LABORATOIRE

INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR 6070

LE TEMPS DANS LE PROFIL UML MARTE

Charles André

Projet AOSTE

Rapport de recherche
ISRN I3S/RR–2007-19–FR

Juillet 2007

Laboratoire d'Informatique de Signaux et Systèmes de Sophia Antipolis - UNSA-CNRS
2000, rte.des Lucioles – Les Algorithmes – Bât Euclide B – B.P. 121 – 06903 Sophia-Antipolis Cedex – France
Tél.: 33 (0)4 92 94 27 01 – Fax: 33 (0)4 92 94 28 98 – www.i3s.unice.fr
UMR6070

RÉSUMÉ :

Le profil UML pour la Modélisation et l'Analyse des Systèmes Temps Réel et Embarqués, appelé MARTE, a été voté en juin 2007. Cette proposition est une contribution à l'approche de développement dirigée par les modèles pour les applications temps réel embarquées. Une partie importante de MARTE concerne la modélisation du temps. Ce rapport présente les concepts définis dans le sous-profil Time de MARTE, conçu par l'auteur, et illustre l'utilisation de ces concepts.

MOTS CLÉS :

temps, UML, profil, ingénierie des modèles

ABSTRACT:

The UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) was voted at the June 2007 OMG technical meeting. This profile is a contribution to the Model Driven Development (MDD) of real-time embedded applications. Time modeling is one of the main concerns addressed by MARTE. This report presents the concepts defined in the Time sub-profile of MARTE-invented by the author-and illustrates their usage.

KEY WORDS :

time, UML, profile, model driven development

Le temps dans le profil UML MARTE

Charles André
Université de Nice Sophia Antipolis / CNRS
Projet AOSTE (I3S/INRIA)
BP 123 — 06903 Sophia Antipolis cedex
charles.andre@unice.fr

Résumé

Le profil UML pour la Modélisation et l'Analyse des Systèmes Temps Réel et Embarqués, appelé MARTE, a été voté en juin 2007. Cette proposition est une contribution à l'approche de développement dirigée par les modèles pour les applications temps réel embarquées. Une partie importante de MARTE concerne la modélisation du temps. Ce rapport présente les concepts définis dans le sous-profil Time de MARTE—conçu par l'auteur—et illustre l'utilisation de ces concepts.

1. Introduction

L'Object Management Group™ (OMG) a introduit, en novembre 2000, l'approche qu'il a nommée Model Driven Architecture (MDA®) [11]. Le but était de faciliter le développement et la maintenance des systèmes essentiellement logiciels. L'approche MDA spécifie séparément les parties des systèmes indépendantes des plates-formes (PIM ou *Platform Independent Models*) et celles liées aux plates-formes (PSM ou *Platform Specific Models*). La modélisation s'appuie sur les standards de l'OMG et en particulier le langage de modélisation UML® [12, 17] (*Unified Modeling Language*™).

UML a pour ambition de modéliser un large éventail de systèmes relevant de différents domaines d'application. Il va bien au-delà de la modélisation d'applications logicielles. Bran Selic, dans un article [21] sur les fondements sémantiques d'UML 2, résume ainsi la situation : *"In essence, standard UML is the foundation for a 'family' of related modeling languages"*. Cette diversité sous-jacente a une conséquence sur la sémantique du langage : pour être aisément adaptable à des domaines différents, UML introduit de nombreux *points de variation sémantique*, c'est-à-dire que plusieurs interprétations peuvent être proposées pour le même élément de modèle. Le mécanisme de *profil* permet, entre autre, de lever les ambiguïtés dues aux points de variation sémantique. Ce qui est nettement plus gênant c'est qu'en l'absence délibérée de sémantique formellement exprimée, la cohérence entre les différentes vues d'un modèle UML n'est pas garantie.

L'Ingénierie Dirigée par les Modèles (IDM), connue aussi sous l'acronyme anglais MDE (*Model Driven Engineering*), reprend l'idée de base du MDA, mais sans les contraintes imposées par l'OMG. L'ouvrage intitulé *"L'ingénierie dirigée par les modèles. Au-delà du MDA"* [5] précise les contours de cette discipline et analyse son impact potentiel sur les pratiques du Génie Logiciel. Au cœur de cette approche on trouve les concepts de *modèle*, *méta-modèle* et *transformation de modèles*. Les modèles préconisés sont dits *productifs* dans le sens où ils doivent être interprétables et manipulables par une machine. Les modèles productifs s'opposent aux modèles dits *contemplatifs*. Ces derniers peuvent servir de support de communication et de compréhension (aspect essentiellement documentaire) mais ne se prêtent pas à des manipulations et analyses formelles par des machines. Les modèles productifs doivent être exprimés dans un langage clairement défini ; c'est le rôle de la méta-modélisation. Dans l'approche IDM, un *méta-modèle* est un modèle d'un langage de modélisation. Un modèle se doit d'être *conforme à* un méta-modèle.

Le MDA et l'IDM conviennent tous les deux qu'un *modèle* est une abstraction d'un système—une représentation simplifiée—construite avec une intention particulière. Les informations contenues dans le modèles sont pertinentes soit pour des utilisateurs soit pour des usages particuliers. UML précise [17, Chap. 17] qu'un modèle est une description d'un système *physique* dont le but est de décrire des aspects logiques ou comportementaux du système physique pour une catégorie de lecteurs. Cette restriction aux systèmes physiques est quelque peu surprenante quand on connaît la propension d'UML à la généralité. L'IDM, quant à lui, ne fait aucune restriction sur la nature des systèmes modélisés.

Le langage de modélisation SysML [16] (Systems Modeling Language™) s'inscrit aussi dans une approche IDM. Il s'adresse plus spécialement aux systèmes complexes. Ces systèmes peuvent inclure du matériel, du logiciel, des personnes, ... Les domaines technologiques couverts sont l'informatique, l'électronique, la

mécanique, la physique, la chimie, ... SysML est un *profil UML*, donc une spécialisation d'UML, dédiée aux systèmes complexes que l'on rencontre en particulier dans les domaines automobile, avionique, automatique et communication. Comparé à UML, SysML permet une meilleure modélisation des exigences et de la traçabilité (*Requirement diagrams*). SysML est également mieux adapté pour l'expression de contraintes sophistiquées liées à des lois physiques (*Parametric diagrams*).

Les systèmes embarqués sont des systèmes dans lesquels des dispositifs de traitement (microprocesseurs, microcontrôleurs, circuits spécialisés, ...) sont enfouis. Les calculateurs font partie intégrante du système et sont en général invisibles de l'extérieur. SysML paraît *a priori* adapté à la modélisation de ces systèmes. Il existe toutefois deux obstacles majeurs pour que les modèles SysML de systèmes embarqués soient "productifs" : une sémantique insuffisamment formelle—héritage d'UML—et une absence de prise en compte des aspects temporels.

La première lacune va bien au-delà de SysML. Elle tient à la nature même de ces systèmes. Un système embarqué comprend du matériel, du logiciel et un *environnement*. Ce qui le distingue d'autres applications mettant en œuvre des calculateurs, c'est que les traitements sont soumis à des contraintes physiques originaires de l'environnement. Comme le soulignent Henzinger et Sifakis [6], cette influence directe de l'environnement sur le comportement du système fait que le logiciel et la plate-forme d'exécution ne peuvent plus être étudiés séparément. L'approche MDA qui distingue PIM et PSM ne convient donc pas. Le cadre de développement adéquat pour ces applications est encore à trouver [6]. Il devra intégrer des paradigmes issus de la conception des circuits, de la conception des logiciels et de l'automatique (commande/contrôle).

Les contraintes auxquelles sont soumis les systèmes embarqués sont souvent des contraintes de *fonctionnement en temps réel* (respect de contraintes temporelles et exigence de comportements prévisibles). Les contraintes temporelles sont d'autant plus difficiles à spécifier et à analyser que les systèmes embarqués modernes sont souvent des applications réparties. L'aspect réparti n'est pas nécessairement lié à des systèmes de grandes tailles. Les réseaux peuvent être de faible dimension (réseaux embarqués dans une automobile), voire microscopiques (NoC, *Network On Chip*). Edward Lee dans un article provocateur [9] clame que pour traiter correctement les systèmes embarqués il faut réinventer l'informatique. Les modèles actuels intégrent mal les concepts de concurrence et de temps ; ils devraient être exprimés directement dans leur sémantique.

Un profil UML pour les systèmes temps réel et embarqués En février 2005, l'OMG a voté la RFP (*Request For Proposals*) MARTE [14]. Cette demande de propositions portait sur un profil UML pour la modélisation et l'analyse des systèmes temps réel et embarqués (*UML profile for Modeling and Analysis of Real-Time and Embedded systems* en anglais). Un consortium appelé proMARTE a soumis une proposition qui a été adoptée le 29 juin 2007. Cette proposition est alors entrée dans sa phase de finalisation : une FTF (*Finalization Task Force*) a été nommée. La FTF doit assurer le suivi des demandes de corrections soumises à l'OMG et fournir les implémentations du profil.

Le "profil UML pour MARTE" (noté simplement "MARTE" dans la suite) a pour objectif d'étendre UML pour l'utiliser dans une approche de développement dirigé par les modèles de systèmes temps réel et embarqués. MARTE fournit des supports pour les étapes de spécification, de conception et de vérification/validation. Ce profil remplace le profil UML SPT [15] (*UML Profile for Schedulability, Performance, and Time*), qui devait être aligné sur UML 2 et étendu. Les retombées attendues de l'usage de ce profil sont de

- fournir une modélisation unifiée pour les parties matérielles et logicielles du système ;
- permettre l'interopérabilité entre les outils de développement utilisés en spécification, en conception, en vérification et en génération de code ;
- faciliter la construction de modèles sur lesquels on peut faire des prévisions quantitatives tenant compte des caractéristiques du matériel et de logiciel.

Cet article se limite à la présentation de la partie de MARTE qui est relative au temps. Les concepts liés au temps et leurs utilisations sont pour l'essentiel définis dans le sous-profil Time de MARTE.

Plan: La présentation débute par une brève revue de modèles de temps. Nous analysons dans une troisième section les concepts de temps présents dans l'UML. La quatrième section définit le méta-modèle de temps proposé dans MARTE. La définition du sous-profil Time et des éléments de modèles associés fait l'objet de la cinquième section. Des exemples sont ensuite détaillés pour illustrer l'usage des concepts introduits. La conclusion tire des enseignements de cette expérience de construction de profil et dégage nos voies de recherche futures.

2. Modélisation du temps

Le temps joue un rôle important dans les systèmes informatiques et techniques. Cependant, chaque domaine peut avoir sa propre modélisation et interprétation du temps. F. Schreiber [19] décrit, dans un article de synthèse, plusieurs aspects du temps en modélisation et définit des ontologies pour le temps dans différents domaines de l'informatique.

Le temps physique

Le terme de *temps physique* est particulièrement ambigu. Sous ce vocable nous désignerons le temps utilisé

dans les lois de la physique et de la mécanique. Dans le modèle de temps de MARTE, ce temps sera supposé donné par une horloge idéale (idealClk, Section 5.4.2).

Le temps dans les circuits numériques

Dans les systèmes numériques, le temps est perçu au travers de circuits spéciaux, appelés *horloges*. Ces circuits engendrent des signaux "périodiques" (tics d'horloge). Toutefois, les systèmes numériques nécessitent souvent plusieurs horloges, ce qui conduit à des problèmes de synchronisation [10].

La répartition spatiale des systèmes introduit un autre problème : celui de la relativité des observations. Ceci conduit à la difficulté—voire l'impossibilité—de s'accorder sur une date d'occurrence pour un même événement.

Temps logique

Le concept d'*horloge logique* introduit par L. Lamport [8] résout partiellement ce problème en calculant une relation d'ordre totale sur les occurrences d'événement dans le système. L'ordre obtenu n'est pas directement lié au temps physique évoqué plus haut. On parle alors de *temps logique*.

Des améliorations ont été apportées aux horloges logiques afin de mieux caractériser les relations de causalité entre événements [20]. Une caractérisation plus fondamentale liant temps et concurrence est proposée dans les travaux de Carl Adam Petri [18] qui modélise les évolutions des systèmes des réseaux représentant des structures d'événements.

Le temps vu comme un ordre partiel d'instants est une idée reprise dans le modèle sémantique du temps de MARTE (Section 4.5).

Pour des évaluations de performances temporelles ou pour des vérifications de propriétés temps réel, un modèle de temps réduit à un poset [1] d'instants, n'est pas suffisant. Il faut alors introduire des synchronisations d'horloges sur le temps physique (voir par exemple le standard et les services proposés dans *Enhanced View of Time Specification* [13]).

Signalons aussi les travaux de H. Kopetz. Dans son livre [7] intitulé *"Real-Time Systems—Design Principles for Distributed Embedded Applications"* il traite de la modélisation du temps dans les systèmes répartis et introduit TTP (*Time-Triggered Protocols*), modèle auquel son nom est attaché. TTP permet de concevoir des applications temps réel, distribuées et tolérantes aux fautes.

Le temps multiforme

Les langages synchrones [3, 4], utilisés dans la programmation des systèmes réactifs, font aussi usage d'un temps logique. En programmation synchrone, le passage du temps physique est représenté au travers d'occurrences d'événements. Il peut s'agir, par exemple, des émissions d'un signal ayant pour source un générateur d'horloge externe. Toutefois, vis à vis du programme, ces signaux ne se distinguent en rien des autres signaux d'entrée. Ainsi en programmation synchrone on peut exprimer les contraintes suivantes :

```
une tâche doit terminer avant 10 ms;
le véhicule doit s'arrêter en moins de
50 m;
```

Dans ces deux expressions '10 ms' et '50 m' ont des fonctions analogues : elles expriment une notion de *deadline*. Cette possibilité d'assimiler les occurrences de n'importe quel événement répétitif [2] à une base de temps est connue sous le nom de *temps multiforme*. Notons que la notion de temps multiforme est également présente dans les *systèmes hybrides* utilisés en automatique.

Le temps dans les systèmes embarqués

Nous finissons ce tour d'horizon par une rapide analyse des relations temps et *modèles de calcul* dans les systèmes embarqués.

Le temps intervient dans les différentes activités liées au développement d'un système embarqué : modélisation, conception, analyse de performance, analyse d'ordonnançabilité, implémentation, etc. Plusieurs modèles de temps sont ainsi nécessaires. Le concept d'ordre entre instants est commun à de nombreuses représentations du temps : quelque chose se produit avant ou après une autre. L'ordre temporel des activités dans un système peut être représenté de façons différentes suivant le niveau de précision exigé. On distingue trois grandes classes d'abstraction du temps pour représenter des flots comportementaux. Les noms donnés à ces classes changent suivant le domaine considéré.

Modèle causal Dans ces modèles on ne s'intéresse qu'aux dépendances causales. La relation d'ordre n'est que partielle en présence de *concurrence*. Les entités concurrentes interagissent par des *communications* (événements ou signaux). Les communications peuvent être complètement asynchrones, bloquantes (l'émetteur attend une réponse ou un accusé de réception) ou synchronisées (rendez-vous, *hand-shake*, etc).

Modèle synchronisé Dans cette vision du temps on rajoute la notion de *simultanéité*. Le temps est divisé en une succession d'instants discrets. Les dépendances causales peuvent exister à l'intérieur d'un instant. Ceci conduit au concept de *réaction instantanée*, une abstraction largement utilisée dans les modèles et langages synchrones [3]. Lorsque les horloges sont liées à des phénomènes réguliers, les "tics d'horloge" deviennent des unités de temps pour des modèles dit à temps discrets. Cette modélisation est utilisée en matériel (niveau RTL). La propagation instantanée correspond alors aux comportements combinatoires. Elle est aussi utilisée dans les formalismes de simulation (MATLAB®/SIMULINK®

[1] Partially Ordered set.

[2] répétitif mais pas nécessairement périodique. La notion de périodicité sous-entend l'existence d'une métrique.

ou dans les langages de description du matériel tels que SystemC, VHDL, Verilog dans lesquels les δ-cycles représentent des relations causales dans l'instant). Elle est également présente dans les modèles logiciels qui s'appuient sur les sémantiques synchrones (Estérel, SCADE, Signal). Une généralisation du domaine synchrone autorise des couplages plus lâches entre entités synchronisées au travers d'un réseau asynchrone. Ceci conduit au concept de GALS (*Globally Asynchronous, Locally Synchronous*) utilisé dans les modèles au niveau système, par exemple dans la modélisation des systèmes sur puces (*System-On-Chip*).

Modèle en temps "physique" Cette classe d'abstraction du temps s'appuie sur des informations quantitatives précises sur la durée d'exécution d'activités. Ces informations sont exploitées, entre autre, pour la vérification des contraintes temporelles. Elles peuvent également être utilisées par les systèmes synchronisés pour évaluer des performances et optimiser des conceptions.

3. UML et le temps

3.1. SPT

Le profil UML *"Schedulability, Performance, and Time"* (SPT) [15] avait pour objectif de combler les lacunes d'UML 1.4 pour les concepteurs et développeurs d'applications temps réel. SPT introduit des notions facilitant la manipulation du temps et les ressources en UML. Ce profil permet d'annoter les éléments de modèle par des informations *quantitatives* relatives au temps. Ces informations sont ensuite utilisées pour des analyses de performance, d'ordonnançabilité ou de vérification du respect de contraintes temps réel.

SPT ne considère qu'un *temps métrique* qui fait implicitement référence au temps physique. SPT introduit les concepts d'instant (*instant*) et de durée (*duration*), ainsi que ceux d'événements et de stimuli liés au temps. SPT modélise également des mécanismes temporels (*clock* et *timer*) et des services associés (démarrage, arrêt, suspension, reprise).

SPT qui s'appuie sur UML 1.4 devait être ré-aligné sur UML 2. Cet alignement est un des objectifs du profil MARTE qui doit remplacer SPT.

3.2. UML 2 Simple Time

UML 2 a rajouté des métaclasses pour prendre en compte une forme simple (voire simpliste) du temps. Ces métaclasses sont regroupées dans le paquetage SimpleTime qui fait lui-même partie du paquetage CommonBehaviors. Nous décrivons le contenu de ce paquetage, des exemples d'utilisation et nous soulignons les limitations de cette modélisation.

Les concepts couverts dans le paquetage SimpleTime sont

– Le concept d'événement temporel (TimeEvent) ;
– La spécification des instants (TimeSpecification) et des durées (Duration) ;
– Le concept d'observation temporelle (Observation) ;
– L'expression de contraintes temporelles.

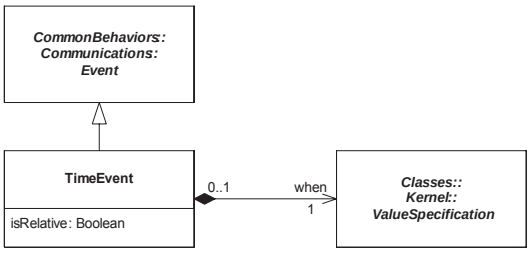

FIG. **1. TimeEvent du paquetage SimpleTime**

TimeEvent Un TimeEvent (Figure 1) est un événement qui spécifie un point dans le temps (un instant). Son attribut booléen isRelative précise si la spécification désigne un temps absolu ou un temps relatif. La spécification elle-même est donnée par la propriété when qui est une ValueSpecification.

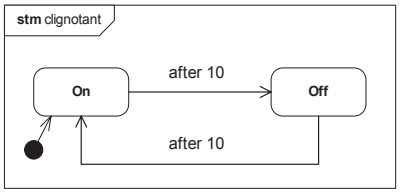

FIG. **2. Machine à état spécifiant le comportement d'un clignotant**

La figure 2 donne un exemple d'utilisation de TimeEvent dans une machine à états d'UML qui spécifie le comportement d'un clignotant. La machine active alternativement les états On et Off toutes les 10 unités de temps. L'étiquette "`after 10`" attachée à la transition de l'état On vers l'état Off indique que le déclencheur (Trigger) de cette transition a pour événement associé un TimeEvent avec isRelative=true. La sémantique est que l'état source doit être quitté 10 unités de temps après y être entré.

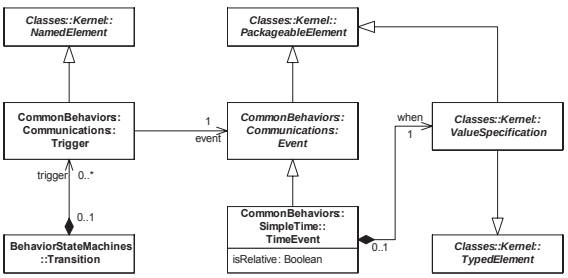

FIG. **3. Métaclasses impliquées dans la modélisation d'un transition déclenchée par un TimeEvent**

Cette simplicité graphique cache une complexité structurelle inhérente à la superstructure d'UML 2.1 [17]. La figure 3 montre différentes métaclasses impliquées dans la modélisation UML d'une transition déclenchée par un TimeEvent. La difficulté majeure vient de la dispersion de ces métaclasses dans de nombreux paquetages. Une métaclasse peut même apparaître dans plusieurs paquetages, ce qui permet leur définition incrémentale. Pour développer correctement des profils UML comme MARTE, il faut être familier avec le méta-modèle UML, ce qui est loin d'être le cas pour les utilisateurs courants.

TimeSpecification et Duration (Figure 4) sont des spécifications de valeurs qui dénotent des instants (pour TimeSpecification) ou des durées (pour Duration). Une valeur peut être spécifiée par une expression faisant intervenir des observations (voir le paragraphe suivant).

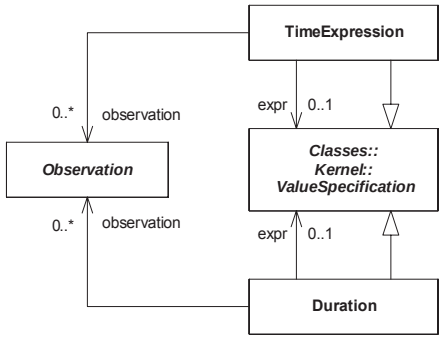

FIG. **4. Spécification de valeurs temporelles**

Observation Observation est une métaclasse abstraite qui généralise TimeObservation et DurationObservation (Figure 5). Son intérêt est de permettre de manipuler les observations relatives aux instants et celles relatives aux durées d'une façon unifiée. L'attribut firstEvent permet, dans les cas les plus simples, de préciser si l'événement observé est lié au début ou à la fin d'une exécution. La définition de cet attribut donnée dans l'UML reste difficile à comprendre et à utiliser dans sa généralité.

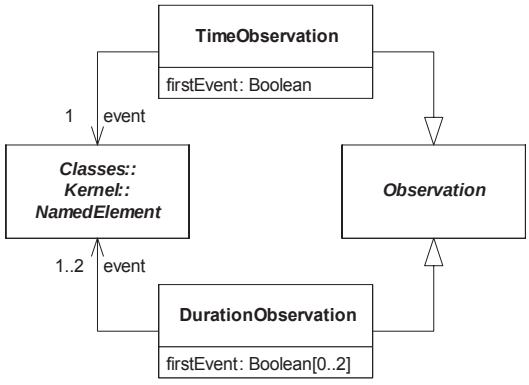

FIG. **5. Spécification des observations temporelles**

Contraintes temporelles Elles sont également définies dans le paquetage SimpleTime. Elles portent sur des instants et/ou des durées. Le paquetage s'est limité aux contraintes dites d'intervalles (appartenance d'un valeur à un intervalle de valeurs).

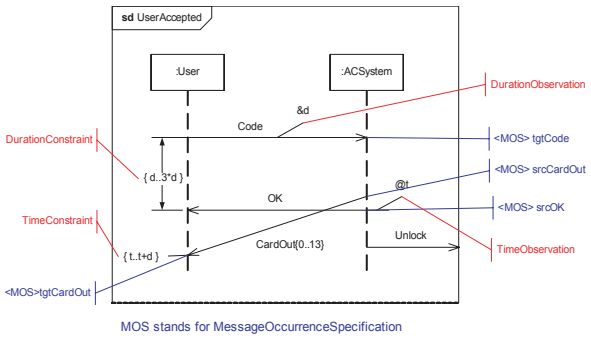

FIG. **6. Diagramme de séquence avec temps (UML 2)**

Pour illustrer l'utilisation des concepts de temps en UML 2, nous adaptons légèrement un exemple donné dans le manuel de spécification d'UML 2 (figure 6). Les annotations en couleur ne font pas partie du modèle ; leur but est seulement de préciser les élements de modèle. Le diagramme de séquence modélise un protocole simplifié d'ouverture de session à l'aide d'une carte. L'utilisateur envoie un *message* nommé Code et d est l'observation de la durée de la transmission. Le système destinataire analyse le code et renvoie deux messages : un message (acceptation dans ce cas) OK et l'éjection de la carte CardOut. L'intervalle de temps entre l'envoi de Code et la réception de OK est contraint : il doit durer entre d et 3*d. Il existe également une contrainte sur l'instant de réception du message CardOut : il doit être compris entre t et t+d. Cette dernière expression montre bien que des observations d'intants (t) peuvent être combinées avec des observations de durées (d). L'expression t+d dénote un instant. La figure 7 représente la contrainte en terme d'instances des métaclasses UML. Ici encore, on constate que la compréhension fine de ces concepts passe par une connaissance approfondie de la structure d'UML et ses métaclasses.

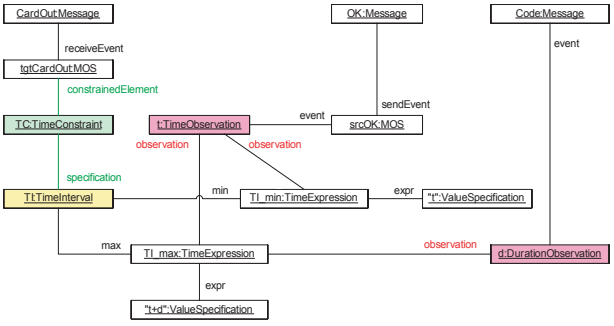

FIG. **7. Modèle d'instance sous-jacent à l'expression d'une contrainte temporelle**

Conclusion sur SimpleTime Le temps reste extérieur à UML : il apparaît indirectement au travers de concepts temporels comme TimeEvent, Observation, TimeExpression, Duration et contraintes temporelles. Notons qu'il n'y a aucune indication précise sur la signification des valeurs temporelles : elles sont exprimées en unité de temps non connues par UML. Ce modèle ne prend pas en compte les effets relativistes que l'on rencontre dans les systèmes répartis. Il ignore également les imperfections des horloges. Il est explicitement dit dans le chapitre CommonBehaviors d'UML 2 que lorsque les phénomèmes mentionnés ci-dessus sont à prendre en compte, il faut utiliser des modèles de temps plus élaborés, fournis par un profil approprié. C'est une des missions du sous-profil Time de MARTE (Section suivante).

4. Le méta-modèle de temps de MARTE

La *vue domaine* (appelée aussi méta-modèle) a pour objectif de spécifier les concepts qui vont être utilisés dans le profil, les relations existant entre ces concepts, ainsi que les contraintes imposées sur leur composition. Ce méta-modèle doit être défini pour répondre aux besoins du domaine, sans se soucier d'UML et des limitations qu'il est susceptible d'imposer (Voir les recommendations de Bran Selic [22] sur l'art et la manière de construire un profil UML).

Dans la vue domaine de MARTE, les concepts de bases sont définis dans le paquetage CoreElements qui est lui-même divisé en deux paquetages : Foundations et Causality. Le premier introduits les concepts d'élément de modèle (ModelElement), de classeur (Classifier) et instance (Instance). Le second traite des concepts comportementaux (CommonBehavior) dans lequel sont introduits les événements, les exécutions et les contextes d'exécution.

Le *domaine de temps* de MARTE réutilise certains de ces concepts. Il a été décomposé en 4 quatre grands sous-domaines

1. TimeStructure qui rassemble les concepts définissant le modèle de temps constitué d'ensembles d'instants, les instants étant partiellement ordonnés.
2. TimeAccess qui regroupe les moyens d'accès à la structure du temps. On y trouve en particulier le concept de Clock.
3. TimeValueSpecification qui permet de dénoter instants et durées.
4. TimeUsage qui introduit les éléments de modélisation couramment utilisés : événements temporels, comportements temporels et contraintes temporelles.

Notre objectif n'est pas ici de décrire dans le détail tous les éléments de modèle que nous avons définis dans le chapitre *Time Modeling* de MARTE. Le lecteur intéressé peut se reporter à la spécification complète de MARTE, disponible sur le site ProMARTE (`http://www.promarte.org`). Le but est plutôt de dégager les contributions essentielles, quitte à simplifier le méta-modèle complet.

4.1. Structure associée au temps

Le modèle de temps retenu dans MARTE (Figure 8) est un ensemble de bases de temps soumis à des contraintes. Une base de temps (TimeBase) est un ensemble totalement ordonné d'instants. A chaque instant est associée une information numérique (date). L'ensemble des instants d'une base de temps peut être *discret* ou *dense*. La vision linéaire (ordre total) du temps apportée par les bases de temps est insuffisante pour la plupart des applications, en particulier pour les applications *multithreads* ou pour les applications réparties. Il faut alors utiliser des bases de temps multiples (MultipleTimeBase) constituées de plusieurs bases de temps. La structure de temps d'une application peut être une hiérarchie (arbre) de bases de temps multiples.

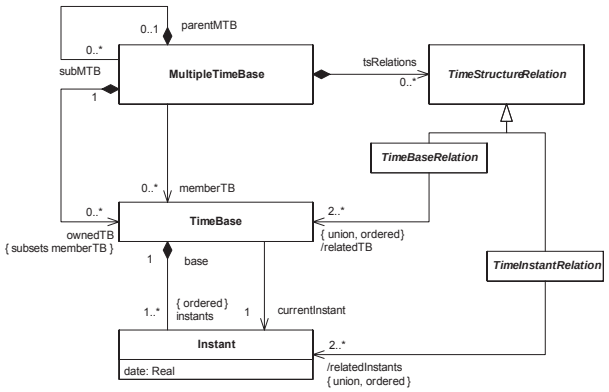

FIG. **8. Structure associée au temps**

Les bases de temps sont *a priori* indépendantes. Elles deviennent dépendantes quand leurs instants se trouvent liés par des relations (relation de coïncidence ou relation de causalité). La classe abstraite TimeInstantRelation a pour sous-classes concrètes les classes CoincidenceRelation et PrecedenceRelation. Pour deux instants appartenant à deux bases de temps distinctes, la première relation indique que ces instants sont coïncidents, alors que la seconde relation indique qu'un instant précède nécessairement l'autre. N'oublions pas que le modèle de temps choisi peut être utilisé lors de la conception du système : la coïncidence n'est donc pas nécessairement interprétée comme un point de l'espace-temps (interprétation relativiste) ; elle peut représenter des points de synchronisations ou de simple choix de conception.

Plutôt que d'imposer des dépendances locales entre instants, on peut imposer directement des dépendances entre bases de temps. Une TimeBaseRelation (ou plus exactement une de ses sous-classes concrètes) spécifie de nombreuses (souvent une infinité de) relations entre instants. Quelques relations entre bases de temps seront données dans les exemples. Une annexe de MARTE

décrit les principales relations entre bases de temps et propose un langage d'expression de ces relations (Clock Constraint Specification Language). Ce langage est brièvement présenté dans la section 5.5.1.

La section 4.5 précise le modèle mathématique associé. Ce modèle ne fait pas partie de la proposition MARTE.

4.2. Accès au temps

Les horloges sont généralement des dispositifs permettant de matérialiser et mesurer la progression du temps physique. MARTE adopte un point de vue plus abstrait : une *horloge* (Clock) est un élément de modèle donnant accès à la structure de temps. Les dispositifs physiques liés au temps sont modélisés dans un autre sous-profil de MARTE (Generic Resource Modeling) : ce sont des TimingResource qui se spécialisent en ClockResource et TimerResource. La figure 9 montre les attributs d'une horloge et ses relations avec d'autres éléments de modèle.

Une horloge fait référence à une base de temps discrète et donc indirectement aux instants de cette base de temps. Une horloge peut accepter différentes unités ; une parmi ces unités joue le rôle d'unité par défaut. On peut également associer à une horloge un événement (clockTick). Les occurrences de cet événement correspondent aux *tics* de l'horloge.

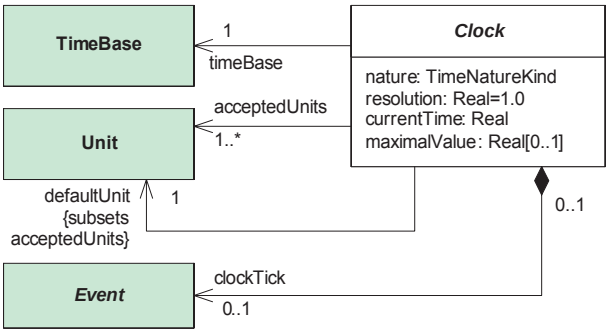

FIG. **9. Accès à la structure de temps : Clock**

Les attributs nature, resolution et maximalValue de Clock caractérisent les valeurs de temps données par l'horloge. La nature peut être discrete ou dense. La résolution indique la granularité de l'horloge (écart minimal entre deux valeurs pour quelles soient distinguables). L'existence d'une valeur maximale correspond au cas où le nombre de valeurs distinctes est fini. Il existe alors un phénomène de repliement des valeurs de temps (les valeurs sont alors généralement données modulo la valeur maximale). Alors que tous ces attributs ont une valeur fixe pour une horloge donnée, l'attribut currentTime change en permanence. Il donne la valeur courante du temps. Bien noter que cette valeur est relative à une base de temps donnée.

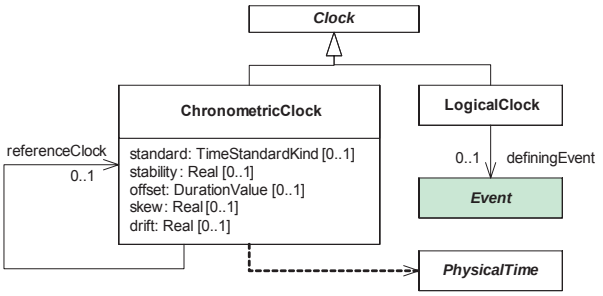

FIG. **10. Horloges logiques et chronométriques**

Clock est une classe abstraite. Il existe deux sous-classes concrètes : LogicalClock et ChronometricClock. Ce qui distingue une horloge logique d'une horloge chronométrique c'est que cette dernière fait référence (souvent implicitement) au temps physique. Une horloge logique peut quant à elle référencer un événement dont les occurrences définissent les instants (les tics) de l'horloge. Il n'y a donc pas nécessité d'une quelconque "régularité" dans les tics d'une horloge logique. Au contraire, à une horloge chronométrique on peut associer des propriétés non-fonctionnelles qui caractérisent ses possibles écarts par rapport à une autre horloge prise comme référence. Les horloges de SPT étaient en fait des horloges chronométiques. Pour avoir plus d'informations sur les propriétés des horloges chronométriques et leurs usages en programmation, consulter le *"Enhanced View of Time Specification"* [13] de l'OMG.

4.3. Spécification des valeurs liées au temps

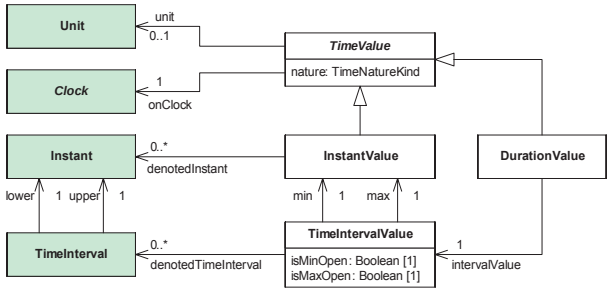

FIG. **11. Valeurs**

Le temps peut être référencé de deux façons en MARTE : soit en tant qu'*instant*, soit en tant que *durée*. Les concepts MARTE correspondant sont respectivement l'InstantValue et la DurationValue. Ces deux classes spécialisent la classe abstraite TimeValue.

Il est important de noter qu'une valeur temporelle (TimeValue) fait nécessairement référence à une horloge. Une *unité* est attachée à une valeur temporelle. Cette unité est une des unités acceptées par l'horloge référencée par la valeur temporelle. Si l'unité n'est pas précisée (propriété unit optionnelle dans le diagramme figure 11), il convient de prendre l'unité par défaut de l'horloge. L'attribut nature de TimeValue précise si les valeurs sont prises

dans un ensemble dense ou discret. Cette distinction est essentiellement sémantique. D'un point de vue pratique les valeurs manipulées par les modèles MARTE restent discrètes, même celles dites "réelles". Notons également que dans MARTE, les valeurs temporelles associées aux horloges logiques sont toujours discrètes.

Une InstantValue dénote 0 ou plusieurs instants de la base de temps associée à l'horloge référencée par la valeur. L'existence possible de plusieurs instants est due à la nature finie des représentations et au "repliement d'horloge" évoqué dans la section 4.2. Une TimeIntervalValue est spécifiée par la donnée de deux valeurs d'instant (min et max) qui doivent référencer la même horloge. Ces valeurs appartiennent ou non à l'intervalle suivant les valeurs prises par les attributs booléens isMinOpen et isMaxOpen.

Une DurationValue spécifie "l'extension" d'un intervalle de temps. La durée est liée à une notion de distance entre instants.

Bien que InstantValue et DurationValue puissent référencer les mêmes horloges et porter les mêmes unités, MARTE les distingue d'un point de vue sémantique. Il n'est en effet pas possible de combiner ces valeurs arbitrairement comme l'indique le tableau ci-dessous :

InstantValue	$\pm$	DurationValue	$\rightarrow$	InstantValue
DurationValue	$+$	InstantValue	$\rightarrow$	InstantValue
DurationValue	$-$	InstantValue		non défini
DurationValue	$\pm$	DurationValue	$\rightarrow$	DurationValue
InstantValue	$-$	InstantValue	$\rightarrow$	DurationValue
InstantValue	$+$	InstantValue		non défini

TAB. **1. Combinaisons de valeurs d'instant et de durée**

4.4. Les entités liées au temps

TimedElement: Le concept d'élément temporel (TimedElement, figure 12) est très général. Il exprime qu'un élément de modèle est lié au temps via une ou plusieurs horloges. La métaclasse correspondante est abstraite. Les spécialisations concrètes de cette métaclasse précisent la sémantique de leur association avec les horloges.

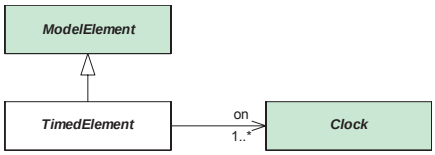

FIG. **12. TimedElement**

TimedEvent: On peut associer des instants aux occurrences d'un événement. MARTE introduit le concept d'occurrence d'événement temporel (TimedEventOccurrence, figure 13) qui est à la fois un élément temporel et une occurrence d'événement. La propriété at spécifie la valeur d'instant d'une occurrence de l'événement considéré. Puisque plusieurs horloges peuvent être référencées (propriété on de l'élément temporel), plusieurs valeurs d'instants sont possibles pour la même occurrence. Généralement on ne retient qu'une seule horloge, mais il n'est pas interdit d'en considérer plusieurs.

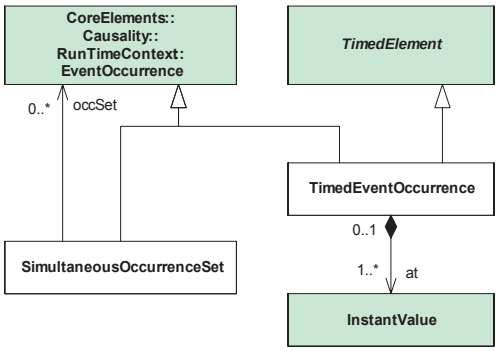

FIG. **13. Occurrences d'événements temporels**

MARTE introduit un concept nouveau : celui d'ensemble d'occurrences simultanées (SimultaneousOccurrenceSet). Ce concept est indispensable quand différents événements doivent être considérés collectivement car leurs effets ne peuvent pas être réduits à la sérialisation des effets de chacun. Un ensemble d'occurrences simultanées *est une* occurrence (EventOccurrence est une généralisation de SimultaneousOccurrenceSet). D'un point de vue UML, cet ensemble d'occurrences simultanées peut donc causer une exécution de comportement. Ceci est utile lors de la conception : différents événements correspondant à un même événement ont pu être introduits dans différentes vues. Il faut alors les unifier. La notion d'occurrences simultanées est aussi d'usage courant en modélisation réactive synchrone [3, 4].

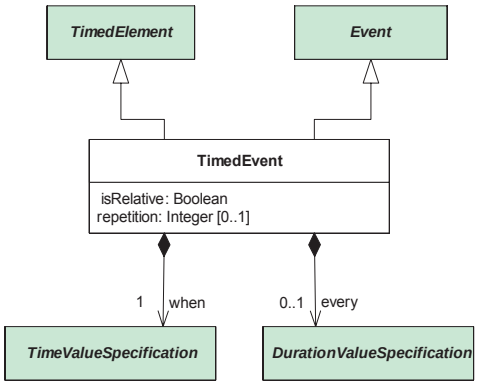

FIG. **14. Événements temporels**

Un événement temporel (TimedEvent) est un événement dont les occurrences sont liées au temps via une horloge (Figure 14). Les occurrences d'un événement temporel sont spécifiées par les propriétés attachées au TimedEvent. L'attribut isRelative indique si

la valeur temporelle donnée par le when est relative ou absolue. La propriété when est une valeur temporelle qui précise l'instant d'occurrence. Cette valeur est une valeur d'instant s'il s'agit d'un temps absolu ; c'est une durée dans l'autre cas. La propriété optionnelle every permet de caractériser les éventuelles occurrences suivantes du même événement. Lorsqu'elle est définie, la propriété every est la valeur de durée qui sépare deux occurrences successives. L'attribut optionnel repetition permet, si nécessaire, de limiter le nombre d'occurrences.

Comparé au TimeEvent d'UML 2, le TimedEvent est explicitement lié à une horloge et il permet de spécifier non seulement la première occurrence de l'événement mais aussi ses occurrences suivantes. Ceci est très commode dans la modélisation d'événements dont les occurrences sont périodiques.

TimedProcessing: Une exécution temporelle (TimedExecution) est un élément de modèle qui spécialise une exécution de comportement (BehaviorExecution). En tant qu'élément temporel (figure 15) une exécution temporelle fait référence à une ou plusieurs horloges. Deux valeurs d'instants startInstant et finishInstant sont associées à une exécution. On peut aussi caractériser une exécution par sa durée. Nous assimilons aussi une transmission de message à une exécution ; la valeur d'instant de début est alors celle de l'instant d'émission, la valeur d'instant de fin étant celle de l'instant de réception.

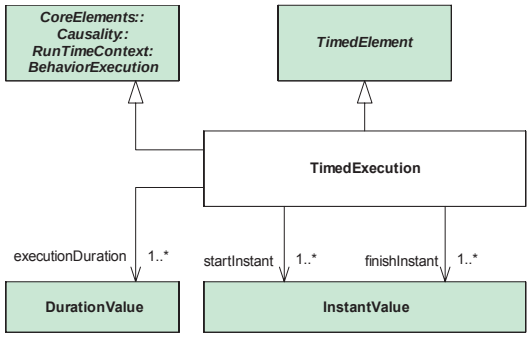

FIG. **15. Exécutions temporelles**

Le traitement temporel (TimedProcessing, figure 16) est un concept générique pour modéliser des activités (au sens large, pas au sens restreint donné par UML) qui ont des instants de début et de fin connus ou une durée connue. De fait, la donnée de deux de ces trois informations est suffisante pour caractériser une exécution particulière. Ce concept s'applique immédiatement aux comportements (TimedBehavior). Il s'étend facilement aux actions (TimedAction) qui ne sont pas en UML des comportements, mais des nœuds d'activité primitifs dont l'exécution entraîne une modification de l'état du système ou le retour d'une valeur. Il s'étend également aux messages (TimedMessage) ; les événements de début et de fin sont nommés événements d'émission et de réception respectivement. Le retard (Delay) est une action temporelle particulière qui représente une action ne faisant rien mais qui a une durée.

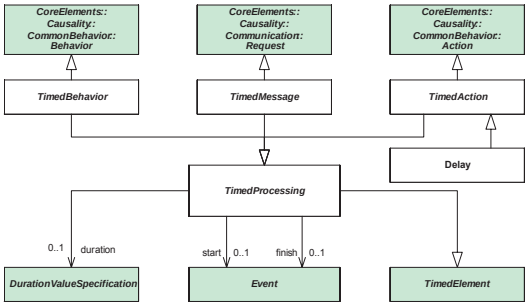

FIG. **16. Traitements temporels**

TimedObservation: Le concept d'observation temporelle (TimedObservation) enrichit et précise celui introduit dans le SimpleTime d'UML 2 (section 3.2). Une observation temporelle (figure 17) est un TimedElement. Elle est donc associée explicitement à une ou plusieurs horloges. Une observation temporelle peut référencer une exécution d'un système ou d'une partie d'un système (CompBehaviorExecution) qui sert de contexte pour cette observation.

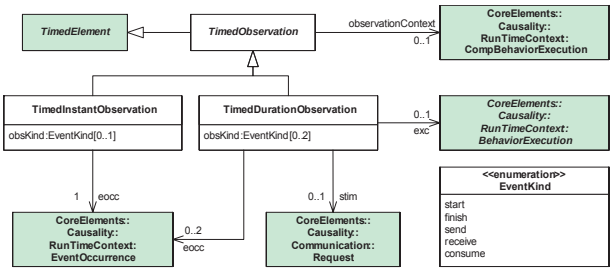

FIG. **17. Observations temporelles**

TimedObservation est une superclasse abstraite pour les observations d'instants (TimedInstantObservation) et les observations de durées (TimedDurationObservation). L'attribut obsKind permet de spécifier le type d'événement considéré dans l'observation temporelle. Pour un comportement, les événements observés peuvent être le début (start) ou la fin (finish) d'une exécution. Pour une requête, les événements possibles sont son émission (send), sa réception (receive) ou sa prise en compte par le récepteur (consume).

Puisqu'une observation d'instant dénote un instant, on observe une occurrence d'événement sur une horloge donnée (propriété eocc). Dans le cas d'une observation de durée, il existe plus de possibilités :

- on peut observer la durée d'une exécution (propriété exc), les événements sont le début et la fin de cette exécution. Une seule horloge est alors considérée.
- on peut observer la durée d'une requête, ou plus exactement de sa transmission ou le retard de sa prise en compte (propriété stim). Les horloges d'émission et de réception peuvent éventuellement être différentes.

– on peut, enfin, observer la durée entre deux événements quelconques, qui peuvent être observés sur des horloges différentes.

Dans le cas où les observations se font sur des horloges différentes il faut être bien conscient des effets relativistes possibles.

TimedConstraint: Une contrainte temporelle (TimedConstraint) est à la fois un élément temporel et une contrainte imposée sur les occurrences d'un événement (TimedInstantConstraint) ou sur la durée d'une exécution ou même sur la distance temporelle entre deux événements (TimedDurationConstraint). Les contraintes sont spécifiées par des prédicats qui contiennent des usages d'observations temporelles. Une forme courante de prédicat est l'appartenance d'une valeur de temps à un intervalle. C'est la seule forme de contrainte que supporte le SimpleTime d'UML. Dans MARTE des contraintes plus sophistiquées sont utilisables. Nous en donnerons certaines dans la section 5.5. De plus MARTE fait explicitement référence à des horloges dans l'expression de ses contraintes.

4.5. Modèle mathématique

Le modèle mathématique pour une horloge est un quintuplé $\langle \mathcal{I}, \preccurlyeq, \mathcal{D}, \lambda, u \rangle$ avec $\mathcal{I}$ un ensemble d'instants, $\preccurlyeq$ une relation d'ordre sur $\mathcal{I}$, $\mathcal{D}$ est un ensemble d'étiquettes, $\lambda : \mathcal{I} \rightarrow \mathcal{D}$ est une fonction d'étiquetage, u un symbole représentant une *unité*. Pour une horloge chronométrique, l'unité peut être la seconde (s), unité de temps du système international (SI) ou une unité dérivée (ms, us, mn, etc.). L'unité la plus employée pour les horloges logiques est le tick, mais clockCycle, executionStep sont aussi courantes. Les temps multiformes peuvent utiliser une unité physique quelconque, comme c'est le cas dans l'application automobile décrite au paragraphe 6.2. Puisque l'horloge fait référence à une base de temps, l'ensemble $\mathcal{I}$ est un oset [3] $\langle \mathcal{I}, \prec \rangle$ avec $\prec = \preccurlyeq \setminus Id$ et Id est la relation identité.

Une *structure de temps* (TimeStructure) est un quadruplé $\langle \mathcal{C}, \mathcal{R}, \mathcal{D}, \lambda \rangle$ avec $\mathcal{C}$ un ensemble d'horloges, $\mathcal{R}$ une relation sur $\bigcup_{a,b \in \mathcal{C}, a \neq b} (\mathcal{I}_a \times \mathcal{I}_b)$, $\mathcal{D}$ est un ensemble d'étiquettes, $\lambda : \mathcal{I}_{\mathcal{C}} \rightarrow \mathcal{D}$ est une fonction d'étiquetage. $\mathcal{I}_{\mathcal{C}}$ est l'ensemble des instants de la structure de temps. $\mathcal{I}_{\mathcal{C}}$ n'est pas la simple union des ensembles d'instants de toutes les horloges. Il est en fait le quotient de cet ensemble par la relation de coïncidence induite par la relation $\mathcal{R}$. La structure de temps spécifie un strict poset $\langle \mathcal{I}_{\mathcal{C}}, \prec_{\mathcal{C}} \rangle$: les instants sont partiellement ordonnés.

Ce modèle mathématique donne une sémantique formelle au temps. Il ne fait pas partie de la spécification de MARTE. Il est toutefois indispensable pour spécifier formellement les relations entre horloges et permettre leur analyse et transformations.

[3] Ordered set

5. Le sous-profil Time de MARTE

5.1. Généralités

Cette section décrit les extensions apportées à UML pour pouvoir prendre en compte les concepts introduits dans la vue domaine du temps. Pour certains concepts on a choisi d'étendre des métaclasses d'UML par des stéréotypes. Pour d'autres on a recours à des spécialisations de stéréotypes définis dans d'autres sous-profils de MARTE. Pour d'autres encore, aucune extension n'est nécessaire.

Les stéréotypes définis dans le sous-profil Time sont volontairement limités en nombre et ils étendent principalement des métaclasses définies dans les paquetages UML::Classes::Kernel et UML::CommonBehaviors.

La figure 18 indique les dépendances du sous-profil Time avec d'autres sous-profils et bibliothèques de MARTE. VSL est un sous-profil de MARTE qui définit un langage (Value Specification Language) pour exprimer, en particulier, les valeurs de propriétés non fonctionnelles. NFPs est le sous-profil de MARTE qui facilite l'accès (déclaration et usage) de propriétés non fonctionnelles. La bibliothèque TimeTypesLibrary contient des énumérations relatives au temps. Quant à la bibliothèque TimeLibrary, c'est une bibliothèque utilisateur. Leurs contenus seront détaillés dans la section 5.4.

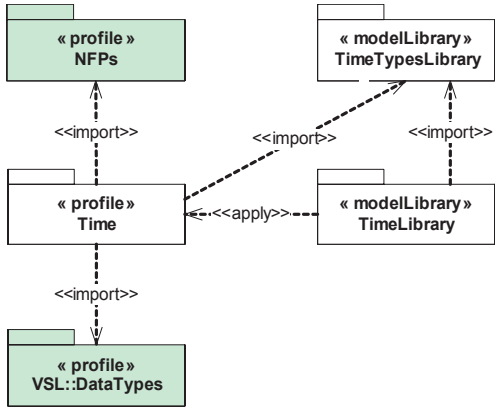

FIG. **18. Dépendances du sous-profil Time**

5.2. Représentation UML

TimedElement et Clock: Les concepts d'instant, de base de temps et de bases de temps multiples constituent le modèle sémantique du modèle de Temps de MARTE. Dans le sous-profil Time, le stéréotype TimedDomain correspond au concept MultipleTimeBase. Le stéréotype ClockType correspond au concept TimeBase et partiellement au concept Clock. Les instants ne sont pas explictement représentés en tant qu'élément de modèle dans le profil. Le stéréotype TimedElement est particulier : il n'étend aucune métaclasse. Ceci ne pose aucun problème car ce stéréotype est *abstrait*. Son intérêt est de factoriser la propriété on qui associe une ou plusieurs Clock à un élément temporel.

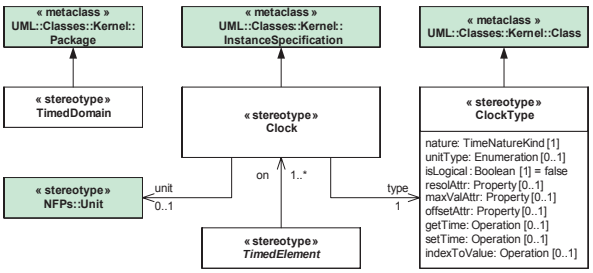

FIG. **19. Clock dans le sous-profil Time**

Les diagrammes tels que celui de la figure 19 ne sont pas suffisants pour la spécification complète du profil. Il faut également des règles de bonne structuration, exprimées de préférence en OCL. Par exemple, une règle impose que le stéréotype Clock soit appliqué à une instance d'une classe elle-même stéréotypée par ClockType. Une autre règle indique que la propriété unit de Clock soit un EnumerationLiteral appartenant à l'énumération spécifiée pour le méta-attribut unitType lors de l'application du stéréotype ClockType. La section 6 contient des exemples d'application de ces stéréotypes.

Notons que le profil exploite la possibilité offerte par UML 2 de typer les attributs des stéréotypes par n'importe quelle métaclasse UML, par exemple Property et Operation. Ceci permet de modifier la sémantique des propriétés et opérations de la classe de base.

TimedValueSpecification: Pour l'expression des valeurs temporelles, nous avons créé le stéréotype TimedValueSpecification de la métaclasse UML::Classes::Kernel::-ValueSpecification. Ce stérotype spécialise le stéréotype TimedElement, ce qui impose qu'une spécification de valeur temporelle fasse référence à au moins une Clock. Le méta-attribut interpretation précise si la valeur dénote un instant ou une durée. La ValueSpecification d'UML offre une grande liberté dans la forme d'expression de la valeur. La syntaxe n'est pas fixée dans UML mais dans l'outil de mise en œuvre d'UML. On voudrait pouvoir spécifier les valeurs temporelles par des expressions comme "`15.2 ms on prClk`", où `prClk` est une Clock. Des possibilités ont été offertes pour cela dans MARTE. Nous reviendrons plus en détail sur ces problèmes dans la section 5.5.

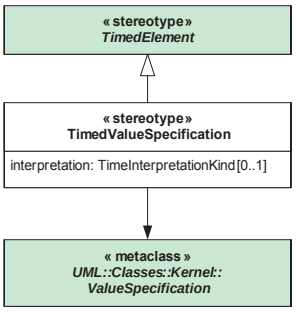

FIG. **20. TimedValueSpecification dans le sous-profil Time**

TimedEvent: Ce stéréotype permet de spécifier des événements dont les occurrences sont explicitement liées à une horloge. TimedEvent est un TimedElement avec une règle imposant qu'il soit associé à une seule Clock. Le stéréotype TimedEvent étend la métaclasse TimeEvent du paquetage UML::CommonBehaviors::SimpleTime. La classe de base apporte les propriétés isRelative et when ; le stéréotype fournit les propriétés every et repetition (revoir la section 4.4 pour la signification des ces propriétés).

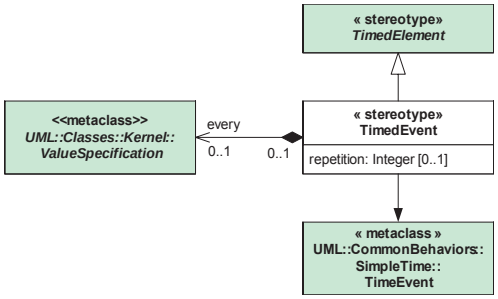

FIG. **21. TimedEvent dans le sous-profil Time**

TimedProcessing: Ce stéréotype représente des activités qui sont explicitement liées à une horloge. Les activités peuvent être des comportements, des actions ou des transmissions de messages. Le stéréotype TimedProcessing étend donc différentes métaclasses UML : Behavior, Action, and Message. Rappelons qu'en UML 2, les machines à états, les activités (UML) et les interactions sont des comportements. On peut donc appliquer le stéréotype TimedProcessing à une StateMachine, une Activity ou une Interaction.

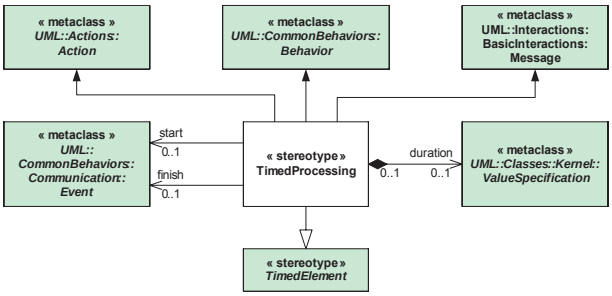

FIG. **22. TimedProcessing dans le sous-profil Time**

TimedObservation: Le stéréotype TimedObservation généralise les stéréotypes TimedInstantObservation et TimedDurationObservation. Il permet donc de référencer ces deux types d'observation. Une TimedObservation est également un TimedElement ce qui la lie explicitement à une ou plusieurs horloges. Le stéréotype TimedInstantObservation (TimedDurationObservation, respectivement) étend la métaclasse TimeObservation (DurationObservation, respectivement) du paquetage UML::CommonBehaviors::SimpleTime. L'attribut obsKind

du stéréotype précise le ou les événements concernés par l'observation.

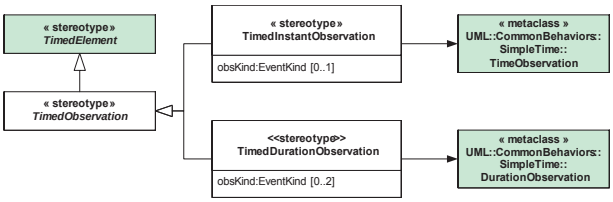

FIG. **23. TimedObservation dans le sous-profil Time**

TimedConstraint et ClockConstraint: Le sous-profil Time introduit deux stéréotypes pour modéliser les contraintes liées au temps. Ces deux stéréotypes spécialisent d'une part TimedElement et d'autre part NfpConstraint défini dans le sous-profil NFPs de MARTE. En tant que TimedElement, ces stéréotypes font explicitement référence à des horloges. En tant que NfpConstraint (un stéréotype de la métaclasse UML : :Classes : :Kernel : :Constraint), ils possèdent un attribut qui précise si la contrainte est requise (required), fournie (offered) ou fait l'objet d'un contrat (contract). Le stéréotype TimedConstraint contraint des valeurs temporelles ; l'attribut interpretation indique s'il s'agit d'une valeur d'instant ou d'une valeur de durée. Le stéréotype ClockConstraint contraint des instants ou des horloges. Il permet donc de modéliser les TimeStructureRelation introduites dans la vue domaine.

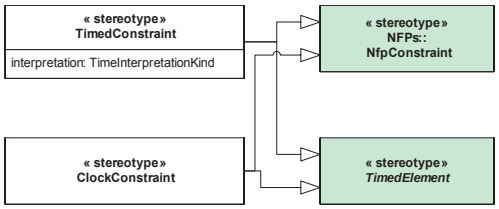

FIG. **24. TimedConstraint et ClockConstraint dans le sous-profil Time**

5.3. Autres stéréotypes relatifs au temps

Le sous-profil GRM (Generic Resource Modeling) fait partie, comme Time, des sous-profils de base de MARTE. GRM définit les concepts nécessaires pour modéliser les plates-formes d'exécution pour les applications temps réel embarquées. Le concept de Resource y est introduit. Il s'agit d'une entité physique ou logique persistante. Une ressource fournit une ou plusieurs services. En MARTE toute ressource peut référencer des horloges (propriété referenceClocks), permettant ainsi de caractériser des services dépendant explicitement du temps. Il est également défini un type de ressource particulier : TimingResource qui spécialise ClockType de Time (Figure 25).

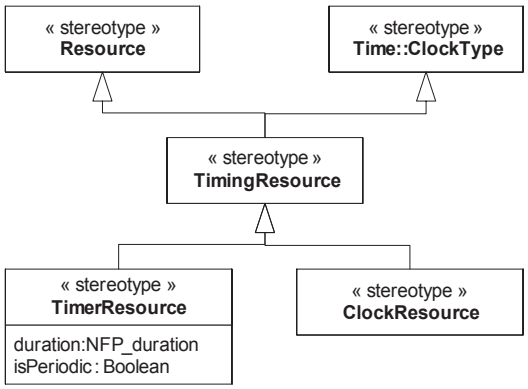

FIG. **25. Stérotypes de GRM liés au temps**

Une TimingResource représente une entité matérielle ou logicielle qui suit et qui indique le passage du temps physique. Ce concept remplace celui de *TimingMechanism* qui était défini dans SPT. Il existe deux spécialisations de ce concept : les stéréotypes ClockResource et TimerResource. En tant que ressources, ces stéréotypes offrent des services, ce qui n'était pas le cas des horloges de Time. Les ClockResource et TimerResource peuvent être démarrés, arrêtés, suspendus, relancés, ... Ils jouent un rôle important dans les systèmes d'exploitation temps réel et comme "horloges physiques" (entités, souvent physiques, gérant le temps dans des applications).

5.4. Les bibliothèques relatives au temps

5.4.1 TimeTypesLibary

La bibliothèque TimeTypesLibary contient les énumérations utilisées dans le sous-profil Time. Ces énumérations sont également utilisables dans les modèles utilisateurs.

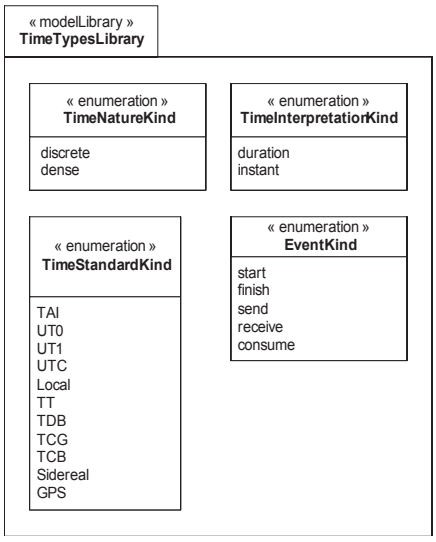

FIG. **26. Bibliothèque des types pour le temps (TimeTypesLibrary)**

L'énumération TimeStandardKind (figure 26) est la seule qui n'ait pas été décrite dans la vue domaine. Elle contient des littéraux qui désignent les divers "systèmes de temps" standards. Les systèmes de temps indiquent le type de dépendance existant entre le temps (chronométrique) considéré et le temps idéal. Ils sont définis par des standards internationaux. Une présentation plus complète est disponible sur le site *Systems of Time*[4].

5.4.2 TimeLibary

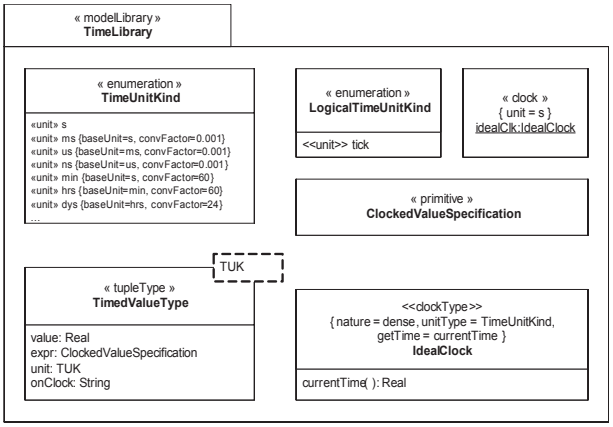

FIG. **27. Bibliothèque pour le temps (TimeLibrary)**

La bibliothèque utilisateur TimeLibrary introduit des unités de temps, des types complexes nouveaux, un type d'horloge et une horloge prédéfinis.

L'énumération TimeUnitKind contient les unités de temps pour les temps chronométriques. La liste des littéraux n'est pas complète dans la figure 27. On notera que chaque EnumerationLiteral est stéréoptypé par «unit». Le stéréotype Unit est défini dans le sous-profil NFPs. Il permet d'une part de distinguer les unités (non limitées aux unités de temps) des autres littéraux d'énumération et d'autre part d'exprimer d'éventuels facteurs de conversion entre unités. L'énumération LogicalTimeUnitKind est assez particulière : elle contient un seul littéral (tick) stéréotypé «unit». Cette unité est généralement utilisée pour les temps logiques. L'utilisateur peut aussi définir ses propres unités de temps (voir l'exemple de la section 6.2).

Le type générique TimedValueType permet de définir des types de valeurs temporelles. Un type nouveau est créé en substituant une énumération contenant des unités au paramètre formel TUK (template binding). Le type lui-même est composé et défini comme un n-uplé. Le stéoréotype TupleType, défini dans le sous-profile VSL::DataTypes, facilite ces créations de types composés. Dans un TimedValueType, le champ onClock du n-uplé contient une chaîne de caractères qui est le *nom* d'une horloge. Le champ expr contient une ClockedValueSpecification qui est une «primitive», c'est à dire un type de donnée primitif. Les valeurs correspondantes sont donc interprétées hors UML. En MARTE, des langages spécialisés ont été proposés pour exprimer ces valeurs (section 5.5).

IdealClock est une classe UML stéréotypée par «clockType». Les valeurs données aux méta-attributs (tagged-values) lors de l'application du stéréotype font que IdealClock type des horloges chronométriques, pour un temps dense, et dont les unités sont des TimeUnitKind (la seconde du Système International et ses dérivées). idealClk est une instance de IdealClock. Cette horloge est supposée *idéale* c'est à dire qu'elle suit fidèlement les évolutions du temps tel que celui utilisé dans les lois de la physique ou de la mécanique. Son unité est la seconde SI. idealClk sert d'horloge de référence pour l'utilisateur lorsqu'il veut définir ses propres horloges chronométriques qui ne sont pas nécessairement parfaites.

5.4.3 BasicNFP_Types

Comme son nom l'indique, la bibliothèque BasicNFP_Types de MARTE propose aux utilisateurs un ensemble de types de donnée applicables à des propriétés non fonctionnelles variées. Trois de ces types de données sont directement liés à la notion de temps : NFP_DateTime, NFP_Duration et NFP_Frequency. Le premier concerne l'expression des dates et heures suivant divers formats ; il n'est donc pas d'un grand intérêt pour les systèmes temps réel. Les deux autres permettent d'exprimer des durées ou des fréquences considérées comme des propriétés non fonctionnelles. Ces types de donnée se révèlent très utiles lorsqu'on désire faire des analyses de performance temporelles ou d'ordonnançabilité.

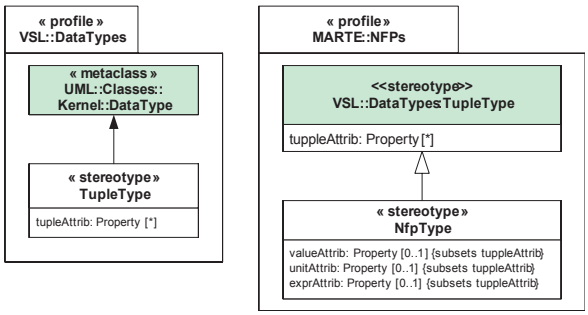

FIG. **28. Stéréotypes TupleType et TNfpType**

Nous allons expliquer comment le type de donnée NFP_Duration a été défini dans MARTE. Cet exemple est significatif de la difficulté qui existe à organiser des profils complexes comme MARTE. La figure 28 contient des extraits des diagrammes définisant les sous-profils VSL::DataTypes et NFPs. Le stéréotype TupleType étend la métaclasse DataType dans le sous-profil VSL::DataTypes. Le sous-profil NFPs spécialise ensuite le stéréotype TupleType pour donner le stéréotype NfpType.

L'étape suivante consiste à définir le type de donnée NFP_CommonType. Ceci est fait dans la bibliothèque utilisateur MARTE_Library::BasicNFP_Types. Le sous-profil

[4] http://tycho.usno.navy.mil/systime.html

NFPs est appliqué à cette bibliothèque, ce qui permet de stéréotyper le dataType NFP_CommonType par NfpType. Des sous-classes de NFP_CommonType sont ensuite définies. La figure 29 contient les types de donnée relatifs au temps. On constate que par héritage, le type de donnée NFP_Duration contient un grand nombre d'attributs.

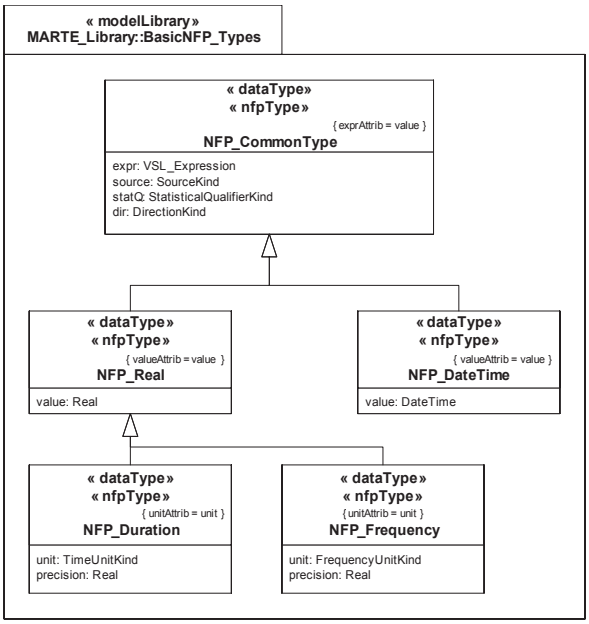

FIG. **29. Types pour propriétés non fonctionnelles temporelles**

5.5. Les langages spécialisés

Ces langages facilitent l'utilisation du profil MARTE.

5.5.1 Expression des valeurs temporelles

Avec MARTE on peut spécifier des valeurs de trois façons :

1. en stéréotypant des valeurs par «TimedValueSpecification»,
2. avec le langage CVSL (Clocked Value Specification Language),
3. avec la partie du langage VSL (Value Specification Language) dédiée aux expressions temporelles.

Ces trois possibilités répondent à des exigences et utilisations différentes.

TimedValueSpecification: C'est un façon simple de spécifier une valeur temporelle. On part d'une ValueSpecification UML et on lui applique le stéréotype TimedValueSpecification. Lors de cette application on précise l'horloge (méta-attribut on) et optionnellement l'interprétation à donner à la valeur : instant ou durée (méta-attribut interpretation). La figure 30 contient un extrait de modèle emf-uml2. LiteralInteger est une sous-classe de la classe abstraite ValueSpecification. L'entier littéral 15 est stéréotypé «timedValueSpecification» avec les valeurs `pr` pour l'attribut de stéréotype on et `duration` pour l'attribut interpretation.

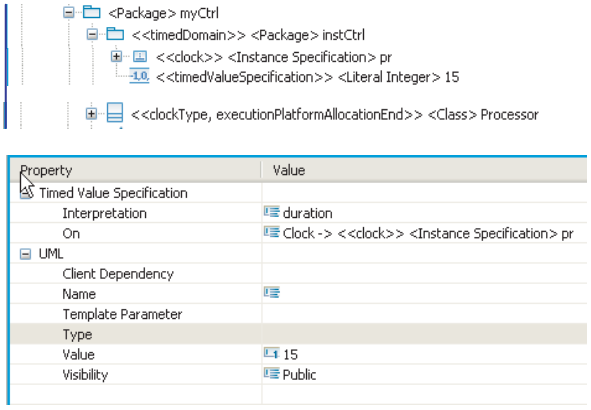

FIG. **30. Exemple de TimedValueSpecification**

CVSL: Une ClockedValueSpecification spécifie un ensemble de valeurs temporelles. Le concept de temps couvrant les deux concepts distincts d'instant et de durée, le méta-modèle de la figure 31 reflète cette dichotomie (InstantValueSpecification et DurationValueSpecification). Une spécification de valeur peut être une instance (InstantInstanceValue ou DurationInstanceValue) ou une expression (InstantExpression ou DurationExpression). Cette expression dénote une ou plusieurs instances. Scaling représente une expression particulière : la multiplication d'une durée par un facteur réel, ce qui donne une durée. Span est une expression qui définit une durée à partir de deux instants. Translation définit un instant à partir d'un instant et d'une durée. Notons enfin que l'on peut spécifier des intervalles de valeurs temporelles (intervalles d'instants ou intervalles de durées).

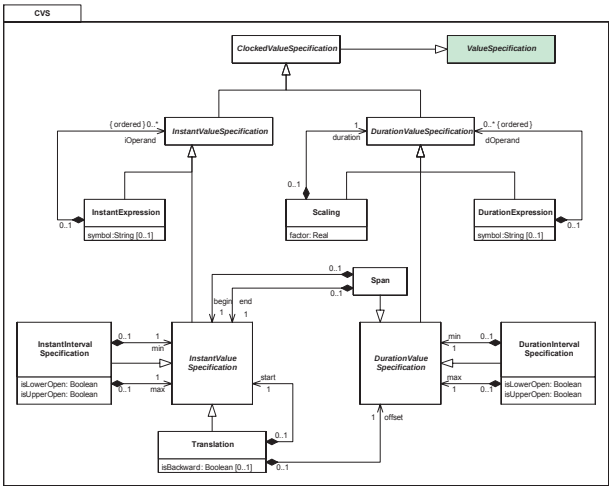

FIG. **31. Clocked Value Specification**

CVSL est une syntaxe concrète, non normative, proposée pour exprimer les valeurs temporelles. Indiquons simplement que ce langage accepte deux notations pour

une instance de valeur temporelle : soit une notation par des n-uplés, soit une notation plus proche du langage naturel (en anglais). Les deux expressions suivantes dénotent la même valeur temporelle en utilisant les deux syntaxes admises :

```
(value=3.5, unit=ms, onClock='idealClk');
3.5 ms on idealClk;
```

CVSL permet également de combiner des valeurs dans des expressions :

```
(value=1.5, unit=ms, onClock='idealClk') +
(value=150, unit=us, onClock='idealClk');
```

Cette expression est évaluable. Compte tenu du facteur de conversion de 0.001 existant entre la milliseconde (`ms`) et la microseconde (`us`), elle donne le résultat (ou une expression équivalente) :

```
(value=1650, unit=us, onClock='idealClk')
```

L'expression qui suit est plus générale, mais n'est pas toujours évaluable.

```
min (15 tick on prClk, 5 ms on idealClk);
```

Pour la calculer il faut connaître des relations entre les deux horloges `prClk` et `idealClk`. L'expression de ces relations d'horloges sera précisée dans la section 5.5.3.

Pour une définition plus complète de la syntaxe de CVSL il faut se reporter à l'annexe C2 de la proposition MARTE.

Expressions temporelles en VSL: VSL (Value Specification Language) est le langage normatif d'expressions de MARTE. Il étend les concepts de ValueSpecification et de DataType d'UML. Nous avons déjà vu que VSL permettait de spécifier des types de donnée composites et leurs instances, qui sont des valeurs composites, sous forme de n-uplés de valeurs (section 5.4.3). Il permet également de spécifier des expressions. Nous limitons ici notre présentation aux expressions temporelles VSL.

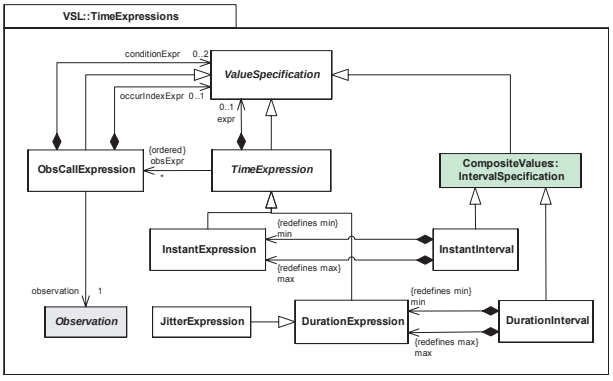

FIG. **32. Expressions temporelles en VSL**

La figure 32 donne la syntaxe abstraite des expressions temporelles de VSL. Une expression temporelle est une spécification de valeur qui peut référencer 0 ou plusieurs *observation call expressions*. Une ObsCallExpression fait référence à une seule Observation (observation d'instant ou de durée, section 4.4). Une expression à valeurs entières peut être utilisée pour spécifier l'index caractérisant une occurrence particulière de l'événement observé. Pour un événement `ev`, `ev[k]` dénote la $k^{\text{ième}}$ occurrence de `ev`.

Comme pour CVSL, les expressions temporelles VSL distinguent les expressions d'instants et celles de durées. Les jigues peuvent être spécifiées comme des expressions de durées spécialisées. Les intervalles d'instants ou de durées sont également définis.

L'exemple qui suit, extrait de la proposition MARTE, montre comment VSL permet d'exprimer des contraintes temporelles de façon bien plus précise qu'UML.

5.5.2 Expression des contraintes de temps

La figure 33 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.

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.

On peut noter sur cet exemple que VSL (comme d'ailleurs CVSL) peuvent spécifier des intervalles fermés ('[..]'), semi-ouverts ('[..[' ou ']..]') ou ouverts (']..[').

VSL ne connaît que les horloges chronométriques. Il faut considérer que les valeurs temporelles données sont relatives à l'horloge idealClk, qui est ici implicite. Si on désire référencer d'autres horloges il faut recourir à CVSL. Ceci sera illustré dans l'exemple de la section 6.2.

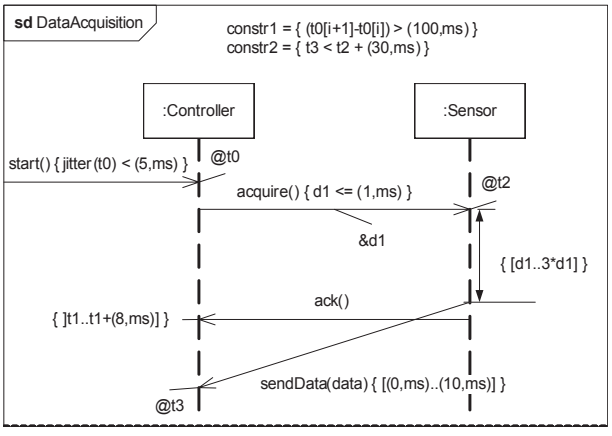

FIG. **33. Contraintes temporelles en VSL**

5.5.3 Expression des contraintes d'horloge

La figure 34 décrit la syntaxe abstraite d'un langage de spécification des contraintes d'horloge (ClockConstraintSpecification, abrégé en CCS).

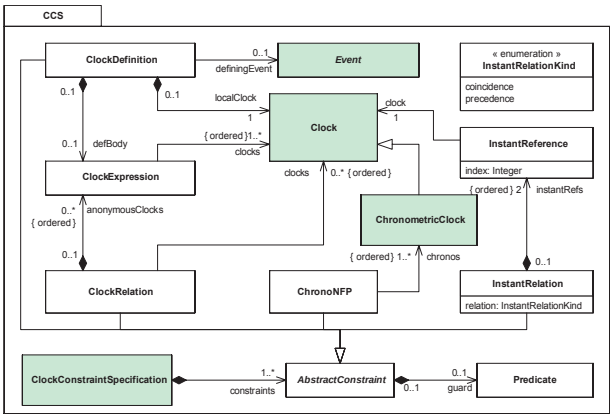

FIG. **34. Contraintes d'horloges**

Une spécification de contrainte consiste en un ensemble non vide de déclarations conditionnelles. Une déclaration conditionnelle est une AbstractConstraint qui possède une garde optionnelle. Lorsque la garde existe, la déclaration n'est effective que si cette garde est évaluée à vrai dans son contexte. Une contrainte abstraite est spécialisée en

- relation d'instants (InstantRelation),
- déclaration d'horloge locale (ClockDefinition),
- relation entre horloges (ClockRelation),
- contraintes sur des propriétés non fonctionnelles d'horloges chronométriques (ChronoNFP).

Une relation entre instants fait référence à deux instants—appartenant à des bases de temps différentes—et impose soit une coïncidence, soit une précédence entre ces instants suivant la valeur de l'attribut relation. Une déclaration d'horloge locale permet d'introduire des horloges intermédiaires, non visibles à l'extérieur de la contrainte. Ces horloges peuvent faciliter l'expression des contraintes d'horloges. La déclaration peut être accompagnée d'une définition (propriété defBody de la classe ClockDefinition). Notons qu'un horloge peut être définie par la donnée d'un événement. Cette possibilité permet de définir des horloges logiques. Une relation entre horloge impose des dépendances entre au moins deux horloges. Elles sont d'usage courant et seront illustrées dans la section 6 consacrée aux exemples. La dernière forme de contrainte s'applique aux propriétés non fonctionnelles des horloges chronométriques. Ces contraintes permettent de caractériser des horloges chronométriques imparfaites en indiquant la stabilité d'une horloge ou bien les décalages (offset), les dérives (skew et drift), les jigues (jitter), etc... lorsque plusieurs horloges sont considérées.

Une syntaxe concrète a également été proposée : c'est le langage déclaratif CCSL (Clock Constraint Specification Language). Ce langage n'est pas normatif. Quelques éléments du langage CCSL seront présentés avec les exemples.

6. Exemples d'utilisation

6.1. Créations d'horloges

Cette section explique comment l'utilisateur peut créer ses propres horloges en appliquant le sous-profil Time de MARTE.

6.1.1 Créations d'horloges chronométriques

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

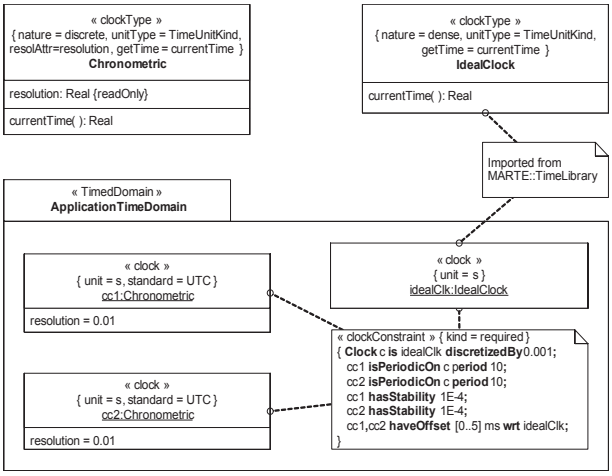

FIG. **35. Spécification d'horloges chronométriques**

Pour commencer, nous définissons un nouveau type d'horloge : Chronometric (figure 35, 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. Trois horloges sont créées 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 35 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 (section 5.5.3). 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 36 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.

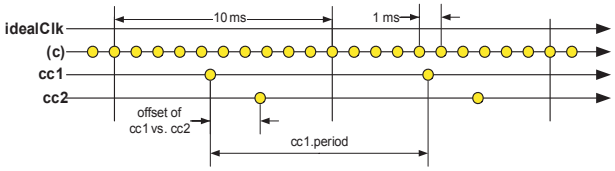

FIG. **36. Instants des horloges cc1 et cc2**

6.1.2 Créations d'horloges logiques

Nous prenons un exemple de contrôle simplifié dans lequel un processeur exécute le même code pour plusieurs contrôleurs. La figure 37 contient les principaux éléments du modèle. Le processeur est un *Voltage Scaling Processor*, c'est-à-dire qu'on peut changer sa fréquence en modifiant sa tension d'alimentation. Pour simplifier on considère que le processeur n'a que deux fréquences de fonctionnement : fh, la plus rapide, lorsque le processeur est en consommation maximale et fl lorsque le processeur est en basse consommation. L'attribut booléen inLowPower de la classe Processor indique le mode de fonctionnement courant du processeur.

Chaque contrôle doit être appliqué périodiquement (attribut period de la classe Controller et doit exécuter le code correspondant à un PID (pidCode qui est un OpaqueBehavior). Le comportement du contrôleur est spécifié par une machine à états nommée ctrlBeh.

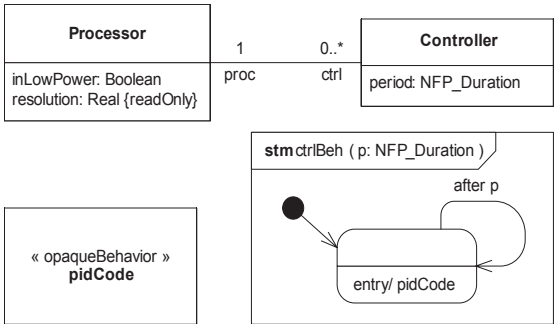

FIG. **37. Diagramme de classes de l'application de contrôle**

pidCode est un comportement qui s'exécute en un nombre fixe et connu de *cycles du processeur*. Ceci peut être modélisé à l'aide d'une *horloge logique*. Pour cela, nous stéréotypons la classe Processor par «clockType». Nous avons ainsi du temps logique (cycles processeur) et du temps chronométrique (périodes d'activation des contrôleurs). Il apparaît que ce mélange des deux types de temps est courant dans les applications de contrôle. C'est une des possibilités offertes par le sous-profil Time de MARTE. Les contraintes d'horloge sont le moyen retenu pour spécifier les dépendances entre les différents temps du modèle.

La figure 38 représente le domaine de temps de l'application. On y trouve les instances, dont les horloges, et une contrainte d'horloge. pr est une instance de Processeur. Nous le stéréotypons «clock». Il joue ainsi le rôle d'une horloge logique. Son unité est l'unité par défaut des horloges logiques : tick. Nous recopions également idealClk depuis la bibliothèque TimeLibrary. Cette horloge donne accès au temps chronométrique servant dans l'expression des périodes. c1 et c2 sont deux instances de Controller. Les périodes respectives sont 1ms et 2ms exprimées avec la syntaxe VSL pour les NFP_Duration. La durée d'exécution de pidCode est 45 cycles processeur. Ceci s'exprime en stéréotypant pidCode par «timedProcessing» et en donnant les valeurs des méta-attributs (`pr` pour on et 45 pour duration). Le TimeEvent tev associé au trigger de la transition de la machine à états ctrlBeh est stéréotypé «timedEvent» en précisant son horloge (on = `idealClk`).

La contrainte d'horloge commence par déclarer l'horloge c locale, chronométrique et discrète de période 1μs. Les deux instructions qui suivent sont conditionnelles. Elles expriment la relation liant les cycles de `pr` au temps

logique suivant que le processeur fonctionne à pleine puissance ou à puissance réduite. La sémantique de '`pr = c filteredBy 0B(1.0^9)`' est que pr est une sous-horloge de c telle que

$$(\forall k \in \mathbb{N})\ pr[1+k] \equiv c[1+10*k]$$

chaque instant de pr est coïncident avec un instant de c. La correspondance est précisée par le mot binaire qui suit le mot clé `filteredBy`. Par exemple, le mot binaire '`0B0.1(1.0^2)`' dénote le mot binaire infini 01100100100... qui présente un préfixe fini 01 et un mot infini périodique de période 100.

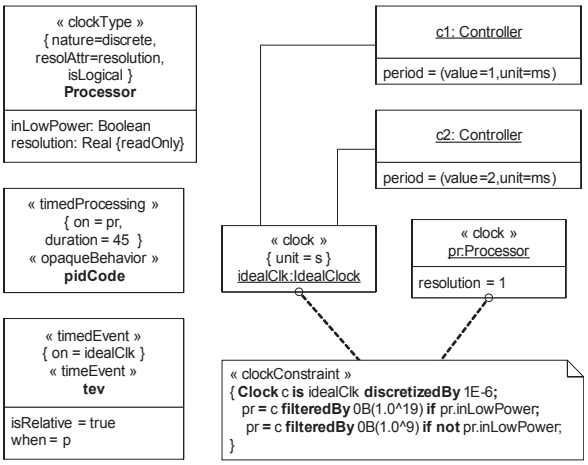

FIG. **38. Application de contrôle avec prise en compte du temps**

6.2. Moteur quatre temps

Cet exemple d'utilisation du modèle de temps de MARTE est extrait d'une application automobile. Une étude plus complète est disponible dans une communication [2] qui modélise et analyse la détection/correction du cliquettis . Nous ne considérons ici que la modélisation du cycle du moteur à explosion 4 temps. Ceci nous permet d'illustrer la notion de temps multiforme, avec des horloges logiques qui utilisent des unités non classiques (des unités d'angle dans cet exemple).

Le cycle du moteur à explosion 4 temps comprend quatre phases : l'admission (*intake*), la compression, la combustion et l'échappement (*Exhaust*). Pour un cylindre, ces phases sont exécutées séquentiellement. Les événements de début et de fin de phase sont liés à la position angulaire de la *came* (*cam*). La came est elle-même liée à la rotation du villebroquin (*crankshaft*). Les positions angulaires, exprimées en °CAM (lire degré came) ou en °CRK (lire degré crank), sont les unités naturelles pour observer, mesurer et spécifier les occurrences d'événements et les traitements relatifs à l'allumage et à l'injection électroniques. La figure 39 contient les horloges logiques que nous proposons.

Il faut en premier définir les unités qui vont être employées. C'est le rôle de l'énumération AngleUnitKind. Cette énumération contient deux littéraux °CAM et °CRK stéréotypés «unit». Il existe une relation, non mentionnée dans la figure, entre les deux unités. Lorsque l'arbre à cames fait un tour complet, le villebroquin fait deux tours complets. On aurait donc pu rajouter une contrainte lors du stéréotypage des littéraux d'énumération :

```
«unit» °CAM {
  baseUnit=°CRK, convFactor=2.0 }
```

ou bien de façon équivalente

```
«unit» °CRK {
  baseUnit=°CAM, convFactor=0.5 }
```

Ces unités sont utilisées dans les horloges de type AngleClock. Ce sont des horloges logiques, à temps discret, et pour lesquelles nous pourrons choisir les attributs de résolution (resolution), d'origine (offset), de valeur maximale (mawimalValue), ainsi que la fonction d'étiquetage des instants (indexToValue()). Cette opération est spécifiée par une contrainte OCL classique :

```
{ context
  AngleClock::angle(k:integer):  Real;
  angle = (offset + (k-1)*resolution)
    rmod maximalValue }
```

`rmod` est l'opération modulo sur les nombres réels (x rmod $y = x - \lfloor x/y \rfloor * y$). Elle utilise comme paramètres les attributs de la classe AngleClock. Le paramètre d'entrée est un entier naturel k qui représente l'indice d'un instant de l'horloge ; la valeur retournée est la valeur d'instant associée à cet indice.

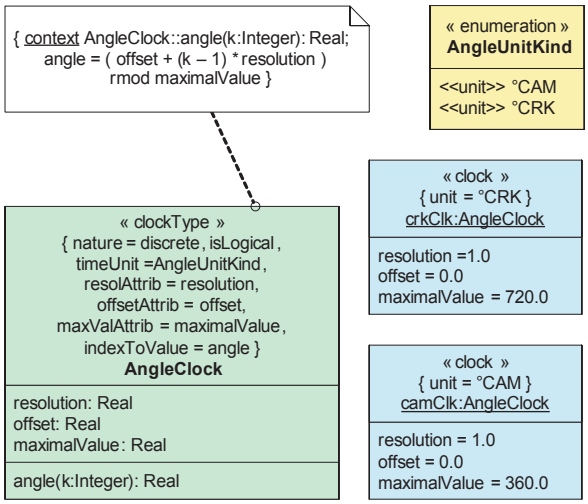

FIG. **39. Horloges logiques pour une application automobile**

Deux horloges crlClk et camClk, instances de AngleClock, sont ensuite créées. La première prend le °CRK comme unité. Sa résolution est de 1°CRK et sa valeur maximale 720°CRK, ce qui correspond à deux révolutions complètes du villebroquin. La seconde horloge a le °CAM pour unité, sa résolution est de 1°CAM et sa valeur maximale de 360°CAM. Compte tenu de la relation 1°CAM = 2°CRK et des résolutions choisies, les deux horloges sont liées par la contrainte d'horloge :

```
camClk = crkClk filteredBy 0b(10)
```

que l'on peut visualiser par la figure 40, dans laquelle les instants sont les gros points. Deux instants coïncidents sont reliés par une liaison verticale de couleur rouge. Quelques valeurs d'instant apparaissent sous la forme d'une annotation attachée à un instant.

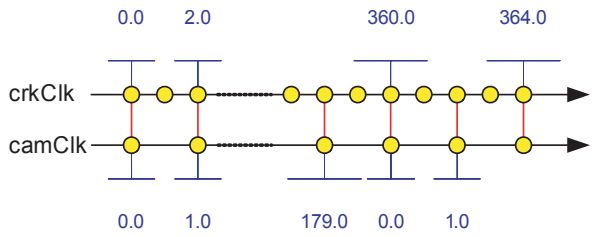

FIG. **40. Relation structurelle entre crkClk et camClk**

La succession des phases peut être représentée par une machine à états UML (figure 41). Dans un premier temps on associe des TimeEvent aux déclencheurs des transitions. Puis ces événements sont stéréotypés «timedEvent» en précisant l'horloge associée (`on = camClk`). La machine à états spécifie alors le comportement suivant :

1. état initial = intake ;
2. après 90°CAM observés sur l'horloge camClk, quitter cet état pour entrer dans l'état Compression ;
3. après 90°CAM observés sur l'horloge camClk, quitter cet état pour entrer dans l'état Combustion ;
4. après 90°CAM observés sur l'horloge camClk, quitter cet état pour entrer dans l'état Exhaust ;
5. après 90°CAM observés sur l'horloge camClk, quitter cet état pour entrer dans l'état intake ;
6. reprendre en 2.

Par rapport à une description du comportement qui aurait fait intervenir des unités de temps usuelles (s ou unités dérivées), la description qui utilise le temps logique présente l'avantage d'être indépendante de la vitesse du moteur.

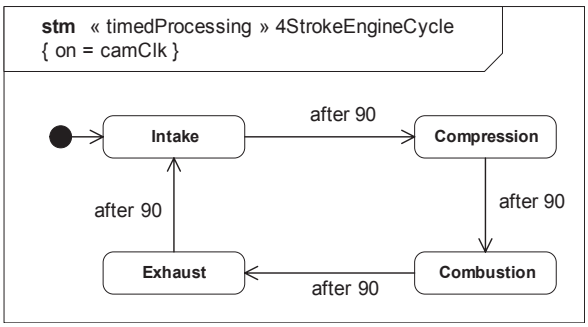

FIG. **41. Machine à états pour le cycle d'un moteur 4 temps**

Le comportement du moteur peut également être décrit en UML sous forme de *diagramme de séquence*. Un diagramme de séquence représente une trace d'exécution. Il se lit de haut en bas. UML 2 a introduit une variante du diagramme de séquence, appelée *Timing Diagram*, proche des chronogrammes utilisés en électronique. Un diagramme de temps montre, en particulier, les changements d'états au cours du temps et les occurrences d'événements ayant causé ou causés par ces occurrences. La lecture se fait cette fois de gauche à droite. MARTE étend ces diagrammes aux temps logiques en les stéréotypant par «timedProcessing».

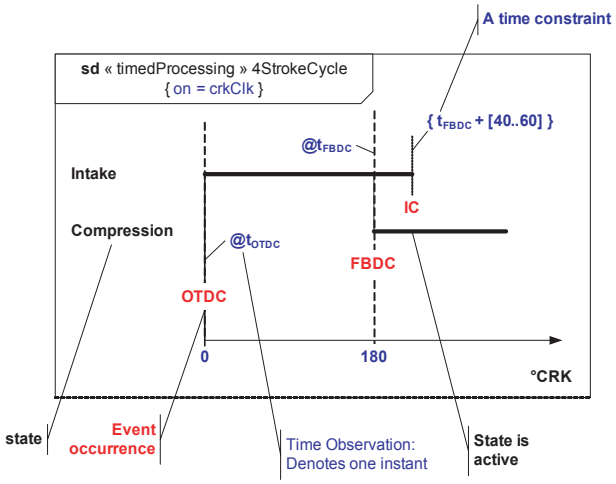

FIG. **42. Representation partielle du Timing-Diagram du cycle 4 temps**

La figure 42 représente les évolutions en début de cycle. Les annotations situées hors du cadre ne relèvent pas d'UML. Dans un diagramme de temps, les états actifs sont représentés par des segments horizontaux. L'échelle de temps est graduée et indique des instants. Dans notre application les indications sont en °CRK et les évolutions sont observées sur l'horloge crkClk, comme l'indique le stéréotype. Les événements sont nommés et leurs occurrences repérées sur l'échelle des temps. Par exemple, l'événement OTDC (Overlap Top Dead Center) se produit à la position angulaire 0°CRK. Pour le cylindre considéré, OTDC (point mort haut) indique que le ville-broquin est dans la position haute. L'événement FBDC (First Bottom Dead Center) se produit à la position angulaire 180°CRK. La figure nomme des observations temporelles (par exemple, t_{OTDC}) et les utilisent dans des contraintes temporelles. La contrainte

```
{ tFDBC + [40..60] }
```

contraint l'occurrence de l'événement IC (Intake valve Closes) à se produire entre $t_{FBDC}+40$ et $t_{FBDC}+60$, valeurs mesurées en °CRK. Noter que dans ce diagramme nous avons pu représenter explicitement des chevauchements d'états (overlap). La phase d'admission continue une certain "temps" alors que la phase de compression a déjà commencé. La contrainte spécifie la durée du chevauchement (en valeur d'angle).

Cet usage des diagrammes de temps avec les temps logiques et la représentation des chevauchements

d'états n'est pas classique mais s'avère très utile en spécification. On peut regréter qu'actuellement les outils de modélisation commerciaux ne supportent pas les diagrammes de temps. La raison est peut être qu'ils ne sont pas dans la culture "génie logiciel". L'existence de profils comme MARTE pourrait favoriser l'apparition de ces diagrammes dans les outils.

Ayant un modèle pour le cycle de combustion d'un cylindre, on voudrait maintenant pouvoir passer facilement à une modélisation pour un moteur constitué de plusieurs cylindres. Prenons, par exemple, le cas d'un moteur 4 cylindres en ligne. Les phases relatives à chaque cylindre vont être décalées dans le temps (en réalité c'est un décalage dans l'espace, puisqu'il s'exprime par un angle). Si on numérote les cylindres de 1 à 4, la séquence d'allumage est 1–3–4–2 pour la géométrie donnée dans la partie supérieure de la figure 43. Ceci signifie que la position angulaire d'allumage d'un cylindre est 720/4=180 °CRK après la position angulaire d'allumage du cylindre qui le précède dans la séquence d'allumage.

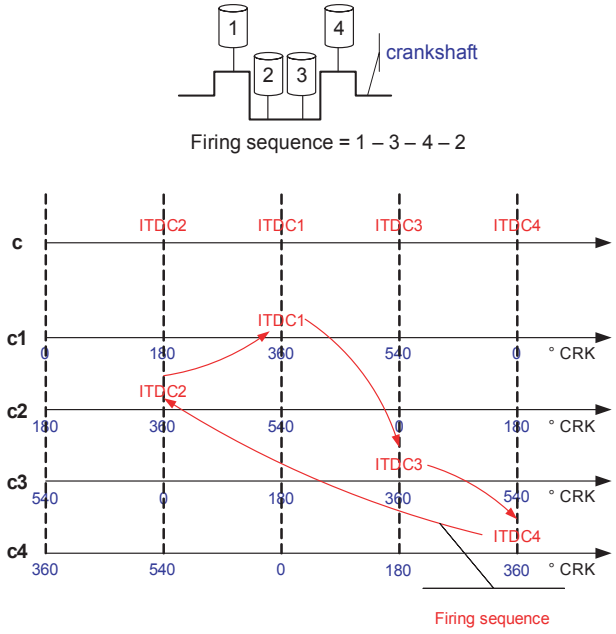

FIG. **43. Séquence d'allumage pour un moteur 4 cylindres**

Pour chaque cylindre l'allumage est déclenché au point mort haut d'allumage (ITDC, Ignition Top Dead Center), soit à 360°CRK de son horloge. La figure 43 représente les horloges associées à chaque cylindre et la séquence d'allumage. Notons qu'il ne s'agit ici que d'une première approximation. Dans la réalité, le point exact d'allumage doit être situé avant ITDC (avance à l'allumage). L'avance, exprimée en °CRK est calculée dynamiquement par l'unité de contrôle en charge de l'allumage. L'article [2], déjà référencé, modélise ce contrôle de façon détaillée.

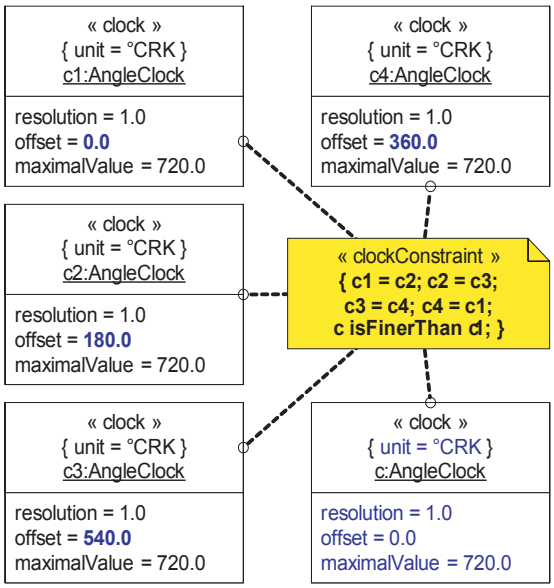

FIG. **44. Horloges logiques pour un moteur 4 cylindres**

Dans notre approche, les décalages d'allumage sont exprimés par des contraintes d'horloges (figure 44). On instancie 4 horloges c1 à c4 de type AngleClock, ayant pour unité le °CRK. L'horloge ck est associée au $k^{\text{ième}}$ cylindre. Les 4 horloges ont même résolution (1°CRK), même valeur maximale (720°CRK), mais diffèrent par leur offset. Compte tenu de la séquence d'allumage nous avons respectivement :

- c1 : `offset=0.0;`
- c3 : `offset=540.0` car elle retarde de 180°CRK par rapport à c1 et $540 = 0 - 180 \pmod{720}$;
- c4 : `offset=360.0` car elle retarde de 180°CRK par rapport à c3 et $360 = 540 - 180 \pmod{720}$;
- c2 : `offset=180.0` car elle retarde de 180°CRK par rapport à c4 et $180 = 360 - 180 \pmod{720}$.

Il reste à présent à créer une horloge logique c pour le moteur et à exprimer les dépendances entre toutes ces horloges. C'est l'objet de la contrainte d'horloge :

```
{
  c1 = c2;
  c2 = c3;
  c3 = c4;
  c4 = c1;
  c isFinerThan c1;
}
```

les quatre premières instructions indiquent que les horloges c1 à c4 ont même structure :

$$\forall k \in \mathbb{N}, c1[k] \equiv c2[k] \equiv c3[k] \equiv c4[k]$$

toutefois, ceci ne signifie pas que les 4 horloges soient identiques : elles n'ont pas la même fonction d'étiquetage qui sont respectivement :

$$\lambda_1(c_1[k]) = (k-1) \pmod{720.0}$$
$$\lambda_2(c_2[k]) = (180.0+k-1) \pmod{720.0}$$
$$\lambda_3(c_3[k]) = (540.0+k-1) \pmod{720.0}$$
$$\lambda_4(c_4[k]) = (360.0+k-1) \pmod{720.0}$$

la dernière instruction indique qu'on peut choisir pour c n'importe quelle horloge qui admet c1 (ou c2, ou c3 ou c4) comme sous-horloge. La contrainte

```
c isFinerThan c1;
```

signifie qu'il existe une application injective de l'ensemble des instants de c1 sur l'ensemble des instants de c qui préserve l'ordre $\preccurlyeq$:

$$\begin{aligned}\forall k_1, k_2 \in \mathcal{I}_{c1},\ \exists j_1, j_2 \in \mathcal{I}_c :\\ (k_1 \equiv j_1) \wedge\\ (k_2 \equiv j_2) \wedge\\ ((k1 \prec_{c1} k2) \Longrightarrow (j_1 \prec_c j_2))\end{aligned}$$

Pour simplifier, nous pouvons choisir c identique à c1.

Notons que les horloges logiques que nous sommes amenés à créer au cours de la conception n'ont pas, en général, vocation à être implémentées. Elles permettent, via la résolution des contraintes d'horloges, de déterminer des ordonnancements corrects. La liaison entre les temps logiques et le temps physique est exprimée par des contraintes supplémentaires. Par exemple, dans le cas du moteur, on peut donner son régime maximal : 4500 rpm (rpm=*Revolutions Per Minute*). A ce régime, le moteur fait 4500/60 tours par seconde, ce qui correspondant à (4500/60)*360=27000°CRK. D'où l'on déduit la relation liant °CRK et s : 1°CRK $\geq$ 37 μs. Cette relation quantitative est exploitée dans l'article [2] pour dimensionner des buffers et choisir la fréquence d'un processeur.

7. Conclusion et perspectives

Le sous-profil Time de MARTE, décrit dans cet article, étend largement le modèle SimpleTime de MARTE. Il permet de faire référence explicitement à *plusieurs* référentiels de temps, par l'intermédiaire d'horloges (stéréotype Clock). Le temps peut être de nature dense ou discrète. MARTE modélise aussi bien le temps physique que les temps logiques. Les horloges donnant accès au temps physique sont appelées des horloges *chronométriques*. MARTE permet d'exprimer des différences dans la perception du temps, différences dues par exemple à la distribution spatiale du système. On peut également modéliser des perceptions imparfaites caractérisées par des propriétés attachées aux horloges : stabilité, décalage, dérive, etc. Les temps logiques, auxquels on accède par des horloges logiques, s'avèrent très utiles en phase de conception. Les horloges logiques peuvent être associées aussi bien à des éléments de modèles matériel (cycles d'un processeur, par exemple) qu'à des éléments applicatifs (itérations contrôlées). Une autre caractéristique du modèle de temps de MARTE est de pouvoir lier *directement* et explicitement des éléments comportementaux (TimedEvent et TimedProcessing) au temps. Il en est de même avec les contraintes (TimedConstraint) et les observations (TimedObservation).

UML ayant des objectifs pragmatiques, il a fallu offrir des facilités d'utilisation des concepts temporels. Cette aide est fournie sous forme de *bibliothèques* et de *langages spécialisés*. Le langage d'expression de contraintes d'horloges permet en particulier d'exprimer de façon concises des dépendances, parfois complexes, entre instants d'horloges différentes. Une représentation purement graphique de ces contraintes n'aurait pas été réaliste. D'un point de vue pratique, nous avons développés des plug-ins Eclipse qui implémentent le profil et font l'analyse syntaxique des langages spécialisés.

Le développement d'un profil UML demande de bonnes connaissances en méta-modélisation mais surtout une grande maîtrise de la spécification UML. C'est un travail d'expert et sa mise en œuvre est très technique. Les contraintes imposées par le standard UML ne nous autorisaient pas d'inclure dans la norme toutes les aspects formels que nous voulions apporter. Ainsi le modèle mathématique de structure de temps (section 4.5) n'apparaît nulle part dans la proposition MARTE : nous avons dû nous en tenir à des sémantiques exprimées en langage naturel (en fait dans un anglais technique). Nous comptons bien poursuivre nos efforts de recherche au-delà de la proposition MARTE. Certaines idées ont été lancées dans le profil Time dans l'espoir qu'elles deviennent plus populaires. Par exemple, le fait de pouvoir associer explicitement une horloge à des éléments de modèle est la voie que nous avons choisie pour faire comprendre que le temps fait partie intégrante de la sémantique de certains éléments. Les informations temporelles ne sont pas de simples annotations attachées à des éléments de modèle : elles en modifie profondément la sémantique. C'est le point de vue que nous défendons dans un de nos articles [1]. Cette proposition va dans le sens des demandes exprimées par E. Lee [9] sur la nécessité d'inclure temps et concurrence dans la sémantique. Notre prochaine étape consitera à spécifier mathématiquement toutes les relations d'horloges et surtout de trouver des moyens efficaces pour extraire des solutions satisfaisant aux contraintes.

Références

[1] C. André, F. Mallet, and R. de Simone. Modeling time(s). In *MoDELS 2007*, October 2007.

[2] C. André, F. Mallet, and M.-A. Peraldi-Frati. A multi-form time approach to real-time system modeling : Application to an automotive system. In *IEEE 2nd International Symposium on Industrial Embedded Systems (SIES'2007)*. IEEE, 2007.

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

[4] 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.

[5] J.-M. Favre, J. Estublier, and M. Blay-Fornarino. *L'ingénierie dirigée par les modèles. Au-delà du MDA*. Hermès, 2006.

[6] T. A. Henzinger and J. Sifakis. The embedded systems design challenge. In *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*. Springer, August 2006.

[7] H. Kopetz. *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston(MA), 1997.

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

[9] E. A. Lee. Absolutely positively on time : What would it take ? *IEEE Computer*, 38(7) :85–87, 2005.

[10] D. G. Messerschmitt. Synchronization in digital system design. *IEEE J. Select. Areas Commun.*, 8 :1404–1419, October 1990.

[11] OMG. *MDA Guide V1.0.1*. Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701, June 2003. OMG document number : omg/03-05-01.

[12] OMG. *UML 2.0 Infrastructure Specification*. Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701., Sept 2003. OMG document number : ptc/03-09-15.

[13] OMG. *Enhanced View of Time Specification, version 1.2*. Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701, October 2004. OMG document number : formal/04-10-04.

[14] OMG. *UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), Request for proposals*. Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701., February 2005. OMG document number : realtime/2005-02-06.

[15] 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).

[16] OMG. *Systems Modeling Language (SysML) Specification*. Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701, April 2006. OMG document number : ad/2006-03-01.

[17] OMG. *UML 2.1 Superstructure Specification*. Object Management Group, Inc., 492 Old Connecticut Path, Framing-ham, MA 01701, April 2006. OMG document number : ptc/2006-04-02.

[18] C. A. Petri. Concurrency theory. In *Petri Nets : Central Models ans their Properties*, volume 254 of *Lecture Notes in Computer Science*, pages 4–24. Springer-Verlag, 1987.

[19] F. A. Schreiber. Is time a real time ? an overview of time ontology in informatics. *Real Time Computing*, F127 :283–307, 1994.

[20] R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations—in search of the holy grail. *Distributed Computing*, 7 :149–174, 1994.

[21] B. Selic. On the semantic foundations of standard UML 2.0. In *SFM-RT 2004*, volume 3185 of *LNCS*, pages 181–199. Springer-Verlag, 2004.

[22] B. Selic. A systematic approach to domain-specific language design using UML. In *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, pages 2–9. IEEE, 2007.