HAL
open science

Modèles de temps et de contraintes temporelles de MARTE et leurs applications

Charles André

To cite this version:

Charles André. Modèles de temps et de contraintes temporelles de MARTE et leurs applications. [Rapport de recherche] RR-7788, INRIA. 2011, pp.22. hal-00639211

HAL Id: hal-00639211

https://inria.hal.science/hal-00639211

Submitted on 8 Nov 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Modèles de temps et de contraintes temporelles de MARTE et leurs applications

Charles André

RESEARCH REPORT
N° 7788
Novembre 2011
Project-Team Aoste

ISSN 0249-6399 ISRN INRIA/RR--7788--FR+ENG

Modèles de temps et de contraintes temporelles de MARTE et leurs applications

Charles André *

Équipe-Projet Aoste

Rapport de recherche n° 7788 — Novembre 2011 — 22 pages

Résumé : Le profil UML MARTE a été créé pour la modélisation et l'analyse des systèmes temps réel embarqués. A ce titre, il permet d'annoter des modèles UML avec des paramètres temporels (durée, période, échéance, etc.). MARTE va bien au-delà de la simple annotation. Il définit un modèle de temps et de contraintes temporelles qui traite à la fois du temps dit physique et du temps logique. Le temps "physique" (appelé chronométrique en MARTE) est utilisé principalement dans les applications multitâches temps réel. MARTE permet aussi la prise en compte de plusieurs référentiels temporels dans une même application ce qui s'avère très utile en modélisation de systèmes répartis ou de systèmes électroniques "multi-horloges". Après une brève présentation du temps chronométrique, nous développons une facette moins connue de MARTE: le temps logique et le langage d'expression de contraintes appelé CCSL. Ceci introduit un modèle de "temps pour la conception" dans lequel la sémantique temporelle est directement intégrée au lieu d'être une simple annotation.

Mots-clés : UML, modèle de temps, contraintes temporelles, MARTE, CCSL

Ce rapport est le texte d'une communication faite par l'auteur à l'École d'Été Temps Réel, Brest, Août 2011

* Université de Nice Sophia Antipolis

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

MARTE Time and Time Constraints Models and their applications

Abstract: The UML profile for Modeling and Analyzing Real-Time Embedded Systems (MARTE) allows the designer to annotate his/her models with temporal parameters (duration, period, deadline,...). MARTE goes far beyond simple annotations. It introduces a model of Time and Timing Constraints, dealing with both physical and logical time. The "physical" time (called Chronometric Time in MARTE) is mainly used in real-time multitask applications. A MARTE model can consider several time-bases, which is useful in distributed applications and in "multi-clock" electronic systems. After a brief presentation of the chronometric time we turn to a less known facet of MARTE: the logical time and CCSL (the language developed for time constraint specifications). The Logical Time appears to be a "time for design" that allows integrating a semantics of time directly in the model.

Key-words: UML, time model, temporal constraints, MARTE, CCSL

1 Introduction

Le *temps* est intimement lié à notre vie biologique (cycles circadiens et annuels, vieillissement), affective (passé, présent, futur) et sociale (emplois du temps, agendas, calendriers). Il intervient dans l'expression des lois physiques. Il joue également un rôle essentiel dans de nombreux systèmes techniques, tels les systèmes temps réel embarqués. Le profil UML MARTE [1] a été créé spécialement pour la modélisation et l'analyse de ces systèmes et il couvre aussi bien les aspects logiciels (tâches, programmes embarqués) que matériels (processeurs, circuits). La spécification de MARTE inclut un modèle de temps et de contraintes temporelles, objet de cette présentation.

Le profil MARTE permet d'annoter des modèles UML avec des informations relatives au temps (valeurs et contraintes temporelles), qui peuvent être attachées à des éléments modélisant du logiciel (traitements à exécuter) ou du matériel (supports des exécutions). Ces informations peuvent ensuite être exploitées par des outils d'analyse de performance temporelle pour évaluer l'efficacité et la correction temporelle (respect des échéances critiques, par exemple) des systèmes modélisés. Ces modélisations temporelles et leurs analyses s'appuient sur un modèle lié au temps dit *physique*, qui est un temps linéaire et métrique, appelé *temps chronométrique* dans MARTE. Ce modèle de temps reprend et précise celui défini dans le profil SPT [2] (Schedulability, Performance, and Time Specification), profil UML désormais remplacé par MARTE. MARTE va bien au-delà en proposant des modèles de temps plus généraux. Ainsi est-il possible de définir *plusieurs référentiels temporels*, ce qui s'avère bien utile pour la modélisation des systèmes *répartis* et des systèmes électroniques *multi-horloges*. Une forme de *temps logique* est également disponible. Utilisé conjointement avec CCSL (Clock Constraint Specification Language, un langage d'expression de contraintes temporelles introduit dans MARTE), le temps logique peut être exploité comme un *temps pour la conception*, permettant d'exprimer des contraintes temporelles complexes, exploitables en vérification formelle de propriétés et en synthèse de circuits.

La section 2 présente les fondements du modèle de temps de MARTE. On y aborde diverses interprétations et représentations du temps, avant de dégager les choix faits pour MARTE. La section suivante étudie le *temps chronométrique* de MARTE. On y apprend comment créer des *horloges chronométriques* et comment exprimer des contraintes temporelles liées au temps physique. La section 4 est consacrée à CCSL. Elle explique comment spécifier des contraintes entre horloges et elle donne une sémantique formelle aux éléments de ce langage. La section 5 adopte un point de vue plus pratique en illustrant, sur un exemple de modélisation d'un filtre d'image, comment utiliser CCSL : choisir des horloges logiques, imposer des contraintes et analyser les comportements temporels du système contraint.

2 Fondements du modèle de temps de MARTE

2.1 Nature du temps

La nature même du Temps a suscité de nombreux débats philosophiques, théologiques et scientifiques. Aucune réponse n'est pleinement satisfaisante. L'article de F. Schreiber intitulé "*Is Time a Real Time ?*" [3] contient de nombreuses références au concept de temps vu par des philosophes et des physiciens. Outre cette facette culturelle, cet article décrit le temps ou plutôt les temps vus par les informaticiens. A ce titre, sa lecture est fortement conseillée pour une meilleure compréhension du modèle

de temps de MARTE. Les lecteurs intéressés par une vision plus humaniste du temps peuvent consulter le livre [4] de J. T. Fraser qui présente ses conceptions du temps dans différents domaines.

Dès l'antiquité, le temps est apparu comme intimement lié au *mouvement* (mouvement des corps célestes en particulier). Le concept d'*événement* est également très présent. Pour certains ce sont carrément les événements qui définissent le temps ; pour d'autres le temps est simplement un conteneur d'événements. Ce dernier point de vue est celui adopté par I. Newton. Dans sa théorie, le temps est une variable indépendante utilisée dans les observations et descriptions des mouvements, ainsi que dans la formulation des lois physiques. Le temps s'écoule indépendamment de tout autre chose (événements, espace, ...). La notion de *durée* y est essentielle et fait l'objet de *mesures*. Avec A. Einstein, le temps perd le caractère privilégié qu'il avait en mécanique classique pour se fondre dans le concept de continuum *espace-temps*. Les événements sont des points de cet espace-temps. La notion de *simultanéité* est profondément modifiée : d'absolue (même valeur de la variable temps), elle devient relative. La *causalité* entre événements s'exprime par une propriété géométrique (inclusion dans un bi-cône de l'espace-temps) ainsi que l'indépendance causale (*concurrent with* en anglais).

2.2 Représentation du temps

Plus que de débattre sur la nature du temps, nous avons besoin de le *représenter*. Cette proposition peut paraître suspecte car elle consiste à représenter quelque chose de mal défini. En fait il s'agit plutôt de représenter des entités liés au temps.

2.2.1 Entités primitives

La première question est le choix des *entités primitives* pour représenter la structure du temps. En informatique on retient généralement trois options :

1. points de temps (instants) ;
2. segments de temps (intervalles) ;
3. occurrences dans le temps (événements).

L'intervalle se définit aisément à partir de deux instants (début et fin). En ce qui concerne les événements on distingue deux approches. La première est une vue absolue dans laquelle le temps est un conteneur d'événements ; le temps (*date*) permet alors de distinguer les événements. La seconde est relative, les événements y précèdent ontologiquement le temps et ils identifient les instants. Le modèle de temps de MARTE s'appuie sur les *instants* (points de temps). Il définit également les *durées* (intervalles de temps). Les instants appartiennent à des bases de temps. Sur une base de temps particulière les instants sont totalement ordonnés. À une *occurrence d'événement*, on peut associer un instant. Noter que MARTE adopte la terminologie UML qui distingue les événements de leurs occurrences.

2.2.2 Structure du temps

On distingue classiquement les modèles de temps continus, denses et discrets. MARTE accepte les deux derniers. La distinction entre dense et continu n'est pas apparue comme nécessaire en MARTE. A un temps discret on associe une base de temps discrète dont les instants sont indexés par les entiers naturels. Dans un même modèle peuvent coexister plusieurs bases de temps aussi bien discrètes que denses. Ces bases

de temps sont *a priori* indépendantes, mais des *contraintes* peuvent être imposées entre instants de différentes bases de temps. Il en résulte alors un modèle de temps dans lequel les instants sont *partiellement ordonnés*.

2.2.3 Relations temporelles

Comme il l'a été indiqué précédemment, les instants appartenant à deux bases de temps différentes peuvent être contraints. MARTE considère principalement deux relations : la *précédence* et la *coïncidence*. La précédence est une relation d'ordre avant-après entre instants. La coïncidence est une relation d'équivalence de simultanéité entre instants ; en quelque sorte elle caractérise un partage d'instants entre bases de temps. MARTE s'appuie donc sur des *relations temporelles entre instants*. Les relations de précédences sont usuelles dans les représentations de relations temporelles, les relations de coïncidence sont plus originales. Les relations temporelles sont parfois spécifiées à partir d'intervalles de temps. C'est en particulier l'approche préconisée par J. F. Allen [5]. La structure partiellement ordonnée des instants en MARTE permet d'exprimer les relations sur intervalles d'Allen. MARTE permet également de spécifier directement des contraintes sur les durées.

Un langage a été proposé pour spécifier commodément les relations entre instants. Il s'agit là d'une des contributions du modèle de temps de MARTE ; elle sera détaillée dans la section 4.

2.2.4 Métriques temporelles

Les aspects logiques de précédence ne sont pas suffisants pour exprimer toutes les propriétés temporelles des systèmes fonctionnant en temps réel. Il est nécessaire d'introduire des éléments quantitatifs. Une solution est de définir une *distance* qui mesure la durée entre deux instants. Dans le cas où la structure temporelle est linéaire, c'est-à-dire que ses instants sont totalement ordonnés, on peut associer à chaque instant un nombre unique qui représente sa distance par rapport à un *instant de référence*. Avec la distance, on définit un espace métrique et une *unité*. On peut également introduire des *granularités* de temps et des *dates*, ainsi que la notion de *période*. Ce modèle permet de représenter le *temps dit physique* et c'est celui qu'on utilise dans les applications temps réel usuelles. Lorsque l'ordre sur le instants n'est que partiel, l'utilisation de ce modèle devient difficile. MARTE offre la possibilité de définir des *temps dits logiques* sur lesquels on introduit des distances à valeurs entières et qui ne sont pas liées à un

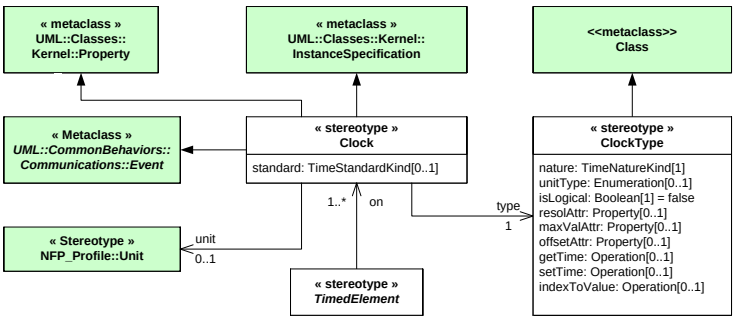

FIGURE 1 – Clock dans le profil MARTE

espace euclidien.

2.3 Le Temps dans le profil MARTE

La proposition initiale du profil UML pour la *"modélisation et l'analyse des systèmes embarqués et temps réel"* (MARTE) date de 2007. Pendant la phase de finalisation qui a suivi, la spécification a subi quelques modifications avant d'être par adoptée par l'OMG en novembre 2009. La version publique la plus récente [1] est la version 1.1 d'août 2010. Une présentation en français [6] contient une description détaillée du modèle de temps inclus dans la version initiale, ainsi que de ses utilisations. Cette présentation reste d'actualité car la finalisation du profil n'a changé que quelques détails au modèle de temps. Le lecteur est donc invité à consulter ce texte pour une connaissance plus approfondie du modèle de temps de MARTE. Nous nous contentons ici d'une présentation sommaire de ce modèle.

MARTE offre la possibilité de lier *explicitement* un élément de modèle au temps ; cet élément devient alors un TimedElement. Ceci se fait en référençant au moins une horloge (Clock dans le vocabulaire MARTE). La figure 1 précise les principaux stérotypes du modèle de temps de MARTE.

Une *Clock* donne accès à la structure du modèle de temps sous-jacent. Elle référence un ClockType, qui joue le rôle de type. Un ClockType regroupe des propriétés communes à un ensemble d'horloges. La propriété nature prend les valeurs *dense* ou *discrete*. Elle précise si les horloges de ce type donnent accès à un temps dense ou discret. La propriété isLogical précise si les horloges donnent accès à un temps chronométrique ou à un temps logique. Un *temps chronométrique* fait implicitement référence au temps physique et possède les propriétés métriques usuelles qui permettent d'effectuer des opérations algébriques sur les valeurs temporelles. C'est le temps qui est utilisé de préférence dans les applications "temps réel". Un *temps logique* par contre n'est pas (directement) lié au temps physique ; il évolue indépendamment de celui-ci. Un *ClockType* définit également les *unités* autorisées (propriété unitType). D'autres propriétés permettent de choisir le type de certains paramètres d'horloge comme par exemple celui de la *résolution* (propriété resolAttr).

Une *Clock* contient des informations plus spécifiques comme l'*unité* effectivement utilisée par l'horloge, ou bien le standard de temps choisi (information effectivement définie seulement pour les horloges de temps chronométrique). La valeur de la *résolution* (granularité du temps) peut être également spécifiée lors de l'instanciation de l'horloge.

Dans la première version du profil, une *Clock* ne pouvait stéréotyper qu'une InstanceSpecification. Lors de la finalisation du profil il est apparu utile d'étendre aux métaclasses Property et Event. La première extension permet d'associer une horloge à un *port*, qui en UML est une Property. La seconde répond au besoin de créer des *horloges logiques* dont les instants coïncident avec les occurrences d'un événement. Il est alors possible de créer des modèles ayant un *temps multiforme*. Ce type de modélisation est usuel dans les *langages synchrones* [7, 8] : n'importe quel événement répétitif (non nécessairement périodique au sens du temps physique) peut servir d'horloge.

Le profil met également à disposition des utilisateurs une bibliothèque pour le temps (TimeLibrary). On y trouve un ensemble d'unités TimeUnitKind utiles pour les horloges chronométriques (définissant la seconde, ses multiples et sous-multiples). IdealClock est un type d'horloge prédéfini qui correspond à du temps chronométrique et dense. Les horloges de ce type sont censées suivre fidèlement le temps "physique". idealClk est une horloge prédéfinie de type IdealClock. Cette horloge est utilisée dans

les applications qui font implicitement référence au temps physique. Des détails sont donnés dans la section suivante.

3 Temps chronométrique

3.1 Création d'horloges chronométriques

Pour créer des horloges chronométriques, le plus simple est d'importer la bibliothèque MARTE::TimeLibrary et d'utiliser l'horloge idéale idealClk et son type IdealClock.

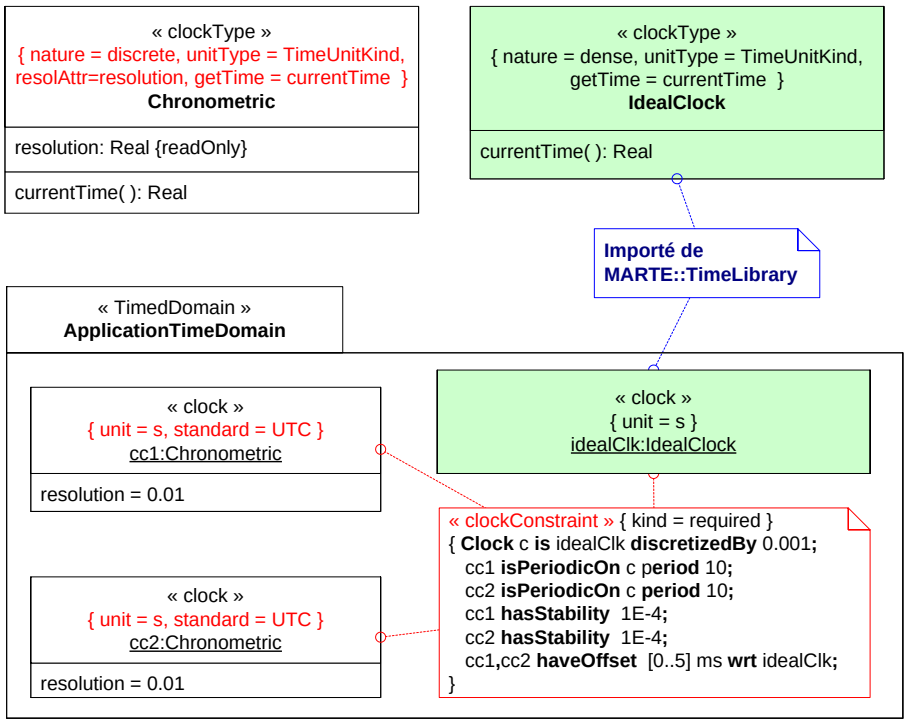

FIGURE 2 – Spécification d'horloges chronométriques

Pour commencer, nous définissons un nouveau type d'horloge : Chronometric (figure 2, partie supérieure). Pour cela, nous créons la classe Chronometric ayant un attribut resolution de type Real et une opération currentTime retournant un Real. Le stéréotype ClockType est ensuite appliqué à cette classe. Les valeurs données aux attributs du stéréotype spécifient que les horloges typées Chronometric sont des horloges chronométriques (c'est l'option par défaut), qu'elles utilisent des unités appartenant à TimeUnitKind, qu'elles accèdent à un temps discret et que resolAttr est la propriété resolution de la classe stéréotypée. Ce dernier choix a pour conséquence que la valeur de la résolution sera précisée dans l'instance elle-même.

Il faut ensuite créer un paquetage auquel on applique le stéréotype TimedDomain. Pour notre illustration, nous créons trois horloges dans ce paquetage : idealClk récopiée de la bibliothèque TimeLibrary et deux instances de Chronometric appelées cc1 et cc2. A ces instances on applique le stéréotype Clock et on précise que leur unité est la

seconde (s) et qu'elles respectent le standard UTC. Cette dernière information ne sera pas exploitée dans l'application ; on aurait pu ne pas spécifier l'attribut standard. Le slot resolution dans les instances cc1 et cc2 contient la valeur 0.01. Ceci signifie que les deux horloges ont une résolution de 0.01s = 10 ms. Arrivé à ce point, nous avons donc une horloge à temps dense (idealClk) et deux horloges discrètes à 100 Hz. Ces horloges sont *a priori* indépendantes.

Rappelons qu'idealClk est "parfaite". Si nous désirons exprimer le fait que les deux autres horloges s'écartent de cette perfection, il faut introduire des *contraintes d'horloges*. La figure 2 contient une telle contrainte. La *tagged-value* kind positionnée à required précise qu'il s'agit d'une contrainte d'exigence, c'est-à-dire qu'il faudra garantir la satisfaction de cette contrainte. Commentons maintenant le corps de la contrainte. Elle est exprimée dans le langage CCSL (voir la section 4). La première instruction définit une horloge locale c qui est une discrétisation de l'horloge idealClk avec un pas de discrétisation de 0.001 s. c est donc une horloge idéale, discrète et de fréquence 1 kHz. Les deux instructions suivantes lient les horloges discrètes cc1 et cc2 à c. La sémantique de '`cc1 isPeriodicOn c period 10`' est

$$(\exists d \in \mathbb{N}) c[d + 10 * (k - 1)] \prec cc1[k] \preccurlyeq c[d + 10 * k]$$

Cette constrainte exprime donc que cc1 à un tic tous les 10 tics de c. Toutefois elle autorise de grandes variations de durées entre deux tics successifs de cc1 : $0 < cc1[k+1] - cc1[k] < 20$ ms. La contrainte '`cc1 hasStability 1E-4`' limite ces variations relatives à 10^{-4}, c'est-à-dire que $10 - 0.001 \leq cc1[k+1] - cc1[k] \leq 10 + 0.001$ en ms mesurées sur idealClk. La stabilité est une *propriété non fonctionnelle* attachée à une horloge chronométrique. Les instructions 3 et 5 spécifient les mêmes caractéristiques pour cc2. Ceci ne signifie par pour autant que cc1 et cc2 soient deux horloges identiques. Elles peuvent présenter un décalage de phase (*offset*). La dernière instruction précise que cet offset, mesuré sur idealClk doit être entre 0 et 5 ms, bornes incluses.

La figure 3 représente *une* structure de temps répondant à ces contraintes. Les gros points représentent les instants (discrets). Ceux d'idealClk ne sont pas distingués car ils appartiennent à un ensemble dense.

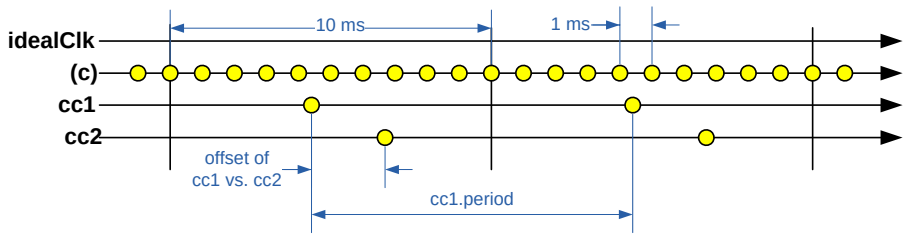

FIGURE 3 – Instants des horloges cc1 et cc2

3.2 Expression de contraintes temporelles chronométriques

Dans des applications de type multi-tâche temps réel tournant sur un processeur unique, une seule horloge chronométrique est suffisante. On peut alors se contenter d'importer idealClk dans son application. Les valeurs des paramètres temporels tels que la période, l'échéance, la durée d'exécution sont alors relatives à cette horloge. MARTE a défini le langage VSL (*Value Specification Language*) pour spécifier des valeurs avec

unités, en particulier celles relatives au temps. Outre les valeurs temporelles, VSL peut spécifier des contraintes sur les dates ou les durées. Il offre également des facilités pour distinguer des occurrences d'événements à l'aide d'indices ainsi qu'une syntaxe pour exprimer des gigues (*jitter*). La figure 4 décrit une interaction sous forme de diagramme de séquence avec des observations d'instants et de durées. Les contraintes temporelles sont exprimées entre une paire d'accolades.

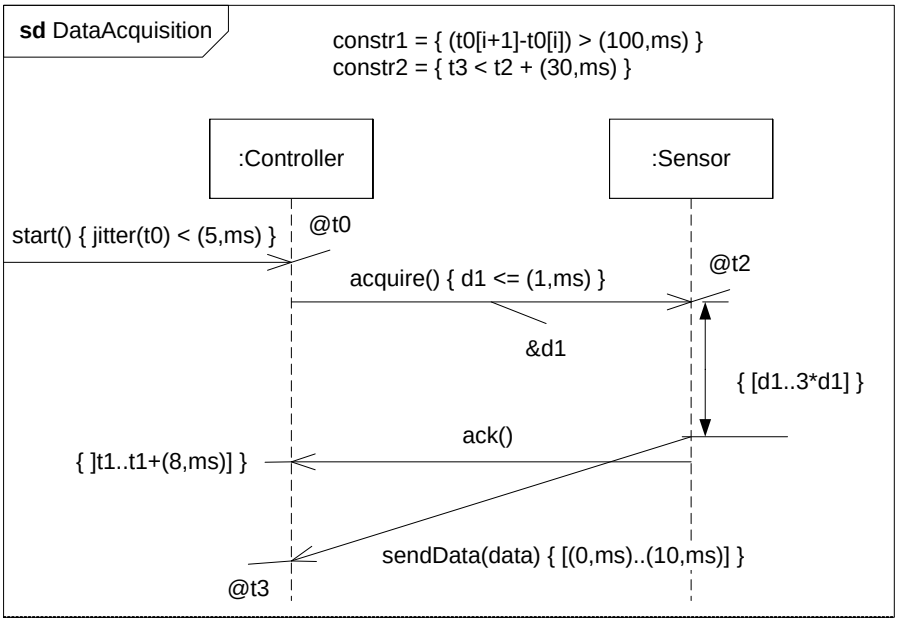

FIGURE 4 – Exemple d'expression de contraintes temporelles en VSL

La séquence est activée par le message `start`. Le contrôleur reçoit ce message à l'instant `t0`. Cet instant est contraint de deux façons : par la contrainte `constr1` donnée dans la partie supérieure du cadre et par la contrainte de gigue écrite à côté du message. Il en résulte que les réceptions de `start` sont périodiques de période 100 ms avec une gigue inférieure à 5 ms. La propagation du message acquire a une durée $d1$ qui est inférieure ou égale à 1 ms. La durée de traitement entre la réception de la demande d'acquisition et l'émission de la valeur acquise (message sendData) est contrainte à l'intervalle $[d1..3 * d1]$ dont la valeur dépend de celle de $d1$. On constate que VSL permet des expressions paramétriques de contraintes.

Plus de détails sur les VSL et son usage sont disponibles dans un guide méthodologique [9].

4 CCSL

4.1 Généralités

Comme nous l'avons indiqué dans la section 2.3, le modèle de temps défini dans MARTE est *multiforme*, c'est-à-dire que, vis-à-vis du système modélisé, tout événement peut définir un référentiel temporel (base de temps). Les instants de cette base

de temps correspondent aux occurrences de l'événement associé et sont donc *strictement ordonnés*. Le temps dit physique (une fois discrétisé) est un cas particulier pour lequel les occurrences de l'événement sont directement liées au temps physique. Si il n'existe pas de relation directe avec le temps physique on parle alors de temps *logique*. Dans la section 2.2.3, qui traitait des relations temporelles, nous avions évoqué l'existence d'un langage permettant d'exprimer les relations entre instants et plus généralement les contraintes d'horloges (stéréotype ClockConstraint de MARTE). Il s'agit de CCSL (*Clock Constraint Specification Language*). Ce langage a été introduit dans la spécification MARTE [1, Annex C]. Depuis, CCSL a évolué indépendamment du profil MARTE. Une syntaxe abstraite et une sémantique formelle ont été définies pour un noyau de CCSL [10–12]. CCSL est maintenant utilisé comme un langage pivot pour spécifier les interactions entre différents modèles de calcul [13]. Un environnement logiciel appelé TimeSquare a également été développé pour spécifier, analyser et visualiser les contraintes CCSL. Il est présenté dans la section 4.6.

CCSL permet de *spécifier* et d'*analyser* le comportement temporel de systèmes dont la modélisation inclut plusieurs (généralement de nombreux) référentiels temporels interdépendants. CCSL utilise le terme d'horloge (*Clock*) plutôt que celui de référentiel temporel. CCSL n'est pas un langage de programmation : il se limite aux aspects contrôle. Les traitements ne sont considérés qu'au travers d'événements particuliers représentant par exemple les activations d'un traitement, la réception ou l'émission d'une donnée. En fait, CCSL permet d'exprimer des *synchronisations complexes* entre systèmes communicants, il se rapproche ainsi des *contrôleurs* définis dans le modèle de composants pour systèmes embarqués hétérogènes appelé 42 [14]. Par son caractère événementiel et réactif, CCSL se rapproche aussi des modèles tels que les réseaux de Petri [15] et les modèles réactifs synchrones [8].

4.2 Relations entre instants

Soit $\mathcal{C}$ l'ensemble des horloges d'un modèle, pour toute horloge $c \in \mathcal{C}$, $\mathcal{I}_c$ est l'ensemble de ses instants. Soit $\mathcal{I} \triangleq \bigcup_{c \in \mathcal{C}} \mathcal{I}_c$ l'ensemble des instants du modèle. Pour un modèle, on définit une relation (*precedence*, notée $\preccurlyeq$) réflexive et transitive sur l'ensemble des instants : $\preccurlyeq \subset \mathcal{I}^2$. De cette relation on dérive quatre nouvelles relations : *Coincidence* ($\equiv \triangleq \preccurlyeq \cap \succcurlyeq$), *Strict precedence* ($\prec \triangleq \preccurlyeq \setminus \equiv$), *Independence* ($\| \triangleq \overline{\preccurlyeq \cup \succcurlyeq}$) et *Exclusion* ($\# \triangleq \prec \cup \succ$).

Une interprétation informelle des relations $\prec$ et $\equiv$ a déjà été donnée (section 2.2.3). Dans un modèle, certaines précédences sont obligatoires, comme celles entre les instants d'une même horloge c :

$$(\forall i, j \in \mathcal{I}_c)\ (i \neq j) \Rightarrow (i \prec j) \vee (j \prec i)$$

Ou bien de façon équivalente $(\mathcal{I}_c, \prec_c)$ est un *ordre total strict*, en introduisant la relation $\prec_c$

$$\prec_c \triangleq \preccurlyeq \cap (\mathcal{I}_c \times \mathcal{I}_c) \tag{1}$$

Énumérer toutes les relations entre instants pour un modèle serait fastidieux, voire impossible puisque l'ensemble des instants d'une horloge est généralement infini. L'idée a donc été d'exprimer directement les relations entre horloges à l'aide d'un langage spécifique : CCSL. La sémantique de CCSL s'exprime en termes de relations entre instants.

4.3 Relations d'horloges

Le noyau de CCSL définit des relations binaires entre horloges.

Sous-horloges

"a est *sous-horloge* de b" ou "b est *super-horloge* de a", noté $a \boxed{\subset} b$, signifie que chaque instant de a est coïncident avec un instant de b et que la projection conserve la relation d'ordre $\prec$. Plus précisément :

$$\begin{aligned} a \boxed{\subset} b \Leftrightarrow (\exists h : \mathcal{I}_a \to \mathcal{I}_b) : \\ (\forall i \in \mathcal{I}_a)\ i \equiv h(i) \\ \wedge\ (\forall i, j \in \mathcal{I}_a)\ (i \prec j) \Rightarrow \big(h(i) \prec h(j)\big) \end{aligned} \tag{2}$$

Dans le cas où "a est sous-horloge de b" et "b est sous-horloge de a", h devient une bijection et on dit que les horloges sont *synchrones* (noté $a \boxed{=} b$).

Précédence stricte d'horloges

"a est *strictement plus rapide* que b" ou "a *précède strictement* b", noté $a \boxed{\prec} b$, signifie que chaque instant de b est précédé strictement par un instant de a et que la projection conserve la relation d'ordre $\prec$. Plus précisément :

$$\begin{aligned} a \boxed{\prec} b \Leftrightarrow (\exists h : \mathcal{I}_b \to \mathcal{I}'_a) : \\ \mathcal{I}'_a \text{ est un segment initial de } (\mathcal{I}_a, \prec_a) \\ \wedge\ h \text{ est une bijection} \\ \wedge\ (\forall i \in \mathcal{I}_b)\ h(i) \prec i \\ \wedge\ (\forall i, j \in \mathcal{I}_b)\ (i \prec j) \Rightarrow \big(h(i) \prec h(j)\big) \end{aligned} \tag{3}$$

Dans le cas d'horloges à temps discret, la relation peut s'exprimer plus simplement :

$$a \boxed{\prec} b \Leftrightarrow (\forall k \in \mathbb{N}^\star)\ (b[k] \in \mathcal{I}_b) \Rightarrow (a[k] \prec b[k]) \tag{4}$$

Dans un souci de simplification, nous nous limitons désormais aux horloges à temps discret.

Précédence d'horloges

"a est *plus rapide* que b" ou "a *précède* b", noté $a \boxed{\preccurlyeq} b$, est une forme affaiblie de la précédence stricte. Dans le cas discret on la définit ainsi :

$$a \boxed{\preccurlyeq} b \Leftrightarrow (\forall k \in \mathbb{N}^\star)\ (b[k] \in \mathcal{I}_b) \Rightarrow (a[k] \preccurlyeq b[k]) \tag{5}$$

C'est à dire que $a[k]$ précède strictement ou coïncide avec $b[k]$.

Exclusion d'horloges

"a et b sont *exclusives*", noté $a \boxed{\#} b$, signifie que les horloges a et b n'ont aucun instant coïncident.

$$a \boxed{\#} b \Leftrightarrow (\forall i \in \mathcal{I}_a, \forall j \in \mathcal{I}_b)\ i \# j \tag{6}$$

4.4 Expressions d'horloges

CCSL permet de spécifier de nouvelles horloges à l'aide d'expressions. Le noyau CCSL définit une dizaine d'expressions de base (voir les rapports [10,12]). Leur sémantique fait intervenir la relation de coïncidence, celle de précédence, ou les deux. Nous présentons un échantillon significatif de ces divers cas, en privilégiant des expressions usuelles plutôt que les expressions primitives.

Expressions basées sur la coïncidence

Ces expressions définissent des sous-horloges d'une horloge donnée en précisant l'application h de l'équation 2.

discretizedBy est une telle expression qui crée des horloges discrètes à partir d'une horloge dense. Elle a été utilisée dans la section 3. Pour $p \in \mathbb{R}_0^+$, "a discretizedBy p" spécifie un ensemble de sous-horloges de a caractérisées par $o \in \mathbb{R}_0^+$ et une application linéaire de $\mathbb{N}^\star \to \mathbb{R}_0^+$ telle que $n \mapsto o + n * p$. o est un décalage initial (*offset*) arbitraire.

filteredBy définit une sous-horloge d'une horloge discrète donnée. L'application entre les deux horloges est caractérisée par un *patron de filtrage* (ou plus simplement *filtre*) codé par un *mot binaire* fini ou infini $w \in \{0,1\}^* \cup \{0,1\}^\omega$. "$a$ filteredBy w", noté $a \blacktriangledown w$, définit la sous-horloge c de a telle que $(\forall k \in \mathbb{N}^\star) c[k] \equiv a[w \uparrow k]$, où $w \uparrow k$ est l'indice du k^{e} 1 dans w. Les mots binaires sont très utilisés pour représenter des séquences d'activations. Ces dernières ont souvent un caractère périodique et peuvent être représentées par des *mots binaires périodiques* dénoté par $w = u(v)^\omega$. u et v sont des mots binaires finis, appelés respectivement *préfixe* et *période*.

Expressions basées sur la précédence

Inf et Sup sont des opérateurs point fixe qui entrent dans cette catégorie. $\mathsf{Inf}(a,b)$ spécifie l'horloge la plus lente parmi toutes celles qui sont plus rapides que a et b ; $\mathsf{Sup}(a,b)$ spécifie l'horloge la plus rapide parmi toutes celles qui sont plus lentes que a et b. Formellement :

$$\mathsf{Inf}(a,b) \boxed{=} \max_{\preccurlyeq} \left\{ d \in \mathcal{C} \mid (d \boxed{\preccurlyeq} a) \wedge (d \boxed{\preccurlyeq} b) \right\} \tag{7}$$

Soit $c = \mathsf{Inf}(a,b)$, une interprétation plus intuitive est que tout instant $c[k]$ coincide avec l'instant $a[k]$ si $a[k]$ précède l'instant $b[k]$, et avec $b[k]$ dans le cas contraire.

$\mathsf{Sup}(a,b)$ se définit de façon duale.

Expressions mixtes

Un premier exemple d'expression faisant intervenir à la fois des relations de précédence et de coïncidence est le sampledOn. L'horloge c définie par l'expression "a sampledOn b", est une sous-horloge de b dont chaque instant coïncide avec un instant de b précédé par un instant de a. a et b étant *a priori* indépendantes, "a sampledOn b" permet de "synchroniser" a sur b. Noter que a et b sont des horloges discrètes. Formellement

$$\begin{aligned} c \boxed{=} a \ \mathsf{sampledOn}\ b \Leftrightarrow \mathcal{I}_c \text{ ensemble maximal :} \\ (\forall i \in \mathcal{I}_c)(\exists j \in \mathcal{I}_b)(\exists k \in \mathcal{I}_a) \\ (j \equiv i) \wedge ({}^\circ j \prec k \preccurlyeq j) \\ \text{avec } {}^\circ j \text{ instant prédécesseur de } j \text{ dans } b. \end{aligned} \tag{8}$$

L'expression "a delayedFor δ on b", avec $\delta \in \mathbb{N}^\star$, est analogue au sampledOn, mais les instants de l'horloge créée (c) subissent un retard de δ instants de b. En se limitant au cas où δ est une constante, la sémantique s'exprime par

$$\begin{aligned} c \boxed{=} a \text{ delayedFor } \delta \text{ on } b \Leftrightarrow \mathcal{I}_c \text{ maximal :} \\ (\forall i \in \mathbb{N}^\star, c[i] \in \mathcal{I}_c) \\ (\exists j \in \mathbb{N}^\star, j > \delta, b[j] \in \mathcal{I}_b) \\ (\exists k \in \mathbb{N}^\star, a[k] \in \mathcal{I}_a) \\ (b[j] \equiv c[i]) \wedge \\ (b[j-\delta-1] \prec a[k] \preccurlyeq b[j-\delta]) \end{aligned} \tag{9}$$

4.5 Relations/Expressions définies par l'utilisateur

À partir des relations et expressions du noyau de CCSL, il est possible d'en construire de nouvelles. À titre d'illustration, considérons la *relation d'alternance* entre deux horloges discrètes a et b, notée $a \boxed{\sim} b$. Dans le cas d'horloges infinies, l'alternance impose

$$a \boxed{\sim} b \Leftrightarrow (\forall k \in \mathbb{N}^\star)\ a[k] \prec b[k] \prec a[k+1] \tag{10}$$

Cette relation peut en fait se construire par composition de deux précédences et d'un retard (voir [12]) :

$$a \boxed{\sim} b \Leftrightarrow (a \boxed{\prec} b) \mid \big(b \boxed{\prec} (a \text{ delayedFor } 1 \text{ on } a)\big)$$

où $\mid$ est la composition de contraintes dont la sémantique est la conjonction logique.

4.6 TimeSquare

TimeSquare un environnement logiciel dédié au modèle de temps de MARTE et à CCSL. Il est disponible sous forme de plug-ins Eclipse téléchargeables depuis le site `http://www-sop.inria.fr/aoste/software/time_square`. Il a été développé pour faciliter d'une part l'application du modèle de temps du profil MARTE à un modèle UML et d'autre part l'utilisation de CCSL en relation avec MARTE ou indépendamment de celui-ci.

TimeSquare est maintenant principalement utilisé pour définir des *Modèles de Calcul et Communication* (MoCC) s'appuyant sur une sémantique formelle. Les comportements temporels y sont spécifiés en CCSL. Ces MoCCs sont ensuite utilisables aussi bien avec des modélisations UML qu'avec des DSL (*Domain Specific Language*). Les fonctionnalités principales de TimeSquare sont

1. Définition de bibliothèques CCSL utilisateur dédiées aux différents MoCCs ;
2. Application de contraintes CCSL à des modèles UML ou à des modèles DSL ;
3. Simulation des modèles contraints et génération de traces d'exécution ;
4. Visualisation des traces sous forme de chronogrammes, exploration de traces et rétro-annotation des modèles.

5 Usage des horloges logiques

5.1 Comment utiliser CCSL

Comme nous l'avons indiqué dans les généralités sur le modèle de temps de MARTE (section 2.3), à un événement on peut associer une horloge logique dont les instants coïncident avec les occurrences de cet événement. Ce lien entre événements et horloges logiques fait que CCSL peut très bien exprimer des contraintes entre occurrences d'événements. Dans un système il existe généralement des *relations de causalité* entre certains événements. Ces relations de causalité peuvent être modélisées en CCSL par des relations de précédence entre les horloges associées. Il est également possible dans un système, décrit comme assemblage de sous-systèmes, qu'un même événement soit nommé et interprété différemment dans deux sous-systèmes distincts. L'unification des événements s'exprime en CCSL par des relations de coïncidence entre instants de horloges associées.

Lors de la conception d'un système, on est amené à raffiner son modèle. Les propriétés de causalité et d'unification d'événements sont des invariants pour un système : les contraintes qu'elles imposent doivent être absolument préservées. À chaque raffinement, des contraintes autres peuvent être rajoutées. Elles peuvent être motivées par des choix de conception. Les aspects temporels de ces contraintes additionnelles sont exprimables en CCSL. Ainsi, les contraintes temporelles imposées au système sont progressivement enrichies. Grâce à la sémantique mathématique de CCSL il est alors possible d'analyser formellement les propriétés temporelles du système.

5.2 Exemple d'application

Cet exemple illustre une approche par composants en UML avec modélisation de contraintes temporelles et analyse des comportements temporels.

L'application

L'application, appelée DigitalFilter, est un filtrage d'image vidéo. DigitalFilter lit une image en mémoire, la filtre et envoie les pixels de sortie vers un périphérique d'affichage. Une image est faite de LPI lignes de PPL pixels chacune. L'image est mémorisée sous forme de mots (Word) contenant PPW pixels. Le filtrage se fait avec conservation du nombre de pixels. La transformation est définie par le produit scalaire (équation 11) dans laquelle k est l'indice d'un pixel dans une ligne, ip le tableau de pixels d'entrée, op le tableau de pixels de sortie, c un tableau des $2L + 1$ coefficients du filtre numérique.

$$op[k] = \sum_{j=-L}^{j=+L} c[j] * ip[k - j] \tag{11}$$

Cette application est tout d'abord modélisée, sous l'environnement Papyrus UML [1] incluant l'implémentation du profil MARTE. Les contraintes temporelles sont ensuite spécifiées en CCSL avec TimeSquare, qui complète Papyrus. Les évolutions temporelles sont visualisées, analysées et éventuellement modifiées, toujours sous TimeSquare. Une dernière étape, non détaillée ici, mais décrite dans la référence [16] consiste à vérifier formellement la conformité d'une implémentation écrite dans le langage synchrone Esterel avec les spécifications CCSL. Nous allons ici illustrer les phases de modélisation

1. http://www.papyrusuml.org

et analyse des comportements temporels. Les valeurs des paramètres retenues pour les simulations sont $L = 2$, $PPW = 4$ et $PPL = 16$ (valeur choisie très inférieure à la valeur réelle afin de rendre plus aisée la visualisation des chronogrammes).

La modélisation

Modélisation UML On définit les types et les classes de l'application. Les types sont ceux associés aux flots échangés : $Word$ entre mémoire et filtre, $Pixel$ entre filtre et périphérique d'affichage et $Control$. Ce dernier type est un peu particulier car il ne porte pas de valeur. Il correspond à des signaux purs dont la présence signale l'occurrence d'un événement particulier, par exemple la sortie du dernier pixel d'une ligne (signal eol). Les classes sont Memory, Filter et Device. Elles sont munies de ports par lesquels les communications se feront. La classe structurée DigitalFilter représente l'application complète (Figure 5).

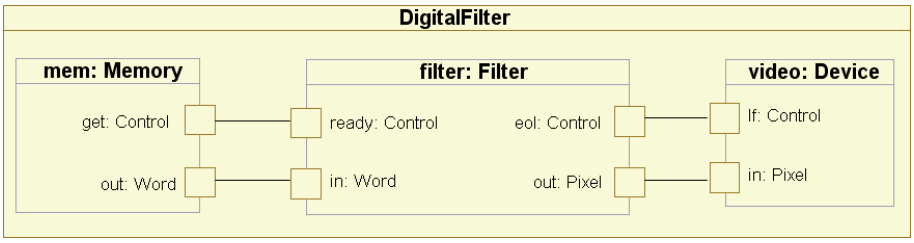

FIGURE 5 – Diagramme de structure

Application du profil MARTE Il s'agit maintenant de préciser la sémantique de certains éléments du modèle. En premier, nous créons des types d'horloge, ce qui se fait en stéréotypant des classes par «clockType». Nous avons choisi de créer un type d'horloge pour chacune des classes qui typent les signaux. La Figure 6 contient une image de la classe Word stéréotypée par «clockType». L'application du stéréotype se fait dans une boîte de dialogue de TimeSquare, ce qui facilite la saisie des valeurs d'attribut pour le stéréotype. Les types d'horloge choisis sont tous logiques à temps discret. Définir des types différents permet de définir des compatibilités et d'éviter de mélanger des horloges qui ne doivent pas l'être.

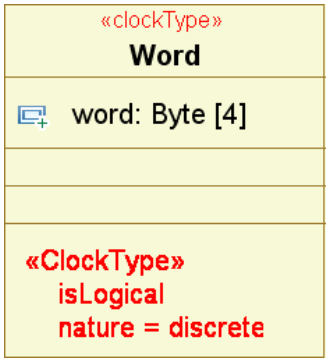

FIGURE 6 – Clock dans le profil MARTE

Des horloges sont ensuite définies. Nous décidons de donner un comportement temporel à chacun des ports, c'est-à-dire de définir les instants de leurs mises à jour. Pour cela, nous associons une horloge logique à chacun des ports (figure 7). En pratique, il suffit d'appliquer le stéréotype «clock» sur les ports. TimeSquare assure automatiquement que l'horloge est correctement typée.

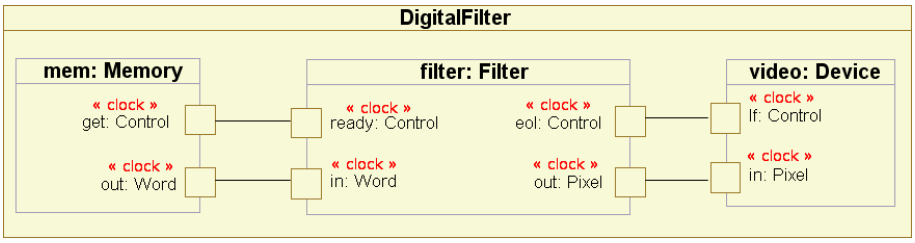

FIGURE 7 – Diagramme de structure avec horloges

Contraintes CCSL Elles permettent d'imposer des contraintes temporelles sur les flots d'information échangés. La figure 9 rassemble les contraintes relatives au filtre sous une forme algébrique qui a l'avantage d'être concise. Il convient de noter que l'utilisateur ne rentre pas les contraintes sous cette forme. Ici encore, des boîtes de dialogues de TimeSquare aident à la spécification (voir la Figure 8).

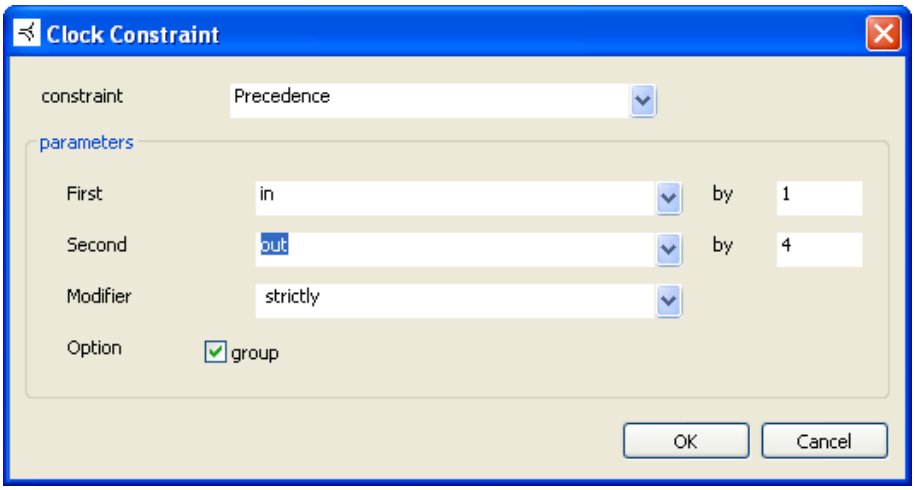

FIGURE 8 – Boîte de dialogue pour la spécification de la contrainte ②

Il existe plusieurs types de contraintes. La contrainte ① est relative au protocole appel-réponse retenu : les occurrences de $ready$ doivent alterner avec celles des mots d'entrée (in). Les contraintes ② et ③ résultent de la structure de données utilisée. La contrainte ② dit que l'émission de quatre pixels de sortie (out) est nécessairement précédée par la réception d'un mot (in) à cause des 4 pixels par mot. C'est une contrainte asynchrone. La contrainte ③ est, par contre, synchrone. Elle indique que les occurrences du signal eol (fin de ligne) coïncident avec la sortie du dernier pixel (out) d'une ligne (ici, le 16e pixel). La contrainte ④ est nettement plus complexe, elle se déduit

$$ready \boxed{\sim} in \qquad ①$$
$$in \boxed{\prec} \left(out/4\right) \qquad ②$$
$$eol \boxed{=} out \blacktriangledown (0^{15}.1)^{\omega} \qquad ③$$
$$in \blacktriangledown \left(0.1^3\right)^{\omega} \boxed{\prec} out \blacktriangledown \left(0^2.(1.0^3)^3.0^2\right)^{\omega} \qquad ④$$
$$out \blacktriangledown \left(0.(1.0^3)^3.0^3\right)^{\omega} \boxed{\prec} in \blacktriangledown \left(0.1^3\right)^{\omega} \qquad ⑤$$

FIGURE 9 – Les contraintes CCSL

du traitement de données réalisé par le filtre (Equation 11). Celui-ci induit des dépendances entre entrées (in) et sorties (out) exprimées par ④. Cette relation n'est qu'une instance d'une relation paramétrique générale modélisant le comportement d'un calcul sur une fenêtre glissante (*sliding window*) de taille $2L+1$, prenant des mots de PPW pixels en entrée, sortant des pixels et ayant WPL mots par ligne. La formulation générale est donnée dans la référence [16]. Les contraintes ① à ④ garantissent que le filtre remplit correctement son rôle. Toutefois le caractère asynchrone de la contrainte ④ autorise de retarder arbitrairement les sorties de pixels. La contrainte ⑤ impose un contrôle de flot supplémentaire sur la sortie des pixels out. Cette contrainte exprime en fait une propriété extra-fonctionnelle sur la limitation de la taille des buffers internes.

Analyse des évolutions temporelles

Évolutions sans la contrainte ⑤ La Figure 10 est une copie d'écran de sortie générée par TimeSquare. Elle montre une évolution possible lorsqu'on n'active que les contraintes de ① à ④. Les instants sont représentés par des impulsions. Les flèches en pointillés visualisent des relations de précédence. TimeSquare sait également visualiser des précédences par paquets en regroupant des instants dans un rectangle à coins arrondis. Quant aux coïncidences, elles sont mises en évidence par des traits verticaux entre instants. Ces renseignements ne sont affichés que sur demande et de manière sélective. Une analyse de cette figure permet de constater que le septième mot entré ne produit des pixels en sortie que bien plus tard. Ce comportement est fonctionnellement correct mais inefficace en termes de mémorisation (*buffers*).

Évolutions avec la contrainte ⑤ Pour la simulation de la Figure 11, la contrainte ⑤ est prise en compte. Le chronogramme montre que les mots d'entrée sont à présent rapidement exploités, c'est-à-dire sans stockage inutile d'information.

Ces analyses de traces d'exécution permettent de trouver des erreurs de conceptions (comportements inattendus) ou des comportements peu performants. Le fait que l'on

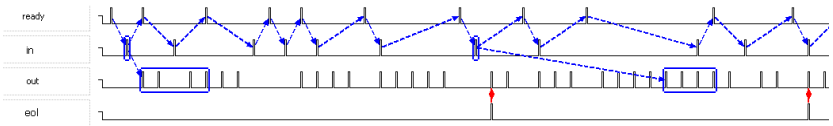

FIGURE 10 – Évolutions temporelles (contraintes ① à ④)

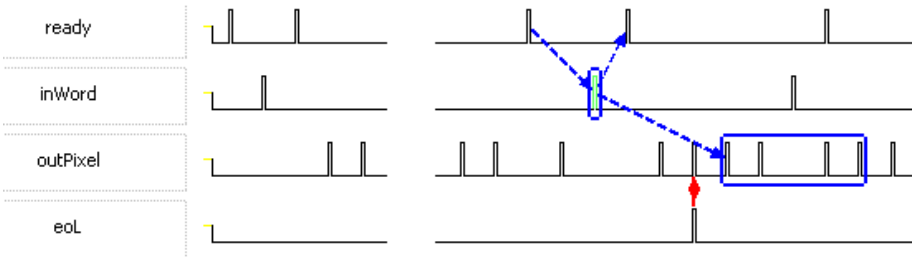

FIGURE 11 – Évolutions temporelles (contraintes ① à ⑤)

travaille uniquement en simulation ne permet pas de prouver formellement des propriétés. Ceci est un autre usage de CCSL traité dans l'article précédemment référencé [16].

6 Conclusion

Nous avons présenté les extensions fournies par le *modèle de temps* de MARTE [1] qui enrichit le modèle de temps simpliste d'UML en facilitant l'ajout d'informations temporelles aux éléments de modèle. Ce profil permet de manipuler les *valeurs* de paramètres temporels tels que la durée d'exécution ou la période. Il peut aussi les contraindre par le biais de *contraintes temporelles*. Ces possibilités sont largement utilisées dans les applications dites *temps réel*. Le profil apparaît alors comme un moyen d'*annoter* des éléments de modèle UML avec des informations temporelles utilisées ultérieurement par des outils dédiés à la simulation, à l'analyse de performance ou à l'analyse d'ordonnançabilité. Ces aspects, qui ont été détaillés dans la section 3, sont relatifs au *temps physique*. Ils sont modélisés par le *temps chronométrique* dans MARTE.

La première innovation importante du modèle de temps du profil MARTE par rapport à UML est la possibilité d'utiliser *plusieurs référentiels temporels* dans un même modèle. Les *horloges* (stéréotype «clock») sont les éléments de modèle donnant accès aux instants de ces référentiels. Le temps modélisé peut être aussi bien discret que dense. En conséquence les horloges sont qualifiées de discrètes ou de denses. Le fait de pouvoir associer explicitement plusieurs horloges à un modèle est une nécessité dans la modélisation des systèmes distribués et des systèmes électroniques multi-horloges. Cette possibilité crée une nouvelle demande : celle d'exprimer les relations possibles entre les horloges. C'est le rôle des *contraintes d'horloges* introduites dans MARTE. La section 4 est une introduction au *langage de spécification des contraintes d'horloges* (CCSL), initialement proposé dans la spécification MARTE. Les profils UML ayant des objectifs pragmatiques, il n'y avait pas lieu de donner une sémantique formelle à ce langage dans la spécification du profil. Ces fondements mathématiques ont été apportés ultérieurement, en dehors des travaux de standardisation. Nous avons donné des fragments d'une *sémantique dénotationnelle* de CCSL. Il existe aussi une sémantique opérationnelle qui est mise en œuvre dans les outils développés autour de CCSL. Cette sémantique n'est pas présentée dans cet article. Elle fait l'objet de rapports spécifiques [10, 12].

Une autre innovation majeure de MARTE est de supporter le *temps logique*. Celui-ci était explicitement exclu du profil SPT [2] (*profile for Schedulability, Performance, and Time Specification*), profil auquel MARTE s'est substitué. Le temps logique, contrairement au temps chronométrique, n'est pas lié au temps physique. En pratique, une *hor-*

loge logique peut être associée à n'importe quel *événement* répétitif : chaque instant de cette horloge discrète coïncide avec une occurrence de cet événement. Au travers de cette association horloge logique/événement, le langage de contrainte CCSL peut être utilisé pour spécifier des contraintes entre occurrences d'événements. Dans le domaine des systèmes embarqués et temps réel, les événements peuvent être liés aux *communications* (envoi ou réception de message), aux *synchronisations* (envoi ou réception de signaux) et plus généralement au *contrôle* (activation ou interruption d'activités). Cet usage de CCSL en modélisation et conception de système numérique a été illustré dans la section 5.

Au-delà d'UML, CCSL est aussi utilisable avec les DSL (*Domain Specific Languages*) et les *Modèles de Calcul et Communication* (MoCC). La modélisation de systèmes modernes, qui sont à la fois complexes et hétérogènes, s'appuie sur l'usage conjoint de différents DSL et/ou MoCC. Pour qu'une telle composition de modèles soit effective, il ne suffit pas de prévoir leur composition structurelle. Il faut également assurer un minimum de cohérence sémantique. Dans un article consacré au *modèle de temps des horloges logiques* [13], CCSL est proposé comme langage pivot pour exprimer des concepts temporels dans différents formalismes et standards. Dans cet article sont étudiés : East-ADL [17] (un langage de description d'architecture dédié au domaine automobile), AADL [18] (*Architecture & Analysis Description Language*, un standard SAE issu du domaine avionique) et SDF [19] (*Synchronous Data Flow*, un MoCC classique). Ce dernier exemple, considère uniquement les aspects temps logique et spécifie en CCSL une sémantique pour SDF.

L'idée que nous défendons est que certains aspects sémantiques doivent être explicites dans un modèle. Les informations temporelles ne peuvent pas se limiter à de simples annotations lorsqu'elles impactent directement la fonctionnalité du système. Les éléments de modèle liés au temps ont une sémantique particulière qui peut être exprimée de façon rigoureuse avec CCSL et se prêter à des analyses et compilations. Un exemple de mise en oeuvre de ces possibilités est la *synthèse de contrôleurs* polychones à partir de spécifications CCSL [20]. Le temps logique utilisé avec CCSL apparaît ainsi comme un *temps pour la conception*, sur lequel on peut effectuer des raffinements/transformations mathématiquement définies. Les relations avec le temps physique, toujours nécessaires dans les systèmes réels, n'apparaissent qu'en fin de conception.

References

[1] OMG. *UML Profile for MARTE, v1.1*. Object Management Group, Août 2010. Document number: PTC/10-08-32.

[2] OMG. *UML Profile for Schedulability, Performance, and Time Specification*. Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701., January 2005. OMG document number: formal/05-01-02 (v1.1).

[3] F. A. Schreiber. Is Time a Real Time? An Overview of Time Ontology in Informatics. *Real Time Computing*, F127:283–307, 1994.

[4] Julius Thomas Fraser. *Time and Time Again, Reports from a Boundary of the Universe*. Koninklijke Brill, 2007.

[5] James F. Allen. Time and Time Again: The Many Ways to Represent Time. *Intl. J. of Intelligent Systems*, 6(4):341–355, July 1991. `ftp://ftp.cs.rochester.edu/pub/u/james/allen-time-again-91.ps.gz`.

[6] Charles André. Le Temps dans le profil UML MARTE. Rapport de recherche ISRN I3S/RR–2007–19–FR, I3S, 07 2007. http://www.i3s.unice.fr/%7Emh/RR/2007/RR-07.19-C.ANDRE.pdf.

[7] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, September 1991.

[8] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1):64–83, January 2003.

[9] A. Cuccuru, C. Mraidha, A. Radermacher, S. Gérard, L. Rioux, T. Vergnaud, and O. Hachet. Methodological Guidelines on the Usage of MARTE VSL for Specification of Time Constraints. 2nd Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2011), Grenoble, France, March 2011. http://www.ecsi.org/sites/default/files/PDF/M-BED2011_paper%2014.pdf.

[10] Charles André. Syntax and semantics of the clock constraint specification language (CCSL). Research Report 6925, INRIA, 05 2009. http://hal.inria.fr/inria-00384077/en/.

[11] Charles André and Frédéric Mallet. Modèles de contraintes temporelles pour systèmes polychrones. *JESA*, 43(7–8–9):725–739, 2009. http://hal.inria.fr/inria-00434462/PDF/jesa09.pdf.

[12] Charles André. Verification of clock constraints: CCSL observers in Esterel. Rapport de recherche RR-7211, INRIA, Feb 2010. http://hal.inria.fr/inria-00458847.

[13] Charles André, Julien DeAntoni, Frédéric Mallet, and Robert de Simone. *The Time Model of Logical Clocks Available in the OMG MARTE Profile*, chapter 7. In Synthesis of Embedded Software. Springer Science + Business Media, June 2010.

[14] Florence Maraninchi and Tayeb Bouhadiba. 42: Programmable models of computation for the component-based virtual prototyping of heterogeneous embedded systems. Technical Report TR-2009-1, Verimag, Centre Équation, 38610 Gières, January 2009. http://www-verimag.imag.fr/index.php?page=techrep-list.

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

[16] Charles André and Frédéric Mallet. Specification and verification of time requirements with CCSL and Esterel. In Christoph Kirsch and Mahmut Kandemir, editors, *Languages, Compilers, and Tools for Embedded Systems ACM SIGPLAN Notices*, volume 44, pages 167–176, Dublin, Ireland, 2009. ACM SIGPLAN/SIGBED. http://hal.inria.fr/inria-00416654/.

[17] The ATESST Consortium. East-ADL2 specification. Technical report, ITEA, March 2008. http://www.atesst.org, 2008-03-20.

[18] P.H. Feiler, D.P. Gluch, and J.J. Hudak. The architecture analysis & design language (AADL): An introduction, 2006.

[19] Edward A. Lee and David G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Computers*, 36(1):24–35, 1987.

[20] Huafeng Yu, J.-P. Talpin, L. Besnard, T. Gautier, H. Marchand, and P. Le Guernic. Polychronous controller synthesis from MARTE's CCSL timing specifications. In *ACM/IEEE Ninth International Conference on Formal Methods and Models for Codesign (MEMOCODE'11)*. ACM, 2011. http://hal.inria.fr/inria-00594942/PDF/memocode11-yu.pdf.

Table des matières

1 Introduction **3**

2 Fondements du modèle de temps de MARTE **3**
2.1 Nature du temps 3
2.2 Représentation du temps 4
2.3 Le Temps dans le profil MARTE 6

3 Temps chronométrique **7**
3.1 Création d'horloges chronométriques 7
3.2 Expression de contraintes temporelles chronométriques 8

4 CCSL **9**
4.1 Généralités 9
4.2 Relations entre instants 10
4.3 Relations d'horloges 11
4.4 Expressions d'horloges 12
4.5 Relations/Expressions définies par l'utilisateur 13
4.6 TimeSquare 13

5 Usage des horloges logiques **14**
5.1 Comment utiliser CCSL 14
5.2 Exemple d'application 14

6 Conclusion **18**

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399