Compte-rendu d'habilitation : Modélisation à haut niveau d'abstraction pour les systèmes embarqués

Matthieu Moy

Compte rendu d'habilitation : Modélisation à haut niveau d'abstraction pour les systèmes embarqués

Matthieu Moy

2 mai 2014

Résumé

Habilitation à diriger des recherches [Moy14], présentée publiquement le 13 mars 2014, devant un Jury composé de Gérard Berry, Rolf Drechsler, Marco Roveri, Samarjit Chakraborty, Benoît Dupont de Dinechin et Frédéric Pétrot

Ce document est un compte-rendu de mon habilitation à diriger des recherches (HDR), qui elle-même résume les travaux que j'ai réalisé sur la modélisation de systèmes embarqués ces dix dernières années au laboratoire Verimag. Une part importante de ces travaux concerne la modélisation transactionnelle des systèmes sur puces et a été réalisée dans le cadre d'une collaboration suivie entre Verimag et STMicroelectronics. Dans un soucis de brièveté, mes publications ne sont pas citées ici ; le lecteur pourra se référer à la bibliographie du manuscrit [Moy14] ou à ma page web [1] pour une liste complète.

1 Introduction : les systèmes embarqués

Les systèmes embarqués peuvent être définis comme des systèmes informatiques autres que des ordinateurs. Alors qu'il pourrait sembler naturel d'associer l'informatique aux ordinateurs, environ 98% des processeurs fabriqués aujourd'hui sont des processeurs embarqués. La plupart des systèmes embarqués diffèrent des systèmes traditionnels en terme de ressources disponibles et par le fort couplage entre matériel et logiciel.

Le besoin d'optimisations de performances (à la fois pour fournir plus de capacité de calcul et pour consommer moins d'énergie) a mené à des architectures où la fonctionnalité est répartie sur plusieurs composants. Dans un système embarqué typique, les parties non-critiques en performance sont implémentées en logiciel, alors que les calculs plus intensifs utilisent des composants matériels dédiés. Une autre tendance est celle des processeurs pluricœurs (*many-core*), comme STHorm [MBF+12] de STMicroelectronics ou MPPA [Kal12] de Kalray. Ces plates-formes cassent le modèle traditionnel de parallélisme symétrique (SMP). Les systèmes embarqués sont donc en général des systèmes très hétérogènes, contenant souvent plusieurs logiciels embarqués compilés pour des jeux d'instructions différents.

Alors que le matériel dédié est indispensable pour de bonnes performances, il y a également de bonnes raisons d'implémenter les parties les plus complexes du système en logiciel. Les modèles de programmation utilisés en programmation logicielle sont beaucoup plus riches et permettent de résoudre des problèmes difficiles algorithmiquement en moins de lignes de code. Le logiciel apporte aussi de la flexibilité dans le flot de conception, permettant des mises à jour à n'importe quel stade du développement, contrairement au matériel qui exige que les bugs soient corrigés avant d'obtenir le premier prototype physique.

En conséquence, les systèmes sont composés d'un grand nombre de composants matériels et logiciels, qui dépendent les uns des autres. Il n'est pas suffisant de valider les blocs individuellement pour assurer le fonctionnement correct du système.

L'interaction avec son environnement pose des contraintes sur la conception d'un système. La plupart des systèmes embarqués sont temps réels, soit mou (l'interaction avec l'environnement *devrait* être faite au bon moment) soit dur (l'interaction *doit* se faire au bon moment).

D'autres propriétés extra-fonctionnelles comme la surface de silicium, la consommation électrique et la température peuvent également avoir une grande importance. Pour les systèmes sans fil, minimiser la consommation est évidemment important car c'est d'elle que dépend l'autonomie sur batterie. Mais même pour un système branché en permanence, la une consommation trop élevée n'est pas acceptable car elle implique des technologies plus coûteuses pour la fabrication de la puce et une durée de vie plus courte, sans compter la problématique environnementale.

Les économies d'énergies peuvent être faites à plusieurs niveaux, y compris les plus proches du silicium (meilleurs algorithmes de synthèse et technologies de fabrication). Mais dans les systèmes modernes, une part importante des économies doit être faite au niveau système : une politique de gestion d'énergie doit

1. `http://www-verimag.imag.fr/~moy/?-Publications-`

être capable d'éteindre les composants quand ils ne sont pas utilisés, et de réduire la tension des composants peu utilisés au minimum nécessaire (ce qui implique de réduire également leur fréquence).

Dans les ordinateurs classiques, ces contraintes peuvent être gérées avec une politique du meilleur effort, c'est à dire faire des ordinateurs ≪ les plus rapides possibles ≫ et consommant ≪ le moins d'énergie possible ≫. Ce n'est en général pas possible avec un système embarqué : un système sur-dimensionné coûtera trop cher, et un système sous-dimensionné peut être inutilisable (par exemple, une télévision numérique qui n'est pas capable de lire le flux vidéo en temps réel n'a que peu d'intérêt). Il est donc important non seulement d'être capable d'optimiser ces propriétés, mais aussi de les évaluer et de valider le système en en tenant compte le plus tôt possible.

Une pratique courante est d'écrire des *modèles*, c'est à dire des version simplifiées du vrai système, avant que celui-ci soit disponible. On peut distinguer plusieurs types de modèles, selon la manière dont ils sont utilisés. En développement dirigé par les modèles, les modèles sont utilisés comme point de départ d'un flot d'implémentation. Mais tous les modèles ne sont pas forcément utilisés pour dériver une implémentation. Les modèles d'environnement physique sont utilisés pour le test ou la vérification, mais ne sont bien sûr pas embarqués dans le système réel. Une tendance relativement récente est le *prototypage virtuel*, qui utilise des modèles exécutables pour valider des choix d'architecture, ou pour développer le logiciel embarqué, mais le prototype lui-même n'est pas utilisé pour l'implémentation du matériel.

Les travaux présentés dans mon manuscrit d'habilitation à diriger des recherches se concentrent sur les modèles qui ne sont pas utilisés directement dans l'implémentation. Les sections suivantes présentent plusieurs catégories d'approches, et proposent une classification de mes contributions dans ces catégories.

1.1 Contributions à plusieurs approches basées sur les modèles

1.1.1 Prototypage virtuel et approches basées sur la simulation

Le *prototypage virtuel* consiste en l'écriture de modèles tôt dans le flot de développement, pour permettre des activités qui n'auraient été possibles que sur le système final (tests d'intégration, développement du logiciel...) les plus tôt possible, en utilisant la simulation. Les prototypes virtuels peuvent être vus comme une spécification exécutable du système réel.

Le niveau de transfert de registre (Register Transfer Level, ou RTL) est le point d'entrée de la synthèse matérielle. Les programmes RTL doivent décrire toute la micro-architecture de la plate-forme à un niveau de détails très fin, et par conséquents ont une vitesse de simulation très lente. Pour un système de grande taille, les simulations RTL sont possibles sur les composants individuels, mais sont beaucoup trop lente pour une utilisation au quotidien au niveau système (par exemple, démarrer un système d'exploitation peut prendre une dizaine d'heures à ce niveau [HYH+11]).

Le niveau transactionnel (Transaction-Level Modeling, ou TLM) [Ghe05, Ope08] a été introduit en réponse à ce problème. On trouve beaucoup de définitions différentes de TLM, et même la définition du standard IEEE [Ope11] est déclinée en plusieurs variantes. Une première définition peut être donnée par : ≪ un modèle qui décrit tout ce qui est nécessaire pour l'exécution du logiciel embarqué, et seulement cela ≫.

Le standard pour la modélisation TLM dans l'industrie est SystemC, qui ajoute à C++ les éléments manquants pour la simulation TLM (le temps simulé, la concurrence, la notion de module, ...). On peut noter que même si les programmes SystemC décrivent des systèmes parallèles, leur exécution est purement séquentielle.

Mon habilitation décrit les contributions suivantes dans le domaine du prototypage virtuel :

- Nous avons développé des techniques de compilations dédiées pour SystemC, qui n'est pas un langage de programmation à part entière, mais une bibliothèque pour C++. Les techniques de compilations usuelles ne s'appliquent pas. L'équivalent d'une partie frontale de compilateur (front-end) doit être capable d'extraire non-seulement les informations sur le comportement du programme, mais aussi l'architecture de la plate-forme qui est construite pendant l'exécution du simulateur, avant le lancement de la simulation à proprement parler.
  Nous présentons des techniques implémentées dans l'outil *PinaVM*, qui utilise LLVM pour exécuter une partie du programme et lier les informations dynamiques obtenues à l'exécution avec les informations statiques extraites par le compilateur C++. Les travaux sur PinaVM sont un prolongement direct de ceux réalisés plus tôt (pendant ma thèse) sur l'outil Pinapa.
- En se basant sur PinaVM, nous avons développé un compilateur optimisant pour SystemC. Ce compilateur utilise les informations récoltées par PinaVM pour réaliser des optimisations comme l'inlining à travers le réseau d'interconnexion SystemC qui ont besoin d'une visibilité sur l'architecture et une d'une connaissance de la sémantique de SystemC.
- Nous avons identifié des limitations du modèle traditionnel de temps et de concurrence en SystemC/TLM en terme de fidélité de simulation. En particulier, nous avons montré que certains bugs logiciels liés au modèle mémoire ne peuvent pas être découverts avec les techniques existantes.
- Nous avons développé un simulateur TLM appelé *jTLM* (pour ≪ Java TLM ≫). Ce simulateur était à l'origine une expérience pour étudier le niveau d'abstraction TLM en dehors des contraintes de SystemC, et a été utilisé ensuite pour développer de nouvelles fonctionnalités comme les *tâches avec*

durée. Les tâches avec durée permettent une parallélisation efficace et donnent une solution partielle au problème de fidélité présentée ci-dessus.
- La notion de tâche avec durée développée à l'origine dans jTLM a ensuite été adaptée au contexte de SystemC, avec les mêmes avantages mais sans le besoin d'un ordonnanceur spécialisé. Le résultat est la bibliothèque `sc-during`, écrite au dessus de l'API SystemC et qui peut s'exécuter sur n'importe quelle implémentation de SystemC conforme à la norme.
- J'ai contribué au développement et à la validation de la ≪ Smart FIFO ≫, qui modélise une file et minimise les changements de contextes en utilisant le découplage temporel. Contrairement aux approches similaires, la ≪ Smart FIFO ≫ garantie que le comportement obtenu avec découplage temporel est équivalent à celui produit sans cette optimisation.
- Nous avons développé des outils pour l'estimation de propriétés extra-fonctionnelles comme la *consommation électrique* et la *température* à différents niveaux d'abstraction. Ces outils permettent de co-simuler un modèle fonctionnel avec un solveur extra-fonctionnel qui modélise la consommation électrique et la dissipation de chaleur. L'interface de co-simulation est très générique et son application aux systèmes à faible couplage temporel a nécessité un travail de modélisation pour éviter de faire apparaître des artefacts de simulation comme des pics de température qui n'apparaissent pas sur le vrai système, mais auraient été exhibés par une modélisation trop naïve.

1.1.2 Vérification formelle de propriétés fonctionnelles

Une limitation évidente des approches basées sur la simulation est que la validation est faite sur un nombre fini d'exécutions. Il peut être nécessaire d'obtenir des garanties plus fortes sur le système, et donc de travailler avec des modèles plus formels.

Nous avons exploré certains aspects de l'*interprétation abstraite*, qui permet la vérification automatique et correcte de systèmes potentiellement non-bornés (par exemple, contenant des valeurs numériques). L'interprétation abstraite associe à chaque point du programme un ensemble de valuations possibles pour ces variables, abstrait en un élément d'un domaine abstrait (par exemple, les intervalles, les polyèdres, ...). L'analyse abstraite est limitée par plusieurs sources d'imprécision (le domaine abstrait, les enveloppes convexes, et l'opérateur d'élargissement qui est en général nécessaire pour assurer la convergence). Changer la manière de parcourir le graphe de flot de contrôle du programme peut changer la précision de l'analyse. L'ordre de parcours des points de contrôle peut être obtenu en interrogeant un solveur *SMT* (Satisfaisabilité Modulo Théorie).

J'ai fait les contributions suivantes au domaine de la vérification formelle :
- Nous avons développé plusieurs outils de vérification formelle pour programmes séquentiels. Certains sont de simples reproduction de l'état de l'art de techniques traditionnelles (interprétation abstraite, model-checking). L'outil PAGAI a démarré comme une implémentation des techniques d'interprétation abstraite de l'état de l'art, en utilisant l'interprétation abstraite conjointement avec le SMT-solving. Il a ensuite été étendu avec de nouvelles techniques qui combinent et améliorent les techniques existantes. PAGAI a aussi permis de comparer les techniques existantes et les nouvelles sur des programmes réels.
- Nous avons développé des outils de vérifications pour programmes SystemC. Empruntant des idées de l'outil LusSy (développé pendant ma thèse), nous avons proposé de nouvelles techniques pour mieux exploiter la sémantique de co-routine de SystemC et les propriétés de la forme SSA (static single assignment, ou affectation unique statique) utilisée comme forme intermédiaire dans les compilateurs modernes.

1.1.3 Formal Models for Non-Functional Properties

Certaines propriétés extra-fonctionnelles peuvent également demander des garanties fortes, et donc bénéficier des méthodes formelles. C'est le cas pour le pire temps d'exécution (Worst Case Execution Time, WCET) d'une tâche, ou le pire temps de traversée (WCTT) d'un évènement dans un système temps réel.

Le point de départ des travaux réalisés sur les modèles formels d'analyse de performance est l'analyse de performance modulaire (Modular Performance Analysis, MPA) par calcul temps réel (Real-Time Calculus, RTC) [TCN00], basé sur la théorie du calcul réseau (Network Calculus) [LBT01]. L'idée de MPA est de modéliser un système avec un ensemble de modules qui communiquent via des flots d'évènements (les données échangées sont abstraites). L'analyse est ensuite faite en manipulant des abstractions de flots d'évènements appelées *courbes d'arrivées* (arrival curves) qui spécifient des bornes sur nombre maximum d'évènements qui peuvent arriver pendant une période de temps donnée. Ces courbes sont en général manipulées par paires : la courbe basse (α^l) spécifie une borne inférieure et la courbe haute (α^u) une borne supérieure. Par exemple, une paire de courbes d'arrivée peut exprimer la propriété ≪ entre 1 et 5 évènements peuvent arriver par seconde, mais pas plus de 10 évènements peuvent arriver sur n'importe quelle intervalle de 5 secondes ≫.

En MPA, l'analyse est faite module par module : les courbes d'arrivées de sortie de chaque module sont calculées en fonction des courbes d'entrée et du modèle de composant. En cas de dépendances circulaires, un point fixe peut être calculé impérativement [JPTY08].

A l'origine, les calculs de MPA étaient faits de manière purement analytique, en utilisant le calcul temps réel (formules d'algèbre max-plus sur les courbes d'arrivées). Ces calculs donnent des garanties fortes avec des algorithmes rapides pour des composants suffisamment simples, mais ne peuvent gérer la notion d'état dans un module. Notre premier objectif avec MPA était de permettre une évaluation de performance plus générale que les performances temporelles, et en particulier permettre de prendre en compte la gestion de l'énergie. Le premier pas, avant d'évaluer la consommation elle-même, est de permettre l'analyse temporelle de systèmes avec une politique de gestion d'énergie. Ces systèmes peuvent placer des composants dans un état d'économie d'énergie dans certaines condition, et ce type de comportement basé sur les états n'est pas modélisable sans faire d'abstraction grossière en calcul temps réel.

Une alternative au calcul temps réel est de faire une analyse locale à chaque module en utilisant une technique plus expressive, typiquement utilisant un formalisme à base d'automates. Ceci peut être fait dans le cadre de MPA tant que l'analyse peut prendre des courbes d'arrivée en entrée, et en produire en sortie. Cette catégorie d'approches a été expérimentée avec un formalisme d'automates spécifique appelé les ≪ event count automata ≫ [PCTT07], et adapté aux automates traditionnels [Upp07, LPT09, LPT10]. Dans ces approches, des adaptateurs depuis le calcul temps réel vers l'autre formalisme doivent être écrits. En entrée, les courbes d'arrivées peuvent être compilées en automates qui produisent des flots d'évènements concrets spécifiés par les courbes d'arrivées. En sortie, un observateur peut calculer des bornes sur le nombres d'évènements produits sur un intervalle de temps donné, et reconstruire une paire de courbes d'arrivées. Avec cette approche, l'analyse locale à un composant peut être complexe, et utiliser des techniques limitées en passage à l'échelle comme le model-checking, mais il est important de noter que l'analyse se fait module par module. La complexité croît donc linéairement en fonction du nombre de module dans le système, même si elle peut être exponentielle en fonction de la taille des modules.

Les contributions faites au cadre de l'analyse de performance modulaire (MPA) suivent deux directions :

- Nous avons proposé deux nouvelles connections du calcul temps réel avec des formalismes et outils basés sur des états. Nous avons amélioré les approches existantes sur les automates temporisées utilisant l'outil de model-checking Uppaal [LPY97], en ajoutant une abstraction basée sur la granularité pour de meilleures performances. Nous avons également développé l'outil ac2lus qui permet d'utiliser le langage Lustre et les outils associés pour faire l'analyse d'un ou plusieurs module. Cet outil permet d'utiliser l'interprétation abstraite et les outils de model-checking basés sur SMT (Satisfaisabilité Modulo Théorie), qui ont un comportement différent des outils comme Uppaal en terme de passage à l'échelle, et permettent donc de résoudre d'autres classes de problèmes.
- Nous avons identifié et défini formellement le problème de la *causalité* dans le calcul temps réel. Ce problème se produit quand les courbes hautes et basses impliquent des contraintes contradictoires (par exemple, qu'au moins 4 évènements mais au plus 3 soient émis pendant un certain intervalle). Nos travaux permettent d'expliquer pourquoi les travaux précédents sur MPA n'ont pas rencontré ce problème, mais nous avons également montré qu'il devait être résolu pour pouvoir interfacer correctement MPA et les outils de vérification formelle comme le model-checking. Nous proposons une transformation appelée la *fermeture causale* (causality closure) qui permet de débarrasser une paire de courbes de ces contraintes contradictoires. Étant donné une paire de courbes non-causale, la fermeture causale permet de calculer une autre paire de courbe qui lui est équivalente mais exprime les contraintes plus précisément et sans contradiction implicite. La fermeture causale est définie dans le cadre le plus général, et les problèmes spécifiques à certaines classes de courbes sont résolus.

2 Conclusion

En résumé, mes recherches ont porté sur la modélisation de haut niveau et la vérification de systèmes embarqués. De nouvelles techniques de vérification de propriétés fonctionnelles et extra-fonctionnelles, et de nouveaux outils pour les évaluer ont été proposés. Les modèles vont des modèles précis et exécutables pour les systèmes sur puces aux modèles analytiques pour l'analyse de performance. Une partie des travaux a porté sur la vérification de logiciel séquentiel.

Un principe directeur de mes recherches passées et futures est d'utiliser des problèmes concrets, et autant que possible en provenance de l'industrie, comme guide. Les travaux ne sont pas toutes applicables directement en pratique, mais vont toujours dans la direction d'un problème pratique et réel. Un exemple de partenariat industriel est la collaboration suivie avec STMicroelectronics depuis 2002 (début de ma thèse, CIFRE STMicroelectronics/Verimag), qui nous a fait découvrir plusieurs domaines de recherches très intéressants.

J'ai apprécié de pouvoir m'attaquer à des problèmes théoriques, mais le développement d'outils et les résultats pratiques sont également un guide précieux. La quasi-totalité de mes publications correspond à un outil ou une fonctionnalité d'outil auquel j'ai directement contribué. Le fait d'implémenter les algorithmes donne un moyen tangible d'évaluer les résultats théoriques, et permet bien souvent d'identifier leurs faiblesses, ce qui donne de nouvelles directions de recherches.

Parmi les directions futures de recherche, il reste beaucoup de pistes intéressantes sur la modélisation SystemC/TLM, et en particulier les améliorations de performances de ces modèles, avec de meilleures techniques

de compilation et de parallélisation, adaptées aux modèles à faible couplage temporel. L'étude des propriétés extra-fonctionnelles reste un challenge intéressant : nous avons beaucoup travaillé sur l'exécution de modèles incluant des informations sur la consommation et la température, mais l'expression de ces informations de manière compositionnelle et abstraite n'est pas totalement résolue aujourd'hui.

Sur le plan plus formel, nos travaux sur le calcul temps réel (RTC) ont donné les fondations théoriques et de nouveaux outils pour utiliser des outils de vérification à l'intérieur d'un système modélisé en RTC. Il serait intéressant d'appliquer ces techniques à des exemples plus gros et plus réalistes. Les travaux sur l'interprétation abstraite et l'outil PAGAI ont ouvert de nouvelles voies sur l'interprétation abstraite modulaire.

Enfin, même si je n'y ai pas contribué directement, une part des travaux de mon équipe concerne l'implémentation de systèmes critiques en utilisant les langages synchrones (Lustre en particulier). Nous démarrons de nouveaux travaux sur l'utilisation des mêmes langages pour des systèmes critiques implémentés sur architecture muliti- ou pluri-cœurs. Ceci demande une vision globale de l'architecture logicielle et matérielle, et fera certainement intervenir des techniques de modélisation et vérification variées.

Références

[Ghe05] F. Ghenassia. *Transaction-level modeling with SystemC : TLM concepts and applications for embedded systems*. Springer Verlag, 2005.

[HYH+11] Chung-Yang (Ric) Huang, Yu-Fan Yin, Chih-Jen Hsu, Thomas B. Huang, and Ting-Mao Chang. SoC HW/SW verification and validation. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, ASPDAC '11, pages 297–300, Piscataway, NJ, USA, 2011. IEEE Press.

[JPTY08] Bengt Jonsson, Simon Perathoner, Lothar Thiele, and Wang Yi. Cyclic dependencies in modular performance analysis. In *EMSOFT*, pages 179–188, New York, NY, USA, 2008. ACM.

[Kal12] Kalray. MPPA : Multi-Purpose Processor Array, 2012.

[LBT01] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus*. Lecture Notes in Computer Science. Springer Verlag, 2001.

[LPT09] Kai Lampka, Simon Perathoner, and Lothar Thiele. Analytic real-time analysis and timed automata : A hybrid method for analyzing embedded real-time systems. In *EMSOFT*. ACM, October 2009.

[LPT10] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata : a hybrid methodology for the performance analysis of embedded real-time systems. *Design Automation for Embedded Systems*, pages 1–35, June 2010.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2) :134–152, October 1997.

[MBF+12] Diego Melpignano, Luca Benini, Eric Flamand, Bruno Jego, Thierry Lepley, Germain Haugou, Fabien Clermidy, and Denis Dutoit. Platform 2012, a many-core computing accelerator for embedded socs : performance evaluation of visual analytics applications. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 1137–1142, New York, NY, USA, 2012. ACM.

[Moy14] Matthieu Moy. *High-level Models for Embedded Systems*. Habilitation à diriger des recherches (HDR), Université de Grenoble, Verimag, 2014.

[Ope08] Open SystemC Initiative (OSCI). *OSCI TLM-2.0 Language Reference Manual*, June 2008.

[Ope11] Open SystemC Initiative. *IEEE 1666 Standard : SystemC Language Reference Manual*, 2011.

[PCTT07] Linh T.X. Phan, Samarjit Chakraborty, P.S. Thiagarajan, and Lothar Thiele. Composing functional and state-based performance models for analyzing heterogeneous real-time systems. In *RTSS*, pages 343–352, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[TCN00] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *International Symposium on Circuits and Systems ISCAS 2000*, volume 4, pages 101–104, Geneva, Switzerland, march 2000.

[Upp07] Uppsala University. Cats tool, 2007.