Une sémantique multi-paradigme pour simuler des modèles SysML avec SystemC-AMS

Daniel Chaves Café[1,2], Filipe Vinci dos Santos[2], Cécile Hardebolle[1], Christophe Jacquet[1] and Frédéric Boulanger[1]

[1] Département Informatique, Supélec, Gif-sur-Yvette, France.
[2] Chaire Thales/Supélec en Conception analogique avancée, Supélec, Gif-sur-Yvette, France.
{daniel.cafe, filipe.vinci, cecile.hardebolle, christophe.jacquet, frederic.boulanger}@supelec.fr

Abstract

Dans le contexte de la modélisation de systèmes, SysML apparait comme un langage pivot de spécification et de documentation. Ses diagrammes permettent la définition de la structure et du comportement de systèmes. La flexibilité de SysML a pour inconvénient qu'il n'existe pas de méthode standard pour définir leur sémantique. Ce problème est flagrant dans la conception de systèmes hétérogènes, où différentes sémantiques opérationnelles peuvent être utilisées. Cet article présente une manière de donner une sémantique opérationnelle aux éléments de SysML sous la forme de transformations vers le langage SystemC-AMS, permettant ainsi la simulation de modèles SysML.

1 Introduction

Un système hétérogène est constitué de composants de différentes natures, qui sont modélisés selon des formalismes distincts. Par exemple, un accéléromètre MEMS a une partie mécanique, une partie analogique et une interface numérique. Dans le domaine numérique, le formalisme à événements discrets est efficace pour la simulation grâce à l'abstraction des phénomènes analogiques qui font qu'une bascule change d'état. Dans le domaine analogique, la modélisation par réseaux de composants électriques permet de décrire la topologie du réseau pour en déduire les équations différentielles. La simulation des systèmes hétérogènes permet de garantir une fabrication correcte dès le premier essai. Le métier d'architecte de systèmes consiste à intégrer différents paradigmes dans un même modèle, ce qui pose des problèmes d'adaptation et fait appel à des compétences pluridisciplinaires et à la maîtrise des outils de simulation.

Les outils de modélisation hétérogène sont encore en phase d'expérimentation et de maturation. Ptolemy II [3] gère l'hétérogénéité par hiérarchie. Chaque composant est considéré comme une boite noire et la sémantique d'exécution et de communication est définie par une entité appelée *director*. Cette entité définit le modèle de calcul (MoC) de chaque composant. Pour arriver à cet objectif, Ptolemy II définit un moteur d'exécution générique [9] avec trois phases distinctes : l'initialisation, l'itération (pre-fire, fire et post-fire) et la finalisation (wrapup). Chaque *director* compose le comportement des blocs en redéfinissant ces phases d'exécution.

Inspiré de Ptolemy II, ModHel'X [1, 7] a été créé pour modéliser explicitement l'adaptation sémantique en rajoutant au moteur d'exécution générique de Ptolemy des phases d'adaptation sémantique. Cela donne un moyen efficace de définir la sémantique des interactions entre différents modèles de calcul. Dans cet article, nous présentons une méthode pour définir la sémantique opérationnelle (les MoCs) ainsi que l'adaptation sémantique pour des modèles hétérogènes SysML par traduction en SystemC-AMS.

SysML est un langage graphique de spécification de systèmes dont les diagrammes facilitent la communication entre différentes équipes d'un projet pluridisciplinaire. La sémantique

opérationnelle de ces diagrammes n'est toutefois pas précisément définie, ce qui est un obstacle à l'exécution de modèles SysML. L'implémentation de cette sémantique peut être réalisée dans un langage capable de simuler des modèles hétérogènes. Comme une première preuve de concept, nous avons choisi d'utiliser SystemC-AMS [6] (Analog and Mixed-Signal) qui est une bibliothèque C++ contenant un noyau de simulation à événements discrets ainsi qu'un ensemble de blocs de base pour la modélisation des systèmes mixtes. Ce langage supporte les modèles de calcul Dicrete Event (DE), Timed DataFlow (TDF) et Continuous Time (CT).

Notre approche consiste à générer du code SystemC-AMS à partir d'un modèle SysML annoté avec des éléments qui donnent la sémantique d'exécution de chaque bloc SysML et la sémantique d'adaptation entre blocs. Nous utilisons des techniques de l'ingénierie dirigée par des modèles (IDM) pour réaliser les transformations de modèles et la génération de code.

2 L'approche

Notre approche consiste à réaliser deux transformations de modèles. En partant d'un modèle SysML, nous appliquons une transformation "model-to-model" (M2M) écrite en ATL (Atlas Transformation Language)[8] pour générer une représentation intermédiaire du système dans le langage SystemC-AMS. La génération de code se fait juste après, en appliquant une deuxième transformation de type "model-to-text" (M2T) écrite en ACCELEO[10].

Nous utilisons des contraintes SysML pour indiquer le modèle de calcul utilisé par un bloc donné. Les mots clés "CT Block", "DE Block" et "FSM Block" sont utilisés pour spécifier l'utilisation des MoCs Continuous Time, Discrete Event et Finite State Machine, respectivement. La sémantique de ces MoCs est décrite dans [9].

Nous appliquons ces MoCs au sous-ensemble des diagrammes SysML qui nous semblent leur correspondre. Un automate par exemple, est modélisé par un diagramme d'états-transitions, un modèle à temps continu peut être representé par un diagramme d'interconnections de blocs où chaque bloc répresente une fonction. D'autres solutions peuvent être imaginées, comme l'utilisation des diagrammes paramétriques pour la définition des équations différentielles.

Nous considerons dans ce travail l'adaptation sémantique entre les différents domaines, par exemple l'échantillonnage entre un sous-système à temps continu et un sous-système à temps discret, que nous exprimons dans des commentaires liés aux ports des modules.

3 Cas d'étude

Pour illustrer l'approche, prenons l'exemple d'un véhicule à vitesse contrôlée, présenté en détail dans [2] et résumé ici. Les diagrammes de la figure 1 montrent comment appliquer une sémantique à un bloc SysML en lui ajoutant des annotations textuelles. Nous nous intéressons à une modélisation haut niveau de la dynamique du système en utilisant des équations de la physique classique. Nous déduisons l'accélération de $F = m \times a$, puis la vitesse et le déplacement en intégrant l'accélération. La dynamique du système est modélisée par le formalisme CT. Nous définissons une équation différentielle dans un diagramme IBD en assemblant des fonctions de base, comme l'intégrateur et le gain (voir figure 1 à gauche). Le contrôle est modélisé par un diagramme états-transitions où la rétroaction de la force dépend de l'état du contrôleur (Accelerate, Hold et Brake). Ces deux formalismes sont non seulement exprimés de manières différentes mais ont aussi des sémantiques opérationnelles différentes.

L'adaptation entre différents domaines est décrite par des commentaires liés aux ports du bloc *i_dynamics* dans le diagramme de droite. Une fois l'interface annotée par le mot clé

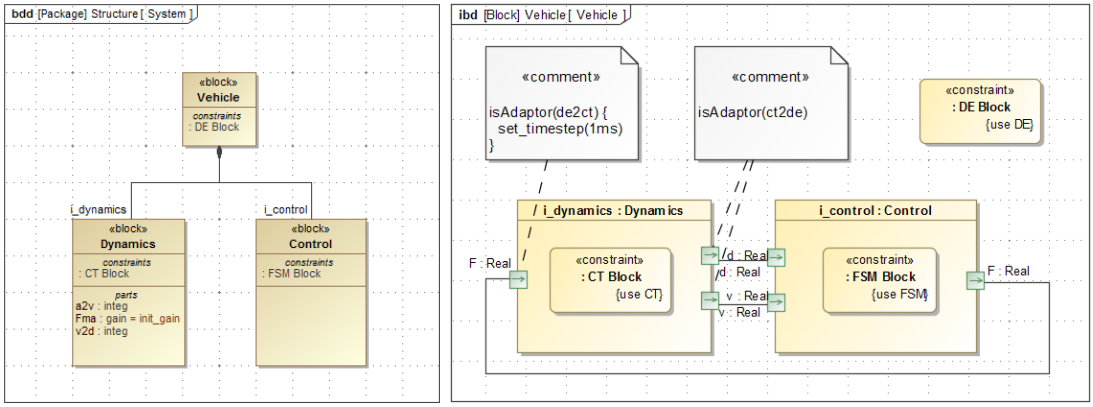

Figure 1: Exemple du véhicule à vitesse contrôlée : modèles SysML annotés

isAdaptor, nous lui appliquons une sémantique d'adaptation spécifique. Dans le cas d'un adaptateur *de2ct* le dernier événement capturé par le port est enregistré et sa valeur répétée à un pas d'échantillonnage fixe, donné par la directive *set_timestep*. En sortie, l'adaptation *ct2de* génère un événement à chaque changement de la valeur produite en sortie.

Cet approche est cependant très dépendante du choix du langage cible (ici SystemC-AMS), de même que les modèles de calcul utilisables. Nous ne pouvons implémenter que les MoCs que SystemC-AMS est capable d'exécuter. Cette limitation n'est pas contraignante car SystemC possède un moteur d'exécution à événements discrets qui supporte une large gamme de modèles de calcul discrets, et la partie AMS supporte les modèles continus. Il est important de souligner que la sémantique des MoCs est prise en compte de deux manières : soit par utilisation directe de la bibliothèque de base de SystemC, soit par implémentation dans la transformation. Dans l'exemple, la sémantique *block dynamics* s'appuie sur l'assemblage de blocs de la bibliothèque LSF (Linear Signal Flow) de SystemC-AMS. Dans le contrôleur, la sémantique de la machine à états est codée par la transformation M2M car SystemC n'a pas de MoC FSM. Cela a une influence sur les performances de simulation comme discuté dans [11].

La même reflexion s'applique aux mécanismes d'adaptation. L'échantillonnage des données et la production d'événements discrets sont réalisés par des adaptateurs spécifiques de la bibliothèque de SystemC-AMS. L'adaptation qui est faite à l'entrée du bloc *dynamics* est traduite en un module SystemC capable de transformer un événement discret de la machine à état en échantillons au pas fixe défini par la commande *set_timestep(1ms)*. L'adaptateur de sortie détecte les changements de valeur pour produire des événements. Une autre adaptation pourrait ne générer que les événements qui correspondent à une transition de l'automate. Cela permettrait une simulation plus performante mais rendrait l'adaptateur dépendant des modules qui lui sont connectés. Les adaptateurs possibles sont également limités par le langage cible. Dans notre exemple, nous n'avons utilisé que les adaptateurs de la bibliothèque SystemC-AMS. La conception d'adaptateurs spécialisées peut être réalisée sous forme de modules dédiés, comme discuté dans [4] avec le concept de *thick adapter*.

Finalement, la simulation de ce système est obtenue par transformation du modèle SysML en code SystemC-AMS qui est ensuite compilé et exécuté. Les résultats de simulation de cet exemple ont été présentés dans [2].

4 Discussions

Cet article définit une première approche pour la modélisation multi-paradigme en SysML ciblant la simulation mixte, que nous avons illustrée sur un modèle à trois formalismes (CT, DE et FSM). Nous avons spécifié la sémantique de chaque bloc SysML avec des annotations textuelles et la sémantique d'adaptation dans les commentaires liés aux ports. Le code exécutable est généré par transformation de modèles en utilisant ATL et ACCELEO.

Cette technique est un premier pas vers un framework générique de génération de code pour d'autres langages, par exemple VHDL-AMS. Notre approche en deux étapes (M2M et M2T) permet d'envisager l'extensibilité à d'autres MoCs par ajout de règles de transformation M2M correspondant au MoC désiré, sans devoir changer le générateur de code (partie M2T). Notre objectif est de découpler au maximum l'aspect sémantique des MoCs et l'aspect génération de code afin de supporter plus facilement de nouveaux MoCs et de nouveaux langages cibles. En ce qui concerne la définition des MoCs et l'adaptation sémantique, nous suivons les travaux en cours sur ce sujet au sein de l'initiative GeMoC [5].

4.1 Réflexions sur l'approche

Depuis la publication de ce travail exploratoire, nous avons amélioré l'intégration de notre approche aux techniques de l'IDM. Nous utilisons désormais des steréotypes pour définir les MoCs au lieu des mots-clés dans les contraintes SysML. Ces contraintes sont désormais réservées à la spécification des équations différentielles dans le formalisme CT. Les annotations des adaptateurs ont aussi évolué vers un langage dédié à la modélisation de leur comportement, ce qui évite de polluer le modèle avec du code spécifique au langage cible.

References

[1] F. Boulanger and C. Hardebolle. Simulation of Multi-Formalism Models with ModHel'X. In *Int. Conference on Software Testing, Verification, and Validation*, pages 318–327. IEEE, 2008.

[2] D. Cafe, F. V. dos Santos, C. Hardebolle, C. Jacquet, and F. Boulanger. Multi-paradigm semantics for simulating SysML models using SystemC-AMS. In *Specification and Design Languages, 2013, Forum on*, pages 82–89. IEEE, 2013.

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

[4] A. Ferrari, L. Mangeruca, O. Ferrante, and A. Mignogna. Desyreml: a sysml profile for heterogeneous embedded systems. *Embedded Real Time Software and Systems, ERTS*, 2012.

[5] GeMoC. On the globalization of Modeling Languages, May 2011. `http://gemoc.org`.

[6] C. Grimm, M. Barnasconi, A. Vachoux, and K. Einwich. An introduction to modeling embedded analog/mixed-signal systems using SystemC AMS extensions. In *DAC*, 2008.

[7] C. Hardebolle and F. Boulanger. ModHel'X: A component-oriented approach to multi-formalism modeling. In *Models in Software Engineering*, pages 247–258. Springer, 2008.

[8] F. Jouault and I. Kurtev. Transforming models with ATL. *Satellite Events at the MoDELS 2005 Conference*, pages 128–138, 2006.

[9] E. A. Lee. Disciplined heterogeneous modeling. In *Model Driven Engineering Languages and Systems*, pages 273–287. Springer, 2010.

[10] J. Musset, E. Juliot, S. Lacrampe, W. Piers, C. Brun, L. Goubet, Y. Lussaud, and F. Allilaire. Acceleo User Guide, 2006.

[11] H. D. Patel and S. K. Shukla. Systemc kernel extensions for heterogeneous system modeling. *The Netherlands: Kluwer Academic Publishers*, pages 9–71, 2004.