Ingénierie dirigée par les modèles : du modèle au circuit logique en utilisant la génération automatique de code avec la plateforme Project DEVS

Clément Foucher, Vincent Albert

Ingénierie dirigée par les modèles : du modèle au circuit logique en utilisant la génération automatique de code avec la plateforme Project DEVS

Clément Foucher Vincent Albert
LAAS-CNRS, Université de Toulouse, UPS, Toulouse, France
clement.foucher@laas.fr vincent.albert@laas.fr

1 Introduction

La conception d'un produit est une opération complexe, sans cesse contrainte par des objectifs de réduction du temps de développement. L'ingénierie système définit un cadre général destiné à optimiser le cycle de développement et à standardiser les échanges entre les différents acteurs.

Une méthode particulièrement utilisée à l'heure actuelle consiste à modéliser le produit le plus en amont possible, afin de permettre de réaliser sur les modèles différents tests, notamment via la simulation. L'ingénierie dirigée par les modèles propose notamment une approche consistant à modéliser le système dès le plus haut niveau puis à raffiner successivement les modèles pour arriver au produit final. La traçabilité entre les différents niveaux de modèle est primordiale dans cette approche, et peut passer idéalement par des transformations automatisées du modèle. Par exemple, la méthode dite de *Rapid Control Prototyping* (RCP) génère automatiquement le code exécutable destiné à la partie contrôle-commande du système depuis les modèles de celui-ci.

Dans cet article, nous présentons la plateforme Project DEVS, qui permet la modélisation et la simulation de modèles à évènement discrets. Nous nous concentrerons plus particulièrement sur la mise en œuvre de la génération de code matérielle, permettant une exécution de la simulation sur FPGA.

2 ProDEVS : du simulateur à la plateforme

Le formalisme DEVS [4] décrit un modèle comme un ensemble de composants de deux types : les composants couplés et les composants atomiques. Un composant couplé contient un ensemble d'autres composants, couplés ou atomiques, et décrit les liens existant entre eux. Un composant atomique, quant à lui, décrit un comportement guidé à la fois par le temps et les évènements extérieurs, et représenté par des états. Sur la base de ces objets, mathématiquement définis, différents algorithmes de simulation (ou simulateurs abstraits) permettent de faire évoluer le système selon des règles strictes.

Project DEVS [1] était initialement ProDEVS, un simulateur à évènements discrets de modèles au formalisme DEVS. ProDEVS permet de saisir ces différents types de composants, dans une approche graphique, pour ensuite les simuler en utilisant les algorithmes définis par Zeigler. Le même modèle peut par exemple être simulé en utilisant le formalisme CDEVS (*Classic DEVS*, nom donné à posteriori au formalisme DEVS initial) ou PDEVS (*Parallel DEVS*).

Par la suite, nous avons fait évoluer ProDEVS dans une approche de plateforme hétérogène. L'interface utilisateur et le cœur de simulation initial, ProDEVS, est ainsi devenu un composant au sein d'un écosystème, Project DEVS. Notre objectif ici est de permettre le déploiement des modèles sur des simulateurs de différentes natures. Ainsi, si le cœur historique de ProDEVS utilisait des modèles décrits en Java pour effectuer la simulation, dans Project DEVS, le choix du langage de description du modèle peut pour l'instant être fait entre Java, C++ ou VHDL. On note dans cette liste à la fois des langages de conception logiciels et de description matérielle. L'idée est de ne pas se restreindre à un type de calculateur, et de permettre une liberté dans la mise en œuvre.

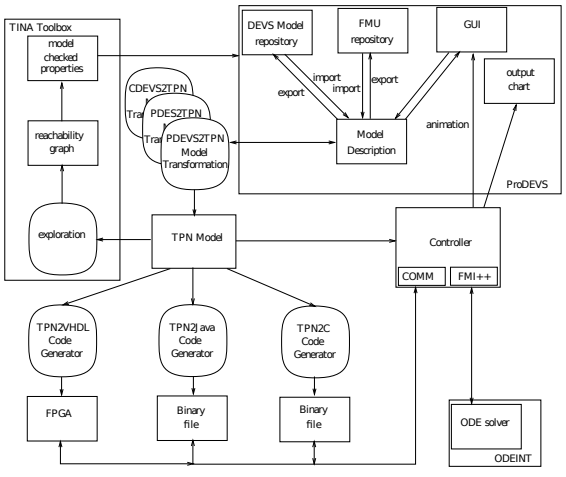

FIGURE 1 : Architecture du simulateur.

Une vision de la plateforme Project DEVS est présentée sur la figure 1. On notera l'interfaçage qui peut être fait avec TINA [2], qui permet la vérification formelle des modèles transformés en Réseaux de Petri, ou encore la possibilité d'importer des modèles au format FMI.

Étape	Durée (sw)	Durée (hw1)	Durée (hw2)
Génération du code	4.3 ms	4.7 ms	4.8 ms
Compilation du code	844.0 ms	102 s	102 s
Programmation de la cible	N/A	3 s	3 s
Durée de la simulation	624.0 ms	6.0 ms	1.0 s
Total	1.5 s	105.0 s	106.0 s

TABLE 1 : Durées pour le modèle du gen-buf-proc.

Étape	Durée (sw)	Durée (hw1)	Durée (hw2)
Génération du code	85.3 ms	1718.6 ms	1717.5 ms
Compilation du code	2.8 s	655 s	663 s
Programmation de la cible	N/A	3 s	3 s
Durée de la simulation	5194.8 s	113.0 ms	8.58 s
Total	5197.7 s	659.8 s	676.3 s

TABLE 2 : Durées pour le modèle du jeu de la vie.

3 Objectifs de la génération matérielle

Dans notre optique, générer un circuit logique plutôt qu'un programme logiciel poursuit plusieurs buts.

Tout d'abord, l'utilisation d'un circuit logique dédié permet d'améliorer les performances de calcul par rapport à un processeur, qui exécute séquentiellement des opérations. Il faut néanmoins compter avec une durée de synthèse du code incompressible pouvant aller jusqu'à plusieurs minutes là où la compilation du code logiciel équivalent ne prends que quelques secondes. Le gain au passage en matériel ne sera donc effectif que dans certains cas, discutés dans la conclusion. Sans pour l'instant garantir le temps réel strict, d'autant plus difficile que notre simulateur utilise un paradigme à évènements discrets dans lequel un pas de temps est variable, l'accélération de la simulation réduit réduit fortement les durées de simulation, permettant de tendre vers ce but. Grâce à cela, nous envisageons d'utiliser notre code dans une approche RCP : génération automatisée du code de commande, puis utilisation du code conjointement avec des capteurs et actionneurs réels.

Ensuite, dans une perspective de génération automatisée du produit final, il est nécessaire de supporter différentes architectures. Dans cet objectif, le code généré ne serait plus une simulation, mais un code fonctionnant en temps réel. Cet objectif rejoint le précédent en l'étendant. Le support de la génération de circuits logiques permet de diversifier les cibles de développement disponibles pour l'utilisateur final.

4 Résultats obtenus sur FPGA

La méthode de génération du circuit logique est présentée dans l'article [3]. Nous indiquons ici les résultats obtenus. La cible est une ZedBoard, carte de prototypage proposant un SoC Zynq associé à différents périphériques, donc une interface Ethernet qui nous permet une communication avec l'interface ProDEVS.

On teste plusieurs scénarios : le scénario *sw* est une exécution logicielle utilisant le simulateur Java de Project DEVS sur un processeur à 3.2 GHz. Le scénario *hw1* utilise un déploiement matériel libre, c'est-à-dire que le matériel produit les résultats de la simulation sans attendre qu'ils soient transmis à l'interface, au risque de la perte de résultats. Enfin, le scénario *hw2* synchronise le matériel avec la remontée des résultats, mettant celui-ci en attente si l'espace de stockage des résultats est plein.

Deux cas de tests sont utilisés : un cas basique sans parallélisme (gen-buf-proc) dans le tableau 1, et un cas fortement parallèle, un jeu de la vie en 8×8 torique dans le tableau 2. La simulation est effectuée jusqu'à 100 000 unités de temps.

5 Conclusion

Lors du passage au matériel, on constate une accélération substantielle de la simulation, pouvant aller jusqu'à un facteur 10 000 sur une exécution « libre », et un facteur 500 avec la remontée de toutes les valeurs intermédiaires. En revanche, ce gain est fortement impacté par la durée de génération du circuit, allant jusqu'à inverser celui-ci sur le cas d'un modèle simple. Le gain est plus probant pour des modèles fortement parallèles tels le jeu de la vie, dans lequel 64 cellules fonctionnent simultanément.

Le gain est également minimisé par la remontée des résultats, qui ralentit lourdement la simulation, jusqu'à un facteur 100. Il est à noter que nous avons choisi de remonter l'intégralité des résultats, c'est-à-dire toutes les valeurs de toutes les variables du modèle à chaque évènement sur celles-ci. Un choix méticuleux dans la remontée des variables, éventuellement assorti d'un échantillonnage de celles-ci pour ne remonter qu'une fraction des valeurs peut amener à de meilleures performances.

Pour le modèle fortement parallèle, on constate également que la durée totale n'est constituée que pour fraction de la simulation à proprement parler, contrairement au logiciel pour lequel la durée totale est principalement constituée de la simulation. Pour ce cas une augmentation du temps final de simulation jouera donc en faveur du matériel.

Références

[1] V. Albert and C. Foucher. Site web de Project DEVS. `https://www.laas.fr/projects/prodevs/`.

[2] B. Berthomieu and F. Vernadat. Time petri nets analysis with tina. In *Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on*, pages 123–124. IEEE, 2006.

[3] C. Foucher and V. Albert. Mise en œuvre d'un simulateur DEVS utilisant des Réseaux de Petri temporisés sur FPGA *(Article accepté)*. Juin 2018.

[4] B. P. Zeigler, T. G. Kim, and H. Praehofer. *Theory of Modeling and Simulation*. Academic Press, Inc., Orlando, FL, USA, 2nd edition, 2000.