EMI : Un Interpréteur de Modèles Embarqué pour l'Exécution et la Vérification de Modèles UML

Valentin Besnard[1], Matthias Brun[1], Philippe Dhaussy[2], Frédéric Jouault[1], et Ciprian Teodorov[2]

[1]ERIS, ESEO-TECH, Angers, France
prenom.nom@eseo.fr
[2]Lab-STICC UMR CNRS 6285, ENSTA Bretagne, Brest, France
prenom.nom@ensta-bretagne.fr

Résumé

Pour faire face à la complexité croissante des systèmes embarqués, les activités de vérification et de validation, et notamment le model-checking, sont de plus en plus sollicités. Les model-checkers permettent de vérifier des propriétés par rapport au modèle formel fourni en entrée mais deux problèmes subsistent généralement. D'une part, les outils de model-checking n'apportent pas l'assurance que les propriétés sont aussi vérifiées sur le code exécutable du système. D'autre part, le modèle formel utilisé par les model-checkers est souvent le résultat d'une transformation de modèle dont l'exactitude n'est pas prouvée. Pour y remédier, cet article présente EMI, un interpréteur de modèles UML visant l'exécution et la vérification de systèmes embarqués à l'aide d'une seule implémentation de la sémantique du langage. En connectant cet outil au model-checker OBP2, diverses activités de vérification peuvent ainsi être menées sur des modèles semi-formels en UML.

Mots clés— UML, Interprétation de modèles, Model-checking, Systèmes embarqués

1 Introduction

Les systèmes embarqués sont de plus en plus complexes à concevoir et s'exposent à de multiples défaillances logicielles (e.g., erreurs de conception, bogues, failles de sécurité). Pour garantir la sûreté de fonctionnement de ces systèmes, leurs développements requièrent des besoins croissants en vérification et validation (V&V). Grâce à l'ingénierie dirigée par les modèles, ces systèmes peuvent être modélisés sous la forme de modèles et vérifiés à un plus haut niveau d'abstraction en utilisant des techniques comme le model-checking. Les model-checkers sont des outils de vérification formelle permettant de vérifier des propriétés par rapport au modèle formel qui leur est fourni. Cependant, l'utilisation classique d'un model-checker pose généralement deux problèmes susceptibles d'impacter la vérification du modèle. (1) Une relation d'équivalence entre le modèle formel et le code exécutable doit être établie et prouvé pour assurer que ce qui est exécuté est bien ce qui a été vérifié.

(2) Le modèle formel est souvent le résultat d'une transformation de modèle complexe qui n'apporte pas la preuve que le modèle obtenu est fidèle au modèle de conception.

Afin d'apporter une réponse à ces deux problématiques, nous présentons dans cette publication l'outil EMI (Embedded Model Interpreter) [1], un interpréteur de modèles UML dédié à l'exécution et à la vérification des systèmes embarqués. EMI interprète directement le modèle UML de conception et sa sémantique opérationnelle est la seule définition de la sémantique du langage qui est utilisée. Cet interpréteur de modèles peut en effet être piloté par des outils de vérification afin de vérifier directement le modèle exécutable du système tout en réutilisant la même implémentation de la sémantique du langage que celle utilisée lors de l'exécution. Notre approche permet d'éviter les transformations de modèle difficiles à prouver et permet d'améliorer la qualité des activités de V&V. Elle offre également l'avantage d'être applicable même si le langage utilisé n'est que semi-formel comme UML. Au stade de prototype, EMI peut être utilisé pour diverses activités de vérification en se connectant au model-checker OBP2 (Observer-Based Prover 2) [3, 4] (`https://plug-obp.github.io/`).

2 Exécution de Modèles UML

L'interpréteur de modèles EMI est l'outil central de cette approche permettant d'exécuter des systèmes embarqués spécifiés sous la forme de modèles UML. Le modèle UML du système est directement chargé dans EMI et exécuté avec une seule implémentation de la sémantique d'UML. Cette implémentation correspond à la sémantique opérationnelle de notre outil qui sera utilisée à la fois pour l'exécution du modèle sur un microcontrôleur embarqué et pour la phase de vérification via des outils de V&V. De sa conception jusqu'à son exécution par EMI, différentes étapes sont nécessaires pour pouvoir interpréter un modèle.

Modélisation. La première étape consiste à modéliser en UML le système embarqué à concevoir. Notre outil supporte un sous-ensemble d'UML auquel le modèle de conception doit se conformer. Ce sous-ensemble permet de décrire la partie structurelle et la partie comportementale du modèle et peut se représenter à l'aide des diagrammes de classes, de structure composite, et d'états-transitions. Le comportement à grains fins est spécifié à l'aide d'un langage d'action permettant de définir précisément les gardes et effets des transitions des machines à états du système. Le langage d'action d'EMI est le langage C enrichi avec des macros C permettant d'accéder aux objets du modèle et à leurs attributs (e.g., état courant de la machine à états, valeurs des attributs).

Sérialisation. Le modèle de conception est ensuite sérialisé en langage C, le langage natif de notre interpréteur de modèles. La sérialisation est une transposition des éléments du modèle en termes d'initialiseurs de structures C. Contrairement à la génération de code, cette étape ne capture pas la sémantique du langage puisqu'elle ne génère que des données et pas de fonctions (sauf pour les expressions du langage d'action définies de façon opaque dans le modèle).

Chargement du modèle. Le modèle sérialisé et le code source de l'interpréteur sont ensuite compilés à l'aide d'un compilateur C afin d'obtenir le binaire exécutable du système. Cette opération peut être vue comme le chargement du modèle UML dans EMI lors de la compilation. Afin de rendre déterministe l'exécution du modèle, la sémantique opérationnelle encodée dans EMI ne doit pas admettre de comportements indéfinis. Pour

chaque point de variation sémantique d'UML, l'interpréteur doit choisir un comportement satisfaisant aux contraintes imposées par UML. Dans EMI, le comportement choisi est soit directement encodé dans l'interpréteur, soit configurable par l'utilisateur au moment de la compilation lorsque plusieurs alternatives sont disponibles.

Exécution. A partir du binaire exécutable généré, le modèle UML peut être exécuté par EMI. L'exécution est pilotée soit par la boucle d'exécution principale de l'interpréteur soit par l'outil de V&V connecté à EMI. Cet interpréteur de modèles peut être exécuté sur un ordinateur équipé d'un système d'exploitation Linux ou en bare-metal (i.e., sans OS) par exemple sur une carte embarquée STM32 discovery.

3 Vérification de Modèles UML

Pour vérifier un modèle UML, des outils de V&V peuvent être connectés à EMI pour piloter l'exécution du modèle. Les activités de V&V portent ainsi directement sur le modèle UML exécutable chargé dans EMI tout en réutilisant la même implémentation de la sémantique opérationnelle que celle utilisée lors de l'exécution du système. Cette approche permet ainsi d'assurer que les propriétés vérifiées par les outils de vérification formelle le sont aussi sur le modèle exécutable puisque le même couple (modèle UML + interpréteur de modèles) est utilisé pour l'exécution et l'analyse du modèle. Cette approche permet également d'assurer la validité des résultats de vérification même pour des langages semi-formels comme UML puisque tous les choix d'implémentation sont capturés dans une seule définition de la sémantique d'exécution.

Interface de communication. L'interpréteur EMI fournit une interface de communication permettant de connecter des outils de V&V. Cette interface de communication permet de récupérer la configuration courante de l'interpréteur (i.e., la partie dynamique du modèle), de mettre l'interpréteur dans une configuration donnée, de collecter l'ensemble des transitions tirables, et de tirer une transition à partir de sa configuration courante. Pour le model-checking, une requête supplémentaire est nécessaire afin de pouvoir évaluer des prédicats sur EMI et ainsi mieux découpler les opérations spécifiques à la vérification de celles spécifiques au langage de modélisation.

Activités de Vérification et Validation. Notre outil, EMI, peut s'interfacer avec le model-checker OBP2 pour effectuer différentes activités de V&V sur des modèles UML :

Model-checking de propriétés LTL OBP2 permet de vérifier des propriétés LTL sur les modèles exécutés par EMI. La vérification de ces propriétés LTL se base sur la composition d'automates de Büchi afin d'analyser des traces infinies. Cette technique nécessite notamment d'évaluer des propositions atomiques (i.e., des prédicats dépendants des objets du modèle) sur EMI. Si une propriété est violée, un contre-exemple sous la forme d'une trace est retourné par le model-checker.

Model-checking avec des automates observateurs Des propriétés de sûreté peuvent être spécifiées dans le langage de modélisation (ici UML) sous la forme d'automates observateurs. La vérification de ces propriétés est ensuite réalisée par composition synchrone avec le système et l'utilisation d'un algorithme d'atteignabilité.

Monitoring Les automates observateurs utilisés pendant la phase de vérification peuvent être déployés sur la cible embarquée avec EMI afin de surveiller le système lors de

son exécution. Malgré l'overhead engendré, le monitoring permet de vérifier les propriétés du système dans son environnement réel, de réagir en cas de défaillance, et d'analyser le problème à posteriori.

Simulation L'interface de simulation permet aux utilisateurs de visualiser la configuration courante d'EMI, de tirer des transitions, et d'explorer différents chemins d'exécution pour mieux comprendre le comportement du système.

Exploration de l'espace d'états OBP2 permet d'explorer l'espace d'états du modèle exécuté par EMI en utilisant un algorithme d'exploration à la volée.

Détection de deadlocks Le model-checker peut également détecter des deadlocks dans le modèle UML et retourner une trace des chemins ayant conduit à ces deadlocks.

Pour toutes ces activités, notre approche permet d'exprimer les résultats de vérification directement avec les concepts du langage de modélisation. Cela offre l'avantage de faciliter l'analyse et la compréhension des résultats par les utilisateurs.

4 Conclusion

Notre interpréteur de modèles UML permet d'améliorer la vérification des systèmes embarqués en appliquant directement les activités de V&V sur le modèle de conception et en utilisant la même sémantique opérationnelle que celle utilisée à l'exécution. De nombreux outils d'exécution de modèles existent, comme le montre l'étude systématique en [2], mais aucun ne se base sur une seule définition de la sémantique d'exécution pour vérifier et exécuter des modèles UML. Afin d'évaluer notre approche, EMI a notamment été mis en oeuvre sur un modèle de contrôleur de passage à niveau [1]. Une analyse plus approfondie reste à mener pour évaluer le passage à l'échelle et les performances de cet outil sur des modèles industriels. Pour enrichir cet outil, plusieurs perspectives sont à l'étude dont l'amélioration du déploiement qui permet de lier le modèle UML aux périphériques de la cible embarquée.

Remerciements Ce projet est partiellement financé par Davidson Consulting. Les auteurs remercient particulièrement David Olivier pour ses conseils et ses commentaires avisés sur le projet.

Références

[1] Valentin Besnard, Matthias Brun, Frédéric Jouault, Ciprian Teodorov, and Philippe Dhaussy. Unified LTL Verification and Embedded Execution of UML Models. In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18)*, Copenhagen, Denmark, October 2018.

[2] Federico Ciccozzi, Ivano Malavolta, and Bran Selic. Execution of UML models : a systematic review of research and practice. *Software & Systems Modeling*, April 2018.

[3] Ciprian Teodorov, Philippe Dhaussy, and Luka Le Roux. Environment-driven reachability for timed systems. *International Journal on Software Tools for Technology Transfer*, 19(2) :229–245, April 2017.

[4] Ciprian Teodorov, Luka Le Roux, Zoé Drey, and Philippe Dhaussy. Past-Free[ze] reachability analysis : reaching further with DAG-directed exhaustive state-space analysis. *Software Testing, Verification and Reliability*, 26(7) :516–542, 2016.