure 12: Violation detection of Cstr 1

The model checker generates a counter-example sequence of 13 reactions (file V1.esi). This counter-example scenario can be displayed as a waveform (see screen copy in Figure 13). Signal OutWord is emitted by the memory, which contains the image, and therefore is the same as signal InWord. This trace confirms the presence, at instant 12, of a spurious signal Ready. This unexpected emission of Ready is caused by the weak preemption of macrostate ProcessInputLine, which lets state WaitForWord emit Ready before leaving the macrostate.

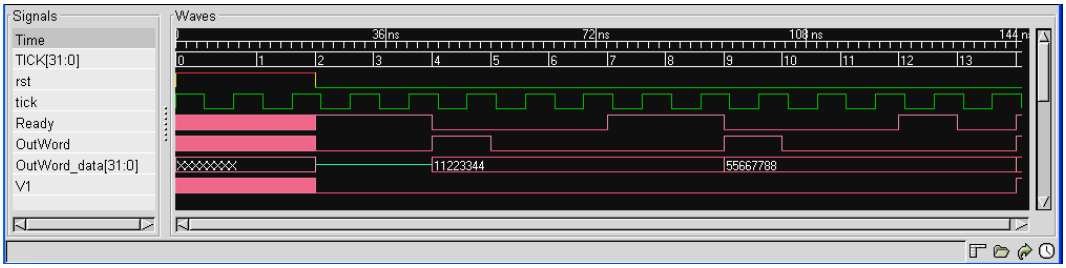

Figure 13: Trace of a violation of Cstr 1

This abnormal behavior is corrected by forbidding the emission of Ready when processing the last input word of a line. This is done by modifying macrostate ProcessOutputLine (Figure 14). A new local signal (EoW) is introduced. It is present only at the end of the processing of the last input word of a line, Ready is conditioned to be emitted only when EoW is absent

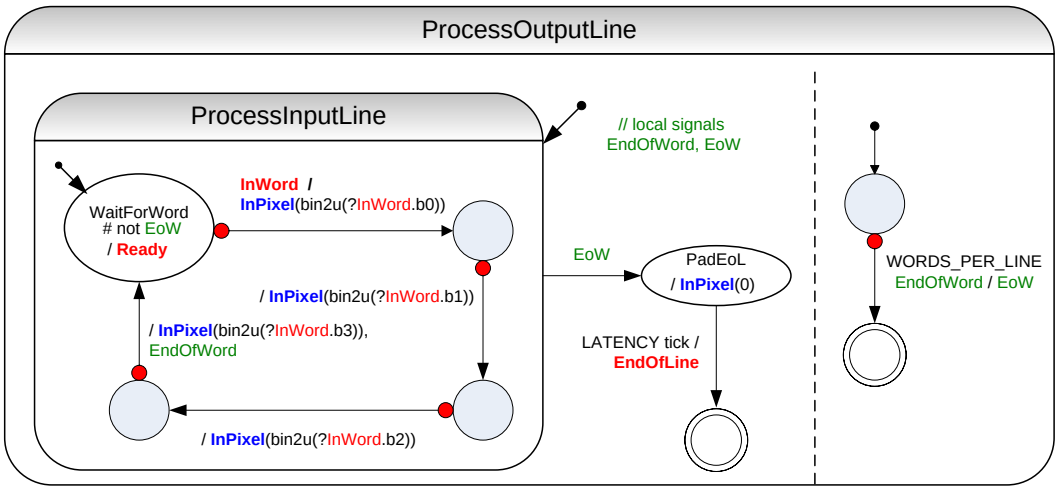

Figure 14: Behavior of the modified Feeder

With the modified program all the CCSL constraints are satisfied. Figure 15 shows the result of applying FMC.

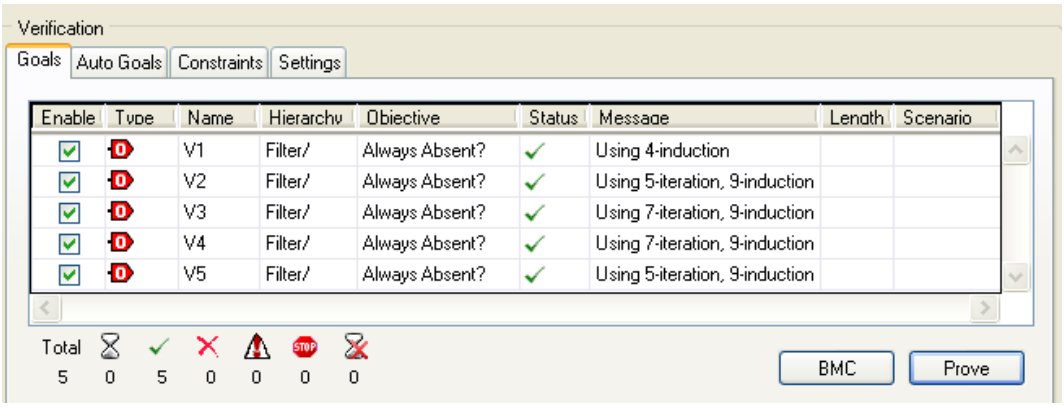

Figure 15: Verification of the modified DF

6 Related Work

In UML, Time is seldom part of the behavioral modeling, which is essentially untimed. By default, events are handled in their arrival order. In UML2, the subpackage Simple Time introduces metaclasses to represent time and duration. This very simple model of

Time explicitely calls for extensions (by an appropriate Profile) to provide both a more sophisticated model and a precise semantics. Several models of Time and Concurrency (outside the scope of UML) have been defined and have inspired our work. We briefly describe them in this section.

There have been several attempts to give a formal semantics to UML constructs. Lots of the existing work focus on behavioral models (activities and interactions) [16, 10, 26] and attempt to give a semantics to UML metaclasses in a transformational way. MARTE Time model includes both structural and behavioral aspects (not the behavior only) and does not focus on one specific diagram. However, we do not consider the whole behavior nor the whole metamodel. Our intent is rather to give a global consistency to timing aspects of a UML model.

MARTE time model, which CCSL relies on, is based on partial ordering of instants. This is close to Petri's work on concurrency theory [22]. Petri's model restricts coincidence to single points in space-time. In CCSL, the coincidence relationship "melts" *a priori* independent points (instants) to reflect design choices and thus is a foundational relation. General Net theory has also influenced this work. The precedence-based relations and especially the clock constraint *synchronization* are inspired from the synchronic distance concept [24].

Petri nets have well-established mathematical foundations and offer rich analysis capabilities. Petri nets support true concurrency and could be used to specify CCSL clock relations (at least some of them). However, it is very difficult to express simultaneity. It is not possible to force two transitions to fire "at the same time". An extension, Time Petri Net [19] adds time intervals to transitions, thus providing a support for simultaneity. Even with that extension, the specification of CCSL constraints is far from straight forward.

MARTE logical time model is also akin to synchronous language time model. Coincidence-based CCSL clock constraints are easily expressed with the language Signal [6]. Signal is a relational language that supports multiclock (polychronous) specifications. A signal is a sequence of values of the same type, which are present at some instants. The set of instants where a signal is present is the clock of the signal. There are two kinds of operators. *Monochronous* operators act only on synchronous signals, *i.e.,* signals that are always present at the same instants. These operators mainly operate on values rather than clocks. *Polychronous* operators act on signals with any clock and their result may have another clock. In MARTE, the Time Structure only refers to the instants and a labeling function can be given to associate values with instants. In this paper, we have focused on the Time Structure ignoring the labeling function (the values). CCSL clocks compare to pure signals (type event in Signal) and CCSL constraints compare to Signal polychronous operators.

A more complete comparison between CCSL, Signal and Time Petri net has been presented in a separate work [17].

ModHel'X [8] is a framework for simulating multi-formalism models. Each model conforms to a specific Model of Computation (MoC). Parts of hierarchical models can be described using different MoCs. The separation is performed by using InterfaceBlocks, which specify the adaptation semantics between the two MoCs. ModHel'X deals with data, control and temporal aspects of models. It relies on imperative OCL to describe an execution

semantics based on delta cycles. Therefore, ModHel'X focuses on simulation and testing aspects and does not cope with formal verification.

7 Conclusion

This paper introduces the formal semantics of UML/MARTE Clock Constraint Specification Language. A simple (but not trivial) example illustrates the use of CCSL in modeling time requirements.

MARTE OMG Specification has proposed a conceptual view of the Time Model with an informal (natural language) semantics and the adequate UML syntax to refer to this Time Model in UML user models. To avoid divergent interpretations a formal semantics is needed, especially for real-time critical systems. MARTE Time Model has also introduced in UML, the notion of logical time, missing in the standard and very useful to digital circuit design. Logical time is a common concept in synchronous languages and Petri nets.

The initial intent of MARTE being to cover both design and analysis, a large set of CCSL constraints have been introduced for convenience on top of a relatively small set of kernel primitives. This paper provides a classification of the constraints based on two fundamental relationships (coincidence and precedence).

This paper presents CCSL as a support for specifying timing requirements on UML models, as well as simulating and analyzing them. The adopted process starts with a UML model. The model is annotated with CCSL constraints by applying the profile MARTE and using its stereotypes in a way not discussed here. The resulting executable specification is assessed and refined by using feedback from TimeSquare simulation. Finally, the implementation is validated through observers generated from the CCSL specification. This process has been applied on an Esterel implementation of a digital filtering application. A specific library of Esterel observers has been developed for verification purpose. Relying on Esterel gives access to its formal verification suite, thus extending TimeSquare capabilities. The same process could also be applied with other implementation languages. Changing the language is mainly a matter of building the library of observers for the target language. Since our observers are written in Esterel, we can also use the Esterel Studio code generation facilities to generate observers in SystemC or in VHDL, thus opening new perspectives.

A Semantics of CCSL

This section discusses the structural operational semantics of CCSL kernel relations and expressions. This is the semantics used in TimeSquare to process valid runs. Its equivalence with the denotational semantics given in Section 4 is not trivial and not addressed here.

A.1 Kernel CCSL

Figure 16 is an excerpt of CCSL kernel meta-model. A clock constraint is a set of clock relations. Binary relations apply to two expressions ($left$ and $right$), the simplest expression

being a simple reference to a clock. We use the wording *relation* to emphasize the fact that CCSL is a declarative language. In particular, *Equality* differs from the imperative *assignment* and is not a directed relationship. As discussed in Section 4 there are three kinds of relations: synchronous (coincidence-based), asynchronous (precedence-based) and mixed.

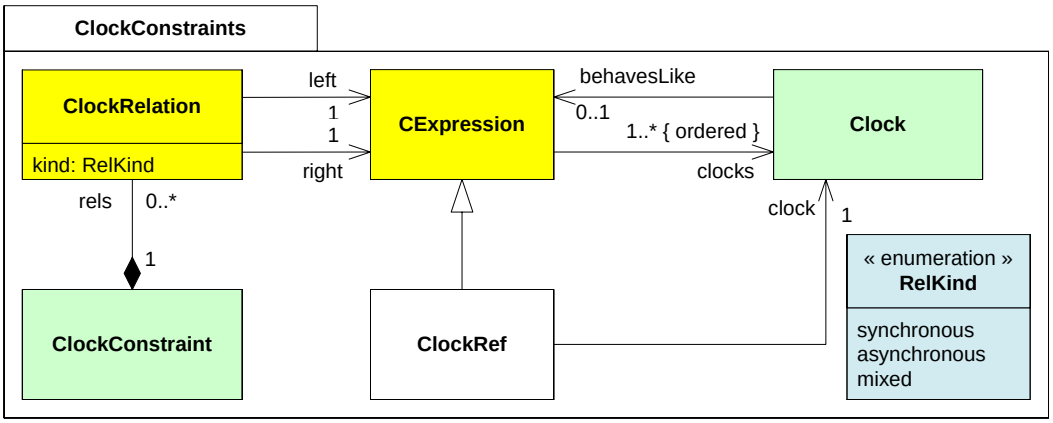

Figure 16: Clock constraint meta-model

We only discuss here the operational semantics of discrete clocks, chronometric clocks are discretized before being handled. For a specification, *i.e.,* a set of discrete clocks C, we associate a function c called *configuration*, $c : C \rightarrow \mathbb{N}$ that gives the current time, *i.e.,* the index of the current instant for each clock. The initial configuration c_0 is so that $(\forall i \in \mathbb{N}^\star)(c_0[i] = 0)$, each clock is initialized at its first instant. A *step* is function that transforms a given configuration into the next one : $step : (C \rightarrow \mathbb{N}) \rightarrow (C \rightarrow \mathbb{N})$.

To process a step, we determine the disabled clocks (D), the set of clocks that CANNOT fire at configuration c. All the remaining clocks are *enabled* and belong to E. E is made of two distinct kinds of clocks, the clocks that MUST fire F and the clocks that MAY fire X (because nothing says they cannot). This computation is done according to the Boolean equations associated with each relation (Annex A.2 and each expression (Annex A.3). Solving the set of Boolean equations gives D and E.

The second step is to apply the simulation policy. The detail of all simulation policies is not given here. The random policy chooses one clock randomly in the set of underdetermined clocks (X) and deduces the new set of clocks that cannot fire or that must fire. This non deterministic process is applied until all clocks from X are either assigned to D or F. All clocks in F are then fired and their index is incremented by 1, whereas the index of all other clocks stalls.

A.2 Clock relations

For each clock t, t denotes the Boolean associated with t. Four rules are given below. Each rule explains how set of Boolean equations are composed from the two Boolean expressions ($b1$ and $b2$) when two clocks $t1$ and $t2$ are in a specific relation.

Sub-clocking

$$\frac{\begin{array}{c} t_1, c \vdash b_1 \\ t_2, c \vdash b_2 \end{array}}{t_1 \boxed{\subset} t_2, c \vdash b_1 \wedge b_2 \wedge (\mathrm{t}_1 \Rightarrow \mathrm{t}_2)} \quad \text{(subclock)}$$

Equality

$$\frac{\begin{array}{c} t_1, c \vdash b_1 \\ t_2, c \vdash b_2 \end{array}}{t_1 \boxed{=} t_2, c \vdash b_1 \wedge b_2 \wedge (\mathrm{t}_1 = \mathrm{t}_2)} \quad \text{(equal)}$$

Strict precedence

$$\frac{\begin{array}{c} t_1, c \vdash b_1 \\ t_2, c \vdash b_2 \\ \beta \triangleq (c(t_1) = c(t_2)) \end{array}}{t_1 \boxed{\prec} t_2, c \vdash b_1 \wedge b_2 \wedge (\beta \Rightarrow \neg \mathrm{t}_2)} \quad \text{(spred)}$$

Non-strict precedence

$$\frac{\begin{array}{c} t_1, c \vdash b_1 \\ t_2, c \vdash b_2 \\ \beta \triangleq (c(t_1) = c(t_2)) \end{array}}{t_1 \boxed{\preccurlyeq} t_2, c \vdash b_1 \wedge b_2 \wedge (\beta \Rightarrow (\mathrm{t}_2 \Rightarrow \mathrm{t}_1))} \quad \text{(pred)}$$

A.3 Clock expressions

Expressions are also described by a set of Boolean equations. The main difference is that a relation applies at each step in the exact same way depending only on the current configuration. Expressions have additional state information (mostly integer counters) and may apply differently depending on both the configuration and their local state. At each step, when the clock t_1 associated with an expression is fired ($t_1 \in F$), the expression is rewritten according to the rewriting rule.

We only give here one example for illustration purpose. It concerns expression filteredBy (denoted ▼). The first two rules (filter1, filter2) give the Boolean equations for the two possible cases (when $n = 1$ and $n > 1$). The last two rules (RWfilter1, RWfilter2) are the rewriting rules for the same two cases.

Clock projection

$$\frac{t_1, c \vdash b_1}{t \cong t_1 \blacktriangledown 1.\sigma, c \vdash b_1 \wedge (\mathrm{t} = \mathrm{t}_1)} \quad (\text{ filter1 })$$

$$\frac{\begin{array}{c} t_1, c \vdash b_1 \\ n > 1 \end{array}}{t \cong t_1 \blacktriangledown n.\sigma, c \vdash b_1 \wedge \neg \mathrm{t}} \quad (\text{ filter2 })$$

$$\frac{t_1 \in F}{t \cong t_1 \blacktriangledown 1.\sigma \rightarrow t \cong t_1 \blacktriangledown \sigma} \quad (\text{ RWfilter1 })$$

$$\frac{t_1 \in F \quad n > 1}{t \cong t_1 \blacktriangledown n.\sigma \rightarrow t \cong t_1 \blacktriangledown (n-1).\sigma} \quad (\text{ RWfilter2 })$$

B Filtering constraint

Constraint ④ applies to the special case where $L = 2$, $PPL = 8$, and $PPW = 4$. Here we consider the general case.

We start with the input/output pixel transformation. Equation 1 implies the following clock relation:

$$(inPixel \blacktriangledown w_1) \boxed{\prec} (outPixel \blacktriangledown w_2) \text{ where}$$
$$w_1 = \left(0^L.1^{PPL-L}\right)^\omega, w_2 = \left(1^{PPL-L}.0^L\right)^\omega \quad (2)$$

Since $inPixel$ is not in the interface, we have to express clock relation 2 directly between $inWord$ and $outPixel$.

$$\left(inWord \blacktriangledown w_3\right) \boxed{\prec} \left(outPixel \blacktriangledown w_4\right) \text{ where}$$
$$w_3 = \left(0^\alpha.1^\beta\right)^\omega$$
$$w_4 = \left(0^\gamma.\left(1.0^{PPW-1}\right)^\beta.0^L\right)^\omega$$
$$\alpha = \lceil L/PPW \rceil$$
$$\beta = WPL - \alpha$$
$$\gamma = PPL - L - \beta * PPW$$

References

[1] C. André. Representation and analysis of reactive behaviors: A synchronous approach. In *Computational Engineering in Systems Applications (CESA)*, pages 19–29. IEEE-SMC, July 1996.

[2] C. André. Computing SyncCharts reactions. *Electronic Notes in Theoretical Computer Science*, 88:3–19, October 2004.

[3] 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.

[4] J-R. Beauvais, E. Rutten, T. Gautier, R. Houdebine, P. Le Guernic, and Y.-M. Tang. Modeling statecharts and activitycharts as signal equations. *ACM Trans. Softw. Eng. Methodol.*, 10(4):397–451, 2001.

[5] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages twelve years later. *Proceedings of the IEEE*, 91(1):64–83, 2003.

[6] 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.

[7] G. Berry. The foundations of Esterel. In C. Stirling G. Plotkin and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.

[8] F. Boulanger and C. Hardebolle. Simulation of multi-formalism models with modhelx. In *ICST*, pages 318–327. IEEE Computer Society, 2008.

[9] 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. Gregory Morrisett and Simon L. Peyton Jones, editors, *POPL*, pages 180–193. ACM, January 2006.

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

[11] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. L., J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.

[12] N. Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic Publishers, Amsterdam, 1993.

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

[14] A. Le Guennec and B. Dion. Bridging UML and safety-critical software development environments. In *Int. Conf. on Embedded and Real-Time Software, ERTS*, 2006.

[15] 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.

[16] X. Li, C. Meng, P. Yu, J. Zhao, and G. Zheng. Timing analysis of UML activity diagrams. In M. Gogolla and C. Kobryn, editors, *UML*, volume 2185 of *Lecture Notes in Computer Science*, pages 62–75. Springer, October 2001.

[17] F. Mallet and C. André. On the semantics of UML/MARTE clock constraints. In *ISORC*, pages 305–312. IEEE Computer Society, March 2009.

[18] K.L. McMillan. Interpolation and SAT-based model checking. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, July 2003.

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

[20] OMG. *Unified Modeling Language, Superstructure*, November 2007. Version 2.1.2 formal/2007-11-02.

[21] OMG. *Systems Modeling Language (SysML) Specification 1.1.* Object Management Group, May 2008. OMG document number: ptc/08-05-17.

[22] C. A. 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, 1987.

[23] D. Potop-Butucaru, S. Edwards, and G. Berry. *Compiling Esterel.* Springer, 2007.

[24] W. Reisig. *Petri nets: an introduction.* Monograph on Theoretical Computer Science. Springer, Berlin, 1985.

[25] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a sat-solver. In W. A. Hunt Jr. and S. D. Johnson, editors, *FMCAD*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer, November 2000.

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

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

[28] T. Weilkiens. *Systems Engineering with SysML/UML: Modeling, Analysis, Design.* The MK/OMG Press, Burlington, MA, USA., 2008.

[29] L. Zaffalon. *Programmation synchrone de systèmes réactifs avec Esterel et les SyncCharts.* Presses Polytechniques et Universitaires Romandes, Lausane (CH), 2005.

Contents

1 Introduction **3**

2 Logical Time **4**
2.1 Clock 4
2.2 Time structure 4

3 Example: Digital Filter **5**
3.1 Informal Specification 5
3.2 Modeling with logical clocks 6
3.3 Clock constraints 7

4 Clock constraints in CCSL **8**
4.1 Coincidence-based clock constraints 9
4.2 Derived coincidence-based clock constraints 9
4.3 Precedence-based clock constraint 11
4.4 Derived precedence-based clock constraints 11
4.5 Mixed constraints 12

5 Design and analysis of the example **12**
5.1 CCSL specification 13
5.2 Running simulations 14
5.3 Esterel/SyncCharts modeling 15
5.4 Checking an Esterel program against a CCSL specification 17
5.4.1 Formal verification tools in Esterel Studio 17
5.4.2 CCSL clock constraints in Esterel 18
5.4.3 Design verification 19

6 Related Work **21**

7 Conclusion **23**

A Semantics of CCSL **23**
A.1 Kernel CCSL 23
A.2 Clock relations 25
A.3 Clock expressions 25

B Filtering constraint **26**

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