EMI : Une approche pour unifier l'analyse et l'exécution embarquée à l'aide d'un interpréteur de modèles pilotable

Application aux modèles UML des systèmes embarqués

Soutenance de thèse de Valentin BESNARD

École doctorale MathSTIC

Mercredi 9 décembre 2020 — ESEO Angers

Rapporteurs

Frédéric BONIOL, ONERA/DTIS

Benoît COMBEMALE, Université de Rennes 1, IRISA

Examinateurs

Isabelle BORNE, Université Bretagne Sud, IRISA

Julien DEANTONI, Université Côte d'Azur, I3S

Frédéric JOUAULT, ESEO

Directeur de thèse

Philippe DHAUSSY, ENSTA Bretagne, Lab-STICC

Encadrants

Matthias BRUN, ESEO

Ciprian TEODOROV, ENSTA Bretagne, Lab-STICC

Invité

David OLIVIER, Davidson Consulting

Sommaire

Sommaire

Contexte

Observations

- Omniprésence des systèmes embarqués dans notre société

Contexte

Observations

- Omniprésence des systèmes embarqués dans notre société
- Complexité croissante des logiciels embarqués

Contexte

Observations

- Omniprésence des systèmes embarqués dans notre société
- Complexité croissante des logiciels embarqués
- Augmentation des risques d'introduction de fautes

Contexte

Observations

- Omniprésence des systèmes embarqués dans notre société
- Complexité croissante des logiciels embarqués
- Augmentation des risques d'introduction de fautes

Besoins

- Assister la conception des logiciels embarqués par les ingénieurs

Ingénieur modélisation

Conçoit le(s) modèle(s)

Modèle

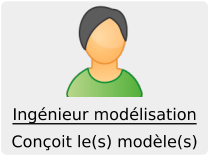
Contexte

Observations

- Omniprésence des systèmes embarqués dans notre société
- Complexité croissante des logiciels embarqués
- Augmentation des risques d'introduction de fautes

Besoins

- Assister la conception des logiciels embarqués par les ingénieurs



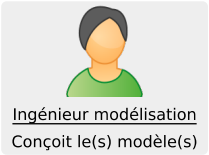
Contexte

Observations

- Omniprésence des systèmes embarqués dans notre société
- Complexité croissante des logiciels embarqués
- Augmentation des risques d'introduction de fautes

Besoins

- Assister la conception des logiciels embarqués par les ingénieurs
- Faciliter la vérification et la validation (V&V) durant les phases de conception



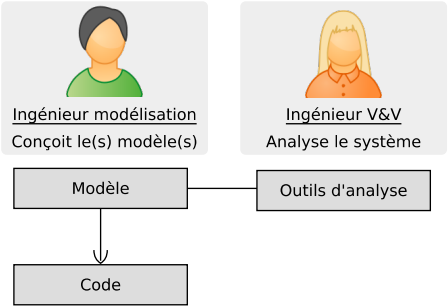
Contexte

Observations

- Omniprésence des systèmes embarqués dans notre société
- Complexité croissante des logiciels embarqués
- Augmentation des risques d'introduction de fautes

Besoins

- Assister la conception des logiciels embarqués par les ingénieurs
- Faciliter la vérification et la validation (V&V) durant les phases de conception



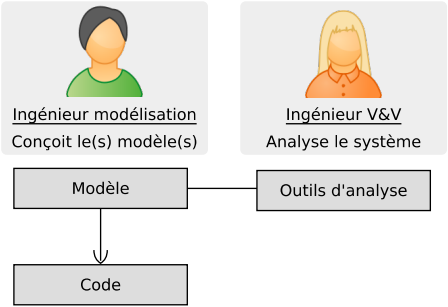
Contexte

Observations

- Omniprésence des systèmes embarqués dans notre société
- Complexité croissante des logiciels embarqués
- Augmentation des risques d'introduction de fautes

Besoins

- Assister la conception des logiciels embarqués par les ingénieurs
- Faciliter la vérification et la validation (V&V) durant les phases de conception
- Garantir que les résultats des activités de V&V obtenus restent applicables aux systèmes opérationnels qui s'exécutent en production



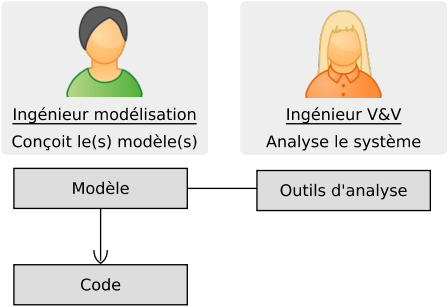
Contexte

Observations

- Omniprésence des systèmes embarqués dans notre société
- Complexité croissante des logiciels embarqués
- Augmentation des risques d'introduction de fautes

Besoins

- Assister la conception des logiciels embarqués par les ingénieurs
- Faciliter la vérification et la validation (V&V) durant les phases de conception
- Garantir que les résultats des activités de V&V obtenus restent applicables aux systèmes opérationnels qui s'exécutent en production
- ... pour différents langages de modélisation



Sommaire

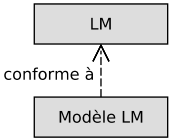
Identification des approches d'analyse et d'exécution

Présentation des approches existantes dans l'état de l'art :

AC#1 Traduction vers modèle vérifiable et code

- Mbeddr [Voe+12]

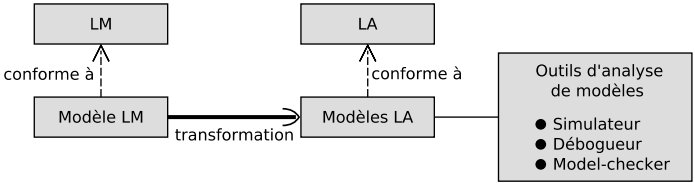
AC#1 : Traduction vers modèle vérifiable et code



Exemple d'outil : Mbeddr [Voe+12]

[Voe+12] Voelter et al., « Mbeddr : An Extensible C-based Programming Language and IDE for Embedded Systems », 2012

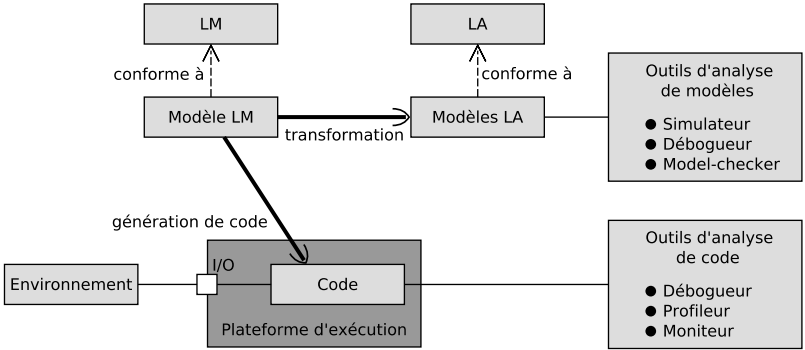
AC#1 : Traduction vers modèle vérifiable et code



Exemple d'outil : Mbeddr [Voe+12]

[Voe+12] Voelter et al., « Mbeddr : An Extensible C-based Programming Language and IDE for Embedded Systems », 2012

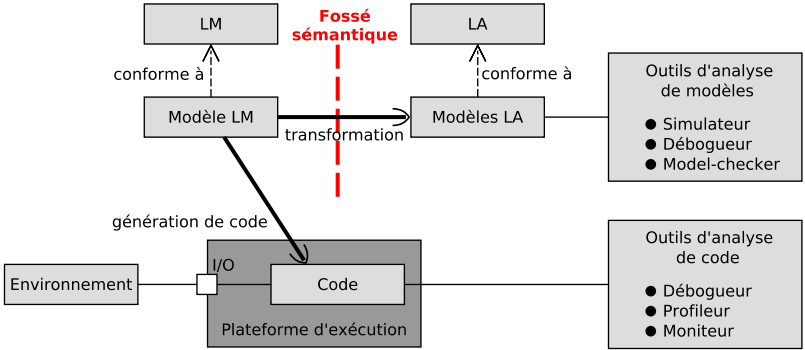
AC#1 : Traduction vers modèle vérifiable et code



Exemple d'outil : Mbeddr [Voe+12]

[Voe+12] Voelter et al., « Mbeddr : An Extensible C-based Programming Language and IDE for Embedded Systems », 2012

AC#1 : Traduction vers modèle vérifiable et code

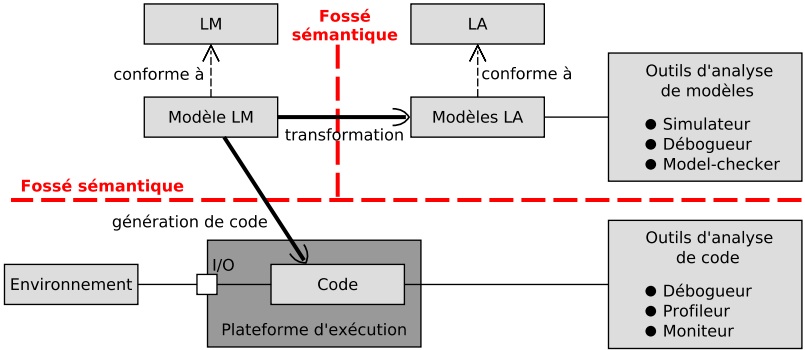


P#1 Fossé sémantique entre le modèle de conception et les modèles d'analyse

Exemple d'outil : Mbeddr [Voe+12]

[Voe+12] Voelter et al., « Mbeddr : An Extensible C-based Programming Language and IDE for Embedded Systems », 2012

AC#1 : Traduction vers modèle vérifiable et code

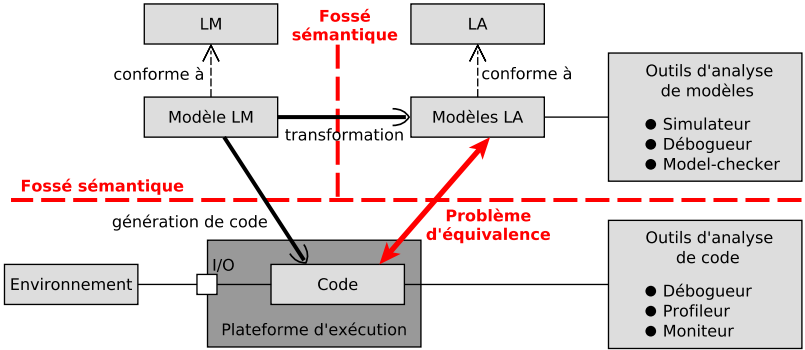


P#2 Fossé sémantique entre le modèle de conception et le code exécutable

Exemple d'outil : Mbeddr [Voe+12]

[Voe+12] Voelter et al., « Mbeddr : An Extensible C-based Programming Language and IDE for Embedded Systems », 2012

AC#1 : Traduction vers modèle vérifiable et code

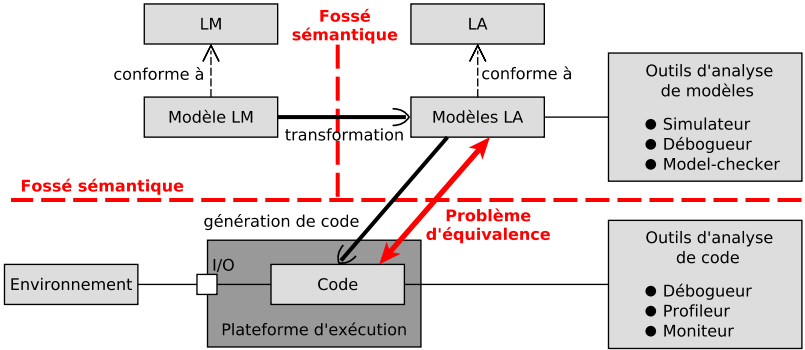


P#3 Problème d'équivalence entre les modèles d'analyse et le code exécutable

Exemple d'outil : Mbeddr [Voe+12]

[Voe+12] Voelter et al., « Mbeddr : An Extensible C-based Programming Language and IDE for Embedded Systems », 2012

AC#2 : Traduction vers modèle vérifiable puis code



Exemple d'outil : RobotChart [Miy+19]

[Miy+19] Miyazawa et al., « RoboChart : modelling and verification of the functional behaviour of robotic applications », 2019

Identification des approches d'analyse et d'exécution

Présentation des approches existantes dans l'état de l'art :

AC#1 Traduction vers modèle vérifiable et code
- Mbeddr [Voe+12]

AC#2 Traduction vers modèle vérifiable puis code
- RobotChart [Miy+19]

Identification des approches d'analyse et d'exécution

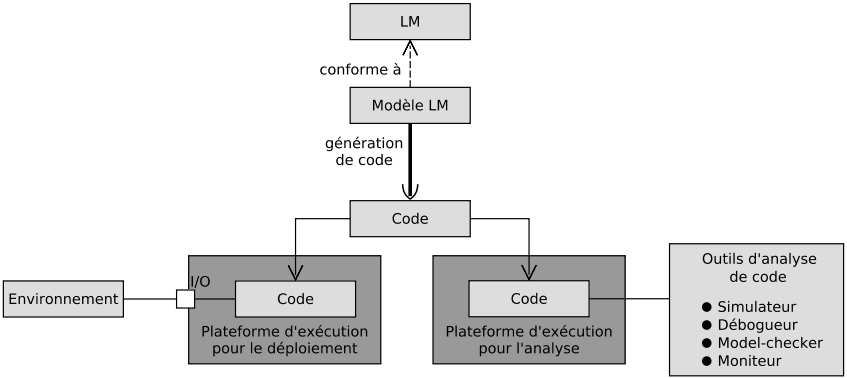
Présentation des approches existantes dans l'état de l'art :

AC#1 Traduction vers modèle vérifiable et code
- Mbeddr [Voe+12]

AC#2 Traduction vers modèle vérifiable puis code
- RobotChart [Miy+19]

AC#3 Traduction vers code vérifiable
- Divine [Bar+17]

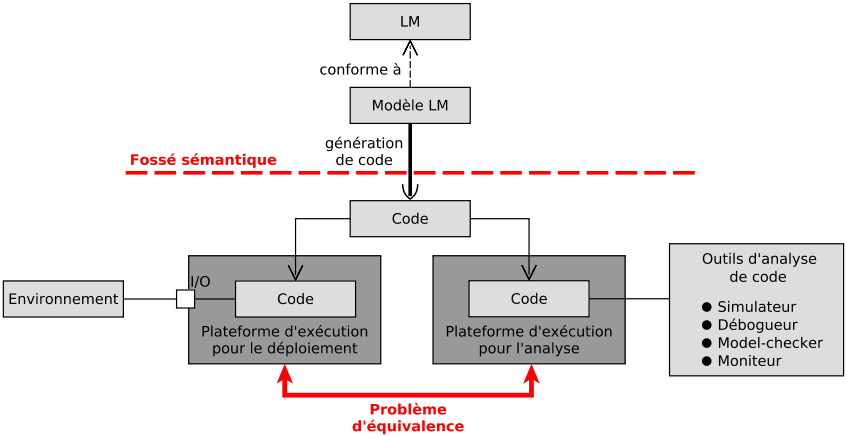
AC#3 : Traduction vers code vérifiable



Exemple d'outil : Divine [Bar+17]

[Bar+17] Baranová et al., « Model Checking of C and C++ with DIVINE 4 », 2017

AC#3 : Traduction vers code vérifiable



Exemple d'outil : Divine [Bar+17]

[Bar+17] Baranová et al., « Model Checking of C and C++ with DIVINE 4 », 2017

Identification des approches d'analyse et d'exécution

Présentation des approches existantes dans l'état de l'art :

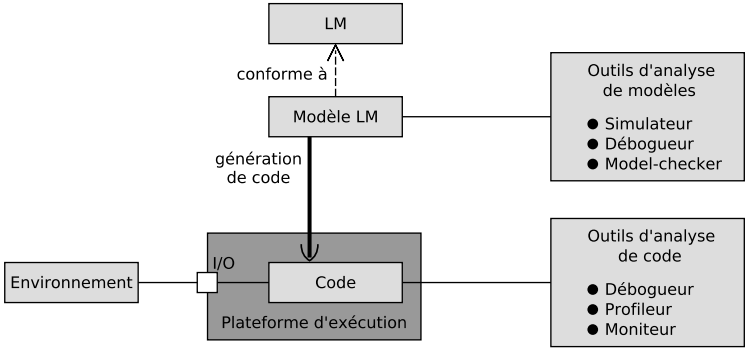
AC#1 Traduction vers modèle vérifiable et code
- Mbeddr [Voe+12]

AC#2 Traduction vers modèle vérifiable puis code
- RobotChart [Miy+19]

AC#3 Traduction vers code vérifiable
- Divine [Bar+17]

AC#4 Traduction modèle vérifiable vers code
- SPOT [DP04] avec générateur de code

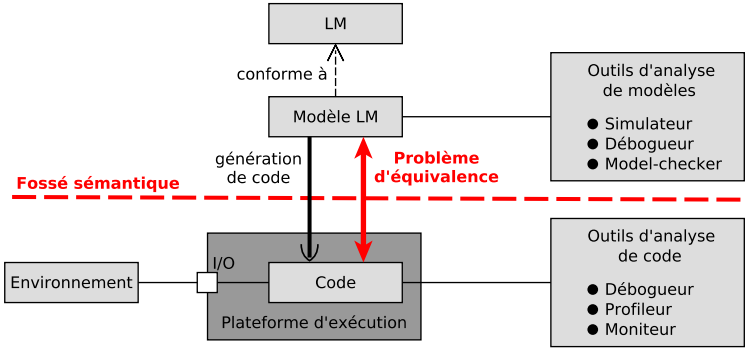
AC#4 : Traduction modèle vérifiable vers code



Exemple d'outil : SPOT [DP04] avec un générateur de code

[DP04] Duret-Lutz et al., « SPOT : An Extensible Model Checking Library Using Transition-Based Generalized Büchi Automata », 2004

AC#4 : Traduction modèle vérifiable vers code



Exemple d'outil : SPOT [DP04] avec un générateur de code

[DP04] Duret-Lutz et al., « SPOT : An Extensible Model Checking Library Using Transition-Based Generalized Büchi Automata », 2004

Identification des approches d'analyse et d'exécution

Présentation des approches existantes dans l'état de l'art :

AC#1 Traduction vers modèle vérifiable et code
- Mbeddr [Voe+12]

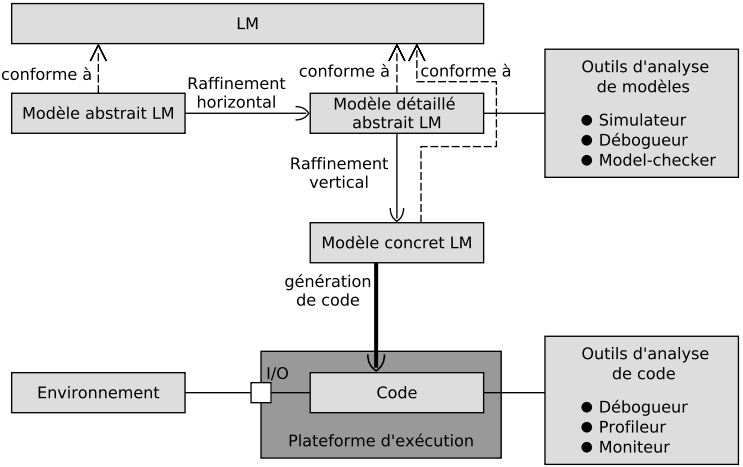
AC#2 Traduction vers modèle vérifiable puis code
- RobotChart [Miy+19]

AC#3 Traduction vers code vérifiable
- Divine [Bar+17]

AC#4 Traduction modèle vérifiable vers code
- SPOT [DP04] avec générateur de code

AC#5 Raffinement de modèles jusqu'au code
- Atelier B (`https://www.atelierb.eu/`)

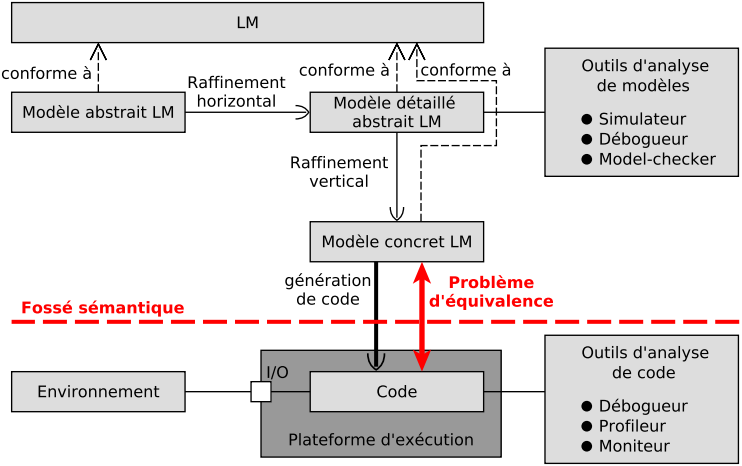
AC#5 : Raffinement de modèles jusqu'au code



Exemple d'outil : Atelier B [1]

1. ClearSy System Engineering, *Atelier B*, URL : `https://www.atelierb.eu/`.

AC#5 : Raffinement de modèles jusqu'au code



Exemple d'outil : Atelier B [1]

1. ClearSy System Engineering, *Atelier B*, URL : `https://www.atelierb.eu/`.

Identification des approches d'analyse et d'exécution

Présentation des approches existantes dans l'état de l'art :

AC#1 Traduction vers modèle vérifiable et code
- Mbeddr [Voe+12]

AC#2 Traduction vers modèle vérifiable puis code
- RobotChart [Miy+19]

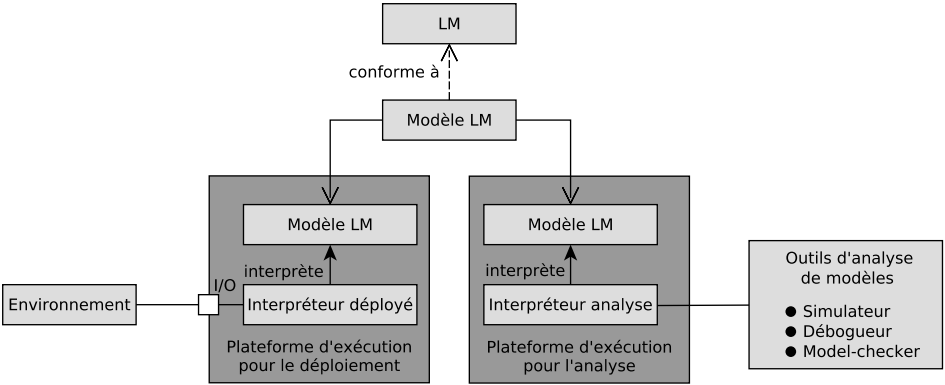
AC#3 Traduction vers code vérifiable
- Divine [Bar+17]

AC#4 Traduction modèle vérifiable vers code
- SPOT [DP04] avec générateur de code

AC#5 Raffinement de modèles jusqu'au code
- Atelier B (`https://www.atelierb.eu/`)

AC#6 Interprétations spécifiques pour vérification et exécution réelle
- Java PathFinder [Bra+00]

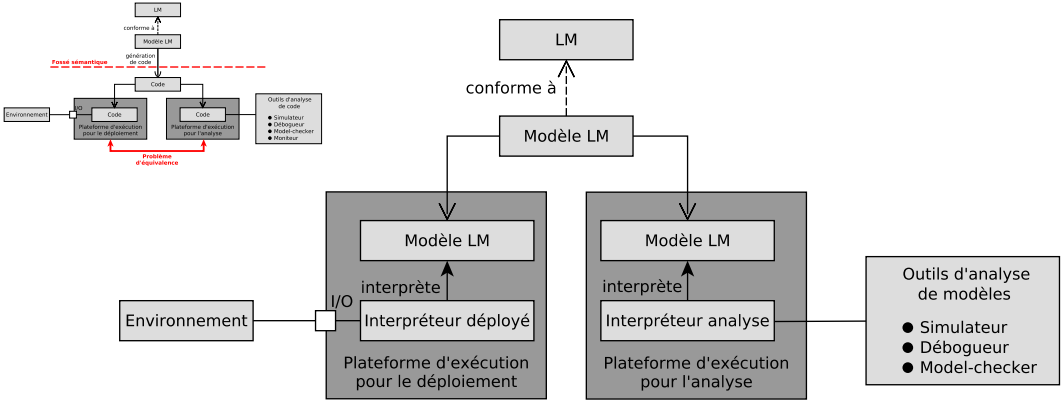
AC#6 : Interprétations spécifiques pour vérification et exécution réelle



Exemple d'outil : Java PathFinder [Bra+00]

[Bra+00] Brat et al., « Java PathFinder - Second Generation of a Java Model Checker », 2000

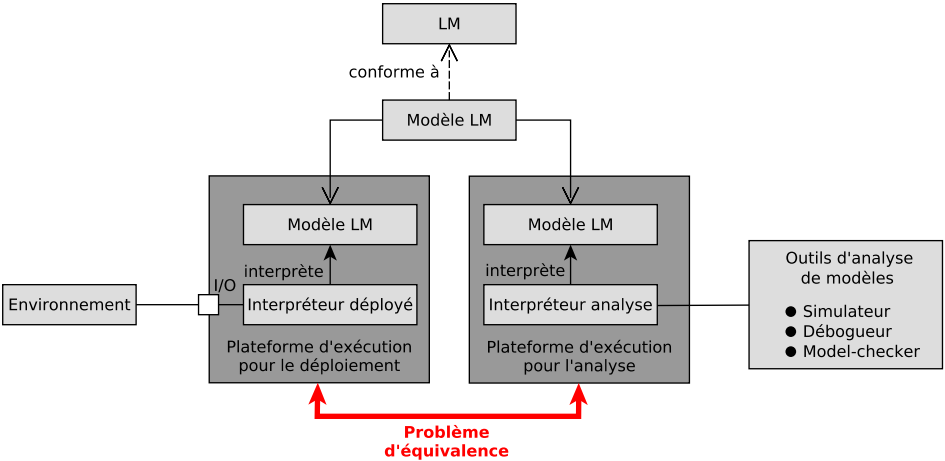
AC#6 : Interprétations spécifiques pour vérification et exécution réelle



Exemple d'outil : Java PathFinder [Bra+00]

[Bra+00] Brat et al., « Java PathFinder - Second Generation of a Java Model Checker », 2000

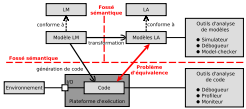
AC#6 : Interprétations spécifiques pour vérification et exécution réelle

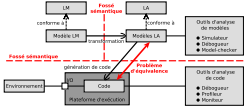


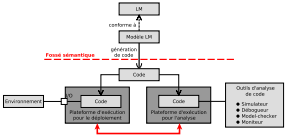
Exemple d'outil : Java PathFinder [Bra+00]

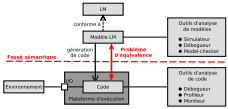
[Bra+00] Brat et al., « Java PathFinder - Second Generation of a Java Model Checker », 2000

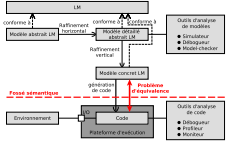
Récapitulatif des problèmes scientifiques

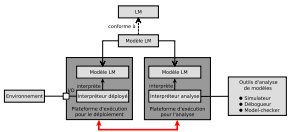




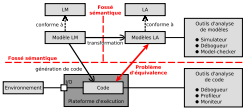


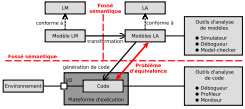


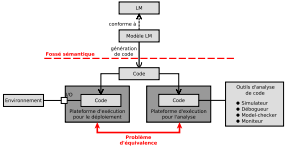




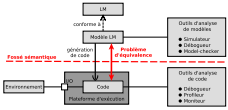
Récapitulatif des problèmes scientifiques

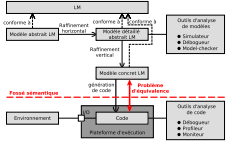


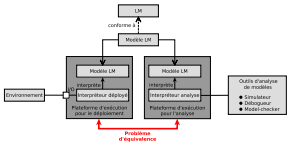




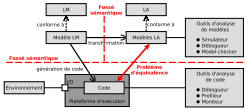
- P#1 Fossé sémantique entre le modèle de conception et les modèles d'analyse
- P#2 Fossé sémantique entre le modèle de conception et le code exécutable
- P#3 Problème d'équivalence entre les modèles d'analyse et le code exécutable

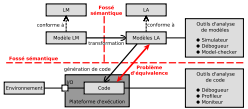


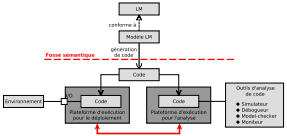




Récapitulatif des problèmes scientifiques







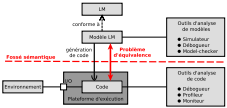
P#1 Fossé sémantique entre le modèle de conception et les modèles d'analyse

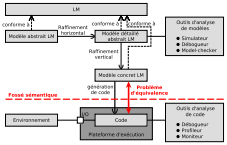
P#2 Fossé sémantique entre le modèle de conception et le code exécutable

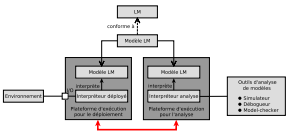
P#3 Problème d'équivalence entre les modèles d'analyse et le code exécutable

	AC#1	AC#2	AC#3	AC#4	AC#5	AC#6
P#1	✗	✗	✓	✓	✓	✓
P#2	✗	✗	✗	✗	✗	✓
P#3	✗	✗	✗	✗	✗	✗

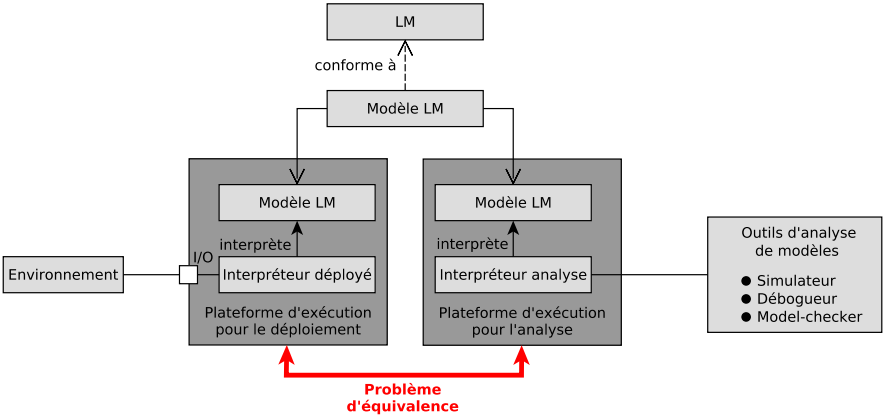
✗ problème présent ✓ problème absent





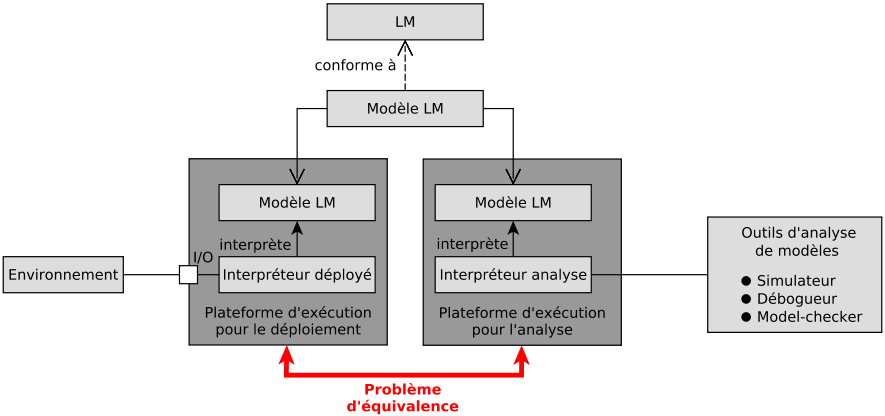


Question de recherche



AC#6 : Interprétations spécifiques pour vérification et exécution réelle

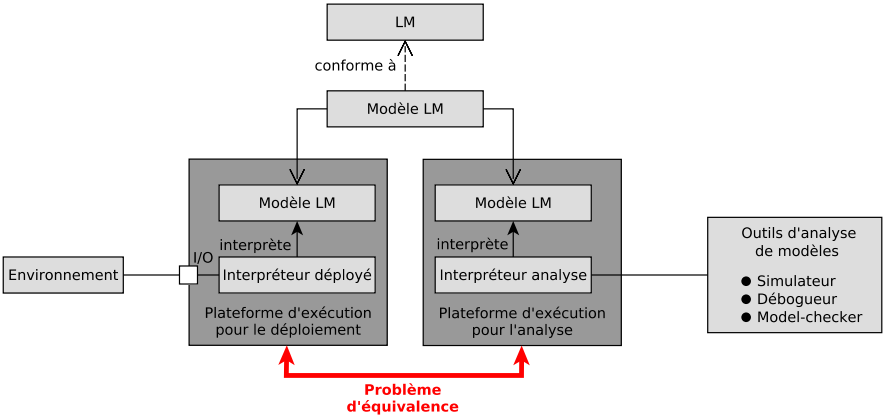
Question de recherche



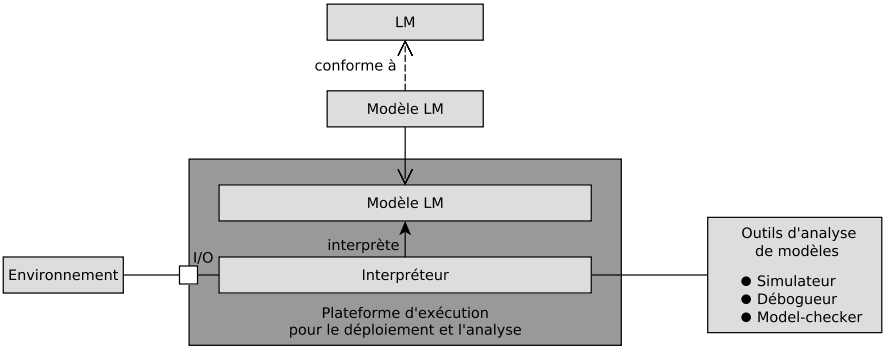
Question de recherche : Est-il possible d'unifier l'analyse et l'exécution embarquée de modèles ?

Sommaire

Aperçu de l'approche EMI



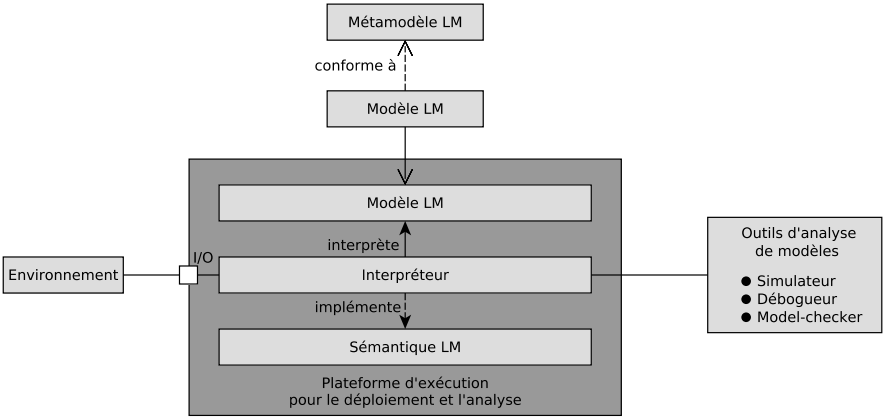
Aperçu de l'approche EMI



- Un seul couple (modèle + sémantique) pour les activités d'analyse et l'exécution embarquée
- Un seul et même environnement de développement pour appliquer toutes ces activités

$\Rightarrow$ **Unification des activités d'analyse et de l'exécution embarquée**

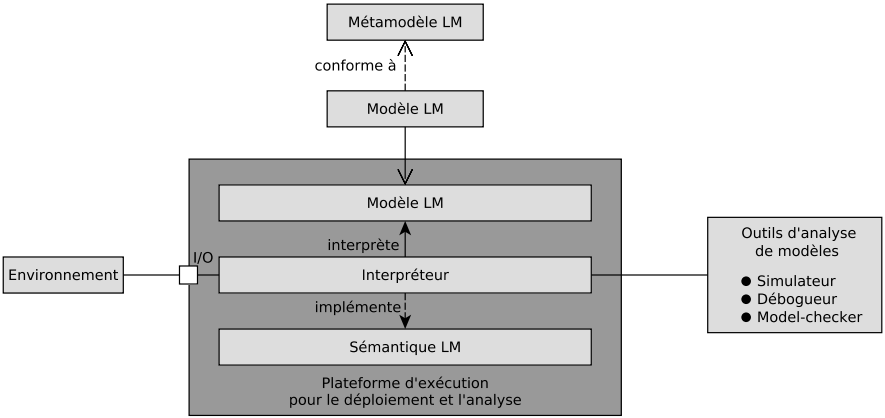
Aperçu de l'approche EMI



- Un seul couple (modèle + sémantique) pour les activités d'analyse et l'exécution embarquée
- Un seul et même environnement de développement pour appliquer toutes ces activités

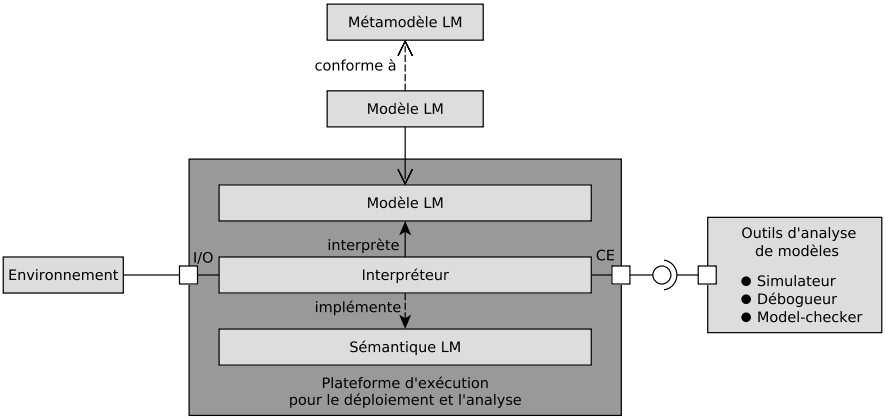
$\Rightarrow$ **Unification des activités d'analyse et de l'exécution embarquée**

Aperçu de l'approche EMI



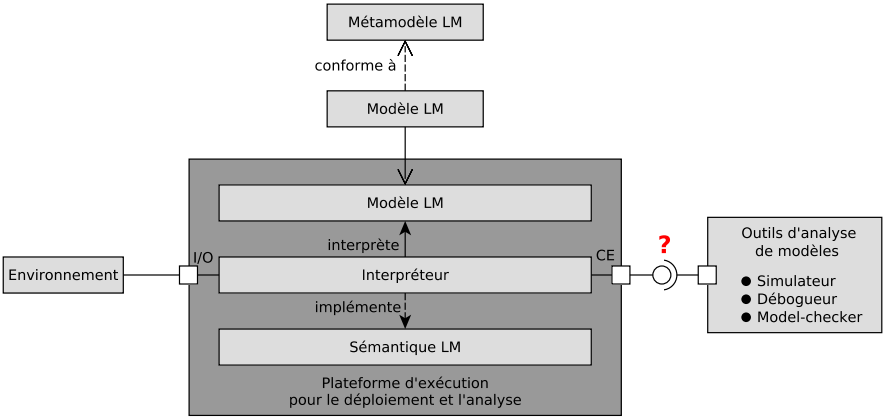
- Besoin de rendre l'interpréteur pilotable pour appliquer toutes ces activités de développement

Aperçu de l'approche EMI



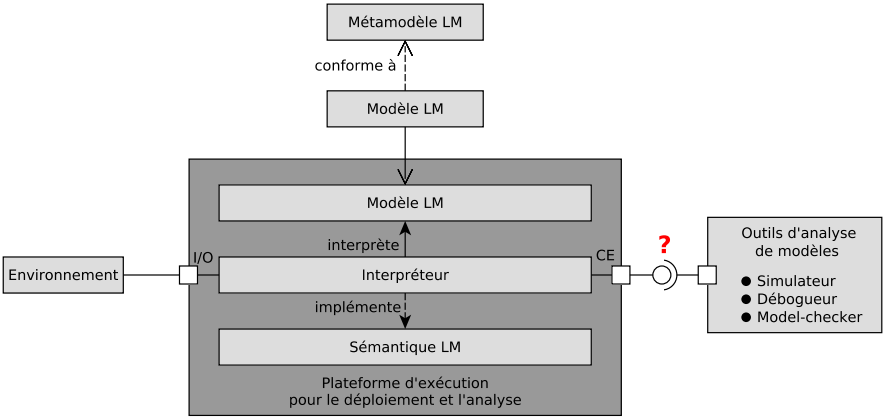
- Besoin de rendre l'interpréteur pilotable pour appliquer toutes ces activités de développement
- Besoin d'interfacer les outils d'analyse avec l'interpréteur

Aperçu de l'approche EMI



- Besoin de rendre l'interpréteur pilotable pour appliquer toutes ces activités de développement
- Besoin d'interfacer les outils d'analyse avec l'interpréteur

Aperçu de l'approche EMI



- Besoin de rendre l'interpréteur pilotable pour appliquer toutes ces activités de développement
- Besoin d'interfacer les outils d'analyse avec l'interpréteur

$\Rightarrow$ **Définir et formaliser les interfaces minimales pour répondre à ces besoins**

Mener diverses activités de développement

Activités de développement :

Exigences à satisfaire :

Mener diverses activités de développement

Activités de développement :

- **Simulation interactive et animation**

Exigences à satisfaire :

Définition

La simulation interactive et l'animation permettent d'explorer différentes traces d'exécution et de visualiser l'évolution des données d'exécution.

Mener diverses activités de développement

Activités de développement :

- **Simulation interactive et animation**

Exigences à satisfaire :

1. Piloter l'exécution du modèle

Définition

La simulation interactive et l'animation permettent d'explorer différentes traces d'exécution et de visualiser l'évolution des données d'exécution.

Mener diverses activités de développement

Activités de développement :

- **Simulation interactive et animation**

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle

Définition

La simulation interactive et l'animation permettent d'explorer différentes traces d'exécution et de visualiser l'évolution des données d'exécution.

Mener diverses activités de développement

Activités de développement :

- **Simulation interactive et animation**

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"

Définition

La simulation interactive et l'animation permettent d'explorer différentes traces d'exécution et de visualiser l'évolution des données d'exécution.

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- **Débogage (omniscient | multivers) ?**

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"

Définition

Le débogage permet de mettre en pause l'exécution et d'observer le contenu de l'état d'exécution courant pendant les pauses. [Bou+17]

[Bou+17] Bousse et al., « Omniscient debugging for executable DSLs », 2017

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- **Débogage (omniscient | multivers) ?**

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"

Définition

Le débogage **omniscient** introduit la possibilité de revenir en arrière sans avoir à redémarrer le moteur d'exécution. [Bou+17]

[Bou+17] Bousse et al., « Omniscient debugging for executable DSLs », 2017

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- **Débogage (omniscient | multivers) ?**

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"

Définition

Le débogage **multivers** permet de définir des points d'arrêts qui peuvent mettre en pause l'exécution du système dans différents chemins d'exécution, appelés univers. [Tor+19]

[Tor+19] Torres Lopez et al., « Multiverse Debugging : Non-deterministic Debugging for Non-deterministic Programs », 2019

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- **Débogage (omniscient | multivers) ?**

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système

Définition

Le débogage **multivers** permet de définir des points d'arrêts qui peuvent mettre en pause l'exécution du système dans différents chemins d'exécution, appelés univers. [Tor+19]

[Tor+19] Torres Lopez et al., « Multiverse Debugging : Non-deterministic Debugging for Non-deterministic Programs », 2019

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- Débogage (omniscient | multivers) ?
- **Détection de *deadlocks***

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système

Définition

La détection de *deadlocks* permet de rechercher des situations de blocage en explorant l'espace d'état du modèle.

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- Débogage (omniscient | multivers) ?
- Détection de *deadlocks*
- ***Model-checking***

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système

Définition

Étant donné un modèle d'états fini du système et une propriété formelle, le *model-checking* vérifie de façon exhaustive que la propriété est satisfaite pour ce modèle. [BK08]

[BK08] Baier et al., *Principles of Model Checking (Representation and Mind Series)*, 2008

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- Débogage (omniscient | multivers) ?
- Détection de *deadlocks*
- ***Model-checking***

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles

Définition

Étant donné un modèle d'états fini du système et une propriété formelle, le *model-checking* vérifie de façon exhaustive que la propriété est satisfaite pour ce modèle. [BK08]

[BK08] Baier et al., *Principles of Model Checking (Representation and Mind Series)*, 2008

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- Débogage (omniscient | multivers) ?
- Détection de *deadlocks*
- ***Model-checking***

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions

Définition

Étant donné un modèle d'états fini du système et une propriété formelle, le *model-checking* vérifie de façon exhaustive que la propriété est satisfaite pour ce modèle. [BK08]

[BK08] Baier et al., *Principles of Model Checking (Representation and Mind Series)*, 2008

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- Débogage (omniscient | multivers) ?
- Détection de *deadlocks*
- ***Model-checking***

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*

Définition

Étant donné un modèle d'états fini du système et une propriété formelle, le *model-checking* vérifie de façon exhaustive que la propriété est satisfaite pour ce modèle. [BK08]

[BK08] Baier et al., *Principles of Model Checking (Representation and Mind Series)*, 2008

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- Débogage (omniscient | multivers) ?
- Détection de *deadlocks*
- *Model-checking*
- **Bisimulation**

Définition

La bisimulation compare les espaces d'état de deux modèles pour vérifier que les deux systèmes se comportent de la même façon.

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- Débogage (omniscient | multivers) ?
- Détection de *deadlocks*
- *Model-checking*
- Bisimulation
- **Exécution réelle**

Définition

L'exécution réelle vise à remplir le besoin pour lequel le système a été conçu.

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- Débogage (omniscient | multivers) ?
- Détection de *deadlocks*
- *Model-checking*
- Bisimulation
- **Exécution réelle**

Définition

L'exécution réelle vise à remplir le besoin pour lequel le système a été conçu.

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- Débogage (omniscient | multivers) ?
- Détection de *deadlocks*
- *Model-checking*
- Bisimulation
- **Exécution réelle**

Définition

L'exécution réelle vise à remplir le besoin pour lequel le système a été conçu.

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- Débogage (omniscient | multivers) ?
- Détection de *deadlocks*
- *Model-checking*
- Bisimulation
- Exécution réelle
- ***Monitoring***

Définition

Le *monitoring* permet de surveiller l'exécution réelle du système afin de détecter des défaillances en ligne.

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système

Mener diverses activités de développement

Activités de développement :

- Simulation interactive et animation
- Débogage (omniscient | multivers) ?
- Détection de *deadlocks*
- *Model-checking*
- Bisimulation
- Exécution réelle
- ***Monitoring***

Définition

Le *monitoring* permet de surveiller l'exécution réelle du système afin de détecter des défaillances en ligne.

Exigences à satisfaire :

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système
10. Déployer des moniteurs sur une cible embarquée

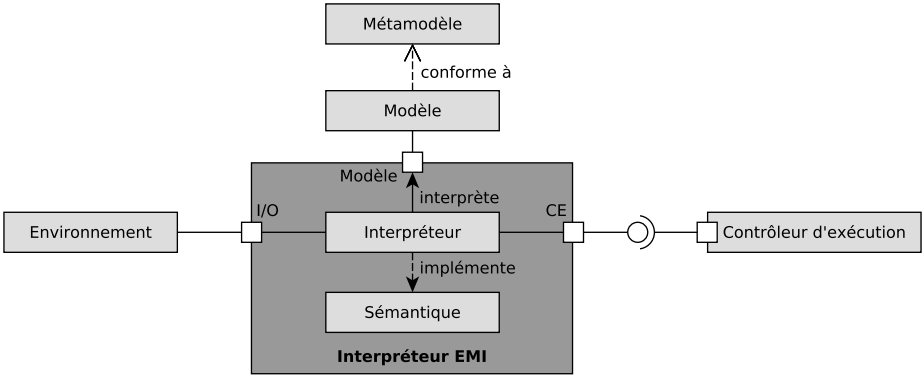
Exigences à satisfaire

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système
10. Déployer des moniteurs sur une cible embarquée

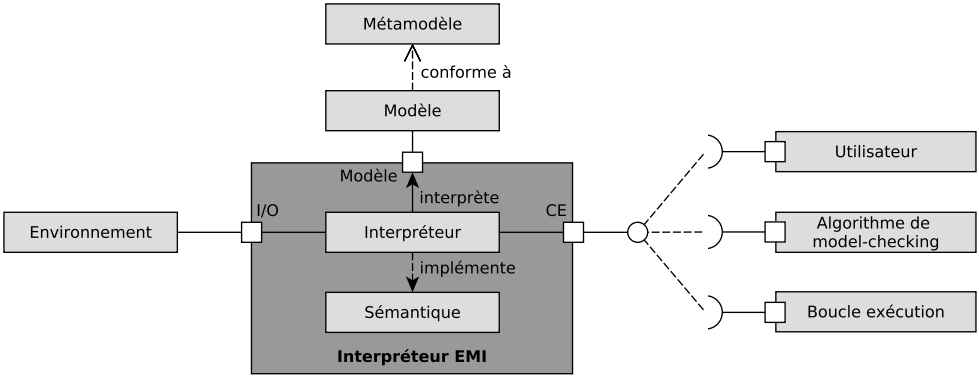
Exigences à satisfaire

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système
10. Déployer des moniteurs sur une cible embarquée

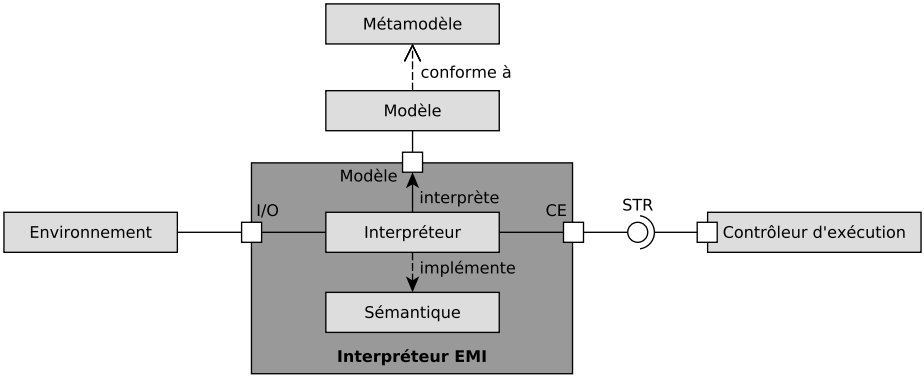
Piloter l'exécution du modèle



Piloter l'exécution du modèle



Piloter l'exécution du modèle



Piloter l'exécution du modèle

Interface de pilotage (STR) en Lean [Mou+15]

```
structure STR (C A : Type) :=
  (initial : set C)
  (actions : C → set A)
  (execute : C → A → set C)
```

[Mou+15] Moura et al., « The Lean Theorem Prover (System Description) », 2015

Piloter l'exécution du modèle

Interface de pilotage (STR) en Lean [Mou+15]

```
structure STR (C A : Type) :=
  (initial : set C)
  (actions : C → set A)
  (execute : C → A → set C)
```

Configuration (C)

Ensemble de données d'exécution à un instant donné

Action (A)

Représentation symbolique des pas d'exécution

[Mou+15] Moura et al., « The Lean Theorem Prover (System Description) », 2015

Piloter l'exécution du modèle

Interface de pilotage (STR) en Lean [Mou+15]

```
structure STR (C A : Type) :=
  (initial : set C)
  (actions : C → set A)
  (execute : C → A → set C)
```

Configuration (C)

Ensemble de données d'exécution à un instant donné

Action (A)

Représentation symbolique des pas d'exécution

[Mou+15] Moura et al., « The Lean Theorem Prover (System Description) », 2015

Piloter l'exécution du modèle

Interface de pilotage (STR) en Lean [Mou+15]

```
structure STR (C A : Type) :=
  (initial : set C)
  (actions : C → set A)
  (execute : C → A → set C)
```

Configuration (C)

Ensemble de données d'exécution à un instant donné

Action (A)

Représentation symbolique des pas d'exécution

[Mou+15] Moura et al., « The Lean Theorem Prover (System Description) », 2015

Piloter l'exécution du modèle

Interface de pilotage (STR) en Lean [Mou+15]

```
structure STR (C A : Type) :=
  (initial : set C)
  (actions : C → set A)
  (execute : C → A → set C)
```

Configuration (C)

Ensemble de données d'exécution à un instant donné

Action (A)

Représentation symbolique des pas d'exécution

[Mou+15] Moura et al., « The Lean Theorem Prover (System Description) », 2015

Piloter l'exécution du modèle

Interface de pilotage (STR) en Lean [Mou+15]

```
structure STR (C A : Type) :=
  (initial : set C)
  (actions : C → set A)
  (execute : C → A → set C)
```

Configuration (C)

Ensemble de données d'exécution à un instant donné

Action (A)

Représentation symbolique des pas d'exécution

[Mou+15] Moura et al., « The Lean Theorem Prover (System Description) », 2015

Piloter l'exécution du modèle

Interface de pilotage (STR) en Lean [Mou+15]

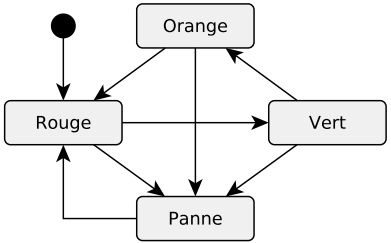
```
structure STR (C A : Type) :=
  (initial : set C)
  (actions : C → set A)
  (execute : C → A → set C)
```

Configuration (C)

Ensemble de données d'exécution à un instant donné

Action (A)

Représentation symbolique des pas d'exécution



[Mou+15] Moura et al., « The Lean Theorem Prover (System Description) », 2015

Piloter l'exécution du modèle

Interface de pilotage (STR) en Lean [Mou+15]

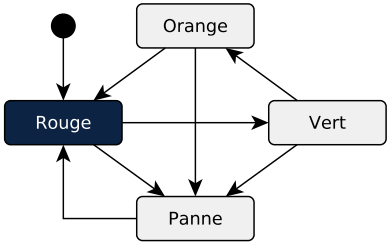
```
structure STR (C A : Type) :=
  (initial : set C)
  (actions : C → set A)
  (execute : C → A → set C)
```

Configuration (C)

Ensemble de données d'exécution à un instant donné

Action (A)

Représentation symbolique des pas d'exécution



1. `str.initial` $= \{\{\texttt{Rouge}\}\}$

[Mou+15] Moura et al., « The Lean Theorem Prover (System Description) », 2015

Piloter l'exécution du modèle

Interface de pilotage (STR) en Lean [Mou+15]

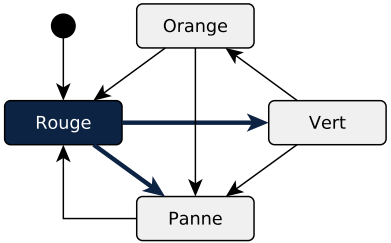
```
structure STR (C A : Type) :=
  (initial : set C)
  (actions : C → set A)
  (execute : C → A → set C)
```

Configuration (C)

Ensemble de données d'exécution à un instant donné

Action (A)

Représentation symbolique des pas d'exécution



1. `str.initial` $= \{\{\texttt{Rouge}\}\}$
2. `str.actions` $\{\texttt{Rouge}\} = \{\rightarrow_{\texttt{Vert}}, \rightarrow_{\texttt{Panne}}\}$

[Mou+15] Moura et al., « The Lean Theorem Prover (System Description) », 2015

Piloter l'exécution du modèle

Interface de pilotage (STR) en Lean [Mou+15]

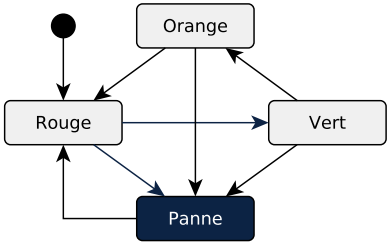
```
structure STR (C A : Type) :=
  (initial : set C)
  (actions : C → set A)
  (execute : C → A → set C)
```

Configuration (C)

Ensemble de données d'exécution à un instant donné

Action (A)

Représentation symbolique des pas d'exécution



1. $\texttt{str.initial} = \{\{\texttt{Rouge}\}\}$
2. $\texttt{str.actions}\ \{\texttt{Rouge}\} = \{\rightarrow_{\texttt{Vert}}, \rightarrow_{\texttt{Panne}}\}$
3. $\texttt{str.execute}\ \{\texttt{Rouge}\} \rightarrow_{\texttt{Panne}} = \{\{\texttt{Panne}\}\}$

[Mou+15] Moura et al., « The Lean Theorem Prover (System Description) », 2015

Piloter l'exécution du modèle

Interface de pilotage (STR) en Lean [Mou+15]

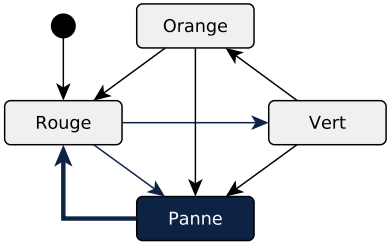
```
structure STR (C A : Type) :=
  (initial : set C)
  (actions : C → set A)
  (execute : C → A → set C)
```

Configuration (C)

Ensemble de données d'exécution à un instant donné

Action (A)

Représentation symbolique des pas d'exécution



1. $\texttt{str.initial} = \{\{\texttt{Rouge}\}\}$
2. $\texttt{str.actions}\ \{\texttt{Rouge}\} = \{\rightarrow_{\texttt{Vert}}, \rightarrow_{\texttt{Panne}}\}$
3. $\texttt{str.execute}\ \{\texttt{Rouge}\}\ \rightarrow_{\texttt{Panne}} = \{\{\texttt{Panne}\}\}$
4. $\texttt{str.actions}\ \{\texttt{Panne}\} = \{\rightarrow_{\texttt{Rouge}}\}$

[Mou+15] Moura et al., « The Lean Theorem Prover (System Description) », 2015

Piloter l'exécution du modèle

Interface de pilotage (STR) en Lean [Mou+15]

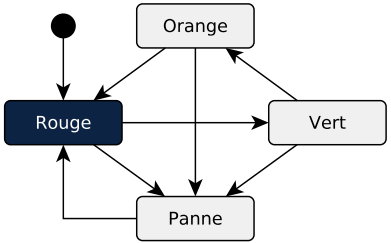
```
structure STR (C A : Type) :=
  (initial : set C)
  (actions : C → set A)
  (execute : C → A → set C)
```

Configuration (C)

Ensemble de données d'exécution à un instant donné

Action (A)

Représentation symbolique des pas d'exécution



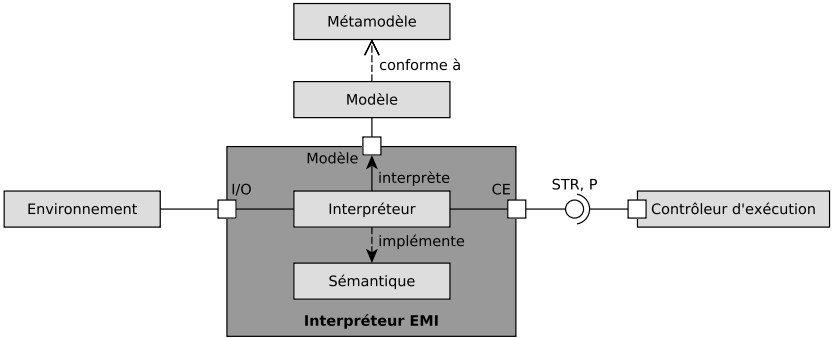
1. $\texttt{str.initial} = \{\{\texttt{Rouge}\}\}$
2. $\texttt{str.actions}\ \{\texttt{Rouge}\} = \{\rightarrow_{\texttt{Vert}}, \rightarrow_{\texttt{Panne}}\}$
3. $\texttt{str.execute}\ \{\texttt{Rouge}\}\ \rightarrow_{\texttt{Panne}} = \{\{\texttt{Panne}\}\}$
4. $\texttt{str.actions}\ \{\texttt{Panne}\} = \{\rightarrow_{\texttt{Rouge}}\}$
5. $\texttt{str.execute}\ \{\texttt{Panne}\}\ \rightarrow_{\texttt{Rouge}} = \{\{\texttt{Rouge}\}\}$

[Mou+15] Moura et al., « The Lean Theorem Prover (System Description) », 2015

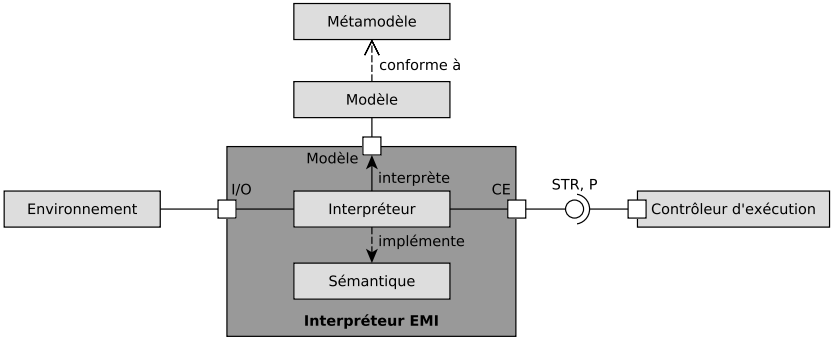
Exigences à satisfaire

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système
10. Déployer des moniteurs sur une cible embarquée

Visualiser l'exécution du modèle



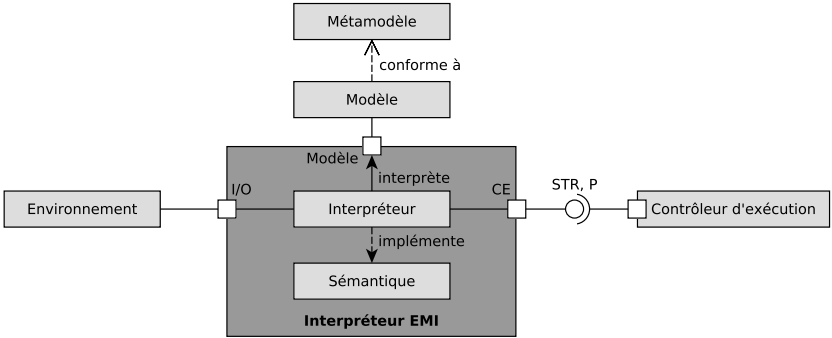
Visualiser l'exécution du modèle



Fonctions de projection

Fournissent une projection de la configuration et des actions **dans les termes du langage de modélisation**

Visualiser l'exécution du modèle



Interface de projection (P)

```
structure P (C A : Type) :=
  (projectC: C → Vc)
  (projectA: A → Va)
```

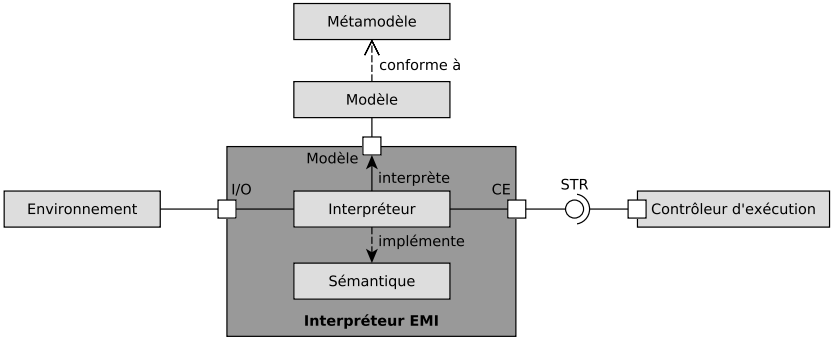
Vues pour la projection

- `Vc` : la vue d'une configuration
- `Va` : la vue d'une action

Exigences à satisfaire

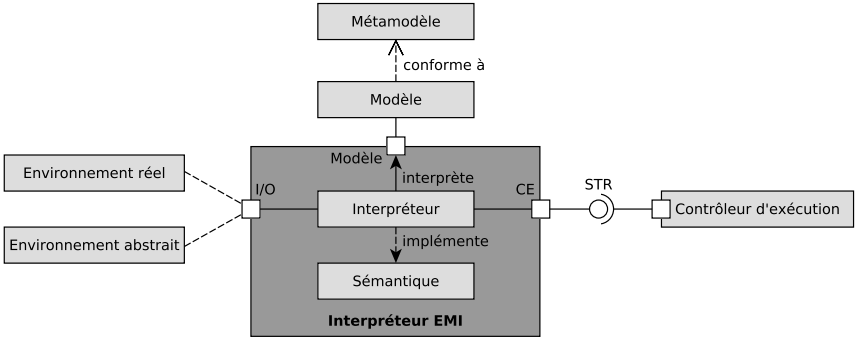
1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système
10. Déployer des moniteurs sur une cible embarquée

Connecter un environnement



[Bes+20] Besnard et al., « Modular Deployment of UML Models for V&V Activities and Embedded Execution », 2020

Connecter un environnement



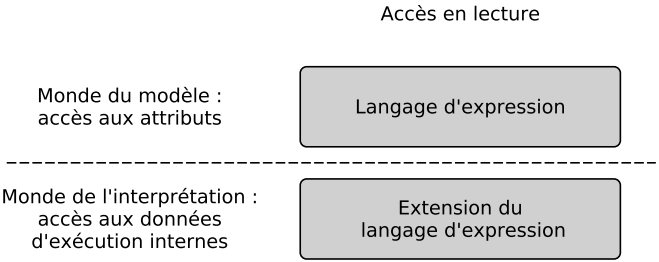
- Utilisation du port I/O pour faire interagir le modèle du système avec différents environnements
- Utilisation du même modèle du système pour toutes les activités → seul l'environnement change

[Bes+20] Besnard et al., « Modular Deployment of UML Models for V&V Activities and Embedded Execution », 2020

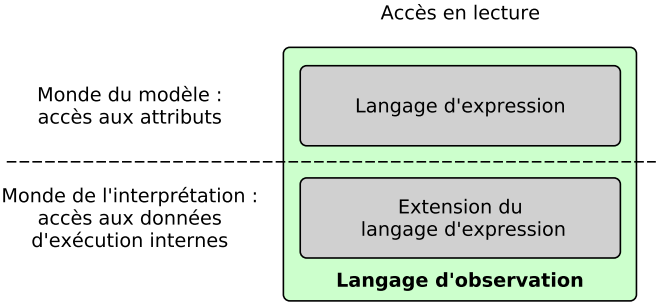
Exigences à satisfaire

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système
10. Déployer des moniteurs sur une cible embarquée

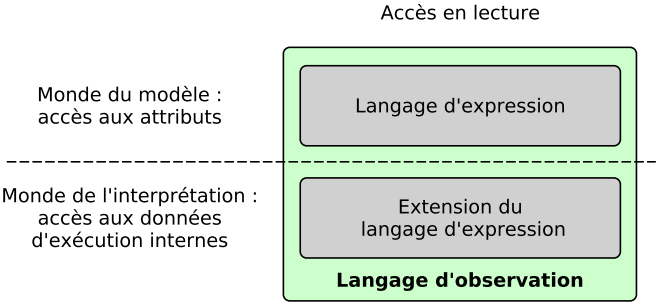
Questionner l'exécution du système



Questionner l'exécution du système



Questionner l'exécution du système



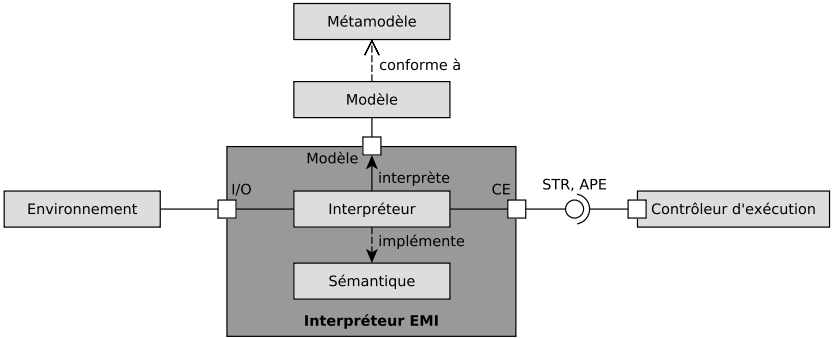
Langage d'observation

Langage sans effet de bord **utilisant les concepts du langage de modélisation**

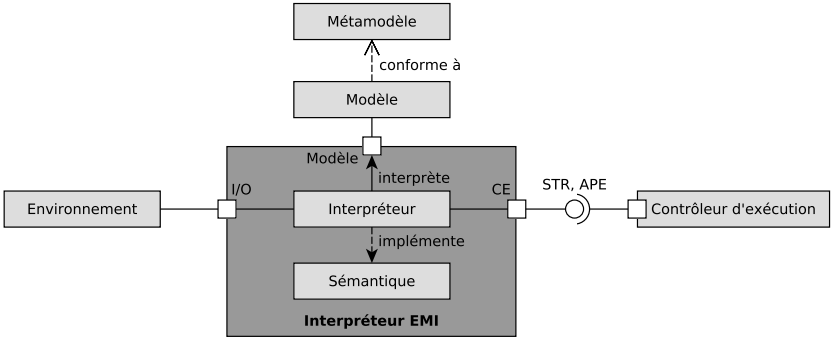
Propositions atomiques

Prédicats permettant de questionner l'exécution du système

Questionner l'exécution du système



Questionner l'exécution du système

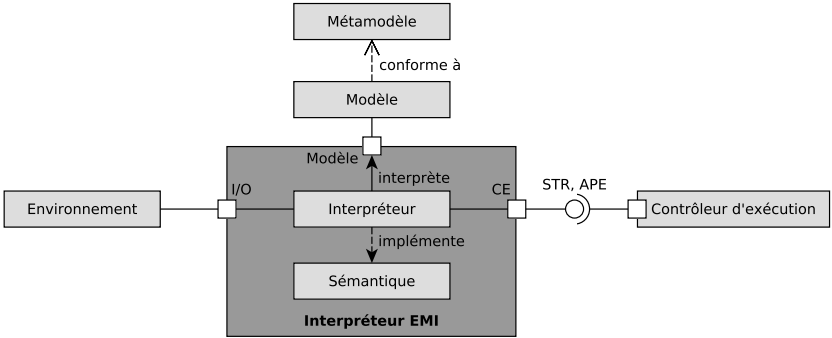


Interface d'évaluation des propositions atomiques (APE)

```
structure APE (C A L : Type) :=
  (eval : L → C → A → C → bool)
```

Avec L le type des propositions atomiques

Questionner l'exécution du système



Interface d'évaluation des propositions atomiques (APE)

```
structure APE (C A L : Type) :=
  (eval : L → C → A → C → bool)
```

Avec L le type des propositions atomiques

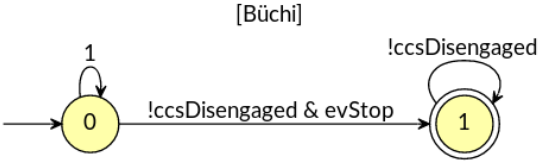
Exigences à satisfaire

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système
10. Déployer des moniteurs sur une cible embarquée

Spécifier des propriétés formelles

Deux types d'automates

- Les automates de Büchi [BK08]
  - Traces d'exécution infinies
  - Condition d'acceptation : détecter une boucle qui passe une infinité de fois par au moins un état d'acceptation (c.-à-d. un état dans lequel on a reconnu une séquence d'évènements donnée).
- Les automates observateurs [OGO06]
  - Traces d'exécution finies
  - Condition d'acceptation : atteindre un état d'acceptation au moins une fois.



[BK08] Baier et al., *Principles of Model Checking (Representation and Mind Series)*, 2008

[OGO06] Ober et al., « Validating timed UML models by simulation and verification », 2006

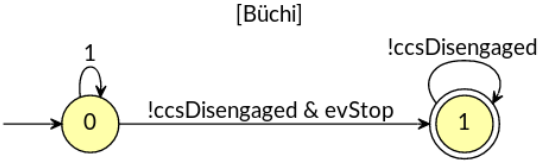
Spécifier des propriétés formelles

Deux types d'automates

- Les automates de Büchi [BK08]
  - Traces d'exécution infinies
  - Condition d'acceptation : détecter une boucle qui passe une infinité de fois par au moins un état d'acceptation (c.-à-d. un état dans lequel on a reconnu une séquence d'évènements donnée).
- Les automates observateurs [OGO06]
  - Traces d'exécution finies
  - Condition d'acceptation : atteindre un état d'acceptation au moins une fois.

Interface permettant de connaitre les états d'acceptation (A_{cc})

```
structure Acc (C : Type) :=
   (accepting : set C)
```



[BK08] Baier et al., *Principles of Model Checking (Representation and Mind Series)*, 2008

[OGO06] Ober et al., « Validating timed UML models by simulation and verification », 2006

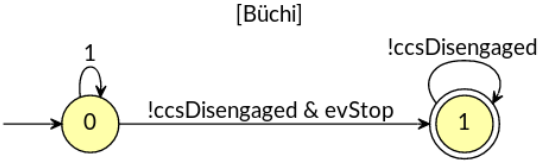
Spécifier des propriétés formelles

Deux types d'automates

- Les automates de Büchi [BK08]
  - Traces d'exécution infinies
  - Condition d'acceptation : détecter une boucle qui passe une infinité de fois par au moins un état d'acceptation (c.-à-d. un état dans lequel on a reconnu une séquence d'évènements donnée).
- Les automates observateurs [OGO06]
  - Traces d'exécution finies
  - Condition d'acceptation : atteindre un état d'acceptation au moins une fois.

Interface permettant de récupérer les propositions atomiques (APC)

```
structure APC (C A L : Type) :=
   (eval : C → A → set L)
```



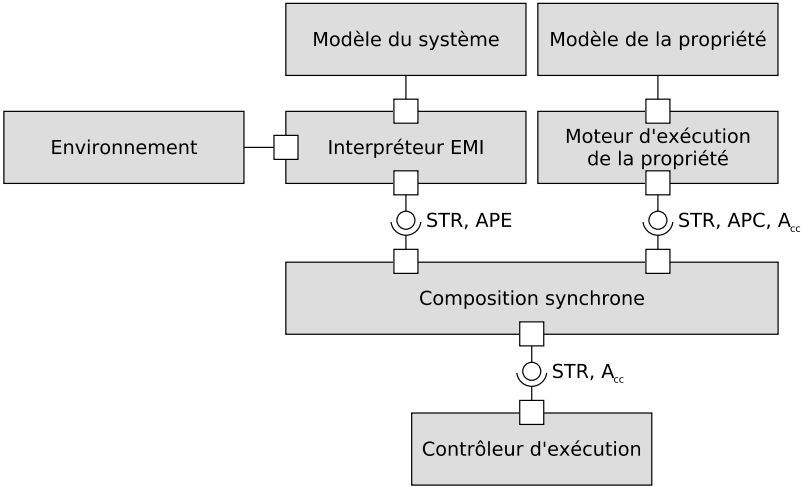
[BK08] Baier et al., *Principles of Model Checking (Representation and Mind Series)*, 2008

[OGO06] Ober et al., « Validating timed UML models by simulation and verification », 2006

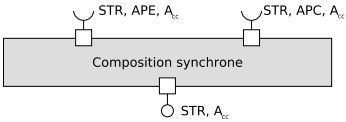
Exigences à satisfaire

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système
10. Déployer des moniteurs sur une cible embarquée

Composer des exécutions



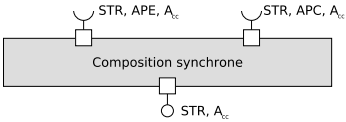
Composer des exécutions — Formalisation en Lean




```
def synchronous_composition (C₁ C₂ A₁ A₂ L₁ : Type)
   (lhs : STR C₁ A₁) (ape : APE C₁ A₁ L₁)
   (rhs : STR C₂ A₂) (apc : APC C₂ A₂ L₁)
: STR (C₁ × C₂) (A₁ × A₂) := {

initial := { c | ∀ (c₁ ∈ lhs.initial) (c₂ ∈ rhs.initial), c = (c₁, c₂) },

actions := λ c, { a | match c with
   | (c₁, c₂) := ∀ (a₁ ∈ lhs.actions c₁) (a₂ ∈ rhs.actions c₂)
      (t₁ ∈ lhs.execute c₁ a₁) (t₂ ∈ rhs.execute c₂ a₂),
      match t₁, t₂ : ∀ t₁ t₂, Prop with
         | t₁, t₂ := ∀ l ∈ apc.eval c₂ a₂,
            ape.eval l c₁ a₁ t₁ = tt → a = (a₁, a₂)
   end end },

execute := λ c a, { t | match c, a with
   | (c₁, c₂), (a₁, a₂) :=  ∀ (t₁ ∈ lhs.execute c₁ a₁)
      (t₂ ∈ rhs.execute c₂ a₂), t = (t₁, t₂)
   end }}
```

Composer des exécutions — Formalisation en Lean

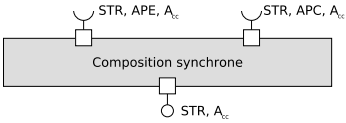
```
def synchronous_composition (C₁ C₂ A₁ A₂ L₁ : Type)
   (lhs : STR C₁ A₁) (ape : APE C₁ A₁ L₁)
   (rhs : STR C₂ A₂) (apc : APC C₂ A₂ L₁)
: STR (C₁ × C₂) (A₁ × A₂) := {
```




```
initial := { c | ∀ (c₁ ∈ lhs.initial) (c₂ ∈ rhs.initial), c = (c₁, c₂) },

actions := λ c, { a | match c with
   | (c₁, c₂) := ∀ (a₁ ∈ lhs.actions c₁) (a₂ ∈ rhs.actions c₂)
      (t₁ ∈ lhs.execute c₁ a₁) (t₂ ∈ rhs.execute c₂ a₂),
      match t₁, t₂ : ∀ t₁ t₂, Prop with
         | t₁, t₂ := ∀ l ∈ apc.eval c₂ a₂,
            ape.eval l c₁ a₁ t₁ = tt → a = (a₁, a₂)
   end end },

execute := λ c a, { t | match c, a with
   | (c₁, c₂), (a₁, a₂) :=  ∀ (t₁ ∈ lhs.execute c₁ a₁)
      (t₂ ∈ rhs.execute c₂ a₂), t = (t₁, t₂)
   end }}
```

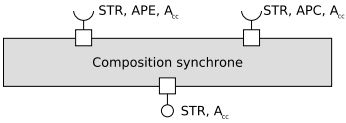
Composer des exécutions — Formalisation en Lean

```
def synchronous_composition (C₁ C₂ A₁ A₂ L₁ : Type)
   (lhs : STR C₁ A₁) (ape : APE C₁ A₁ L₁)
   (rhs : STR C₂ A₂) (apc : APC C₂ A₂ L₁)
: STR (C₁ × C₂) (A₁ × A₂) := {
```



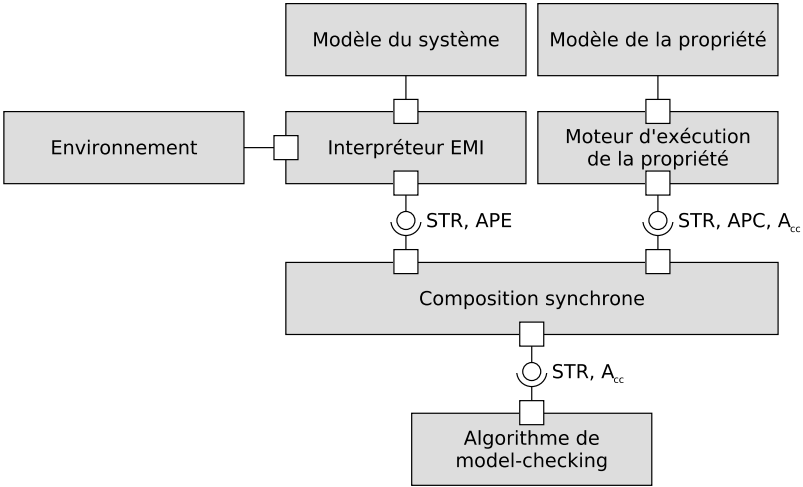

```
initial := { c | ∀ (c₁ ∈ lhs.initial) (c₂ ∈ rhs.initial), c = (c₁, c₂) },

actions := λ c, { a | match c with
   | (c₁, c₂) := ∀ (a₁ ∈ lhs.actions c₁) (a₂ ∈ rhs.actions c₂)
      (t₁ ∈ lhs.execute c₁ a₁) (t₂ ∈ rhs.execute c₂ a₂),
      match t₁, t₂ : ∀ t₁ t₂, Prop with
         | t₁, t₂ := ∀ l ∈ apc.eval c₂ a₂,
            ape.eval l c₁ a₁ t₁ = tt → a = (a₁, a₂)
   end end },

execute := λ c a, { t | match c, a with
   | (c₁, c₂), (a₁, a₂) :=  ∀ (t₁ ∈ lhs.execute c₁ a₁)
      (t₂ ∈ rhs.execute c₂ a₂), t = (t₁, t₂)
   end }}
```

Composer des exécutions — Formalisation en Lean

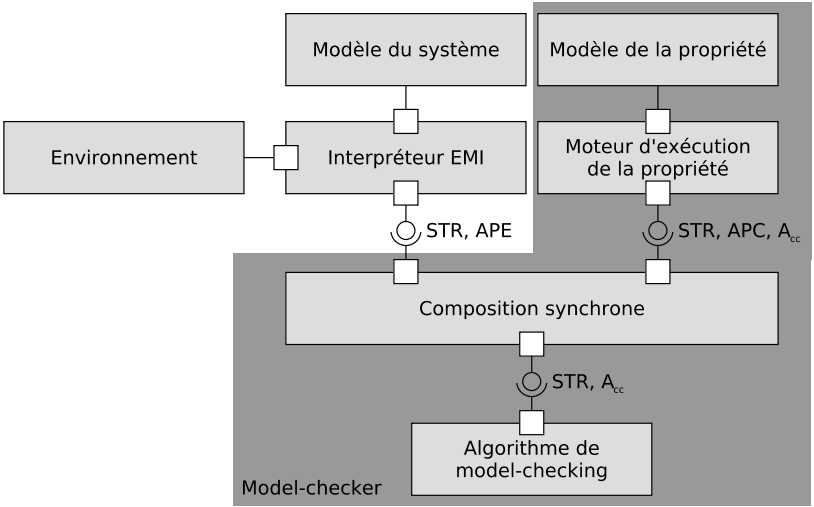



```
def synchronous_composition (C₁ C₂ A₁ A₂ L₁ : Type)
   (lhs : STR C₁ A₁) (ape : APE C₁ A₁ L₁)
   (rhs : STR C₂ A₂) (apc : APC C₂ A₂ L₁)
: STR (C₁ × C₂) (A₁ × A₂) := {

initial := { c | ∀ (c₁ ∈ lhs.initial) (c₂ ∈ rhs.initial), c = (c₁, c₂) },

actions := λ c, { a | match c with
   | (c₁, c₂) := ∀ (a₁ ∈ lhs.actions c₁) (a₂ ∈ rhs.actions c₂)
      (t₁ ∈ lhs.execute c₁ a₁) (t₂ ∈ rhs.execute c₂ a₂),
      match t₁, t₂ : ∀ t₁ t₂, Prop with
         | t₁, t₂ := ∀ l ∈ apc.eval c₂ a₂,
            ape.eval l c₁ a₁ t₁ = tt → a = (a₁, a₂)
   end end },

execute := λ c a, { t | match c, a with
   | (c₁, c₂), (a₁, a₂) :=  ∀ (t₁ ∈ lhs.execute c₁ a₁)
      (t₂ ∈ rhs.execute c₂ a₂), t = (t₁, t₂)
   end }}
```

Piloter l'exécution avec des algorithmes de *model-checking*



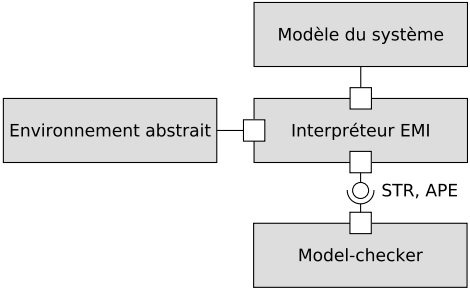
Piloter l'exécution avec des algorithmes de *model-checking*



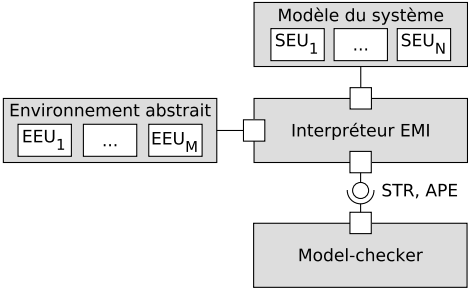
Exigences à satisfaire

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système
10. Déployer des moniteurs sur une cible embarquée

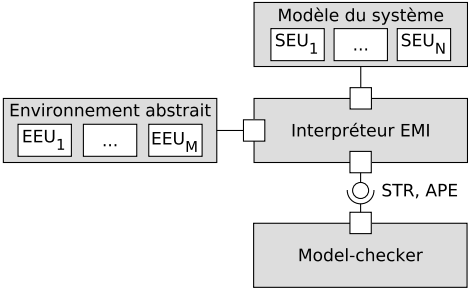
Prendre en compte des hypothèses d'analyse



Prendre en compte des hypothèses d'analyse



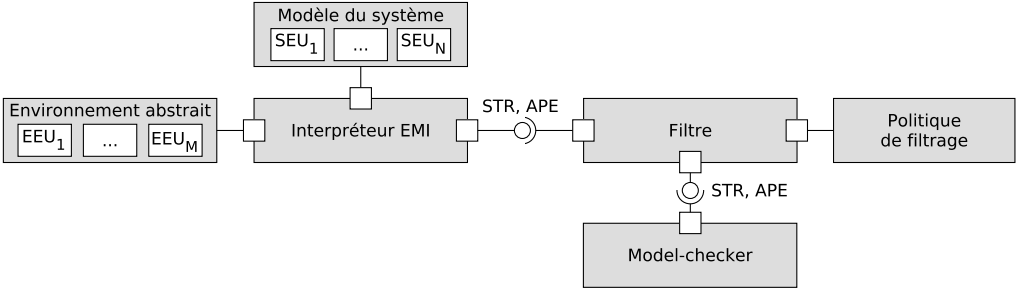
Prendre en compte des hypothèses d'analyse



Exemples d'hypothèse d'analyse

- Hypothèse de réactivité : l'exécution du système est infiniment plus rapide que les réactions de l'environnement → privilégie l'exécution des actions du système
- Hypothèse visant à supprimer les débordements sur les files de messages

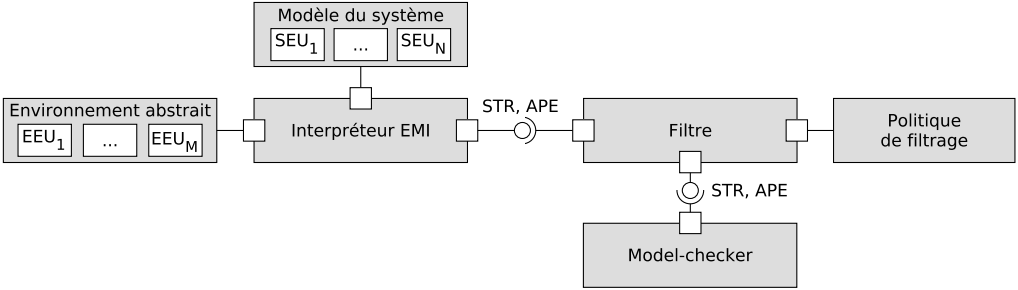
Prendre en compte des hypothèses d'analyse



Exemples d'hypothèse d'analyse

- Hypothèse de réactivité : l'exécution du système est infiniment plus rapide que les réactions de l'environnement → privilégie l'exécution des actions du système
- Hypothèse visant à supprimer les débordements sur les files de messages

Prendre en compte des hypothèses d'analyse



Politique de filtrage (avec S le type de l'état d'exécution de la politique de filtrage)

```
structure FilteringPolicy (C A S : Type) :=
  (initial : S)
  (selector : set A)
  (apply : S → C → set A → set (S × A))
  (subset : ∀ s c 𝔸 (sa ∈ (apply s c 𝔸)), prod.snd sa ∈ 𝔸)
```

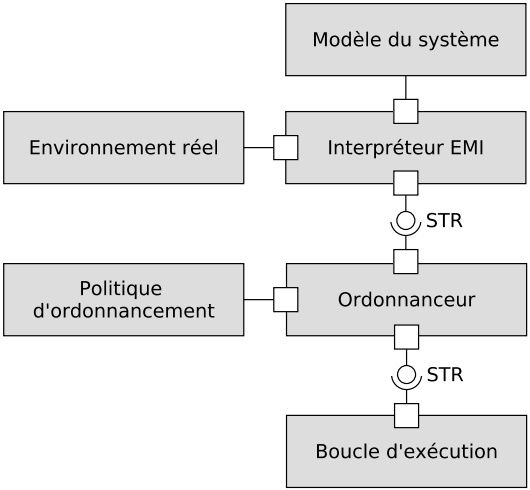
Exigences à satisfaire

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système
10. Déployer des moniteurs sur une cible embarquée

Déployer et ordonnancer l'exécution sur une cible embarquée

Boucle d'exécution

1. Récupère le prochain pas d'exécution via l'ordonnanceur (`str.actions`)
2. Exécute ce pas d'exécution (`str.execute`)



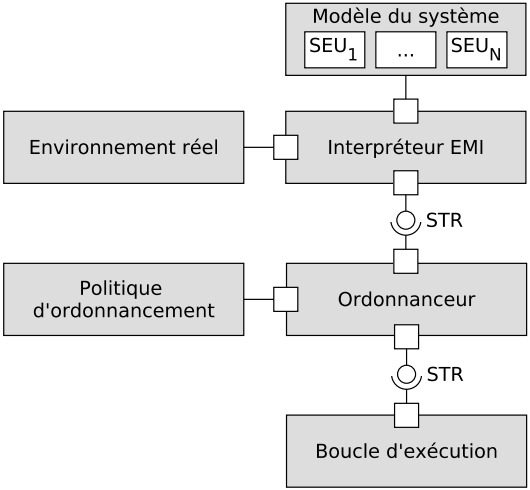
Déployer et ordonnancer l'exécution sur une cible embarquée

Boucle d'exécution

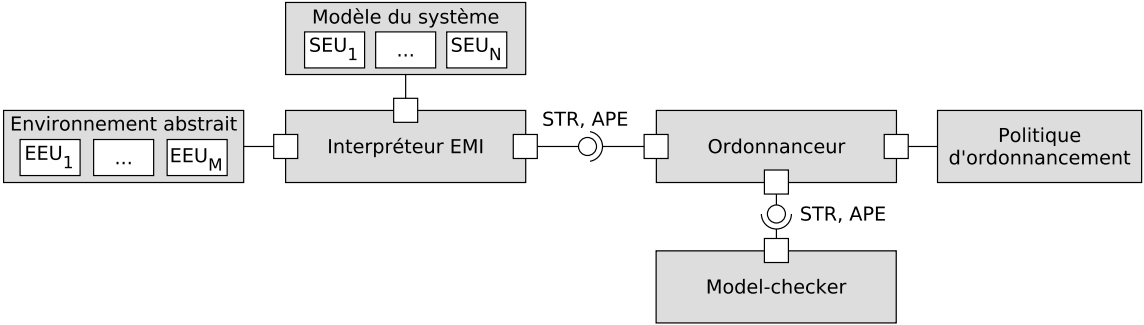
1. Récupère le prochain pas d'exécution via l'ordonnanceur (`str.actions`)
2. Exécute ce pas d'exécution (`str.execute`)

Ordonnancer l'exécution du système

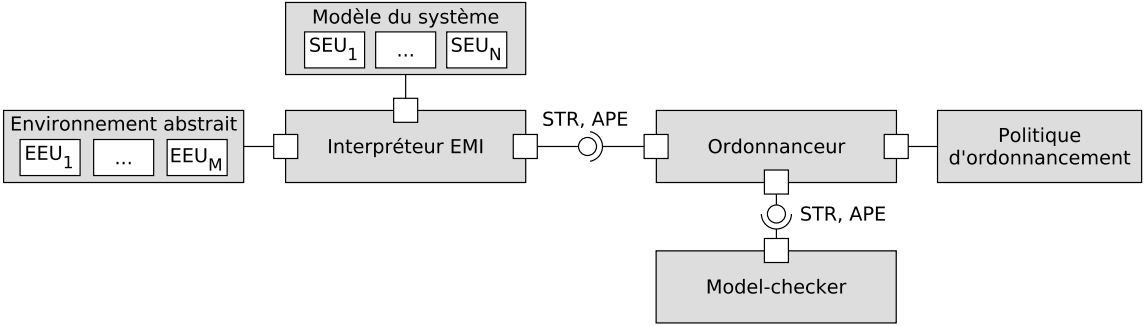
- Différentes unités d'exécution s'exécutent de manière concurrente → non-déterminisme
- Choix d'un seul chemin d'exécution avec l'ordonnanceur



Inclure l'ordonnanceur dans la boucle de vérification



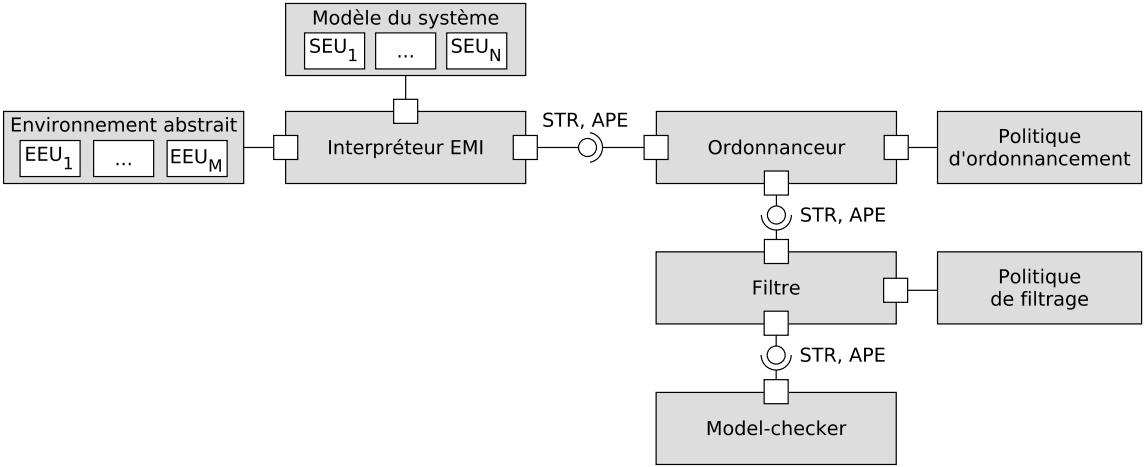
Inclure l'ordonnanceur dans la boucle de vérification



Politique d'ordonnancement

```
structure UnicoreSchedulingPolicy (C A S : Type)
  extends (FilteringPolicy C A S) :=
  (unique : ∀ s c 𝔸 (a ∈ (apply s c 𝔸)) (b ∈ (apply s c 𝔸)), a = b)
```

Inclure l'ordonnanceur dans la boucle de vérification



Exigences à satisfaire

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système
10. Déployer des moniteurs sur une cible embarquée

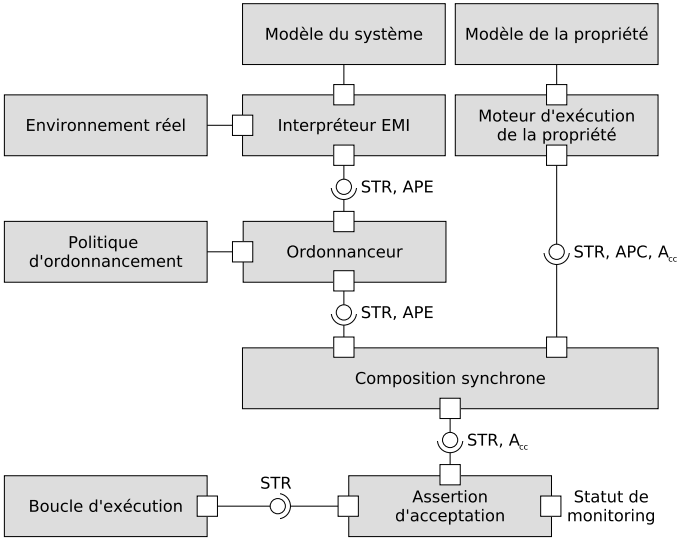
Déployer les moniteurs sur une cible embarquée

Caractéristiques des moniteurs

- Automates observateurs déterministes et complets
- Utilisation des mêmes automates qu'en *model-checking*
- Observation de la trace d'exécution courante

Déploiement des moniteurs

- Sans génération de code
- Sans instrumentation de code



[Bes+19a] Besnard et al., « Verifying and Monitoring UML Models with Observer Automata : A Transformation-Free Approach », 2019

Exigences à satisfaire

1. Piloter l'exécution du modèle
2. Visualiser l'exécution du modèle
3. Connecter un "environnement"
4. Questionner l'exécution du système
5. Spécifier des propriétés formelles
6. Composer des exécutions
7. Prendre en compte des hypothèse d'analyse pour le *model-checking*
8. Déployer l'interpréteur sur une cible embarquée
9. Ordonnancer l'exécution du système
10. Déployer des moniteurs sur une cible embarquée

Sommaire

Expérimentations

V#1 Effort nécessaire pour concevoir un interpréteur pilotable et embarquable □

Expérimentations

V#1 Effort nécessaire pour concevoir un interpréteur pilotable et embarquable □

V#2 Généricité de l'interface de contrôle d'exécution □

Expérimentations

V#1 Effort nécessaire pour concevoir un interpréteur pilotable et embarquable ☐

V#2 Généricité de l'interface de contrôle d'exécution ☐

V#3 Facilité de l'analyse de l'exécution de modèles ☐

Expérimentations

V#1 Effort nécessaire pour concevoir un interpréteur pilotable et embarquable

- Concevoir différents interpréteurs conformes à l'approche EMI
  - Implémenter l'interface de contrôle d'exécution
  - Définir une sémantique pilotable
  - Déployer l'interpréteur sur une cible embarquée

V#2 Généricité de l'interface de contrôle d'exécution

V#3 Facilité de l'analyse de l'exécution de modèles

1. Concevoir des interpréteurs

EMI-UML : interpréteur embarqué de modèles UML

- Basé sur un sous-ensemble d'UML :
  - Classes
  - Structures composites
  - Machines à états

1. Concevoir des interpréteurs

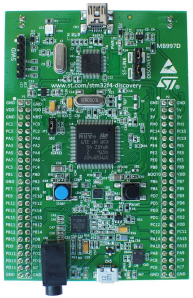
EMI-UML : interpréteur embarqué de modèles UML

- Basé sur un sous-ensemble d'UML :
  - Classes
  - Structures composites
  - Machines à états
- Utilisation du langage C comme langage hôte

1. Concevoir des interpréteurs

EMI-UML : interpréteur embarqué de modèles UML

- Basé sur un sous-ensemble d'UML :
  - Classes
  - Structures composites
  - Machines à états
- Utilisation du langage C comme langage hôte
- Déploiement sur :
  - PC de développement (Linux)
  - Cible embarquée en *bare-metal* (sans OS)



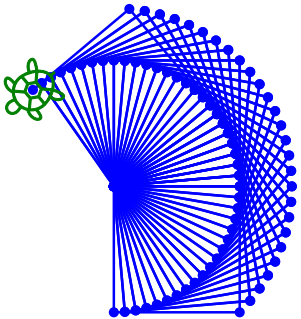
Cible embarquée STM32

[Bes+19b] Besnard et al., « EMI : Un Interpréteur de Modèles Embarqué pour l'Exécution et la Vérification de Modèles UML », 2019
[Bes+17] Besnard et al., « Towards one Model Interpreter for Both Design and Deployment », 2017

1. Concevoir des interpréteurs

EMI-UML : interpréteur embarqué de modèles UML

- Basé sur un sous-ensemble d'UML :
  - Classes
  - Structures composites
  - Machines à états
- Utilisation du langage C comme langage hôte
- Déploiement sur :
  - PC de développement (Linux)
  - Cible embarquée en *bare-metal* (sans OS)



Et d'autres interpréteurs...

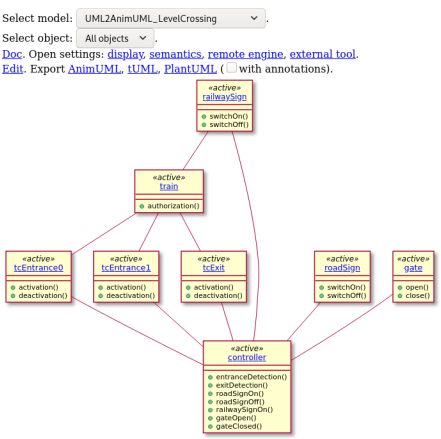
- EMI-LOGO : interpréteur de modèles LOGO

[Jou+20] Jouault et al., « Designing, Animating, and Verifying Partial UML Models », 2020
[Bes+19b] Besnard et al., « EMI : Un Interpréteur de Modèles Embarqué pour l'Exécution et la Vérification de Modèles UML », 2019
[Bes+17] Besnard et al., « Towards one Model Interpreter for Both Design and Deployment », 2017

1. Concevoir des interpréteurs

EMI-UML : interpréteur embarqué de modèles UML

- Basé sur un sous-ensemble d'UML :
  - Classes
  - Structures composites
  - Machines à états
- Utilisation du langage C comme langage hôte
- Déploiement sur :
  - PC de développement (Linux)
  - Cible embarquée en *bare-metal* (sans OS)



Et d'autres interpréteurs...

- EMI-LOGO : interpréteur de modèles LOGO
- AnimUML : interpréteur Web de modèles UML [Jou+20]

[Jou+20] Jouault et al., « Designing, Animating, and Verifying Partial UML Models », 2020
[Bes+19b] Besnard et al., « EMI : Un Interpréteur de Modèles Embarqué pour l'Exécution et la Vérification de Modèles UML », 2019
[Bes+17] Besnard et al., « Towards one Model Interpreter for Both Design and Deployment », 2017

1. Concevoir des interpréteurs

Plusieurs cas d'études pour EMI-UML

- Une interface d'un régulateur de vitesse [Bes+19a]
- Un contrôleur de passage à niveau [Bes+18]
- Un train d'atterrissage [BW14]
- Un défibrillateur cardiaque [FDD17]
- Un robot joueur de football [Bes+19c]

Utilisation d'EMI-UML et d'AnimUML par des étudiants-ingénieurs en travaux dirigés et en projets

[Bes+19a] Besnard et al., « Verifying and Monitoring UML Models with Observer Automata : A Transformation-Free Approach », 2019
[Bes+18] Besnard et al., « Unified LTL Verification and Embedded Execution of UML Models », 2018
[BW14] Boniol et al., « The Landing Gear System Case Study », 2014
[FDD17] Ferretti et al., « Open-source automated external defibrillator », 2017 (`https://github.com/Pyponou/defibrillator`)
[Bes+19c] Besnard et al., « A Model Checkable UML Soccer Player », 2019

1. Concevoir des interpréteurs

Plusieurs cas d'études pour EMI-UML

- Une interface d'un régulateur de vitesse [Bes+19a] $-----\rightarrow$ automobile
- Un contrôleur de passage à niveau [Bes+18] $---------\rightarrow$ ferroviaire
- Un train d'atterrissage [BW14] $-----------------\rightarrow$ aéronautique
- Un défibrillateur cardiaque [FDD17] $--------------\rightarrow$ santé
- Un robot joueur de football [Bes+19c] $--------------\rightarrow$ robotique

Utilisation d'EMI-UML et d'AnimUML par des étudiants-ingénieurs en travaux dirigés et en projets

[Bes+19a] Besnard et al., « Verifying and Monitoring UML Models with Observer Automata : A Transformation-Free Approach », 2019
[Bes+18] Besnard et al., « Unified LTL Verification and Embedded Execution of UML Models », 2018
[BW14] Boniol et al., « The Landing Gear System Case Study », 2014
[FDD17] Ferretti et al., « Open-source automated external defibrillator », 2017 (`https://github.com/Pyponou/defibrillator`)
[Bes+19c] Besnard et al., « A Model Checkable UML Soccer Player », 2019

Expérimentations

V#1 Effort nécessaire pour concevoir un interpréteur pilotable et embarquable

- Concevoir différents interpréteurs conformes à l'approche EMI
  - Implémenter l'interface de contrôle d'exécution
  - Définir une sémantique pilotable
  - Déployer l'interpréteur sur une cible embarquée

V#2 Généricité de l'interface de contrôle d'exécution

V#3 Facilité de l'analyse de l'exécution de modèles

Expérimentations

V#1 Effort nécessaire pour concevoir un interpréteur pilotable et embarquable ☑

- Concevoir différents interpréteurs conformes à l'approche EMI
  - Implémenter l'interface de contrôle d'exécution
  - Définir une sémantique pilotable
  - Déployer l'interpréteur sur une cible embarquée

V#2 Généricité de l'interface de contrôle d'exécution ☐

V#3 Facilité de l'analyse de l'exécution de modèles ☐

Expérimentations

V#1 Effort nécessaire pour concevoir un interpréteur pilotable et embarquable ☑

- Concevoir différents interpréteurs conformes à l'approche EMI
  - Implémenter l'interface de contrôle d'exécution
  - Définir une sémantique pilotable
  - Déployer l'interpréteur sur une cible embarquée

V#2 Généricité de l'interface de contrôle d'exécution ☐

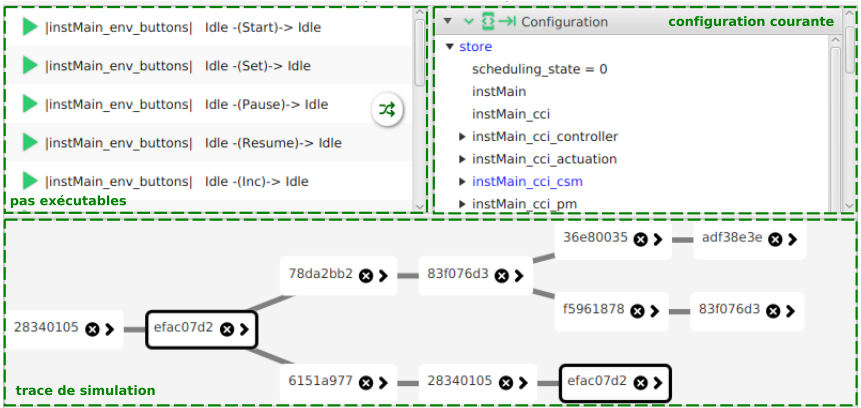
- Mener diverses activités d'analyse en connectant ces interpréteurs à différents outils de V&V

V#3 Facilité de l'analyse de l'exécution de modèles ☐

2. Mener diverses activités de développement EMI-UML EMI-LOGO AnimUML

Activités d'analyse avec le *model-checker* OBP2 [BGT20] :

- Simulation interactive ou animation

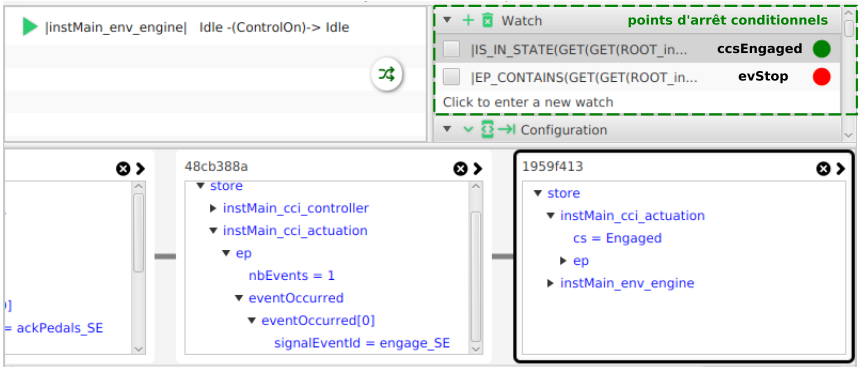


[BGT20] Brumbulli et al., « Automatic Verification of BPMN Models », 2020

2. Mener diverses activités de développement EMI-UML EMI-LOGO

Activités d'analyse avec le *model-checker* OBP2 [BGT20] :

- Simulation interactive ou animation
- Débogage omniscient et multivers

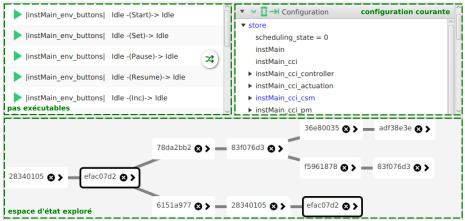


[BGT20] Brumbulli et al., « Automatic Verification of BPMN Models », 2020

2. Mener diverses activités de développement EMI-UML EMI-LOGO AnimUML

Activités d'analyse avec le *model-checker* OBP2 [BGT20] :

- Simulation interactive ou animation
- Débogage omniscient et multivers
- Détection de *deadlocks*

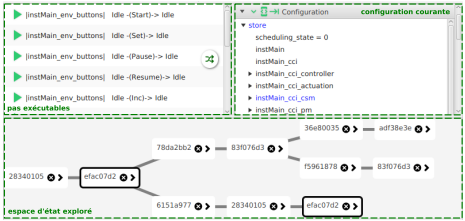


[BGT20] Brumbulli et al., « Automatic Verification of BPMN Models », 2020

2. Mener diverses activités de développement EMI-UML EMI-LOGO AnimUML

Activités d'analyse avec le *model-checker* OBP2 [BGT20] :

- Simulation interactive ou animation
- Débogage omniscient et multivers
- Détection de *deadlocks*
- *Model-checking* LTL

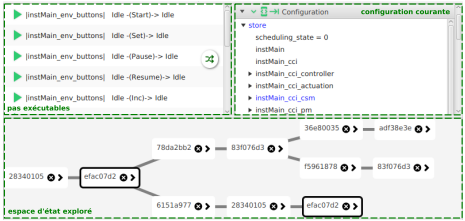


[BGT20] Brumbulli et al., « Automatic Verification of BPMN Models », 2020

2. Mener diverses activités de développement EMI-UML

Activités d'analyse avec le *model-checker* OBP2 [BGT20] :

- Simulation interactive ou animation
- Débogage omniscient et multivers
- Détection de *deadlocks*
- *Model-checking* LTL
- *Model-checking* avec des automates UML



[BGT20] Brumbulli et al., « Automatic Verification of BPMN Models », 2020

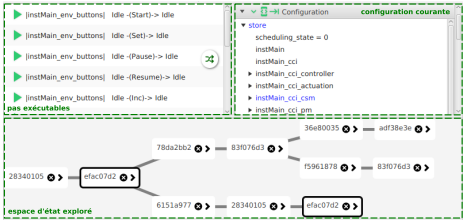
2. Mener diverses activités de développement

AnimUML

Activités d'analyse avec le *model-checker* OBP2 [BGT20] :

- Simulation interactive ou animation
- Débogage omniscient et multivers
- Détection de *deadlocks*
- *Model-checking* LTL
- *Model-checking* avec des automates UML

Bisimulation avec AnimUML



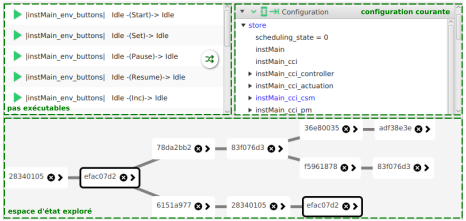
[BGT20] Brumbulli et al., « Automatic Verification of BPMN Models », 2020

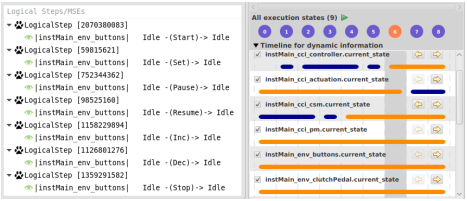
2. Mener diverses activités de développement EMI-UML

Activités d'analyse avec le *model-checker* OBP2 [BGT20] :

- Simulation interactive ou animation
- Débogage omniscient et multivers
- Détection de *deadlocks*
- *Model-checking* LTL
- *Model-checking* avec des automates UML

Bisimulation avec AnimUML
Débogage avec Gemoc Studio [Bou+16]





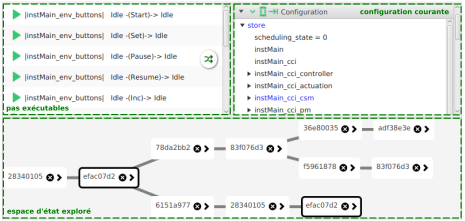
[Bou+16] Bousse et al., « Execution Framework of the GEMOC Studio (Tool Demo) », 2016
[BGT20] Brumbulli et al., « Automatic Verification of BPMN Models », 2020

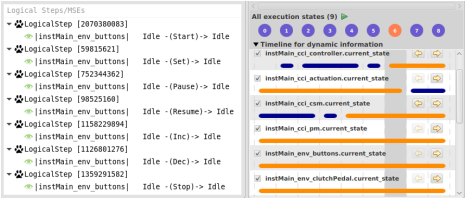
2. Mener diverses activités de développement EMI-UML

Activités d'analyse avec le *model-checker* OBP2 [BGT20] :

- Simulation interactive ou animation
- Débogage omniscient et multivers
- Détection de *deadlocks*
- *Model-checking* LTL
- *Model-checking* avec des automates UML

Bisimulation avec AnimUML
Débogage avec Gemoc Studio [Bou+16]
Model-checking avec l'outil Menhir [FTL20]



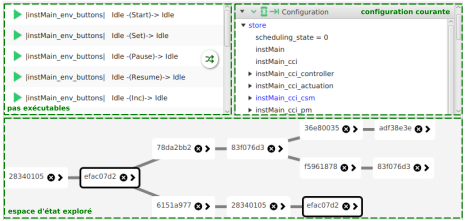


[FTL20] Fournier et al., « Menhir : Generic High-Speed FPGA Model-Checker », 2020
[Bou+16] Bousse et al., « Execution Framework of the GEMOC Studio (Tool Demo) », 2016
[BGT20] Brumbulli et al., « Automatic Verification of BPMN Models », 2020

2. Mener diverses activités de développement EMI-UML EMI-LOGO

Activités d'analyse avec le *model-checker* OBP2 [BGT20] :

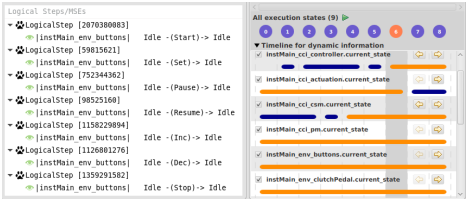
- Simulation interactive ou animation
- Débogage omniscient et multivers
- Détection de *deadlocks*
- *Model-checking* LTL
- *Model-checking* avec des automates UML



Bisimulation avec AnimUML
Débogage avec Gemoc Studio [Bou+16]
Model-checking avec l'outil Menhir [FTL20]

Activités sur la plateforme d'exécution :

- Exécution embarquée sur une cible STM32

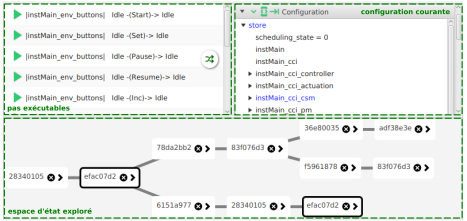


[FTL20] Fournier et al., « Menhir : Generic High-Speed FPGA Model-Checker », 2020
[Bou+16] Bousse et al., « Execution Framework of the GEMOC Studio (Tool Demo) », 2016
[BGT20] Brumbulli et al., « Automatic Verification of BPMN Models », 2020

2. Mener diverses activités de développement EMI-UML

Activités d'analyse avec le *model-checker* OBP2 [BGT20] :

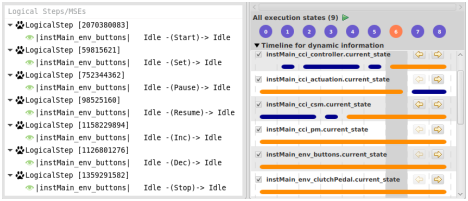
- Simulation interactive ou animation
- Débogage omniscient et multivers
- Détection de *deadlocks*
- *Model-checking* LTL
- *Model-checking* avec des automates UML



Bisimulation avec AnimUML
Débogage avec Gemoc Studio [Bou+16]
Model-checking avec l'outil Menhir [FTL20]

Activités sur la plateforme d'exécution :

- Exécution embarquée sur une cible STM32
- *Monitoring* avec des automates observateurs



[FTL20] Fournier et al., « Menhir : Generic High-Speed FPGA Model-Checker », 2020
[Bou+16] Bousse et al., « Execution Framework of the GEMOC Studio (Tool Demo) », 2016
[BGT20] Brumbulli et al., « Automatic Verification of BPMN Models », 2020

Expérimentations

V#1 Effort nécessaire pour concevoir un interpréteur pilotable et embarquable ☑

- Concevoir différents interpréteurs conformes à l'approche EMI
  - Implémenter l'interface de contrôle d'exécution
  - Définir une sémantique pilotable
  - Déployer l'interpréteur sur une cible embarquée

V#2 Généricité de l'interface de contrôle d'exécution ☐

- Mener diverses activités d'analyse en connectant ces interpréteurs à différents outils de V&V

V#3 Facilité de l'analyse de l'exécution de modèles ☐

Expérimentations

V#1 Effort nécessaire pour concevoir un interpréteur pilotable et embarquable ☑

- Concevoir différents interpréteurs conformes à l'approche EMI
  - Implémenter l'interface de contrôle d'exécution
  - Définir une sémantique pilotable
  - Déployer l'interpréteur sur une cible embarquée

V#2 Généricité de l'interface de contrôle d'exécution ☑

- Mener diverses activités d'analyse en connectant ces interpréteurs à différents outils de V&V

V#3 Facilité de l'analyse de l'exécution de modèles ☐

Expérimentations

V#1 Effort nécessaire pour concevoir un interpréteur pilotable et embarquable ☑

- Concevoir différents interpréteurs conformes à l'approche EMI
  - Implémenter l'interface de contrôle d'exécution
  - Définir une sémantique pilotable
  - Déployer l'interpréteur sur une cible embarquée

V#2 Généricité de l'interface de contrôle d'exécution ☑

- Mener diverses activités d'analyse en connectant ces interpréteurs à différents outils de V&V

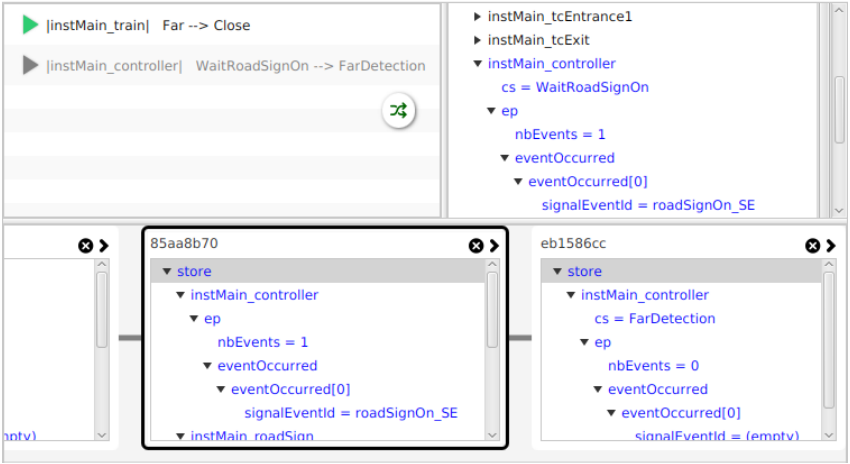
V#3 Facilité de l'analyse de l'exécution de modèles ☐

- Adapter la spécification et la vérification des propriétés aux besoins des ingénieurs
  - Utiliser nos opérateurs pour configurer l'architecture d'analyse
  - Réutiliser les concepts du langage de modélisation

3. Adapter l'architecture d'analyse aux besoins des ingénieurs

Pour tous nos interpréteurs :

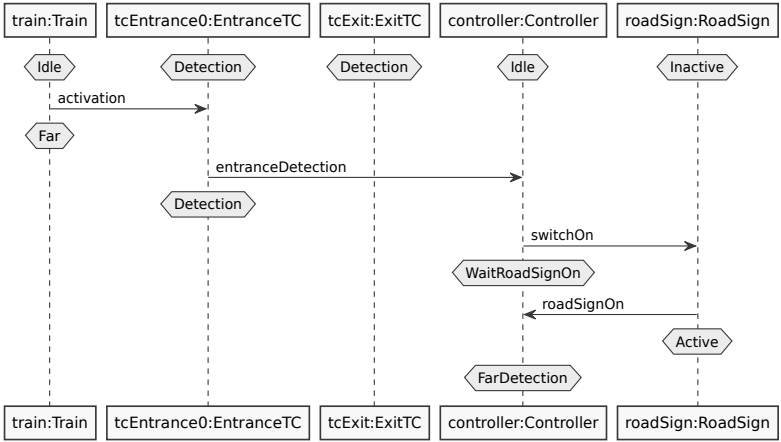
- Expression des résultats d'analyse en termes des concepts du langage de modélisation



3. Adapter l'architecture d'analyse aux besoins des ingénieurs

Pour tous nos interpréteurs :

- Expression des résultats d'analyse en termes des concepts du langage de modélisation



3. Adapter l'architecture d'analyse aux besoins des ingénieurs

Pour tous nos interpréteurs :

- Expression des résultats d'analyse en termes des concepts du langage de modélisation
- Définition d'un langage d'observation avec un mécanisme du langage hôte

```
"IS_IN_STATE(GET(GET(ROOT_instMain, cci), actuation), STATE_Actuation_Disengaged)"
"EP_CONTAINS(GET(GET(ROOT_instMain, cci), csm), SIGNAL_stop)"
```

3. Adapter l'architecture d'analyse aux besoins des ingénieurs

Pour tous nos interpréteurs :

- Expression des résultats d'analyse en termes des concepts du langage de modélisation
- Définition d'un langage d'observation avec un mécanisme du langage hôte

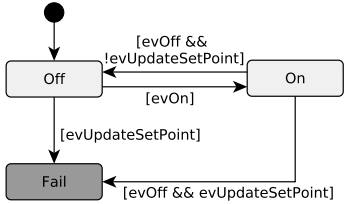
Pour l'interpréteur EMI-UML :

- Encodage des propriétés sous forme de machines à états UML [Bes+19a]

```
"(!evUpdateSetPoint W evOn) &&
([] (evOff -> (!evUpdateSetPoint W evOn)))"
```

Avec *evUpdateSetPoint*, *evOn* et *evOff* des propositions atomiques exprimées dans le langage d'observation

Propriété LTL



Automate observateur en UML

[Bes+19a] Besnard et al., « Verifying and Monitoring UML Models with Observer Automata : A Transformation-Free Approach », 2019

3. Adapter l'architecture d'analyse aux besoins des ingénieurs

Pour tous nos interpréteurs :

- Expression des résultats d'analyse en termes des concepts du langage de modélisation
- Définition d'un langage d'observation avec un mécanisme du langage hôte

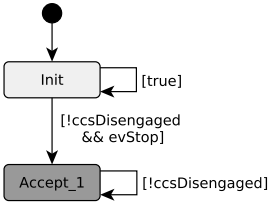
Pour l'interpréteur EMI-UML :

- Encodage des propriétés sous forme de machines à états UML [Bes+19a]

`"[] (evStop -> (<> ccsDisengaged))"`

Avec *evStop* et *ccsDisengaged* des propositions atomiques exprimées dans le langage d'observation

Propriété LTL



Automate de Büchi en UML

[Bes+19a] Besnard et al., « Verifying and Monitoring UML Models with Observer Automata : A Transformation-Free Approach », 2019

3. Adapter l'architecture d'analyse aux besoins des ingénieurs

Pour tous nos interpréteurs :

- Expression des résultats d'analyse en termes des concepts du langage de modélisation
- Définition d'un langage d'observation avec un mécanisme du langage hôte

Pour l'interpréteur EMI-UML :

- Encodage des propriétés sous forme de machines à états UML [Bes+19a]
- Prise en compte de l'hypothèse de réactivité en *model-checking*

[Bes+19a] Besnard et al., « Verifying and Monitoring UML Models with Observer Automata : A Transformation-Free Approach », 2019

3. Adapter l'architecture d'analyse aux besoins des ingénieurs

Pour tous nos interpréteurs :

- Expression des résultats d'analyse en termes des concepts du langage de modélisation
- Définition d'un langage d'observation avec un mécanisme du langage hôte

Pour l'interpréteur EMI-UML :

- Encodage des propriétés sous forme de machines à états UML [Bes+19a]
- Prise en compte de l'hypothèse de réactivité en *model-checking*
- Possibilité de couper des chemins d'exécution selon des invariants

[Bes+19a] Besnard et al., « Verifying and Monitoring UML Models with Observer Automata : A Transformation-Free Approach », 2019

3. Adapter l'architecture d'analyse aux besoins des ingénieurs

Pour tous nos interpréteurs :

- Expression des résultats d'analyse en termes des concepts du langage de modélisation
- Définition d'un langage d'observation avec un mécanisme du langage hôte

Pour l'interpréteur EMI-UML :

- Encodage des propriétés sous forme de machines à états UML [Bes+19a]
- Prise en compte de l'hypothèse de réactivité en *model-checking*
- Possibilité de couper des chemins d'exécution selon des invariants
- Utilisation d'un ordonnanceur avec différentes politiques d'ordonnancement

[Bes+19a] Besnard et al., « Verifying and Monitoring UML Models with Observer Automata : A Transformation-Free Approach », 2019

Expérimentations

V#1 Effort nécessaire pour concevoir un interpréteur pilotable et embarquable ☑

- Concevoir différents interpréteurs conformes à l'approche EMI
  - Implémenter l'interface de contrôle d'exécution
  - Définir une sémantique pilotable
  - Déployer l'interpréteur sur une cible embarquée

V#2 Généricité de l'interface de contrôle d'exécution ☑

- Mener diverses activités d'analyse en connectant ces interpréteurs à différents outils de V&V

V#3 Facilité de l'analyse de l'exécution de modèles ☐

- Adapter la spécification et la vérification des propriétés aux besoins des ingénieurs
  - Utiliser nos opérateurs pour configurer l'architecture d'analyse
  - Réutiliser les concepts du langage de modélisation

Expérimentations

V#1 Effort nécessaire pour concevoir un interpréteur pilotable et embarquable ☑

- Concevoir différents interpréteurs conformes à l'approche EMI
  - Implémenter l'interface de contrôle d'exécution
  - Définir une sémantique pilotable
  - Déployer l'interpréteur sur une cible embarquée

V#2 Généricité de l'interface de contrôle d'exécution ☑

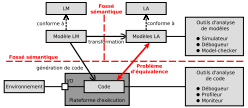
- Mener diverses activités d'analyse en connectant ces interpréteurs à différents outils de V&V

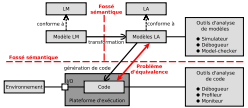
V#3 Facilité de l'analyse de l'exécution de modèles ☑

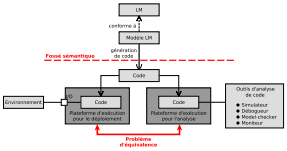
- Adapter la spécification et la vérification des propriétés aux besoins des ingénieurs
  - Utiliser nos opérateurs pour configurer l'architecture d'analyse
  - Réutiliser les concepts du langage de modélisation

Sommaire

Conclusion



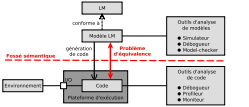


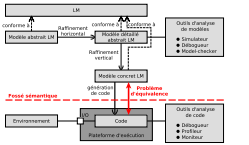


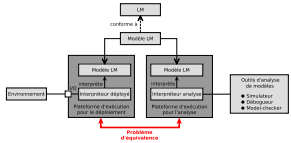
P#1 Fossé sémantique entre le modèle de conception et les modèles d'analyse

P#2 Fossé sémantique entre le modèle de conception et le code exécutable

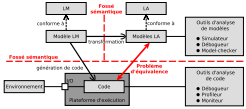
P#3 Problème d'équivalence entre les modèles d'analyse et le code exécutable

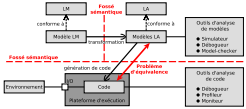


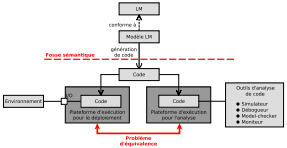


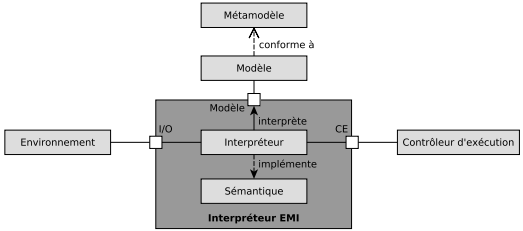


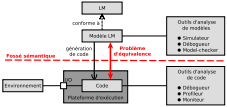
Conclusion

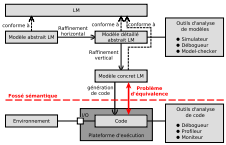

LM
LA
conforme à
Modèle LM
transformation
Modèles LA
Fossé sémantique
génération de code
Problème d'équivalence
Environnement
I/O
Code
Plateforme d'exécution
Outils d'analyse de modèles
Simulateur
Débogueur
Model-checker
Outils d'analyse de code
Débogueur
Profileur
Moniteur

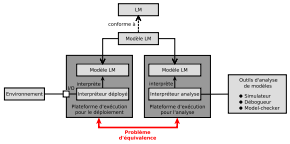

LM
LA
conforme à
Modèle LM
transformation
Modèles LA
Fossé sémantique
génération de code
Problème d'équivalence
Environnement
I/O
CE
Code
Plateforme d'exécution
Outils d'analyse de modèles
Simulateur
Débogueur
Model-checker
Outils d'analyse de code
Débogueur
Profileur
Moniteur


LM
conforme à
Modèle LM
Fossé sémantique
génération de code
Code
Environnement
I/O
Code
Plateforme d'exécution pour le déploiement
Plateforme d'exécution pour l'analyse
Problème d'équivalence
Outils d'analyse de code
Simulateur
Débogueur
Model-checker
Moniteur

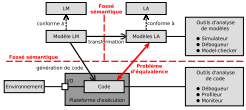

Métamodèle
conforme à
Modèle
Modèle
interprète
Environnement
I/O
Interpréteur
CE
Contrôleur d'exécution
implémente
Sémantique
Interpréteur EMI

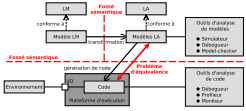

LM
conforme à
Modèle LM
Fossé sémantique
génération de code
Problème d'équivalence
Environnement
I/O
Code
Plateforme d'exécution
Outils d'analyse de modèles
Simulateur
Débogueur
Model-checker
Outils d'analyse de code
Débogueur
Profileur
Moniteur

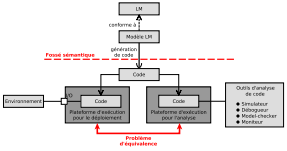

LM
conforme à
Modèle abstrait LM
Raffinement horizontal
conforme à
Modèle abstrait LM
Raffinement vertical
Modèle concret LM
Fossé sémantique
génération de code
Problème d'équivalence
Environnement
I/O
Code
Plateforme d'exécution
Outils d'analyse de modèles
Simulateur
Débogueur
Model-checker
Outils d'analyse de code
Débogueur
Profileur
Moniteur

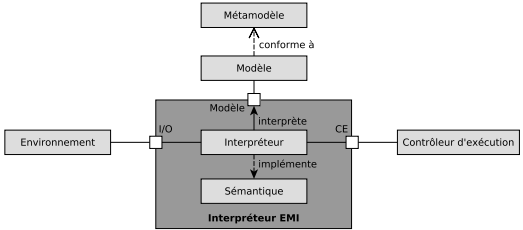

LM
conforme à
Modèle LM
Modèle LM
Modèle LM
interprète
interprète
Environnement
I/O
Interpréteur déploiement
Interpréteur analyse
Plateforme d'exécution pour le déploiement
Plateforme d'exécution pour l'analyse
Problème d'équivalence
Outils d'analyse de modèles
Simulateur
Débogueur
Model-checker

Conclusion





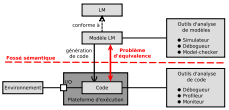


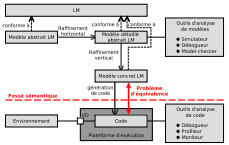


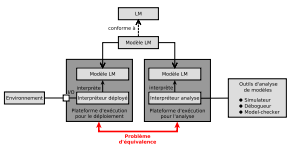
Contributions

1. Unifier l'analyse et l'exécution embarquée de modèles
2. Faciliter l'adoption des techniques de vérification formelle par les ingénieurs

... pour différents langages de modélisation logicielle ($\rightarrow$ ingénieur langage)







Perspectives

Live modeling permet d'adapter la modélisation du système à l'exécution.
Partial modeling permet de modéliser et d'exécuter des modèles partiels (c.-à-d. incomplets).

Défis scientifiques

- Modifier le modèle déployé en :
  - Enrichissant l'interface de contrôle d'exécution
  - Utilisant des techniques de chargement du modèle à la volée
- Modifier la sémantique du langage de modélisation pour :
  - Relâcher certaines contraintes
  - Étendre la sémantique d'exécution
- Retrouver un état d'exécution cohérent

EMI : Une approche pour unifier l'analyse et l'exécution embarquée à l'aide d'un interpréteur de modèles pilotable

Application aux modèles UML des systèmes embarqués

Soutenance de thèse de Valentin BESNARD

École doctorale MathSTIC

Mercredi 9 décembre 2020 — ESEO Angers

Rapporteurs
Frédéric BONIOL, ONERA/DTIS
Benoît COMBEMALE, Université de Rennes 1, IRISA
Examinateurs
Isabelle BORNE, Université Bretagne Sud, IRISA
Julien DEANTONI, Université Côte d'Azur, I3S
Frédéric JOUAULT, ESEO

Directeur de thèse
Philippe DHAUSSY, ENSTA Bretagne, Lab-STICC
Encadrants
Matthias BRUN, ESEO
Ciprian TEODOROV, ENSTA Bretagne, Lab-STICC
Invité
David OLIVIER, Davidson Consulting