MÉTA-MODÉLISATION D'UNE MÉTHODE DE CONCEPTION DE SYSTÈMES HYBRIDES INTÉGRANT UNE DOUBLE REPRÉSENTATION CONTINUE ET DISCRÈTE DU TEMPS

Laurent Piétrac[1], Bruno Denis[2] et Jean-Jacques Lesage[2]
[1] : LAI, INSA de Lyon, bat. 303,
20 av. Albert Einstein 69621 VILLEURBANNE CEDEX
e-mail : laurent.pietrac@lai.insa-lyon.fr
[2] : LURPA, ENS de Cachan, 61 av. du président Wilson 94235 CACHAN CEDEX
e-mail : {Lesage, Denis}@lurpa.ens-cachan.fr

Résumé

La conception des systèmes automatisés de production nécessite l'utilisation de langages permettant de modéliser leur évolution temporelle. Lorsque ces systèmes sont hybrides, une approche classique consiste à utiliser conjointement deux langages, dédié pour l'un à la partie discrète et pour l'autre à la partie continue de ce système. Le problème crucial est alors d'intégrer ces deux langages, et donc souvent deux modélisations du temps, au sein d'une méthode cohérente. Cet article présente notre approche de méta-modélisation formelle utilisée pour définir rigoureusement des langages et leur intégration au sein de méthodes multi-langages. Nous y abordons plus spécifiquement les problèmes posés par la double perception, discrète et continue, du temps. Une méthode de conception de la commande des systèmes dynamiques hybrides intégrant une classe particulière de réseaux de Petri et des équations différentielles est utilisée comme exemple illustratif de notre approche.

Mots-clés

Méthode multi-langages, systèmes hybrides, temps logique, temps physique, langage formel Z, méta-modélisation.

I INTRODUCTION

La modélisation du comportement temporel des systèmes automatisés de production (SAP) est particulièrement importante. Elle occupe de ce fait une place privilégiée dans toutes les phases de conception : depuis la conception préliminaire où on s'intéresse essentiellement à la vérification des performances attendues du SAP, jusqu'à la conception détaillée de la commande qui est dévolue à la représentation du comportement dynamique des équipements de production. La modélisation comportementale des systèmes dynamiques hybrides (SDH) pose un problème particulier puisqu'elle nécessite un double point de vue discret et continu de l'évolution du système, c'est pourquoi une approche très répandue (voir [1] par exemple) consiste à utiliser deux langages complémentaires : un langage adapté à la modélisation des systèmes à événements discrets, et un adapté aux systèmes continus. Pour une large classe de SDH, cette approche est plus modulaire, plus flexible que celle utilisant un langage unique permettant d'appréhender simultanément les deux facettes des SDH (tels que les réseaux de Petri hybrides ou les automates hybrides). Elle permet en outre de modéliser des systèmes ayant une partie continue particulièrement complexe [6].

Pour intégrer deux langages au sein d'une même méthode, il faut tout d'abord définir formellement chacun des langages utilisés. Pour cela nous utilisons le concept de méta-modélisation que nous mettons en oeuvre avec le langage Z. Ces deux points sont abordés dans la première section. Nous détaillerons ensuite les différents points de vue temporels qui sont utilisés pour la conception des SDH. La section IV présentera une méthode intégrant plusieurs de ces points de vue ; cette méthode multi-langages nous servira d'exemple illustratif de notre approche. Cette méthode, inspirée de [2], est basée sur l'utilisation conjointe d'une classe particulière de réseaux de Petri et d'équations différentielles. Au travers de la méta-modélisation de chacun de ces deux langages, puis de leur intégration méthodologique, nous montrerons dans la section V comment le langage formel Z nous permet d'appréhender deux abstractions différentes du temps.

II LA MÉTA-MODÉLISATION

La conception des SAP nécessite la construction de plusieurs modèles abordant chacun des aspects différents du système. Ces modèles sont construits à l'aide de langages et de méthodes dont la rigueur conditionne pour une large part la qualité des modèles produits. Beaucoup de langages et de méthodes sont à l'origine définis textuellement, ce qui ne permet pas d'assurer la rigueur de ces définitions. Pour en rigoriser la définition, une approche récente consiste à construire des modèles de ces langages et méthodes : ces modèles sont appelés des méta-modèles, et les langages utilisés pour ce faire, des méta-langages.

I.1 Les approches de méta-modélisation

Chronologiquement, les premiers méta-langages utilisés ont été des langages de représentation structurée de données issus du génie logiciel et de l'informatique de gestion : NIAM [5], les modèles conceptuels de données de Merise [11] ou des modèles Entités-Relations étendus [7]. En complément de ces approches, des travaux plus récents se sont intéressés à la modélisation de la dynamique des langages ou des mécanismes de construction des modèles [8] [9].

Toutes ces approches ont l'inconvénient de ne couvrir chacune qu'une partie des besoins de méta-modélisation. En effet, dans [15], nous avons montré que les besoins de méta-modélisation sont naturellement de définir aussi bien la syntaxe que la sémantique interne des langages, mais également la construction de modèles, leur vérification et leur algorithme de jeu. De plus un langage est souvent intégré au sein d'une méthode de conception multi-langages. Ces mécanismes d'intégration sont eux aussi porteurs d'une syntaxe et d'une sémantique qu'il convient de méta-modéliser. Dans la section suivante, nous allons présenter le langage Z et ses possibilités pour la méta-modélisation.

I.2 La méta-modélisation avec Z

Le langage Z fut créé par Jean-Raymond Abrial en France et développé par une équipe du Programming Research Group de l'université d'Oxford. Ce langage est basé sur une théorie axiomatique des ensembles, sur la logique du premier ordre et sur la notation par schémas qui permet de structurer les spécifications [12]. Ce langage formel est désormais internationalement utilisé pour spécifier le comportement attendu de logiciels, notamment dans le domaine des systèmes critiques. En effet, les théories mathématiques utilisées dans le langage Z permettent d'apporter la preuve de la cohérence de la spécification écrite.

Grâce aux théories mathématiques qu'il utilise, le langage Z peut être utilisé pour méta-modéliser la syntaxe et la sémantique de langages [16]. De plus il permet de méta-modéliser les techniques de construction associées aux langages, ainsi que leur intégration au sein de méthodes multi-langages [17]. Dans cet article, nous allons montrer qu'il est également capable de supporter plusieurs abstractions du temps.

III LA VARIABLE TEMPS DANS LES SDH

Un système dynamique hybride est caractérisé par un comportement qui est à la fois continu et discret ; il s'agit par exemple de séquentialiser les phases successives de transformation chimique sur un produit. L'espace d'états d'un tel système est alors lui-même hybride puisque constitué du continuum des états pris par les variables continues du système (débit, pression, PH… utilisées dans le procédé chimique) et de l'espace discret des états pris par ses variables discrètes (événements ou seuils de niveaux sur des variables continues permettant de gérer la recette d'élaboration du produit, de changer les modes de marche ou de prendre en compte les pannes…). Les langages utilisés pour caractériser ces espaces d'états retiennent des vues différentes de la variable temps. Pour les langages dédiés aux systèmes continus, le temps peut être vu comme une variable réelle (comme dans les équations différentielles) ou entière (comme dans les équations aux différences). Il est important de remarquer que cette discrétisation du temps n'implique pas pour autant la discrétisation de l'espace d'états qui est décrit. Pour les langages dédiés aux systèmes à événements discrets, le temps peut souvent être ramené à une notion implicite au travers d'un ensemble de dates d'occurrences d'événements entre lesquelles est définie une relation d'ordre total (c'est le « temps logique » utilisé par exemple dans les réseaux de Petri non temporisés). Lorsqu'il est nécessaire de manipuler des intervalles de temps, et plus seulement des dates d'événements, il est cette fois nécessaire de prendre en compte un « temps physique », le plus souvent défini comme un temps continu rationnel positif ou nul (c'est le cas pour exprimer les temporisation des RdP). Lorsque la modélisation comportementale d'un SDH est réalisée en utilisant conjointement deux langages intégrant ces deux visions différentes du temps (des équations différentielles et des RdPT par exemple) se pose alors de manière particulièrement aiguë le problème de définition cohérente de la variable temps. De nombreux auteurs se sont attachés à résoudre ce problème en définissant par exemple des techniques de représentation discrète de l'évolution temporelle de variables continues linéaires [13], des approximations de modèles continus [10] ou des techniques de composition de plusieurs variables temps discrètes ou continues considérées comme des variables locales [4]. Dans cet article, nous allons plus particulièrement nous intéresser à la définition du temps sous son aspect continu et discret événementiel dans le langage Z, de manière à montrer que la méta-modélisation offre un cadre formel unifié pour la définition des méthodes de conception des SDH. Mais tout d'abord, nous décrivons dans la section suivante la méthode de conception de la commande des SDH que nous avons retenue pour illustrer notre approche. Nous allons notamment y décrire les différentes notions de temps qu'elle intègre.

IV UNE MÉTHODE MULTI-LANGAGES DE CONCEPTION DES SDH

L'objectif des travaux de David Andreu [2] est d'avoir une approche globale, hiérarchisée et modulaire, des aspects temps-réel de la conduite des procédés hybrides, et notamment des procédés de traitement par lots. Le modèle du système, constitué du procédé et de la commande, résulte de l'utilisation conjointe de deux langages : les réseaux de Petri et les équations différentielles.

Ce modèle est structuré en trois parties (figure 1). Un superviseur traduit la recette d'élaboration du produit en consignes pour les régulateurs continus et en événements pour les automates programmables. Le superviseur est également chargé de la surveillance du procédé. Les contrôleurs locaux (régulateurs et automates), en boucle fermée avec le procédé, génèrent les ordres de commande. Un générateur d'événements observe l'évolution du procédé et transmet au superviseur l'ensemble des événements pertinents détectés. Seule la modélisation du superviseur sera étudiée dans cet article.

Le modèle de supervision est en fait constitué d'un modèle de la commande (discret, avec un point de vue « temps logique ») et d'un modèle hybride du procédé, appelé « modèle de référence hybride » (avec les deux points de vue temporels : « temps logique » et « temps physique » défini comme un temps continu réel). Ce « modèle de référence hybride » est essentiellement utilisé à des fins de surveillance. Piloté par le modèle de la commande, il permet la détection de dérives du procédé (comparaison avec les événements détectés par le générateur d'événements) ainsi

que la vérification de la faisabilité de l'ordonnancement prévu (par simulation).

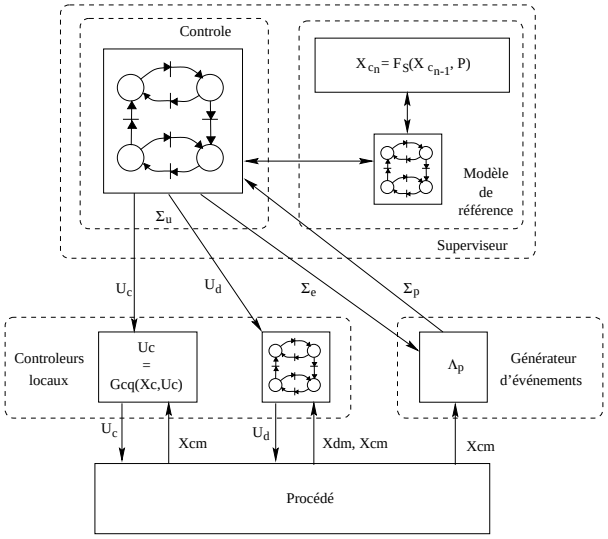

Figure 1 : supervision basée sur un modèle hybride [2]

Notre objectif, en méta-modélisant cette méthode, est de montrer que notre approche permet d'étudier les problèmes d'intégration de différents langages utilisant différents points de vue sur le temps. Nous avons donc choisi d'adapter légèrement la méthode utilisée afin quelle nous permette d'atteindre notre objectif, tout en restant abordable dans le volume de cet article.

Notre simplification porte sur la classe de réseaux de Petri utilisée dans le superviseur. Les auteurs ont en effet retenu une classe particulière, et complexe, de RdP à objets [2]. De manière à limiter la taille des méta-modèles construits, nous ne retiendrons que la caractéristique « synchronisés » des RdP généralisés (par association d'événements aux transitions). Des fenêtres temporelles seront également associées aux transitions afin de contrôler la durée de marquage des places. Dans la section suivante, nous construisons progressivement les méta-modèles en abordant tout d'abord le méta-modèle du réseau de Petri généralisé (section V.1), puis dans la section V.2 celui de la classe de réseaux de Petri retenue. Nous étudierons ensuite le méta-modèle des équations différentielles, et enfin celui de l'intégration de tous ces méta-modèles.

V MÉTA-MODÉLISATION DES DEUX LANGAGES

I.3 Réseaux de Petri généralisés

Un réseau de Petri généralisé (RdPG) est défini dans [14] comme un 5-uplet $PN = (P, T, F, W, M_0)$. Ce langage permet, à travers l'évolution de son marquage, de représenter la dynamique du système modélisé. Le temps n'est donc utilisé que pour définir une chronologie des tirs de transitions et des marquages des places : c'est un « temps logique ». Nous allons maintenant présenter une définition de ce langage par méta-modélisation formelle avec Z.

Un RdPG est alors défini par le méta-modèle suivant :

[PLACE, TRANSITION]

USE ::= build | play

PN

$P : \mathbb{F}\ PLACE$
$T : \mathbb{F}\ TRANSITION$
$arcTP : TRANSITION \leftrightarrow PLACE$
$arcPT : PLACE \leftrightarrow TRANSITION$
$WarcTP : TRANSITION \times PLACE \rightarrow\!\!\!\!\!\!\bullet\ \mathbb{N}_1$
$WarcPT : PLACE \times TRANSITION \rightarrow\!\!\!\!\!\!\bullet\ \mathbb{N}_1$
$M0 : PLACE \rightarrow\!\!\!\!\!\!\bullet\ \mathbb{N}$
$M : PLACE \rightarrow\!\!\!\!\!\!\bullet\ \mathbb{N}$
$enabled : \mathbb{F}\ TRANSITION$
$usePN : USE$

$\mathrm{dom}(arcTP) \subseteq T$
$\mathrm{ran}(arcTP) \subseteq P$
$\mathrm{dom}(arcPT) \subseteq T$
$\mathrm{ran}(arcPT) \subseteq P$
$\mathrm{dom}(WarcTP) = arcTP$
$\mathrm{dom}(WarcPT) = arcPT$
$\mathrm{dom}(M0) = P$
$\mathrm{dom}(M) = P$

Ce schéma défini un réseau de Petri généralisé comme un ensemble de places (*P*), de transitions (*T*), d'arcs (*arcTP* et *arcPT*), de poids (*WarcTP* et *WarcPT*), d'un marquage initial (*M0*) et d'un marquage courant (*M*). Les arcs et les poids sont représentés sous la forme de deux fonctions, car en Z les ensembles sont typés. Ceci augmente légèrement la taille du méta-modèle, mais a l'avantage de permettre la preuve de la cohérence des opérations spécifiées. La deuxième partie du schéma (en dessous du trait horizontal médian) est utilisée pour définir les contraintes sur les ensembles utilisés. La dynamique du RdPG (l'évolution de son marquage) est définie par le schéma suivant. Ce schéma spécifie tous les états des ensembles et des individus après une évolution (décoré par un prime) en fonction des états avant cette évolution, y compris pour les ensembles non modifiés. Ceci nécessite de choisir celle des transitions validée qui est tirée ($Fired? \in enabled$). La variable *usePN* sert à vérifier que cette opération est permise, son utilité ne sera pas montrée ici (le lecteur pourra pour cela consulter [15].

Evolution

ΔPN
$Fired? : TRANSITION$

$usePN = play$
$Fired? \in enabled$
$P' = P$
$T' = T$
$arcTP' = arcTP$
$arcPT' = arcPT$
$WarcTP' = WarcTP$
$WarcPT' = WarcPT$
$M0' = M0$
$\forall p : PLACE \mid p \in P \bullet$
$(p \in arcPT\sim(\{Fired?\}) \cap arcTP(\{Fired?\})$
$\wedge\ M'(p) = M(p) - WarcPT(p \mapsto Fired?) + WarcTP(Fired? \mapsto p))$
$\vee$
$(p \in arcPT\sim(\{Fired?\}) \setminus arcTP(\{Fired?\})$

$\wedge\ M'(p) = M(p) - WarcPT(p \mapsto Fired?))$

$\vee$

$(p \in arcTP(\{Fired?\}) \setminus arcPT\sim(\{Fired?\})$

$\wedge\ M'(p) = M(p) + WarcTP(Fired? \mapsto p))$

$\vee$

$(p \notin arcPT\sim(\{Fired?\}) \cap arcTP(\{Fired?\})$

$\wedge\ M'(p) = M(p))$

$enabled' = \{t : TRANSITION \mid (\forall\ p : PLACE \bullet$

$(p,t) \in arcPT' \wedge WarcPT'(p,t) < M'(p)) \vee t \notin \text{ran}(arcPT') \bullet t\}$

$usePN' = usePN$

Dans ce schéma, le marquage du réseau est représenté à deux instants successifs : avant et après l'évolution. Le « temps logique » du RdPG est donc méta-modélisé à travers deux dates successives : ceci est suffisant pour décrire la dynamique d'évolution temporelle du réseau. Par contre, l'historique des dates de franchissement des transitions n'est pas représenté. Ceci est volontaire, car nous avons considéré que cet historique ne fait pas partie du langage, mais fait partie des méthodes d'utilisation de ce langage, à des fins de vérifications de propriétés par exemple [17]. Dans ces méthodes, cet historique serait méta-modélisé en définissant une relation d'ordre total entre les marquages atteignables (par exemple grâce à une suite).

I.4 Enrichissement du RdP généralisé

Dans la méthode retenue, la partie continue du modèle de référence est modélisée par des équations différentielles : le temps est ici un « temps physique » évoluant dans $\mathbb{R}$. Les réseaux de Petri utilisés intègrent des fenêtres temporelles dont les bornes sont déterminées à partir de la résolution de ces équations différentielles. Ces réseaux doivent donc utiliser, en plus du « temps logique », le même point de vue sur le temps que ces équations différentielles. Par conséquent, pour ces réseaux, le temps sera modélisé par une variable prenant ses valeurs dans $\mathbb{R}$. Nous utilisons ici les extensions proposées par [3] qui nous permettent d'utiliser $\mathbb{R}$ comme nouveau type.

$TIME == \mathbb{R}$

L'écoulement du temps est représenté à travers la variable *date* et le schéma *Time* :

Time

. $date : TIME$

Le schéma *PNTE* défini la classe de réseau de Petri utilisé par le 4-uplet $PNTE = \{PN, event, condition, window\}$, où :

- *PN* : RdP généralisé (comme défini dans la section V.1),
- *event* : un ensemble d'événements externes associés aux transitions,
- *condition* : application $TRANSITION \nrightarrow event$ associant les événements aux transitions,
- *window* : application $TRANSITION \nrightarrow \mathbb{R} \times \mathbb{R}$ associant une fenêtre temporelle aux transitions.

Inclure le nom du schéma *PN* dans le schéma *PNTE* revient à inclure tous les éléments de *PN* dans *PNTE* et ceci de façon très lisible. La variable *pndate* sera utilisée dans le schéma décrivant l'évolution du marquage pour déterminer le temps de marquage d'une place.

PNTE

. PN

. $event : \mathbb{F}\ EVENT$

. $window : TRANSITION \nrightarrow TIME \times TIME$

. $condition : TRANSITION \nrightarrow EVENT$

. $pndate : TIME$

. $\text{dom}(window) = T$

. $\text{dom}(condition) = T$

. $\text{ran}(condition) = event$

L'évolution d'un *PNTE* ne se fait plus à partir du choix d'une transition à tirer, mais à partir du choix de l'occurrence d'un événement *Event?*. Le schéma suivant spécifie cette évolution qui correspond au tir de la transition *Fired*. Cette évolution ne peut se faire que si le marquage du Réseau permet le tir de cette transition ($Fired \in enabled$) et si elle est associée à l'événement choisi ($condition(Fired) = Event?$). Le tir de la transition *Fired* ne peut en outre avoir lieu que si le « temps physique » écoulé depuis le tir précédent ($date - pndate$) est compris dans l'intervalle de temps associé à la transition ($t1 < (date - pndate) \wedge t2 > (date - pndate)$). Si aucune fenêtre temporelle n'est associée à la transition ($Fired \notin \text{dom}\ window$), seules les deux premières conditions sont vérifiées.

Dans ce schéma, l'évolution du marquage n'est pas différente de celle spécifiée dans le schéma décrivant l'évolution des réseaux de Petri généralisés. Ce sont les conditions de tir qui ont changées. Le langage Z nous a donc permis de faire coexister dans le même méta-modèle deux représentations du temps, un « temps logique » et un « temps physique ». Ces deux abstractions du temps sont utilisées pour décrire l'évolution du marquage d'un réseau.

PNTE_Evolution

$\Phi PNTE$

$\Xi Time$

$event? : EVENT$

$Event? \in event$

$usePN = play$

$\exists\ Fired : TRANSITION \mid$

$fired \in enabled \wedge$

$condition(Fired) = Event? \wedge$

$((Fired \in \text{dom}\ wondow \wedge$

$(\forall\ t1, t2 : TIME \mid (t1,t2) = window(Fired) \bullet$

$t1 < (date - pndate) \wedge$

$t2 > (date - pndate))) \vee$

$Fired \notin \text{dom}\ window) \bullet$

$\forall\ p : PLACE \mid p \in P \bullet$

$(p \in arcPT\sim(\{Fired?\}) \cap arcTP(\{Fired?\})$

$\wedge\ M'(p) = M(p) - WarcPT(p \mapsto Fired?) + WarcTP(Fired? \mapsto p))$

$\vee$

$(p \in arcPT\sim(\{Fired?\}) \setminus arcTP(\{Fired?\})$

$\wedge\ M'(p) = M(p) - WarcPT(p \mapsto Fired?))$

$\vee$

$(p \in arcTP(\{Fired?\}) \setminus arcPT\sim(\{Fired?\})$

$\wedge\ M'(p) = M(p) + WarcTP(Fired? \mapsto p))$
$\vee$
$(p \notin arcPT\sim(\{Fired?\}) \cap arcTP(\{Fired?\})$
$\wedge\ M'(p) = M(p)))$
$enabled' = \{t : TRANSITION \mid (\forall p : PLACE \bullet$
$(p,t) \in arcPT' \wedge WarcPT'(p,t) < M'(p)) \vee t \notin \text{ran}(arcPT') \bullet t\}$
$usePN' = usePN$
$pndate' = date$

I.5 Méta-modélisation des équations différentielles

Le modèle de référence utilisé dans la méthode retenue est constitué de deux modèles : un réseau de Petri et un système d'équations différentielles. Chaque équation différentielle est utilisée pour modéliser un comportement continu du système. Le Réseau de Petri modélise les changements de comportements continus du système. A chaque place du RdP est donc associée au plus une équation différentielle qu'elle valide ou inhibe. Toutes les équations sont linéaires, du premier degré, et à coefficients constants.

Ce type d'équation est mathématiquement défini. Sa méta-modélisation n'a donc d'intérêt que dans le cadre d'une intégration avec un méta-modèle d'un langage du domaine discret, ce qui est notre cas. Dans ce cadre le méta-modèle a pour objectif de spécifier les éléments qui caractérisent chaque équation, afin de permettre sa résolution. Quelle que soit l'équation différentielle utilisée dans notre système, elle a toujours la même forme : la dérivée par rapport au temps (évaluant dans $\mathbb{R}$) de la variable M_r est égale à une valeur constante q. Pour résoudre le système, il faut donner une valeur initiale M_{r0} et une valeur finale M_{r1}. Finalement, ce qui caractérise chaque système d'équations, c'est q et M_{r1}. La valeur de M_{r0} est donnée au moment de la résolution. Le schéma *Continue* représente l'état de ce système.

Continue
$q : \mathbb{R}$
$M_{r1} : \mathbb{R}$

Sa résolution est très simple : elle peut être spécifiée par un schéma.

Resolution
$\Xi Continue$
$M_{r0}? : \mathbb{R}$
$Tsol! : TIME$

$Tsol! = (Mr1 - Mr0?) / q$

L'intérêt de ce schéma est de nous permettre de spécifier le mécanisme de résolution, mécanisme qui sera intégré au schéma spécifiant l'évolution globale du modèle intégré, dans la section suivante.

VI MÉTA-MODÉLISATION DE L'INTÉGRATION

Le système est modélisé (voir section IV) comme la réunion d'une commande discrète et d'un modèle de référence scindé en une partie discrète et un ensemble de parties continues. Le réseau de Petri temporel à événement est défini par le schéma *PNTE* qui est maintenant utilisé comme type. Chaque partie continue est définie par le schéma *Continue* qui est également utilisé comme type. Le schéma *System* représente le système global. Il inclut deux sous-systèmes modélisés chacun par une instance de *PNTE* et un ensemble de sous-systèmes modélisés chacun par une instance de *Continue*. Il comprend également deux fonctions permettant d'associer chaque équation différentielle à une transition et à une place du réseau de Petri de référence.

System
$Commande : PNTE$
$Ref\_discret : PNTE$
$Ref\_cont : \mathbb{F}\ Continue$
$winpla : PLACE \rightarrow Continue$
$wintra : TRANSITION \rightarrow Continue$
---
dom $winpla \subseteq Ref\_discret.P$
dom $wintra \subseteq Ref\_discret.T$

La résolution de l'équation active à un instant donné se fait au moment du tir de la transition associée à l'événement reçu par le système. Cette résolution et le tir de la transition sont spécifiés dans le schéma *System_Evolution*. Cette évolution ne peut avoir lieu que sur occurrence d'un événement *Event?* et nécessite l'information $M_{r0}?$ qui sera utilisée comme condition initiale pour la résolution de l'équation. Elle se fait entre deux dates de « temps logique » successives, avec des conditions sur les dates du « temps physique » défini comme un temps continu réel. La partie écrite en gras regroupe toutes les conditions initiales nécessaires à l'évolution du système (nous y retrouvons les conditions initiales nécessaires au tir des deux Réseaux).

System_Evolution
$\Delta System$
$\Phi Commande$
$\Phi Ref\_discret$
$\Xi Time$
$Event? : EVENT$
$M_{r0}? : \mathbb{R}$
$Fired1, Fired2 : TRANSITION$

$\exists$ *Fired1*, *Fired2* : *TRANSITION* |
***Fired1* $\in$ *Commande.enabled* $\wedge$**
***Commande.condition*(*Fired1*) = *Event?* $\wedge$**
((*Fired1* $\in$ dom *Commande.window* $\wedge$
($\forall$ *t1*, *t2* : *TIME* |
(*t1*, *t2*) = *Commande.window*(*Fired1*) $\bullet$
***t1* < (*date* – *Commande.pndate*) $\wedge$**
***t2* > (*date* – *Commande.pndate*))) $\vee$**
***Fired1* $\notin$ dom *Commande.window*)**
$\vee$
***Fired2* $\in$ *Ref_discret.enabled* $\wedge$**
***Ref_discret.condition*(*Fired2*) = *Event?* $\wedge$**
((*Fired2* $\in$ dom *Ref_discret.window* $\wedge$
($\forall$ *t1*, *t2* : *TIME* |
(*t1*, *t2*) = *Ref_discret.window*(*Fired2*) $\bullet$
***t1* < (*date* – *Ref_discret.pndate*) $\wedge$**
***t2* > (*date* – *Ref_discret.pndate*))) $\vee$**
***Fired2* $\notin$ dom *Ref_discret.window*) $\bullet$**
($\forall$ *pnte* : *PNTE* $\bullet$

$$
\begin{aligned}
&\textbf{\textit{Event?}} \in \textbf{\textit{pnte.event}} \land \\
&\textbf{\textit{pnte.usePN}} = \textbf{\textit{play}} \bullet \\
&(\forall\, \textbf{\textit{p}} : \textbf{\textit{PLACE}}, \textbf{\textit{t}} : \textbf{\textit{TRANSITION}} \mid \\
&\textbf{\textit{p}} \in \textbf{\textit{pnte.P}} \land \textbf{\textit{t}} \in \{\textbf{\textit{Fired1}}, \textbf{\textit{Fired2}}\} \bullet \\
&\quad (p \in pnte.arcPT{\sim}(\{t\}) \cap pnte.arcTP(\{t\}) \\
&\quad \land pnte.M'(p) = pnte.M(p) - pnte.WarcPT(p \mapsto t) \\
&\qquad + pnte.WarcTP(t \mapsto p)) \\
&\qquad \lor \\
&\quad (p \in pnte.arcPT{\sim}(\{t\}) \setminus pnte.arcTP(\{t\}) \\
&\quad \land pnte.M'(p) = pnte.M(p) - pnte.WarcPT(p \mapsto t)) \\
&\qquad \lor \\
&\quad (p \in pnte.arcTP(\{t\}) \setminus pnte.arcPT{\sim}(\{t\}) \\
&\quad \land pnte.M'(p) = pnte.M(p) + pnte.WarcTP(t \mapsto p)) \\
&\qquad \lor \\
&\quad (p \notin pnte.arcPT{\sim}(\{t\}) \cap pnte.arcTP(\{t\}) \\
&\quad \land pnte.M'(p) = pnte.M(p))) \\
&\quad pnte.enabled' = \{t : TRANSITION \mid (\forall\, p : PLACE \bullet \\
&\qquad (p,t) \in pnte.arcPT' \land pnte.WarcPT'(p,t) < pnte.M'(p)) \\
&\qquad \lor t \notin \mathrm{ran}(pnte.arcPT') \bullet t\} \\
&\quad pnte.usePN' = pnte.usePN \\
&\quad pnte.pndate' = date) \\
&(\exists_1\, p : PLACE \mid Ref\_discret.M'(p) \neq 0 \land p \in \mathrm{dom}\ winpla \bullet \\
&Ref\_discret.window' = \{wintra{\sim}(winpla(p)) \mapsto \\
&\quad (0{,}95*(winpla(p).Mr1 - Mr0?) / winpla(p).q, \\
&\qquad 1{,}05*(winpla(p).Mr1 - Mr0?) / winpla(p).q)\} \lor \\
&Ref\_discret.window' = \{\}) \\
&Commande.window' = \{\}
\end{aligned}
$$

Le schéma spécifie ensuite l'évolution du marquage de chaque Réseau de Petri, et ceci de façon identique à celle utilisée dans le schéma *PNTE_Evolution*. Enfin ce schéma spécifie les nouvelles fenêtres temporelles associées aux transitions susceptibles d'être tirées, et supprime toutes les autres (6 dernières lignes du schéma).

Ce schéma définit donc rigoureusement une méthode de modélisation de systèmes dynamiques hybrides. A partir de cette définition rigoureuse, cette méthode peut être facilement implantée. Le méta-modèle constitue alors, pour le programmeur, un cahier des charges univoque, qu'il doit respecter, tout en restant libre des algorithmes programmés.

VII CONCLUSION

Dans cet article, nous avons montré que le langage Z dispose des primitives nous permettant de méta-modéliser deux abstractions du temps : le temps discret et le temps continu. A travers [16], [15], [17], et cet article, nous avons successivement abordé aussi bien la syntaxe que la sémantique des langages et des méthodes de conception des SDH. Nous avons donc montré la capacité du langage Z à être utilisé comme langage de méta-modélisation, permettant ainsi l'étude des langages et des méthodes de conception des SDH. Nos travaux futurs consisterons à utiliser ce méta-langage pour définir de nouvelles méthodes rigoureuses de conception de systèmes dynamiques discrets ou hybrides, notamment dans le cadre de la modélisation de la dynamique d'évolution de la commande des systèmes.

RÉFÉRENCES

[1] « Automatisation des processus mixtes : les systèmes dynamiques hybrides », AFCET – SEE, 1998.

[2] D. ANDREU, « Commande et supervision des procédés discontinus : une approche hybride », thèse de doctorat de l'Université Paul Sabatier de Toulouse, 1996.

[3] R. BARDEN, S. STEPNEY et D. COOPER, Z in practice, Prentice Hall, 1994.

[4] A. BENVENISTE, Compositional and uniform modeling of hybrid systems, IEEE trans. on automatic control, vol. 43, n°4, avril 1998, pp. 579-583.

[5] E. BON-BIEREL, Contribution à l'intégration des modèles de systèmes de production manufacturière par méta-modélisation, thèse de doctorat de l'Univ. de Nancy I, 1998.

[6] R. CHAMPAGNAT *et al*, A gas storage example as a benchmark for hybrid modelling : a comparative study, APII--JESA, vol. 32, n° 9-10, 1998, pp. 1233-1253.

[7] F. COUFFIN, Modèles de données de référence et processus de spécialisation pour l'intégration des activités de conception en Génie Automatique, thèse de doctorat de l'École Normale Supérieure de Cachan, 1997.

[8] A. J. GALLOWAY et S. J. O'BRIAN, A formal abstract syntax for Ward-Mellor SA/RT Essential Models, rapport technique TEES-SCM-4-94, Université de Teesside, 1994, ftp.tees.ac.uk/pub/a.galloway/tech.report/tees-scm-4-94.ps.Z

[9] D. M. GEE, Formal specification of visual languages, 1995, http://lion.unn.ac.uk/~davidg/papers/fspec.ps.Z

[10] S. KOWALEWSKI, S. ENGELL, J. PREUSSIG et O. STURSBERG, Verification of logic controllers for continuous plants using timed condition/event-system models, Automatica, vol. 35, n° 3, avril 1999, pp. 505-518.

[11] J.-J. LESAGE, B. DENIS et G. TIMON, Une approche formelle de la modélisation intégrée, La conception an 2000 et au-delà : outils et technologie, Strasbourg, France, 24-27 novembre 1992.

[12] D. LIGHTFOOT, La spécification formelle avec Z, Teknea, 1994, traduit de Formal Specification Using Z par H. HABRIAS et P.-M. DELPECH.

[13] J. LUNZE, B. NIXDORF et J. SCHRÖDER, Deterministic discrete-event representations of linear continuous-variable systems, Automatica, vol. 35, n° 3, mars 1999, pp.371-384.

[14] T. MURATA, Petri Nets : Properties, Analysis and Applications, proceedings of the IEEE, vol. 77, n° 4, 1989.

[15] L. PIÉTRAC, Apport de la méta-modélisation formelle pour la conception des systèmes automatisés de production, doctorat de l'École Normale Supérieure de Cachan, 1999.

[16] L. PIÉTRAC, B. DENIS et J.-J. LESAGE, Formalization of the design of control systems, ISRAM'96, WAC'96, Montpellier, France, 27-30 Mai, 1996.

[17] L. PIÉTRAC, B. DENIS et J.-J. LESAGE, A formal meta-modelling approach for the design of automated manufacturing systems, IWFMM'99, pp. 65-74, Zaragoza, Espagne, septembre 1999.