Outillage pour la modélisation, la vérification et la génération d'applications temporisées et embarquées

Pierre-Emmanuel Hladik, Silvano Dal Zilio, Olivier Pasquier, Sébastien Pillement, Bernard Berthomieu

Outillage pour la modélisation, la vérification et la génération d'applications temporisées et embarquées

Pierre-Emmanuel Hladik[1], Silvano Dal Zilio[1], Olivier Pasquier[2], Sébastien Pillement[2] et Bernard Berthomieu[1]

[1]LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France
[2]Lunam Université, Université de Nantes, UMR CNRS 6164, Institut d'Électronique et de Télécommunications de Rennes (IETR), Polytech Nantes, France
{bernard,dalzilio,pehladik}@laas.fr, {olivier.pasquier, Sebastien.Pillement}@univ-nantes.fr

Résumé

Cet article présente un travail en cours pour mettre en place une chaîne d'outils dédiée à la conception, la vérification et l'exécution de systèmes embarqués temps réel. Ce travail se base sur la méthode MCSE et les modèles qu'elle préconise pour la description d'applications. Une traduction du modèle dans le langage formel Fiacre est appliquée pour ensuite vérifier le système à l'aide du model-checker Tina. Afin de faciliter cette analyse et la génération d'un exécutif, la notion de *Logical Execution Time* est utilisée pour décrire le comportement temporel. Nous présentons ces différentes méthodes et outils avant d'exposer l'état d'avancement des différents composants de la chaîne.

1 Introduction

Le projet présenté dans cet article est en cours de réalisation et vise la mise en place d'une chaîne d'outils pour la modélisation, la vérification et la génération d'applications embarquées temps réel. Une des motivations principales de ce travail est d'intégrer une approche formelle basée sur le model-checking dans une démarche de conception orientée modèles. Ce travail porte sur la branche descendante du cycle de développement classique, dit en V, c'est-à-dire de la spécification des besoins à l'implantation. Cette réalisation devrait déboucher sur un outil facile à prendre en main et pouvant être employé, dans un premier temps, pour des projets d'enseignement ou comme démonstrateur.

L'approche développée dans ce projet a pour cadre la méthode MCSE, proposée par Jean-Paul Calvez dans les années 1990 [4]. Elle offre l'avantage, par rapport à d'autres approches, d'être simple, de couvrir complètement le cycle de développement et d'avoir un langage de modélisation avec une syntaxe restreinte, mais expressive. Une partie de la chaîne logicielle repose sur la boîte à outils Tina [2], qui est utilisée comme outil de vérification formelle, mais aussi comme format intermédiaire pour l'exécution.

L'étude présentée dans cet article se focalise uniquement sur la génération d'un exécutable logiciel à partir d'un modèle décrit dans le langage MCSE. Ce travail est une première étape pour aborder des problématiques d'aide à la conception conjointe matérielle-logicielle de systèmes embarqués, c.-à-d. des méthodes pour aider un concepteur à définir l'architecture d'un système en répartissant les composants entre le matériel et le logiciel et tout en garantissant les exigences fonctionnelles et non-fonctionnelles.

Cet article est organisé de la manière suivante : la section 2 introduit la méthodologie MCSE ; la section 3 présente les éléments qui composent la chaîne d'outils ; la section 4 s'attarde sur les hypothèses sous-jacentes à la machine d'exécution ; la section 5 fait le point sur les avancées et l'état du développement des éléments de la chaîne ; et pour finir la section 6 conclut le travail et ouvre sur les suites envisagées.

2 La méthodologie MCSE

Le processus de développement d'un systèmes embarqués peut être décrit de manière classique par un cycle en V. La conception y est considérée comme un raffinement des composants du système dont le résultat sert de référence pour le programmeur (ou l'outil) en charge d'écrire le code spécifique à la plate-forme d'exécution. La branche montante du cycle est consacrée à l'intégration des composants du systèmes. À cela s'ajoute des étapes de vérification et de validation par des tests lors des phases en aval et par des méthodes ad-hoc pendant la conception. Ces dernières activités sont particulièrement importantes afin de corriger au plus tôt les éventuelles erreurs de conception.

Pour mener à bien ces étapes de développement, il est nécessaire de disposer d'un cadre méthodologique. De nombreuses propositions existent dans le domaine des systèmes embarqués depuis les années 1970 telles que SADT [10] ou plus récemment ARCADIA [9]. Au tournant des années 1990, Jean-Paul Calvez a proposé dans [4] une méthode baptisée Méthodologie de Conception des Systèmes Embarqués (MCSE). C'est une méthode de conception descendante appliquée aux systèmes embarqués qui couvre les phases de spécification, de conception, d'implantation et de validation. Cette méthode est supportée par CoFluent Studio [1] racheté en 2011 par Intel, pour la modélisation et la simulation de systèmes électroniques embarqués.

Conjointement à la méthode MCSE, un langage spécifique de modélisation à base de composants est défini et permet de couvrir les dimensions parallèles et séquentielles d'une application. La dimension parallèle est liée à la structure fonctionnelle basée sur quatre types d'éléments : la fonction et les trois relations possibles que sont l'événement, la variable partagée et la file de messages. La dimension séquentielle est liée au comportement de chaque fonction qui est basé sur des éléments tels que l'opération, la boucle, le test, l'entrée et la sortie. À ces éléments peut être associée une notion de temps.

La sémantique du langage MCSE n'est pas formellement définie et la méthode ne préconise pas des techniques spécifiques de vérification ou de validation. Par contre, la méthode est particulièrement intéressante de part son caractère efficace et dédié à la conception des systèmes embarqués. Ainsi, sa syntaxe réduite permet de raisonnablement proposer un outils couvrant la totalité du modèle et d'en fixer la sémantique formellement.

3 Chaîne d'outils

Le travail présenté ici n'aborde que la partie vérification et génération de code pour des applications logicielles embarquées à partir d'un modèle fonctionnel décrit en MCSE. La vérification ne porte que sur des propriétés temporelles et comportementales du systèmes. Les aspects numériques tels que le domaine des valeurs ou la simulation des algorithmes sont traités par d'autres outils tels que ceux disponibles dans CoFluent.

1. `http://www.intel.com/content/www/us/en/cofluent/intel-cofluent-studio.html` [consulté 02/2016].

Afin de disposer d'un outillage adapté, nous nous appuyons sur la boîte à outils Tina [2]. Celle-ci est dédiée à la vérification formelle de systèmes décrits en réseaux de Petri temporels et offre des méthodes pour construire les graphes de comportements ainsi que des model-checkers pour les logiques temporelles LTL et CTL. Tina peut être couplé avec Fiacre [1], langage formel de plus haut niveau que les réseaux de Petri qui permet de modéliser des systèmes d'un point de vue comportemental et temporel. Plusieurs compilateurs sont proposés pour traduire un modèle Fiacre vers des outils d'analyse. Dans le cas de Tina, un modèle Fiacre est compilé par l'outil `frac` sous la forme d'un *Time Transition Systems* (TTS).

Le travail présenté dans cet article consiste à organiser, définir et produire les outils nécessaires pour réaliser la chaîne décrite par la figure 1. Cette chaîne s'inscrit dans la phase de conception d'un cycle de développement de MCSE et comprend les étapes suivantes : (i) le résultat de la conception du système est modélisé selon le modèle fonctionnel MCSE, (ii) le modèle fonctionnel est traduit en Fiacre, (iii) le modèle est ensuite compilé à l'aide de l'outil `frac` sous forme d'un TTS qui servira d'entrée à Tina ; (iv) le modèle est utilisé par Tina afin de vérifier les propriétés temporelles et comportementales du système ; (v) un générateur, baptisé `hippo`, produit un exécutable depuis le TTS garantissant la même sémantique que le modèle vérifié.

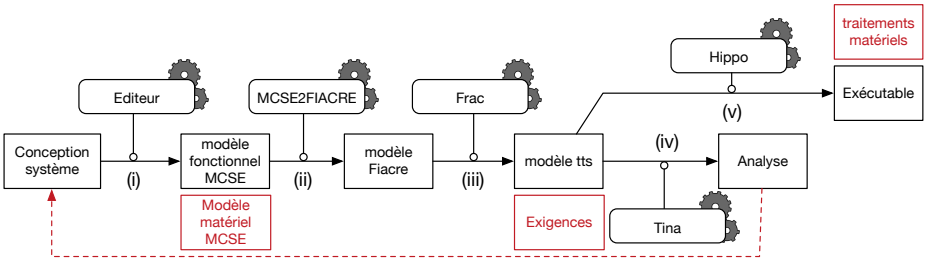

FIGURE 1 – Chaîne d'outils pour la conception, la vérification et l'exécution d'un système embarqué avec des contraintes temporelles. En rouge, les éléments non traités dans cet article.

Cette chaîne est construite suivant des principes classiques pour vérifier le comportement de systèmes temporels. L'intérêt de passer par Fiacre est double [1] : d'une part simplifier la traduction du modèle fonctionnel MCSE en utilisant les notions de processus, de synchronisation et de composants offertes par Fiacre (que l'on ne retrouve pas dans les formalismes bas-niveau comme les réseaux de Petri ou les automates), et d'autre part de permettre l'utilisation d'outils de vérification autres que Tina sans avoir besoin de redéfinir la traduction.

Une fois cette chaîne réalisée, ce travail sera complété par d'autres outils pour par exemple (voir éléments en rouge sur la figure 1) : remonter les résultats de l'analyse au niveau du modèle MCSE, exprimer et traduire automatiquement les exigences en propriétés vérifiables, prendre en considération les composantes matérielles du système dans l'analyse et la génération.

Actuellement, les outils de vérification sont disponibles et matures, c'est-à-dire que la partie (iii)-(iv) de la chaîne existe et est outillée [2]. Les étapes (i) et (ii) suivent une démarche basée sur l'ingénierie des modèles avec la définition d'un méta-modèle du langage MCSE afin de produire un éditeur sous la forme d'un plugin Eclipse à l'aide du projet Sirius [2]. Ce méta-modèle est aussi employé lors de l'étape (ii) afin d'écrire une transformation entre les modèles MCSE et Fiacre. La génération de code de l'étape (v) est détaillée dans la section suivante.

2. Les outils liés à Tina sont disponibles sur `http://projects.laas.fr/tina/` et ceux permettant de compiler du Fiacre sur `http://projects.laas.fr/fiacre/`.

2. Sirius est un projet soutenu par la fondation Eclipse et dédié à la conception d'interfaces graphiques. `https://projects.eclipse.org/projects/modeling.sirius` [consulté en février 2016]

4 Hypothèses d'exécution

La génération d'un exécutable depuis un modèle est un problème largement traité dans la littérature du génie logiciel. Cependant, au delà des méthodes issues du développement logiciel, il existe des problématiques propres aux systèmes embarqués, en particulier pour rester fidèles aux contraintes temporelles exprimées dans le modèle.

Une de ces problématiques est la différence sémantique pouvant exister entre le modèle produit par le concepteur, le modèle vérifié et l'exécutable. Les travaux de Mathias Brun [3] et de Cédrick Lelionnais [7] exposent clairement les différentes approches possibles. Dans notre cas, nous générons un code à partir du modèle le plus proche de celui vérifié, à savoir le TTS. Le support d'exécution n'est alors qu'une machine virtuelle pouvant exécuter le format TTS et garantissant ainsi le respect de la sémantique. Cette approche est similaire à ce qu'offre, par exemple, BIP [11] pour des systèmes non temporisés.

Un second point spécifique à l'exécution d'un modèle temporisé est la prise en charge du temps, aussi bien au niveau des modèles vérifiés que de l'exécutable. Classiquement, deux approches sont distinguées en fonction de la manière dont l'écoulement du temps est considéré et du moment de la prise en considération des événements, les démarches synchrone et asynchrone [12].

Il existe une troisième approche, dite *asynchrone conduite par le temps* (*Time-Triggered*) et qui sépare les aspects logiques (fonctionnalité et chronométrage) des aspects physiques (placement et ordonnancement). La production des événements et leur prise en compte par le système se fait à des instants connus sous l'hypothèse synchrone (temps logiquement nul), mais à la différence des approches synchrones, les traitements ne sont pas supposés se réaliser entre chaque instant. Christoph Kirsch et Raja Sengupta dans [6] décrivent ce principe à l'aide du *Logical Execution Time* (LET), qui est la durée séparant l'instant où un traitement peut commencer à s'exécuter et la date où il doit produire son résultat. Ainsi la date de production des données est parfaitement connue et ne dépend pas de la durée effective du traitement. Il est par contre nécessaire lors de la génération du système exécutable de s'assurer que l'exécution d'un traitement respecte bien son LET. Le *framework* pour exécutifs temps réel OASIS [8] est un exemple mettant en œuvre cette démarche tout comme Giotto [5] ou Ptide [13].

Cette approche est intéressante pour deux raisons : (i) l'abstraction du temps physique facilite la vérification du comportement d'un système, en particulier en ignorant la préemption ; (ii) la notion de LET offre plus de souplesse que l'approche purement synchrone. Par contre, l'ordonnancement du système doit d'être vérifié lors de la génération afin de garantir que les traitements respectent bien leur LET.

La sémantique d'exécution d'un modèle fonctionnel MCSE est définie comme étant séquentielle au sein de chaque fonction et concurrentes entre les fonctions. Au niveau du temps, les synchronisations (attente, message) l'écriture de données partagées et les éléments de contrôle (boucle, condition, etc.) sont supposés à temps nuls (équivalent de l'hypothèse synchrone) et les *opérations* (équivalent à un traitement) sont associées à une durée.

L'hypothèse LET est alors particulièrement adaptée pour gérer et vérifier le modèle MCSE, mais nécessite une légère adaptation de son comportement. Pour cela, il est nécessaire d'ajouter une échéance relative à chaque opération (c.-à-d. la date à laquelle l'opération doit être terminée en fonction de sa date de démarrage) en plus de sa durée d'exécution. La sémantique d'exécution de MCSE est ainsi alors modifiée en introduisant une lecture et une écriture (c'est ce dernier point qui est nouveau) des *opérations* à des instants parfaitement définis et donc vérifiables.

5 Outils et travaux en cours

Plusieurs outils de la chaîne sont encore au stade de développement ou de preuve de concept. Nous donnons dans cette section un aperçu rapide des solutions envisagées et de l'avancée des travaux.

Méta-modèle MCSE. Le premier élément de la chaine a pour objectif de saisir le modèle de l'application en se basant sur une approche multi-fonctions (parallélisme) et comportemental (séquentiel) avec notion de temps. Il est en cours de développement à l'aide de l'environnement Sirius. Il s'agit de redéfinir un méta-modèle du modèle fonctionnel préconisé par la méthodologie MCSE. L'objectif de ce travail est d'avoir un méta-modèle ouvert mais en aucun cas de concurrencer les possibilités de CoFluent Studio qui en plus permet de faire de la simulation fonctionnelle et de l'extraction de performances en générant un code basé sur SystemC.

Transformation du modèle MCSE vers Fiacre. Cette transformation couvre tous les aspects comportementaux du modèle fonctionnel de MCSE ce qui permet ainsi d'en fixer formellement la sémantique. Un méta-modèle de Fiacre, en plus de celui de MCSE, a été produit et les règles de transformations ont été écrites à l'aide du langage ATL. Cependant, le prototype existant doit être revu pour prendre en considération les hypothèses sur l'exécution ainsi que les dernières modifications du méta-modèle MCSE. Par la suite, il serait aussi intéressant d'assurer, lors de cette transformation, une traçabilité des propriétés temporelles et comportementales afin de pouvoir faire remonter les éventuels échecs au niveau du modèle manipulé par le concepteur (cette traçabilité existe déjà entre le TTS et Fiacre).

Moteur d'exécution. Pour une question de place, nous n'entrerons pas dans les détails techniques de la machine assurant l'exécution du TTS. La version prototype est basée sur un Linux patché temps réel (Xenomai) pour gérer l'ordonnancement des traitements et pour avoir accès à des mécanismes qui garantissent la cohérence des états aux instants logiques. La machine d'exécution est alors une simple couche intermédiaire entre le système d'exploitation et les traitements applicatifs (idem aux couches OASIS et Giotto). Son rôle est de contrôler l'exécution des threads qui encapsulent les traitements et de faire avancer l'état du TTS en garantissant sa cohérence. Une version statique au niveau mémoire est implantée et sera portée sur un microcontrôleur sous OSEK afin d'en évaluer les surcoûts temporels et mémoriels.

La prise en considération d'événements asynchrones extérieurs au système a aussi été implantée, mais nécessite une étude plus approfondie pour valider l'architecture retenue. Cela rejoint les difficultés pour modéliser et vérifier un système dans son environnement.

Analyse du Logical Execution Time. La notion de LET attachée aux traitements impose de vérifier le comportement temporel du système du point de vue de l'ordonnançabilité. Actuellement cet aspect n'a pas été traité, mais plusieurs pistes sont envisagées pour attaquer ce point. Le comportement temporel des traitements étant bien spécifié dans le modèle, il est possible d'utiliser les approches dites analytiques pour évaluer l'ordonnançabilité. Un autre moyen serait de conduire l'analyse par model-checking en ajoutant des motifs propres au comportement de l'ordonnanceur dans le modèle Fiacre ou dans le TTS ; mais cela est au risque d'une explosion du nombre d'états du système.

6 Conclusion

Cet article présente un travail en cours sur la production d'une chaîne outillée pour la modélisation, la vérification et la génération d'applications embarquées temps réel. La méthodologie MCSE sert de cadre à cette étude et guide la démarche mise en place. Nous avons montré que certains outils sont matures, comme ceux pour la vérification, alors que d'autres sont encore au stade de prototype (éditeur, transformation de modèle et machine d'exécution) voire de projet (validation de l'ordonnancement).

Ce travail permettra à terme de disposer d'une chaîne complète pour générer un exécutable temps réel. Cet outillage aura l'avantage d'être inscrit dans un processus de conception et d'assurer la cohérence entre les outils. À plus long terme, cette chaîne servira de base pour aborder d'autres problèmes telle que la gestion des propriétés (traçabilité et sémantique) ou la modélisation de l'environnement.

Références

[1] B. Berthomieu, J.-P. Bodeveix, P. Farail, M. Filali, H. Garavel, P. Gaufillet, F. Lang et F. Vernadat : Fiacre : an intermediate language for model verification in the topcased environment. *In ERTS 2008*, 2008.

[2] B. Berthomieu, P.-O. Ribet et F. Vernadat : The tool TINA – construction of abstract state spaces for Petri nets and time petri nets. *Int. Journal of Production Research*, 2004.

[3] M. Brun : *Contribution à la considération explicite des plates-formes d'exécution logicielles lors d'un processus de déploiement d'application.* Thèse de doctorat, Univ. de Nantes, 2010.

[4] J.-P. Calvez : *Spécification et Conception des Systèmes : une Méthodologie.* Masson, 1990.

[5] T. Henzinger, B. Horowitz et C. Kirsch : Giotto : a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1), 2003.

[6] C. Kirsch et R. Sengupta : *Handbook of Real-Time and Embedded Systems*, chapitre The Evolution of Real-Time Programming. Chapman and Hall/CRC, 2007.

[7] C. Lelionnais, J. Delatour, M. Brun, O. Henri Roux et C. Seidner : Formal synthesis of real-time system models in a MDE approach. *International Journal on Advances in Systems and Measurements*, 7, 2014.

[8] S. Louise, M. Lemerre, C. Aussagues et V. David : The oasis kernel : A framework for high dependability real-time systems. *In IEEE HASE*, 2011.

[9] P. Roques : Mbse with the arcadia method and the capella tool. *In 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[10] D. T. Ross : *Applications and extensions of SADT : structured analysis and design technique.* SoftTech, 1984.

[11] J. Sifakis : A framework for component-based construction. *In Third IEEE International Conference on Software Engineering and Formal Methods*, 2005.

[12] F. Simonot-Lion et Y. Trinquet : *Encyclopédie de l'informatique et des systèmes d'information*, chapitre Exemples de systèmes temps réel et choix d'implémentation. Vuibert, 2005.

[13] J. Zou : *From Ptides to PtidyOS, Designing Distributed Real-Time Embedded Systems.* Thèse de doctorat, UC Berkeley, 2011.