Unification de la Vérification et de l'Exécution Embarquée de Modèles

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

Valentin BESNARD

ERIS, ESEO-TECH, Angers, France
en collaboration avec ENSTA Bretagne et Davidson

Sommaire

1. Contexte
2. Approche Classique vs Notre Approche
3. Application au Langage UML
4. Exemple
5. Conclusion

Sommaire

1 Contexte

2 Approche Classique vs Notre Approche

3 Application au Langage UML

4 Exemple

5 Conclusion

Contexte

Observations

- Complexité croissante des systèmes embarqués
- Émergence de nouveaux besoins et de nouvelles applications

Contexte

Observations

- Complexité croissante des systèmes embarqués
- Émergence de nouveaux besoins et de nouvelles applications

Conséquences sur les programmes logiciels

- Augmentation du risque de défaillances logicielles
- Augmentation des exigences de sûreté de fonctionnement et de sécurité

Contexte

Observations

- Complexité croissante des systèmes embarqués
- Émergence de nouveaux besoins et de nouvelles applications

Conséquences sur les programmes logiciels

- Augmentation du risque de défaillances logicielles
- Augmentation des exigences de sûreté de fonctionnement et de sécurité

Conséquence sur le développement logiciel

- Besoin croissant en vérification et validation

Sommaire

1 Contexte

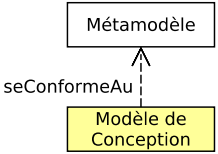
2 Approche Classique vs Notre Approche

3 Application au Langage UML

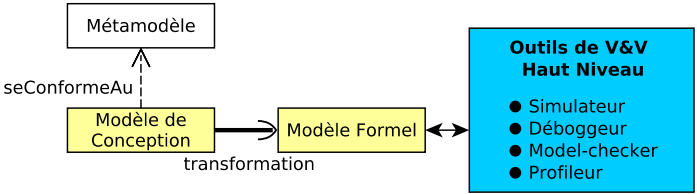
4 Exemple

5 Conclusion

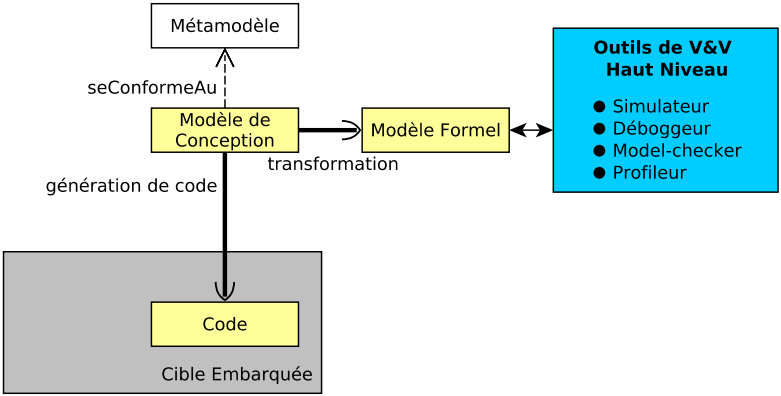
Approche Classique



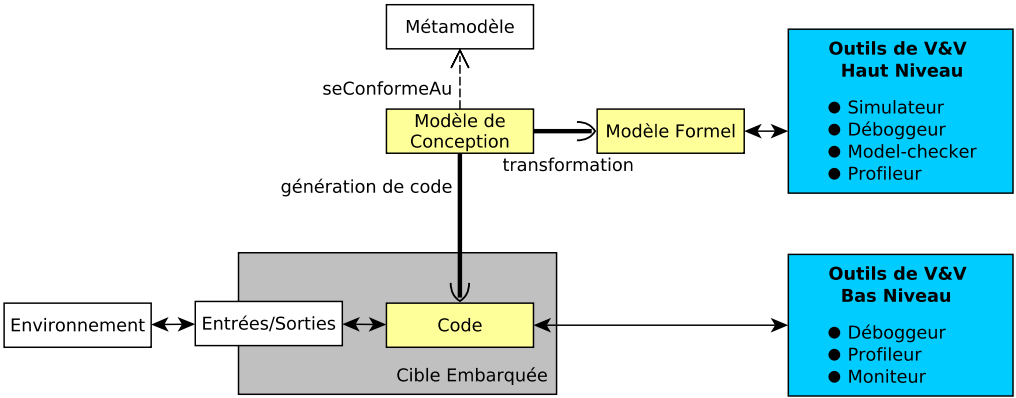
Approche Classique



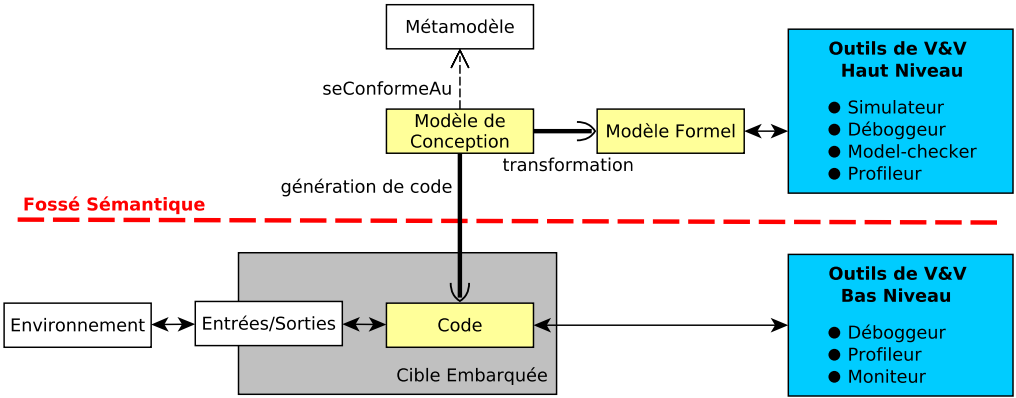
Approche Classique



Approche Classique

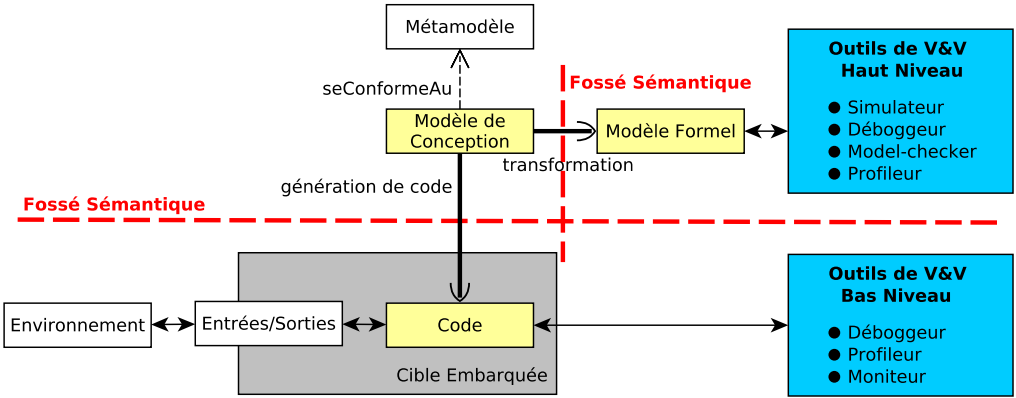


Trois Problématiques



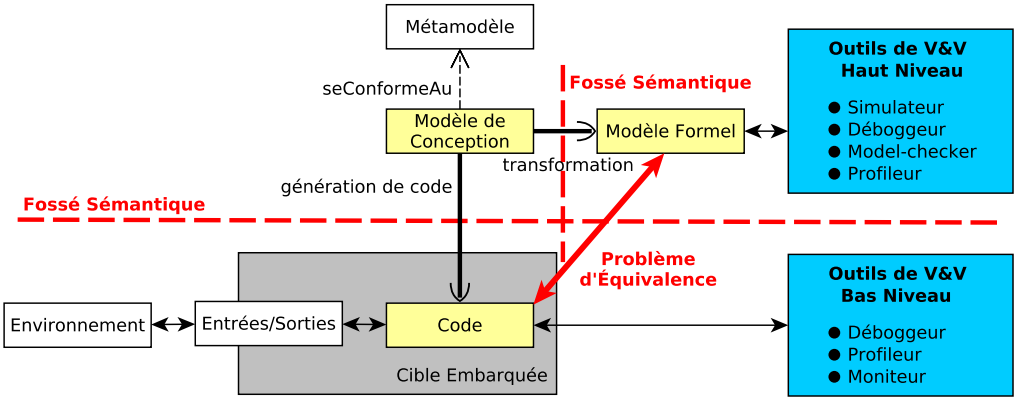
1ière problématique : Fossé sémantique entre le modèle de conception et le code exécutable.

Trois Problématiques



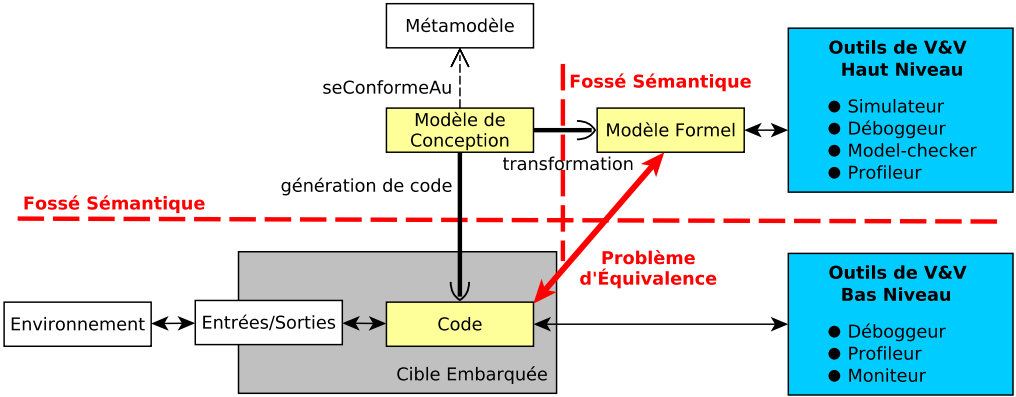
2ième problématique : Fossé sémantique entre le modèle de conception et le modèle formel.

Trois Problématiques



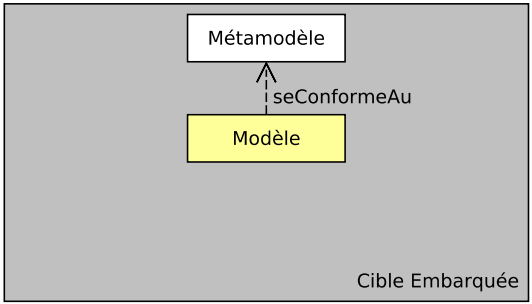
3ième problématique : Une relation d'équivalence entre le modèle formel et le code exécutable doit être établie, prouvée, et maintenue.

Trois Problématiques

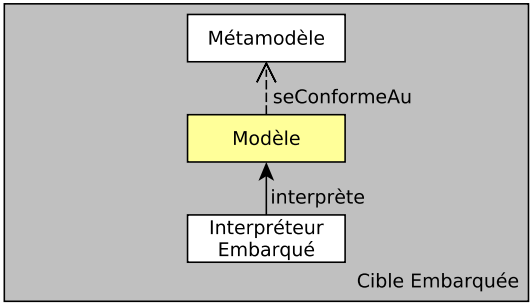


Cause principale de ces problèmes : De multiple implémentations de la sémantique du langage à cause des transformations de modèles.

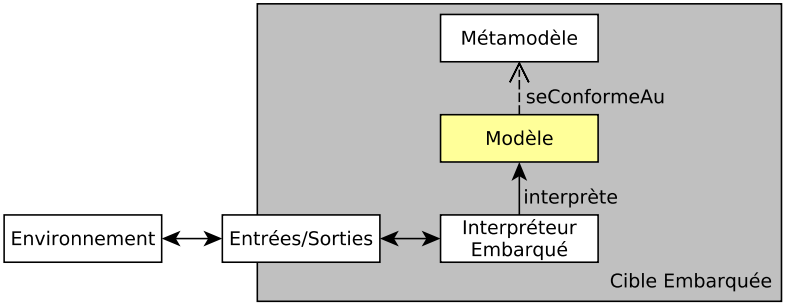
Notre Approche



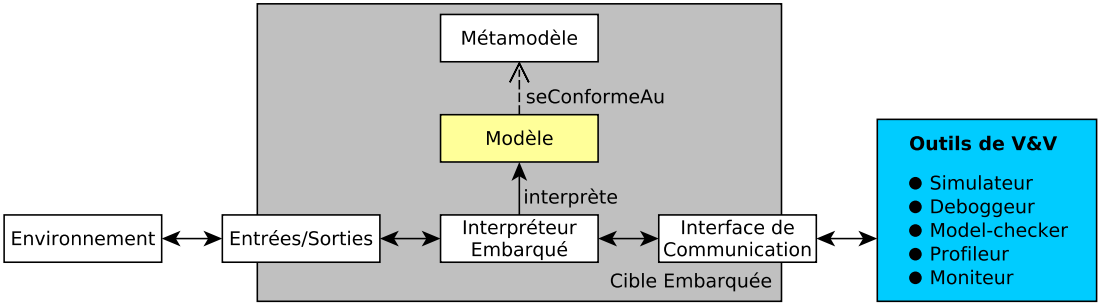
Notre Approche



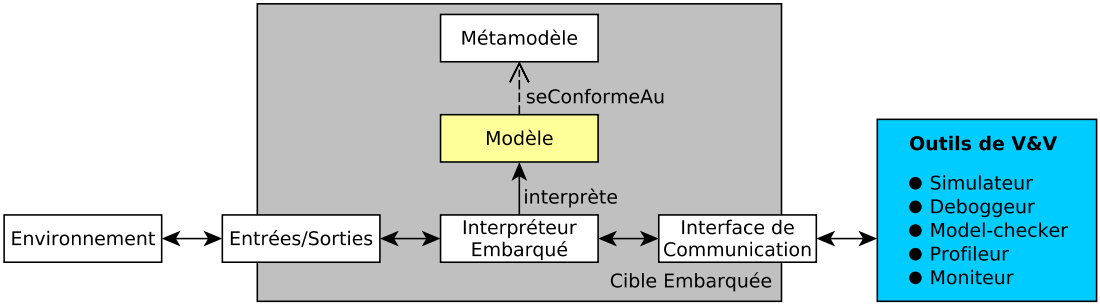
Notre Approche



Notre Approche



Notre Approche



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

État de l'Art

Outils pour l'exécution de modèles :

- Étude systématique [10]
- Rational Software Architect [13] et Rhapsody [11]
- Interpréteurs fUML Moka [16] et Moliz [15]
- Compilateurs GUML [8] et UniComp [9]
- GEMOC Studio [7]

Approches alternatives :

- Débogueurs [2] et [12]
- Compilateur certifié CompCert [14]
- Event-B [1]
- SCADE [3]

Sommaire

1 Contexte

2 Approche Classique vs Notre Approche

3 Application au Langage UML

4 Exemple

5 Conclusion

Application au Langage UML

- UML : Unified Modeling Language
- Un langage semi-formel avec des points de variation sémantique
  - La sémantique opérationnelle de l'interpréteur sert de référence
  - Les choix d'implémentation permettent de déterminer le comportement du système
  - Préservation de la relation d'équivalence
- Utilisation d'un sous-ensemble d'UML pouvant se représenter par :
  - Diagrammes de classes
  - Diagrammes de structure composite
  - Machines à états

Framework EMI

- EMI : Embedded Model Interpreter
- 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
- Connexion d'OBP2 [1] à EMI pour mener diverses activités d'analyse :
  - Simulation
  - Model-checking de propriétés LTL
  - Model-checking et monitoring avec des automates observateurs

1. https://plug-obp.github.io/

Sommaire

1 Contexte

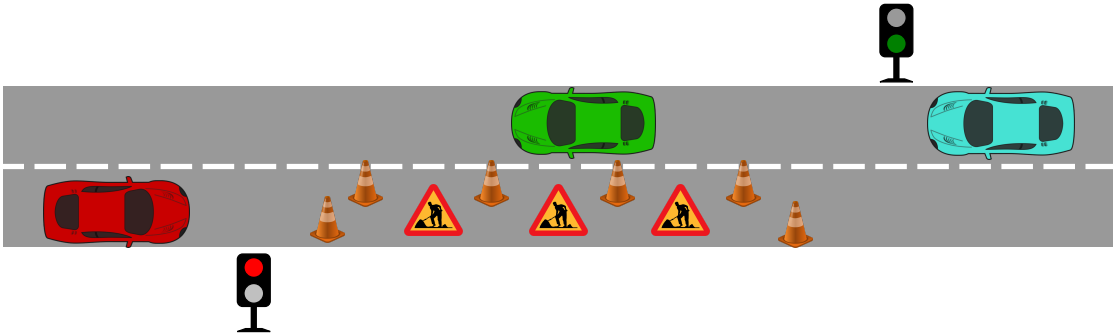
2 Approche Classique vs Notre Approche

3 Application au Langage UML

4 Exemple

5 Conclusion

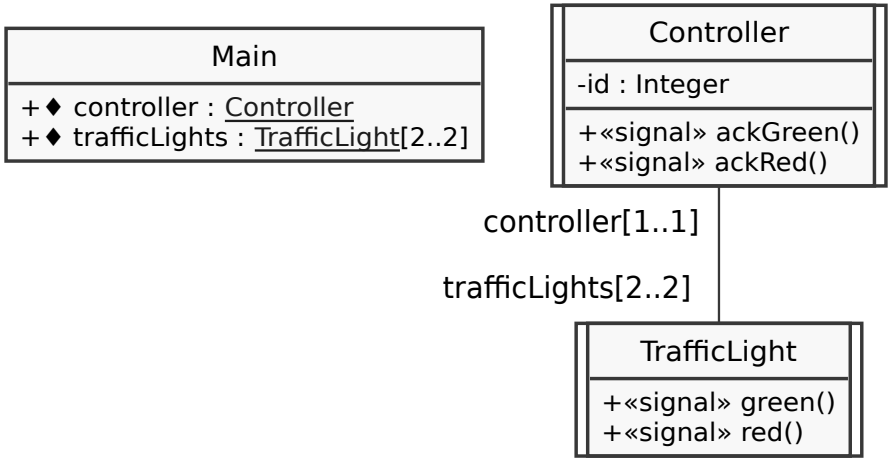
Exemple : Gestion de Feux de Signalisation



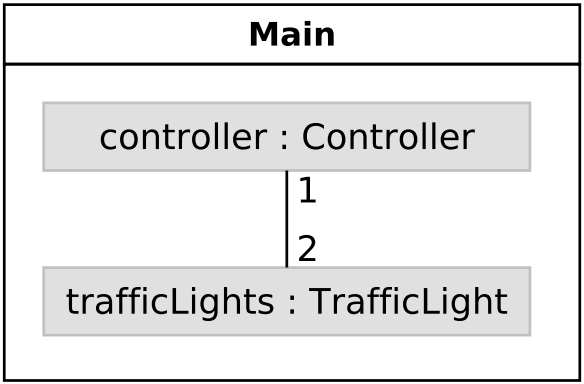
Objectif

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

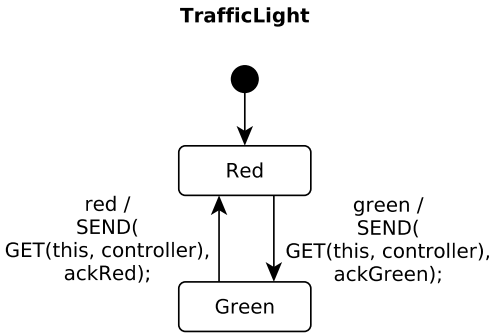
Gestion de Feux de Signalisation (Diagramme de classes)

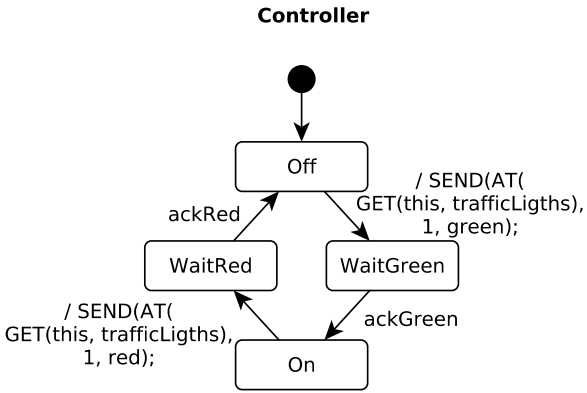


Gestion de Feux de Signalisation (Diagramme de composite structure)



Gestion de Feux de Signalisation (Machines à états)





Exigences pour la Gestion de Feux de Signalisation

En langage naturel

1. Le premier feu est vert infiniment souvent.
2. Les deux feux ne sont jamais verts tous les deux simultanément.

En LTL

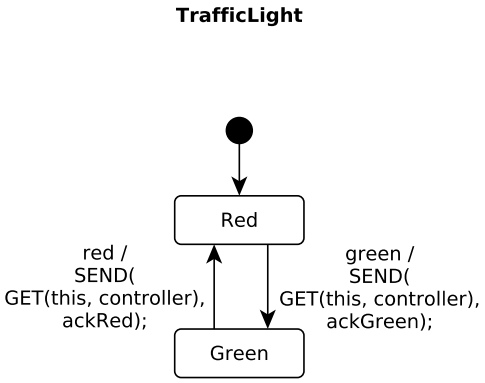
1. `"[] <> firstIsGreen"`
2. `"[] !(firstIsGreen and secondIsGreen)"`

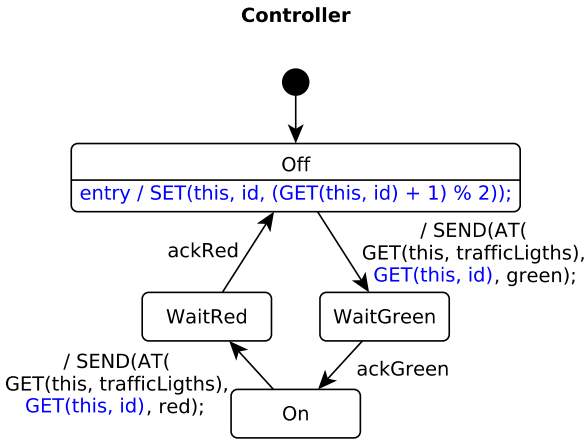
Résultats

1. Rejected
2. Verified

Analyse du contre-exemple de la propriété 1

Gestion de Feux de Signalisation (Machines à états v2)



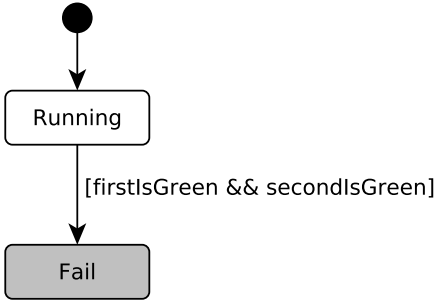


Les deux propriétés sont maintenant vérifiées.

Monitoring de la propriété 2

Monitoring de propriétés de sûreté avec des automates observateurs

- Encodage de la propriété dans un automate observateur en UML
- Déploiement de l'automate observateur avec le système
- Monitoring de la propriété à l'exécution



Sommaire

1 Contexte

2 Approche Classique vs Notre Approche

3 Application au Langage UML

4 Exemple

5 Conclusion

Conclusion

Contributions

- Utilisation d'une seule implémentation de la sémantique du langage
- Les propriétés vérifiées pendant la phase de vérification le restent lors de l'exécution

⇒ Contribution à l'amélioration de la vérification des systèmes embarqués

Perspectives

- Évaluation de l'interpréteur de modèles (e.g., performances, utilisabilité, limites)
- Mise en oeuvre de l'outil sur un cas d'étude industriel

Merci de votre attention

Bibliographie I

Jean-Raymond Abrial.

Modeling in Event-B : System and Software Engineering.
Cambridge University Press, New York, NY, USA, 2013.

Mojtaba Bagherzadeh, Nicolas Hili, and Juergen Dingel.

Model-level, Platform-independent Debugging in the Context of the Model-driven Development of Real-time Systems.
In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 419–430, New York, USA, 2017. ACM.

Gérard Berry.

SCADE : Synchronous Design and Validation of Embedded Control Software.
In S. Ramesh and Prahladavaradan Sampath, editors, *Next Generation Design and Verification Methodologies for Distributed Embedded Control Systems*, pages 19–33, Dordrecht, 2007. Springer Netherlands.

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)*, 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.

Bibliographie II

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.

Erwan Bousse, Thomas Degueule, Didier Vojtisek, Tanja Mayerhofer, Julien Deantoni, and Benoit Combemale.

Execution Framework of the GEMOC Studio (Tool Demo).
In *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*, SLE 2016, pages 84–89, New York, NY, USA, 2016. ACM.

Asma Charfi Smaoui, Chokri Mraidha, and Pierre Boulet.

An Optimized Compilation of UML State Machines.
In *ISORC - 15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, Shenzhen, China, April 2012.

Federico Ciccozzi.

Unicomp : A Semantics-aware Model Compiler for Optimised Predictable Software.
In *Proceedings of the 40th International Conference on Software Engineering : New Ideas and Emerging Results*, ICSE-NIER '18, pages 41–44, New York, NY, USA, 2018. ACM.

Federico Ciccozzi, Ivano Malavolta, and Bran Selic.

Execution of UML models : a systematic review of research and practice.
Software & Systems Modeling, April 2018.

Bibliographie III

Eran Gery, David Harel, and Eldad Palachi.
Rhapsody : A Complete Life-Cycle Model-Based Development System.
In Michael Butler, Luigia Petre, and Kaisa Sere, editors, *Integrated Formal Methods*, pages 1–10, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

Padma Iyenghar, Elke Pulvermueller, Clemens Westerkamp, Juergen Wuebbelmann, and Michael Uelschen.
Model-Based Debugging of Embedded Software Systems, pages 107–132.
Springer New York, New York, NY, 2017.

Daniel Leroux, Martin Nally, and Kenneth Hussey.
Rational Software Architect : A tool for domain-specific modeling.
IBM systems journal, 45(3) :555–568, 2006.

Xavier Leroy.
The CompCert C verified compiler : Documentation and user's manual.
Intern report, Inria, June 2017.

Tanja Mayerhofer and Philip Langer.
Moliz : A Model Execution Framework for UML Models.
In *Proceedings of the 2nd International Master Class on Model-Driven Engineering : Modeling Wizards*, MW '12, pages 3 :1–3 :2, New York, NY, USA, 2012. ACM.

Sebastien Revol, Géry Delog, Arnaud Cuccurru, and Jérémie Tatibouët.
Papyrus : Moka overview, 2018.
`https://wiki.eclipse.org/Papyrus/UserGuide/ModelExecution`.

Bibliographie IV

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.