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

18ième journées Approches Formelles dans l'Assistance au Développement de Logiciels à Toulouse, France

Valentin BESNARD [1] Matthias BRUN [1] Philippe DHAUSSY [2]
Frédéric JOUAULT [1] Ciprian TEODOROV [2]

[1] ERIS, ESEO-TECH, Angers, France

[2] Lab-STICC UMR CNRS 6285, ENSTA Bretagne, Brest, France

Ce projet est partiellement financé par Davidson.

Sommaire

Sommaire

Introduction

Contexte

- Complexité croissante des systèmes embarqués
- Besoin croissant en vérification et validation

EMI : Embedded Model Interpreter

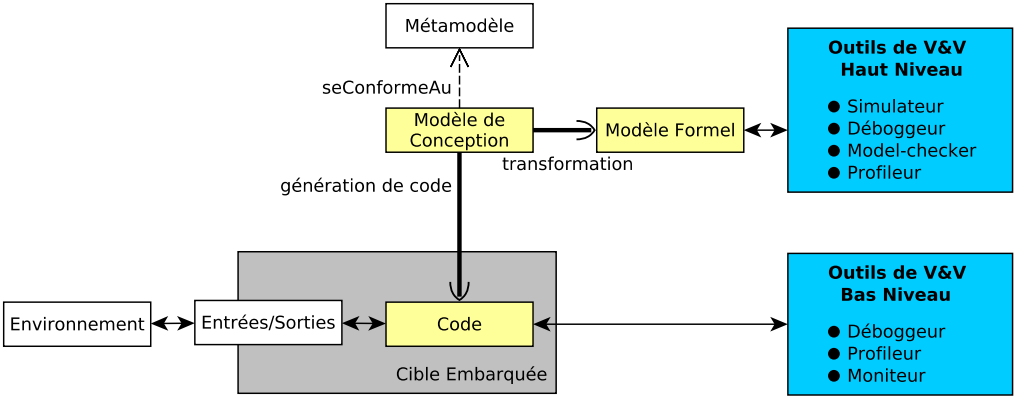
Un interpréteur de modèles UML qui permet :

- L'exécution d'un modèle UML
- La vérification de ce modèle grâce à des techniques de vérification formelle (avec le model-checker OBP2 [a]).

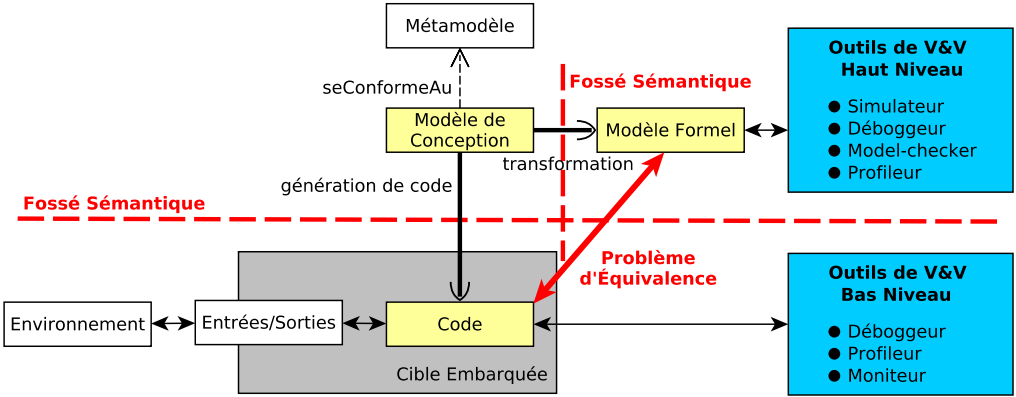
a. (`https://plug-obp.github.io/`)

Sommaire

Approche Classique

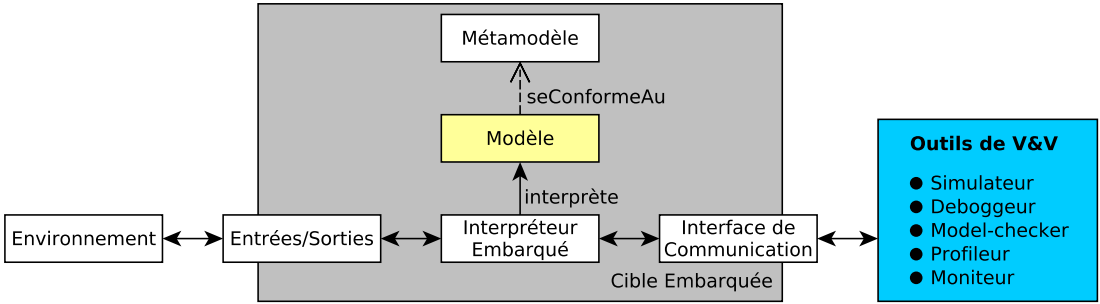


Trois Problématiques



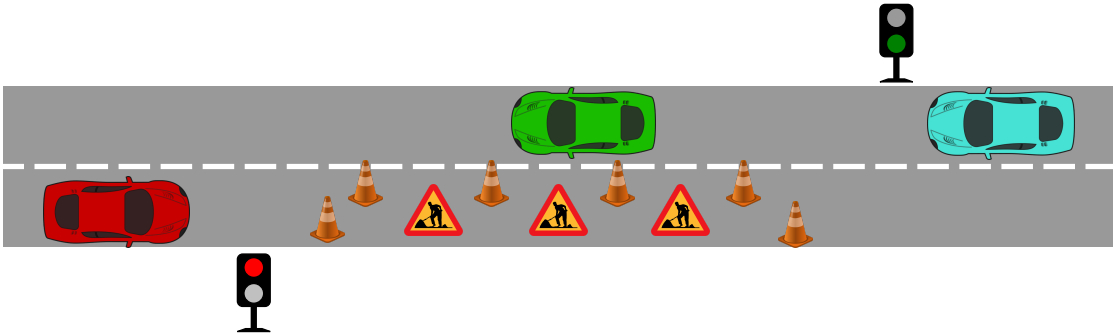
Cause principale de ces problèmes : De multiple implémentations de la sémantique du langage à cause des transformations de modèles.

Notre Approche



Une seule implémentation de la sémantique du langage
pour toutes les activités : simulation, exécution, et vérification.

Exemple : Gestion de Feux de Signalisation



Objectif

Garantir la sécurité des usagers et la fluidité de la circulation.

Sommaire

Processus de Développement

Processus de développement d'un modèle avec EMI

1. Modélisation du système logiciel en UML
2. Sérialisation du modèle en C
3. Chargement du modèle en mémoire
4. Utilisation du binaire exécutable

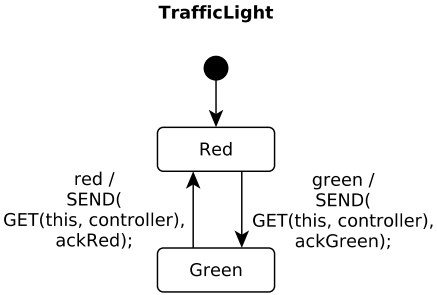
Modélisation du système logiciel en UML

Utilisation d'un sous-ensemble d'Eclipse UML pour modéliser :

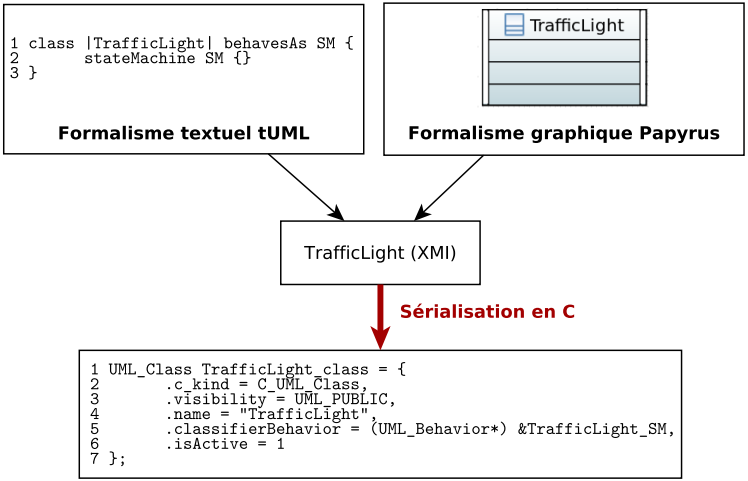
- La partie structurelle du système
  - Diagrammes de classes
  - Diagrammes de structure composite
- La partie comportementale du système
  - Machines à états

Utilisation d'un langage d'action basé sur le C pour définir précisément :

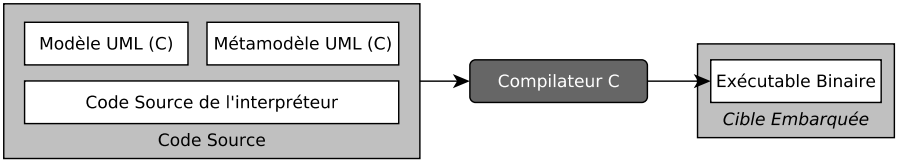
- Les gardes des transitions des machines à états
- Les effets des transitions des machines à états



Sérialisation du modèle UML en C



Chargement du modèle et Exécution



Chargement du modèle

- Chargement lors de la compilation
- Un seul chargement du modèle pour toutes les activités

Execution

- Sur un ordinateur équipé d'un OS Linux
- Sur une cible embarqué en bare-metal (i.e., sans OS)

Sommaire

Interface de Communication

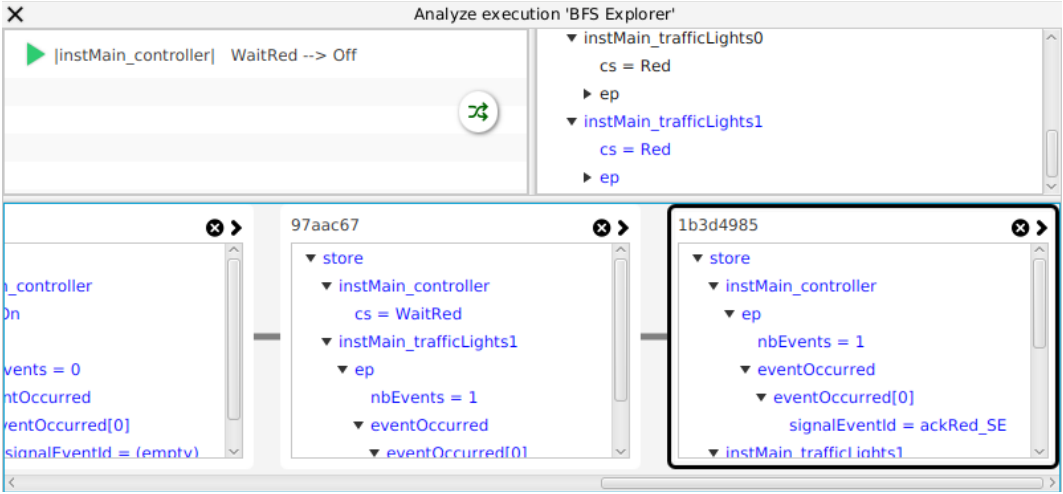
Configuration

- L'ensemble des données dynamique du modèle
- L'état exécutable du modèle

Deux fonctionnalités principales :

- Pilotage de l'interpréteur
  - *Get configuration*
  - *Get fireable transitions*
  - *Fire transition*
  - *Set configuration*
- Évaluation de prédicats
  - *Evaluate predicate*

Simulation


Analyze execution 'BFS Explorer'
|instMain_controller| WaitRed --> Off
instMain_trafficLights0
cs = Red
ep
instMain_trafficLights1
cs = Red
ep
97aac67
store
instMain_controller
cs = WaitRed
instMain_trafficLights1
ep
nbEvents = 1
eventOccurred
1b3d4985
store
instMain_controller
ep
nbEvents = 1
eventOccurred
eventOccurred[0]
signalEventId = ackRed_SE
instMain_trafficLights1

Model-checking de propriétés LTL

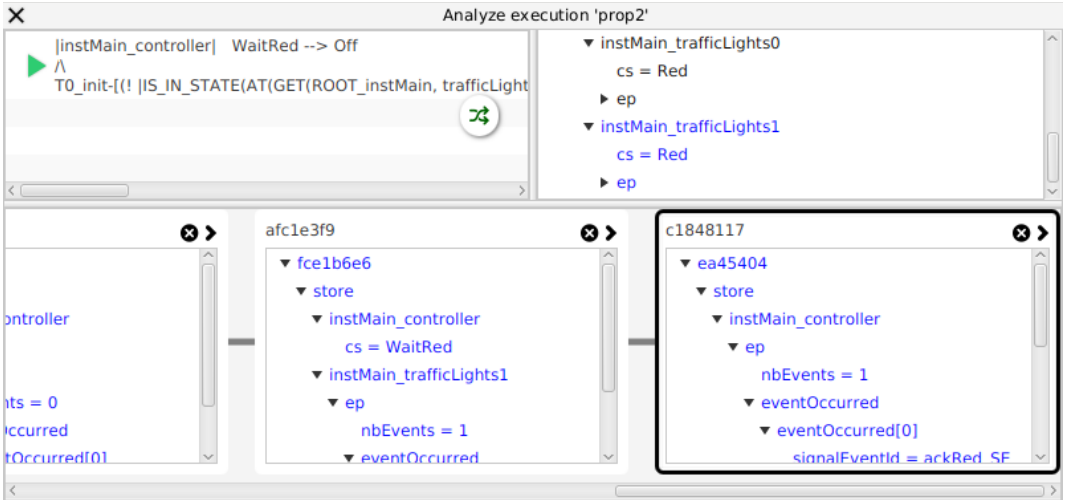
Exemple de propriété

- En langage naturel : Les deux feux ne sont jamais verts tous les deux simultanément.
- En LTL : `"[] !(firstIsGreen and secondIsGreen)"`

Processus utilisé pour le model-checking de propriété LTL

- Transformation de la propriété LTL en automate de Büchi
- Composition de cet automate avec l'automate de Büchi du système
- Recherche d'un contre-exemple

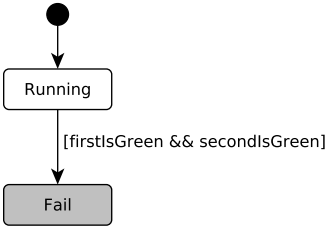
Model-checking de propriétés LTL



Model-checking et monitoring avec des automates observateurs

Exemple de propriété

- En langage naturel : Les deux feux ne sont jamais verts tous les deux simultanément.
- Avec un automate observateur : ci-contre



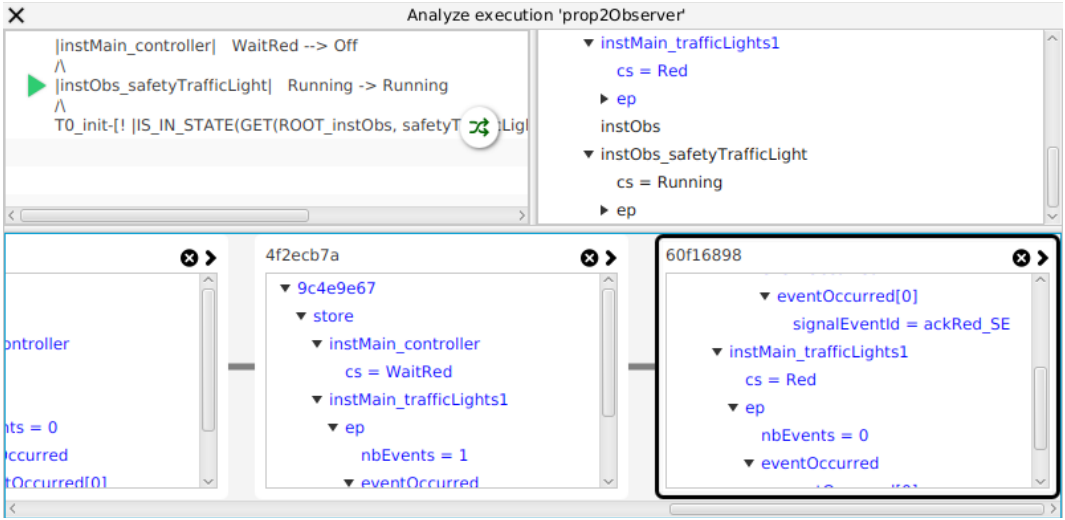
Processus utilisé pour le model-checking avec des automates observateurs

- Expression de la propriété sous la forme d'un automate observateur
- Composition de cet automate avec l'automate fini du système
- Utilisation d'un algorithme d'atteignabilité

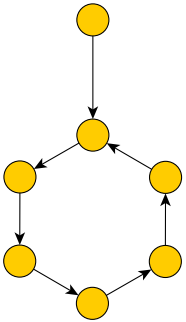
Processus utilisé pour le monitoring

- Déploiement des automates observateurs sur le système réel
- Analyse de la trace d'exécution courante

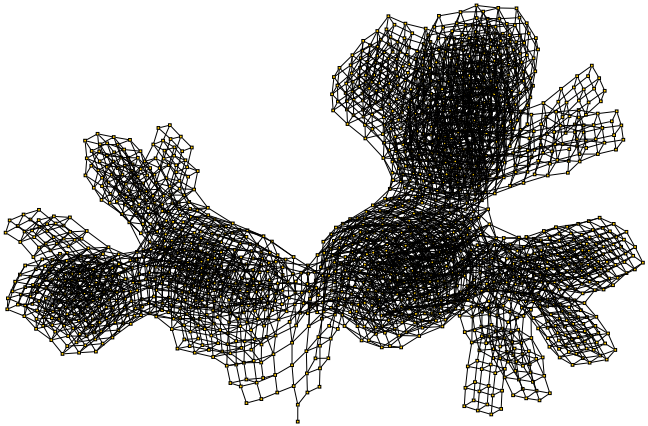
Model-checking avec des automates observateurs



Exploration de l'espace d'état et Détection de deadlocks



(a) Modèle de gestion de feux de signalisation



(b) Modèle de passage à niveau

Sommaire

Conclusion

EMI permet la vérification de modèles UML de systèmes embarqués en :

- Appliquant directement les activités de V&V sur le modèle de conception
- Utilisant la même sémantique opérationnelle que celle utilisée à l'exécution

Perspectives

- Analyser les performances de l'outil
- Améliorer le déploiement des modèles UML pour :
  - Avoir des modèles plus modulaire
  - Améliorer le lien entre les modèles UML et les périphériques de la cible embarquée
- Appliquer l'outil sur un cas d'étude industriel

Merci de votre attention

Bibliographie I

Valentin Besnard, Matthias Brun, Philippe Dhaussy, Frédéric Jouault, David Olivier, and Ciprian Teodorov.
Towards one Model Interpreter for Both Design and Deployment.
In *3rd International Workshop on Executable Modeling (EXE 2017)*, Austin, United States, September 2017.

Valentin Besnard, Matthias Brun, Frédéric Jouault, Ciprian Teodorov, and Philippe Dhaussy.
Embedded UML Model Execution to Bridge the Gap Between Design and Runtime.
In *MDE@DeRun 2018 : First International Workshop on Model-Driven Engineering for Design-Runtime Interaction in Complex Systems*, Toulouse, France, June 2018.

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.

Bibliographie II


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


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.


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.