Un méta-modèle pour la représentation de systèmes à structure dynamique applicable à la reconfiguration dynamique partielle

Clément Foucher — Min Zhu

LAAS-CNRS, Université de Toulouse, UPS, Toulouse

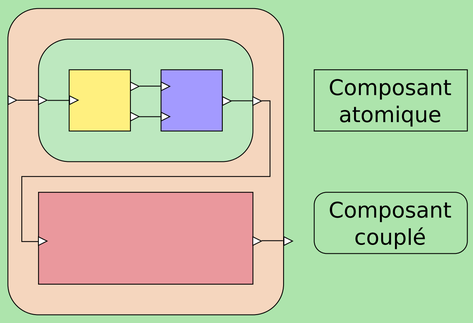
Contact : clement.foucher@laas.fr

DEVS et Parallel DEVS

DEVS (Discrete Event System Specification) est un formalisme mathématique de modélisation et de simulation de systèmes à évènements discrets.

DEVS permet de représenter des systèmes hiérarchiques basés sur des composants. Les composants peuvent être de deux types : couplés ou atomiques. Un composant atomique exprime un comportement basé sur la notion d'états et de transitions entre ces états, les transitions pouvant être déclenchées par le temps ou un évènement d'entrée. Un composant couplé définit un groupe de composants et leurs relations.

Il existe de multiples variantes du formalisme original, et nous nous basons sur la



Un composant PDEVS atomique est défini comme suit :

$$M = < X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, \tau >$$

Avec

- X et Y respectivement les ensembles des valeurs d'entrée et de sortie
  - Avec $X = \{(p, v) \mid p \in InPorts, v \in X_p\}$
  - et $Y = \{(p, v) \mid p \in OutPorts, v \in Y_p\}$
    - Avec $InPorts$ et $OutPorts$ les ensembles des ports d'entrée/sortie
    - et X_p et Y_p les ensembles de valeurs autorisées pour un port p
- S l'ensemble des états possibles du composant
- $\delta_{ext} : Q \times X \to S$ la fonction de transition externe
  - Avec $Q = \{(s, e) \mid s \in S, 0 \leq e \leq \tau(s)\}$ l'état total du composant
    - Avec e le temps écoulé depuis la dernière transition
- $\delta_{int} : S \to S$ la fonction de transition interne
- $\delta_{con} : Q \times X \to S$ la fonction de transition confluente
- $\lambda : S \to Y$ la fonction de sortie
- $\tau : S \to \mathbb{R}^+_{0,\infty}$ la fonction d'avancement du temps

Un composant PDEVS couplé est défini comme suit :

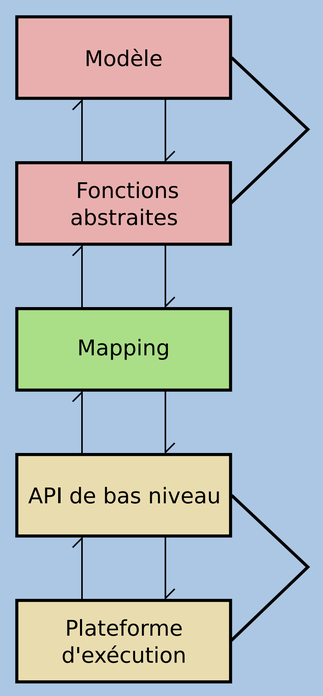
$$N = < X, Y, D, \{M_d\}, EIC, EOC, IC >$$

Avec

- X, Y comme définis pour un composant atomique
- D l'ensemble des noms des composants contenus dans le couplé
- $\{M_d\}$ l'ensemble des composants du couplé, avec $d \in D$
- EIC la fonction de couplage des entrées, qui lie les entrées du composant couplé avec des entrées de composants contenus dans le couplé : lie $p_N \in InPorts_N$ à $p_d \in InPorts_d, d \in D$
- EOC la fonction de couplage des sorties, qui lie des sorties de composants contenus dans le couplé avec les sorties du composant couplé : lie $p_d \in OutPorts_d, d \in D$ à $p_N \in OutPorts_N$
- IC la fonction de couplage interne, qui lie des entrées et des sorties de composants contenus dans le couplé : lie $p_a \in OutPorts_a$, $a \in D$ à $p_b \in InPorts_b$, $b \in D, a \neq b$

Approche MDA

Model-Driven Architecture — Architecture dirigée par les modèles



Le PIM (Platform-Independent Model — Modèle indépendant de la plateforme) est, comme son nom l'indique, un modèle d'application ne dépendant pas d'une plateforme d'exécution.

Le PSM (Platform-Specific Model — Modèle spécifique à une plateforme) correspond à un modèle d'application destiné à s'exécuter sur une plateforme particulière. Il est obtenu en mettant en correspondance un PIM avec un PDM.

Le PDM (Platform Definition Model — Modèle de définition de la plateforme) décrit la plateforme d'exécution, et en propose une abstraction qui permet son utilisation par différentes applications.

Problématique

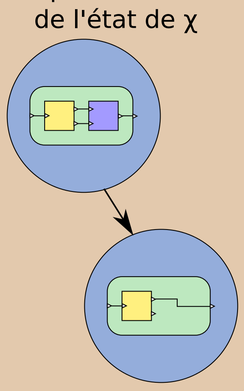
Différentes extensions de DEVS permettent la représentation de modèles à structure dynamique, mais celles-ci ne sont pas aisément utilisables dans une approche MDA. Notamment dans la phase de mapping entre le PIM et le PDM, qui doit être automatisable.

Pour cela, il est nécessaire que l'interface de bas niveau du PIM soit rapidement rapprochable de l'interface de haut niveau du PDM. Or, l'approche généralement utilisée dans DEVS et ses variantes est une approche mathématique abstraite basée sur des fonctions « boîte noire », dont la définition est libre.

Par exemple, dans DSDE (Parallel Dynamic Structure DEVS), le changement de structure du modèle est réalisé par un composant spécifique : χ, le « network executive ». Il s'agit d'un composant spécial, dont chaque état correspond à un état structurel du modèle. L'état structurel du modèle est obtenu par l'application de la fonction γ : $S_\chi \to \Sigma^*$ (S_χ étant l'état de χ et Σ^* l'ensemble des structures possibles pour le modèle) qui permet de réaliser la correspondance entre l'état du composant et l'état structurel du modèle.

Si cette approche peut être mathématiquement parfaite pour un PIM, car elle permet de définir arbitrairement n'importe quelle fonction γ, il est en revanche difficile de lui faire correspondre une réalité dans un PDM.

Exemple de représentation de l'état de χ



Fonctions de structure dynamique

Plutôt que de définir un composant spécial associé à une fonction complexe, nous avons choisi de raisonner sur des fonctions manipulant les ensembles contenus dans les composants couplés.

En effet, la structure du modèle est définie grâce aux composants couplés. Et comme chacun de ces composants contient des ensembles (de composants, de ports et de connexions), nous définissons différentes fonctions manipulant ces ensembles.

Il faut également identifier précisément à quel élément du modèle se réfère un appel de fonction. Pour cela, nous définissons la notion d'identifieur (id) unique, et applicable à tout élément d'un modèle. Les composants atomiques sont alors munis d'une fonction de sortie supplémentaire, λ_{SC}, qui permet d'associer un appel de

Fonctions de stucture dynamique λ_{SC} définies dans PrDEVS

- addComponent : $T \times ID_N \to ID$
- removeComponent : $ID_{PR} \to \emptyset$
- addPort : $ID_N \times P_{type} \times P_{dir} \to ID$
- removePort : $ID_N \times ID_P \to \emptyset$
- addConnection : $ID_{PR} \times ID_P \times ID_{PR} \times ID_P \to \emptyset$
- removeConnection : $ID_{PR} \times ID_P \times ID_{PR} \times ID_P \to \emptyset$
- getContext : $ID_{PR} \to ID_\Phi$
- setContext : $ID_{PR} \times ID_\Phi \to \emptyset$

Ports

Dans PDEVS, les ports ne sont définis qu'indirectement. En effet, les entrées et les sorties des composants sont définies par des couples *(port, valeur autorisée)*, les ports étant uniquement définis par leurs noms.

Afin de permettre la manipulation des ensembles de ports, nous définissons les ports en tant qu'objets mathématiques de la façon suivante :

$$P^{id} = < Direction, DataType >$$

Avec

$Direction \in P_{dir} = \{in, out\}$

$DataType \in P_{type}$

Avec P_{type} un ensemble arbitraire, par exemple $\mathbb{N}$, $\mathbb{R}$, $\mathbb{B} = \{True, False\}$, etc.

Racine, bibliothèques et types

Dans PrDEVS, la racine du modèle est définie de la manière suivante :

$$PRDEVS = < L, L_\Phi, C^{Top} >$$

Dans lequel C^{Top} est un composant couplé, L représente la bibliothèque de composants et L_Φ la bibliothèque d'états.
La bibliothèque L est un ensemble de composants disponibles pour ajout dans le modèle.
Les composants de la bibliothèque ne possèdent pas d'identifiant, mais un type. La bibliothèque est alors définie comme suit :

$$L = \{C_t \mid t \in T\}$$

T étant l'ensemble des types de composants définis dans le modèle.

En dehors de la bibliothèque (*i.e.* dans le modèle), les composants conservent bien entendu leur type, mais celui-ci n'apparait plus nécessairement dans la notation, l'identifiant étant suffisant.

Tous les composants d'un même type partagent la même définition et le même comportement, mais peuvent être dans un état différent.

État du système

Dans les variantes dynamiques de DEVS, la notion d'état du système n'est que survolée : par exemple, dans DSDE, l'état des composants du modèle après un changement de structure est défini comme étant le même qu'avant le changement de structure pour ceux qui n'ont pas été modifiés, et l'état initial pour ceux qui ont été ajoutés.

Or, dans de nombreux cas, il peut être nécessaire de ne pas forcément faire démarrer un composant dans son état initial, ou de forcer son changement. Par exemple, si on replace dans le modèle un composant qui en avait été préalablement ôté, on peut souhaiter le remettre dans l'état dans lequel il était avant sa suppression.

Pour cela, nous définissons une bibliothèque d'états, permettant de sauvegarder et de restaurer des états en utilisant les fonctions de structure dynamiques. Celle-ci est définie comme suit :

$$L_\Phi = \{Q^{id}_t \mid id \in ID_\Phi, t \in T\}$$

Ici, L_Φ stocke les états sauvegardés au fil de l'exécution du modèle. Elle peut également contenir statiquement des états qui seront utilisés à l'exécution.

PDM matériel

La définition d'un PDM matériel était un des objectifs principaux de ces travaux. Nous nous basons sur les FPGA à reconfiguration dynamique partielle pour la prise en charge de la structure dynamique.

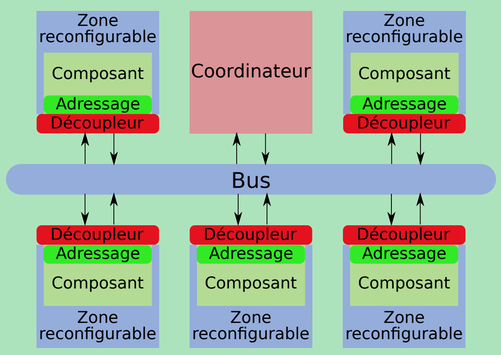
Un coordinateur est implanté dans le système, et permet l'ordonnancement et le contrôle des composant, ainsi que la mise en œuvre des fonctions de structure dynamique

La notion de zone reconfigurable permet de créer des zone d'accueil pour les composants du modèle. Ces zones sont reliées par un bus, constitué de trois composantes : le bus de contrôle, le bus de communication et le bus de structure dynamique. Le bus de structure dynamique permet notamment aux composants de réaliser des appels de fonction de structure dynamique auprès du coordinateur.

La gestion de la structure dynamique est réalisée à plusieurs niveaux : l'ajout ou la suppression d'un composant agit par reconfiguration partielle d'une zone reconfigurable, les modifications de ports ou de connexions par la mise à jour de tables d'adressage, et les actions sur le contexte par remplacement de la mémoire des composants, via le bus de structure dynamique.

La hiérarchie est mise à plat dans le système physique, mais reste présente de manière virtuelle au niveau du coordinateur. On n'a donc dans le système physique que des composants atomiques, mais ceux-ci peuvent continuer à interagir avec des ports de composants couplés, le coordinateur effectuant des translations dans l'espace d'adressage pour mettre en correspondance les ports virtuels des composants couplés avec les ports réels des composants atomiques.

Des tables d'adressages locales à chaque composant, lui indiquant les connexions de ses ports et tenues à jour par le coordinateur, permettent la communication entre les composants sans avoir à transiter à chaque fois par le coordinateur.



PIM et PDM

L'indépendance du modèle de haut niveau vis à vis de la plateforme permet, depuis un même modèle abstrait, de cibler n'importe quelle plateforme d'exécution pour laquelle un PDM PrDEVS a été défini.

Nous avons défini un PDM pour FPGA dotés de reconfiguration dynamique partielle, et un PDM logiciel pour les langages orientés objets.

Tout modèle exprimé dans le formalisme PrDEVS sera donc un PIM qui pourra être transformé en PSM pour chaque plateforme.

PDM logiciel

Nous avons choisi pour le PDM logiciel une approche orientée objet. Celle-ci convient parfaitement, tant les notions de PrDEVS sont proches du monde objet. Ce n'est d'ailleurs pas un hasard, car nous nous sommes inspirés de l'approche objet pour créer le formalisme, afin d'importer ces notions dans le PDM matériel.

La création d'un PDM logiciel orienté objet est donc aisée, car tous les mécanismes inhérents à la création et la suppresison de composants font naturellement partie de l'approche de programmation orientée objet.

Ainsi, les composants, les ports ou encore les connexions peuvent être représentés par des objets. Pour les composants, on exploite la notion d'héritage, en utilisant une classe racine pour tous les composants, qui rassemble les caractéristiques commune aux composants atomiques et couplés. Les différents types de composants définis dans un modèle PrDEVS hériteront ensuite de ces types racine.

Les ensembles seront représentés par des listes ou des tableaux dynamiques, selon la notion la plus adaptée au langage utilisé.

Enfin, les notions de création et de suppression correspondent de manière naturelle aux opérations « new » et « delete » qui font partie du socle de la programmation orientée objet.