Unified LTL Verification and Embedded Execution of UML Models

Scientific Seminar, MOCS Team

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

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

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

This work has been partially funded by Davidson.

Table of Contents

Table of Contents

Context

Observations

- Increasing complexity of embedded systems
- Emergence of new needs and applications
- Connection of these systems to networks (IoT)

Context

Observations

- Increasing complexity of embedded systems
- Emergence of new needs and applications
- Connection of these systems to networks (IoT)

Consequences on software programs

- More prone to uncertain behaviors, security flaws, and design mistakes
- More safety and security requirements

Context

Observations

- Increasing complexity of embedded systems
- Emergence of new needs and applications
- Connection of these systems to networks (IoT)

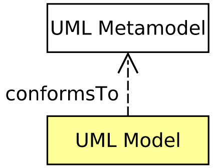
Consequences on software programs

- More prone to uncertain behaviors, security flaws, and design mistakes
- More safety and security requirements

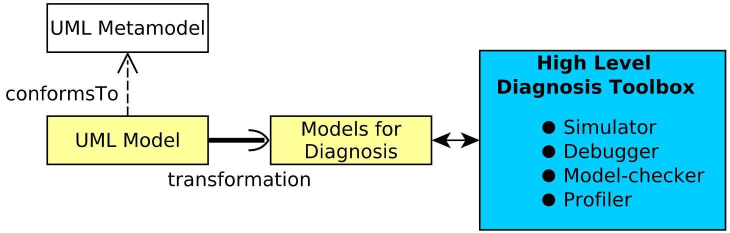
Consequence on software development

- Increasing need of verification and validation

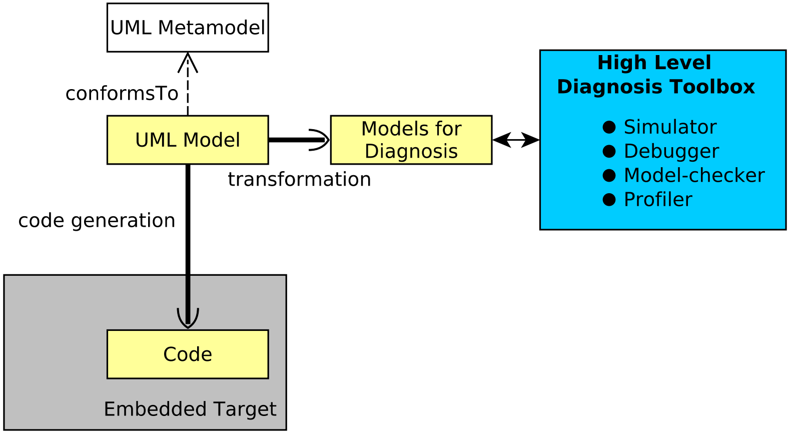
Classical UML-based Approaches



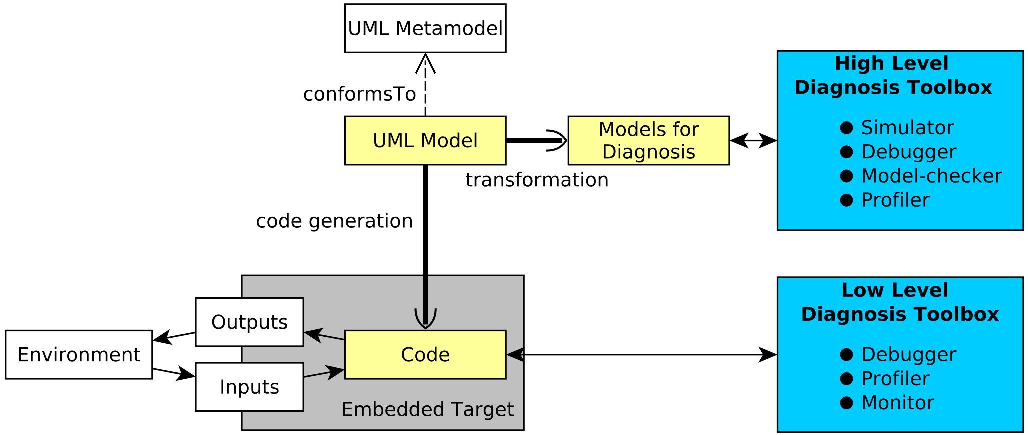
Classical UML-based Approaches



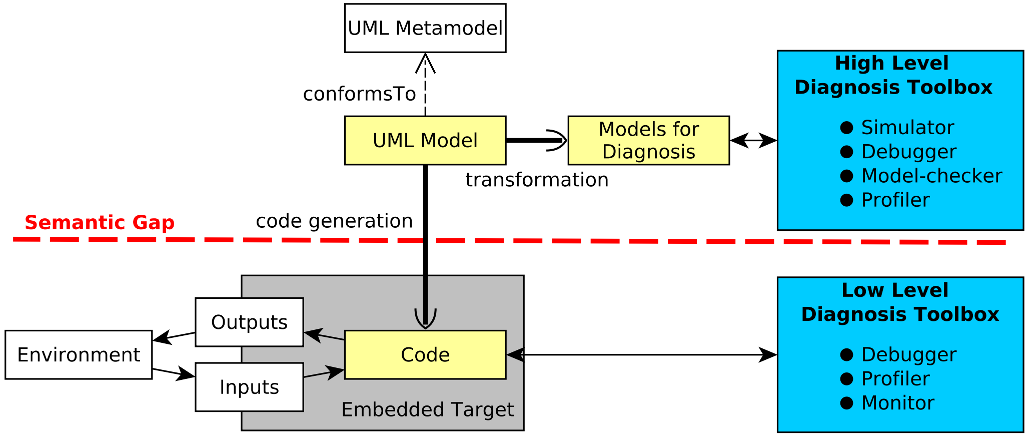
Classical UML-based Approaches



Classical UML-based Approaches

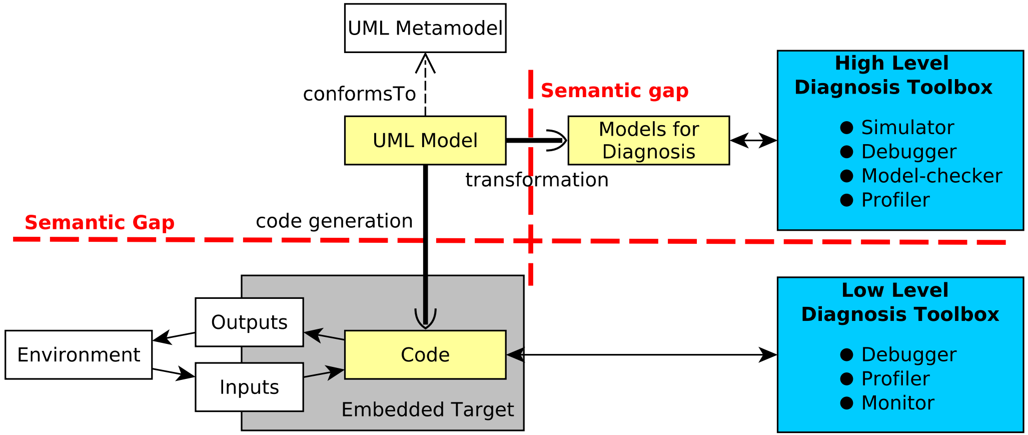


Some Problems



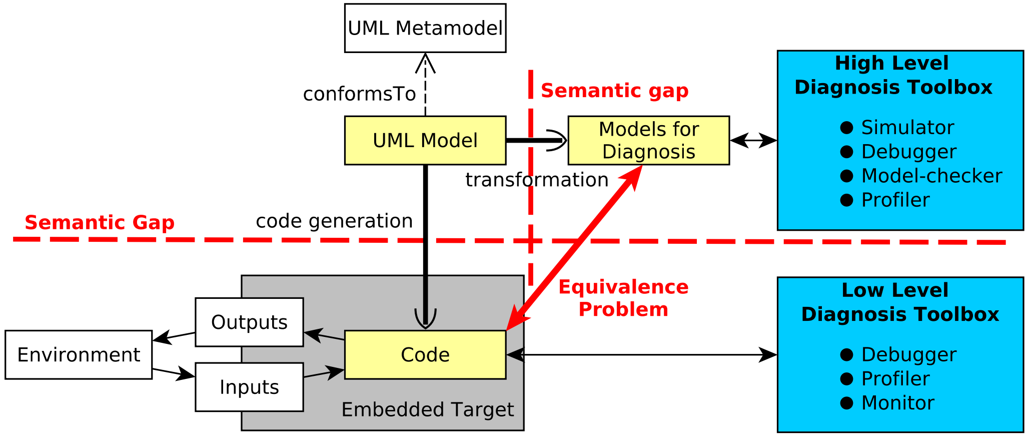
First issue: Semantic gap between design model and executable code.

Some Problems



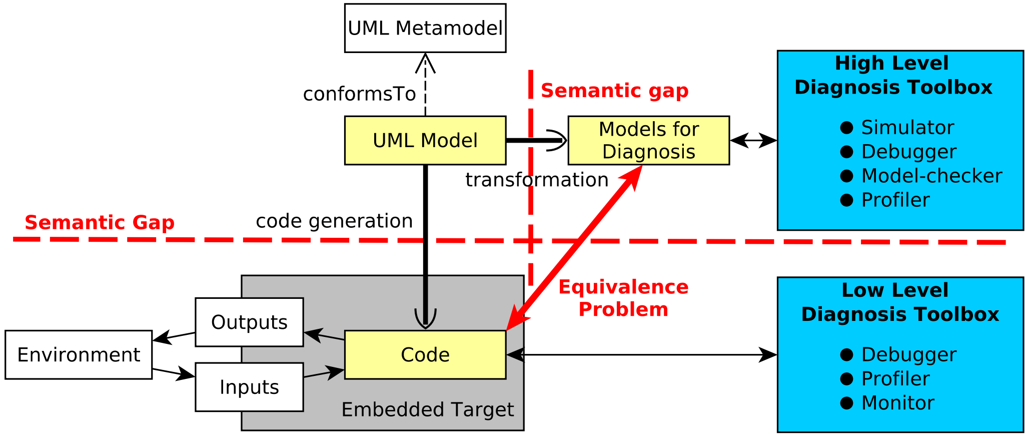
Second issue: Semantic gap between design model and diagnosis model.

Some Problems



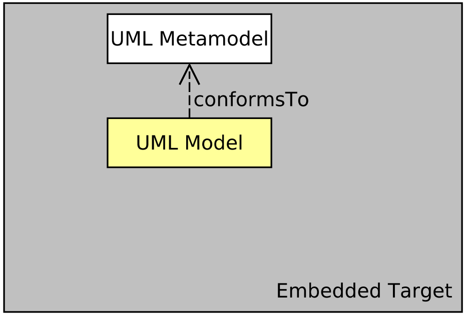
Third issue: An equivalence relation between verified formal models and deployed code should be built, proven, and maintained.

Some Problems

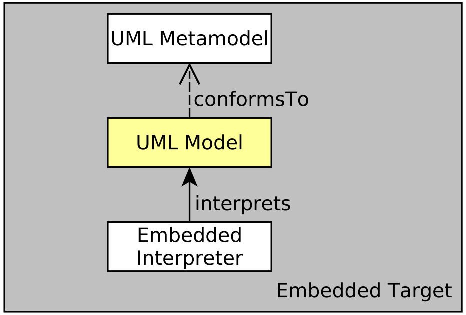


Main cause of these problems: Multiple definitions of the modeling language semantics.

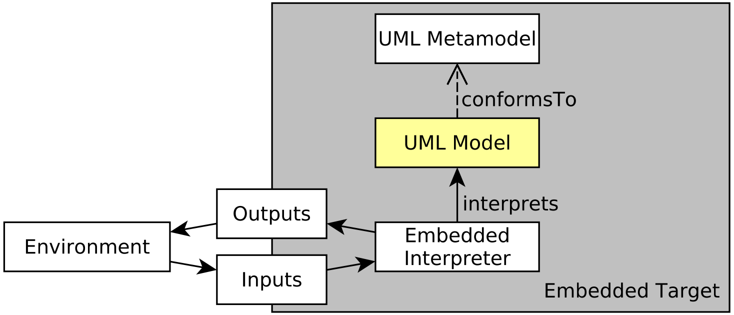
Our Approach: A Unified Modeling Language Semantics



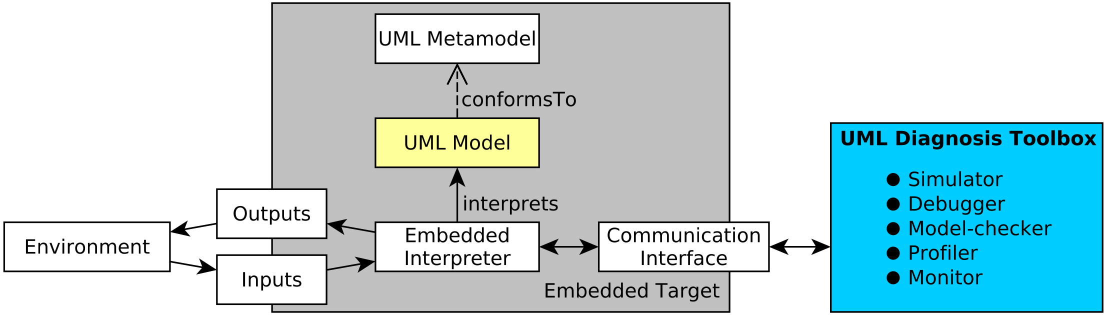
Our Approach: A Unified Modeling Language Semantics



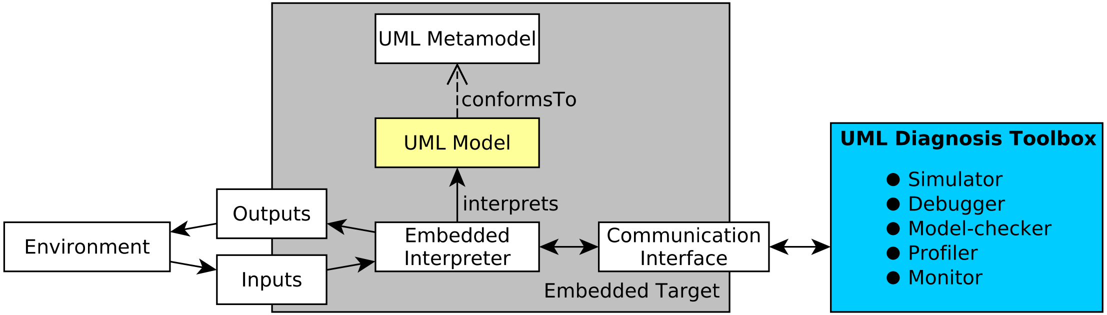
Our Approach: A Unified Modeling Language Semantics



Our Approach: A Unified Modeling Language Semantics

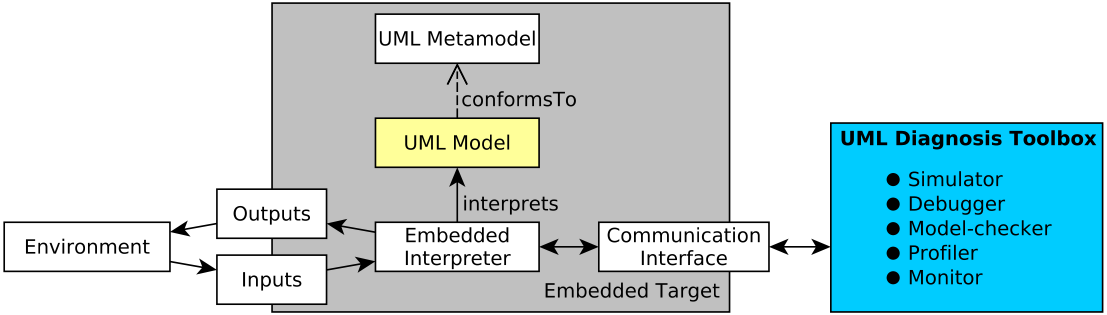


Our Approach: A Unified Modeling Language Semantics



Other tools are able to execute and analyze models:
GEMOC Studio [Bousse et al., 2016], Moliz [Mayerhofer et al., 2012], Moka [Revol et al. 2018], GUML [Charfi et al, 2012], Unicomp [Ciccozzi, 2018], Mbeddr [Voelter et al., 2012], etc.

Our Approach: A Unified Modeling Language Semantics



A single implementation of the language semantics
for all activities: simulation, execution, and diagnosis.

Results

Simulation

- Trace-based simulation

Execution

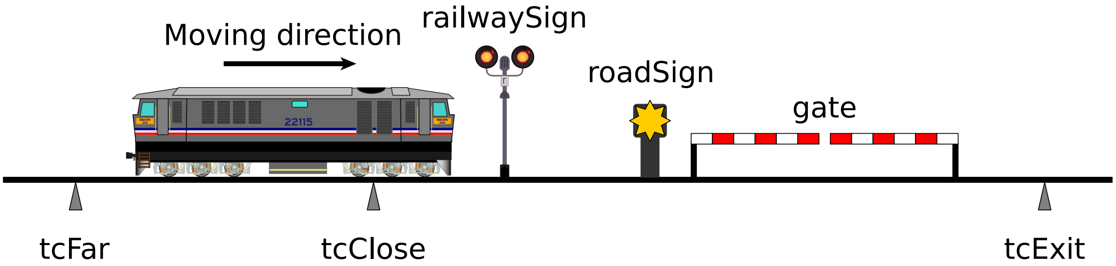
- On bare-metal (without operating system) embedded targets
- On desktop computers

Diagnosis

- State-space exploration
- Deadlock detection
- Model-checking of Linear Temporal Logic (LTL) properties

Table of Contents

Level Crossing Overview



Goal

Ensure safety during the passage of the train

Level Crossing Model Requirements

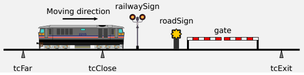
Deadlock detection

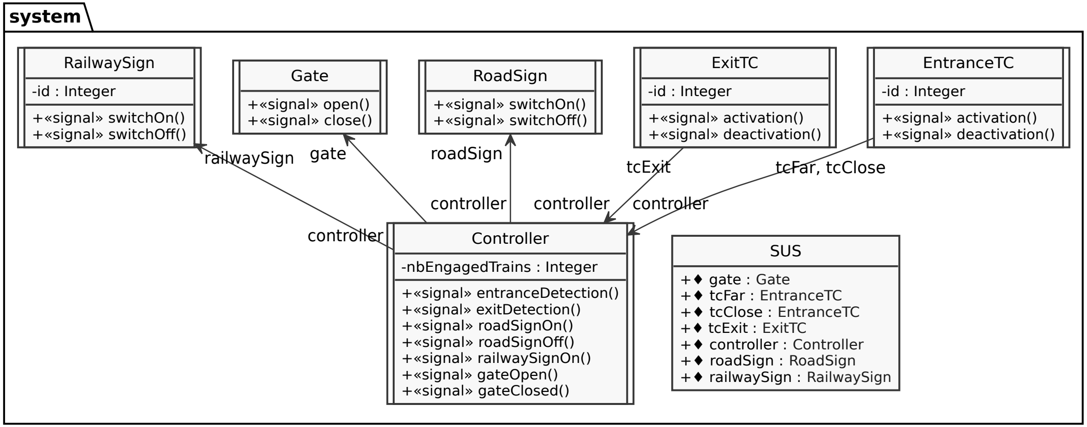
- Ensuring that the model is deadlock free.

System requirements

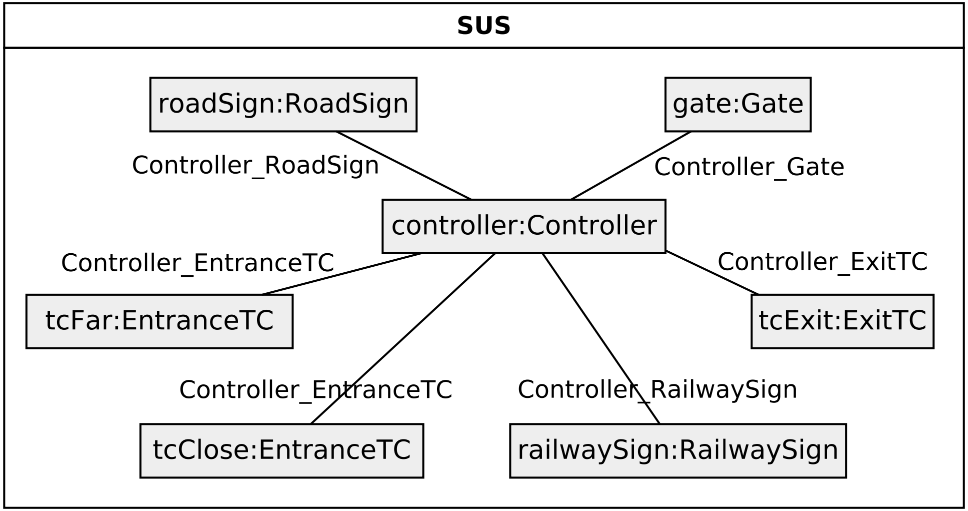
1. The Gate is closed when the Train is on the level crossing.
2. The light of the RoadSign is active when the Train is on the level crossing.
3. The Gate finally opens after being closed.
4. The light of the RoadSign is finally turn off after being activated.

Level Crossing Model (Class Diagram)

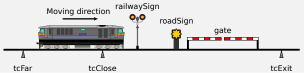


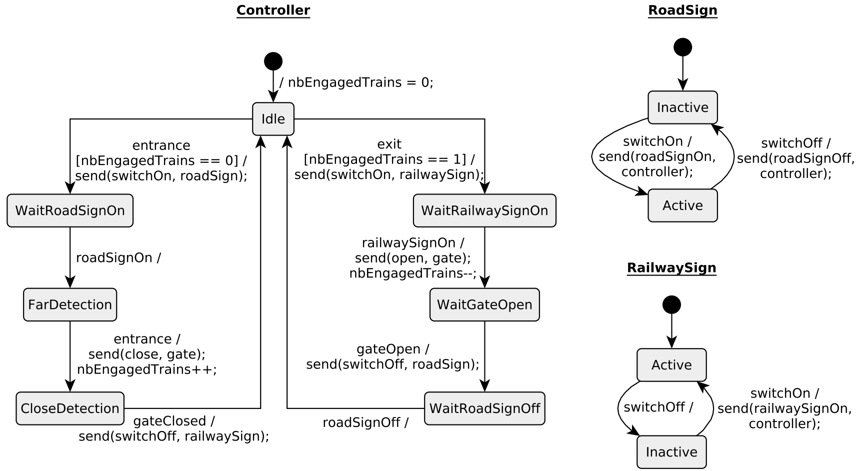


Level Crossing Model (Composite Structure Diagram)



Level Crossing Model (State Machines)




Controller
/ nbEngagedTrains = 0;
Idle
entrance [nbEngagedTrains == 0] / send(switchOn, roadSign);
exit [nbEngagedTrains == 1] / send(switchOn, railwaySign);
WaitRoadSignOn
WaitRailwaySignOn
roadSignOn /
railwaySignOn / send(open, gate); nbEngagedTrains--;
FarDetection
WaitGateOpen
entrance / send(close, gate); nbEngagedTrains++;
gateOpen / send(switchOff, roadSign);
CloseDetection
WaitRoadSignOff
gateClosed / send(switchOff, railwaySign);
roadSignOff /
RoadSign
Inactive
switchOn / send(roadSignOn, controller);
switchOff / send(roadSignOff, controller);
Active
RailwaySign
Active
switchOff /
switchOn / send(railwaySignOn, controller);
Inactive

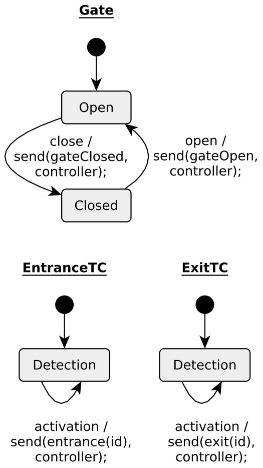
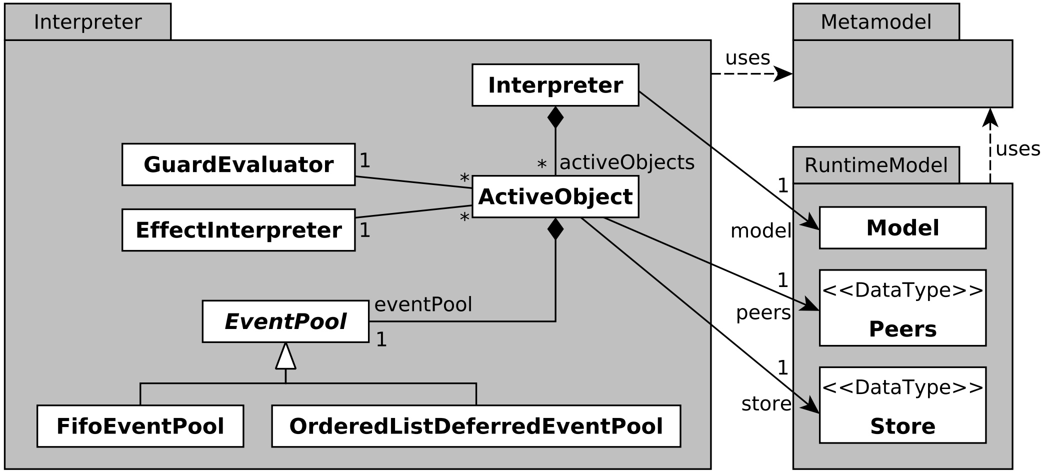

Gate
Open
close / send(gateClosed, controller);
open / send(gateOpen, controller);
Closed
EntranceTC
Detection
activation / send(entrance(id), controller);
ExitTC
Detection
activation / send(exit(id), controller);

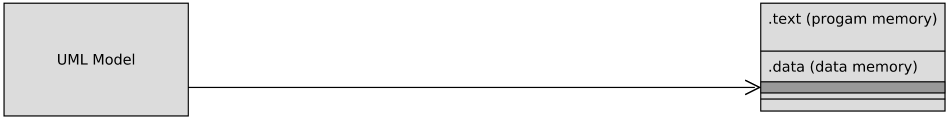
Table of Contents

UML Interpreter Design

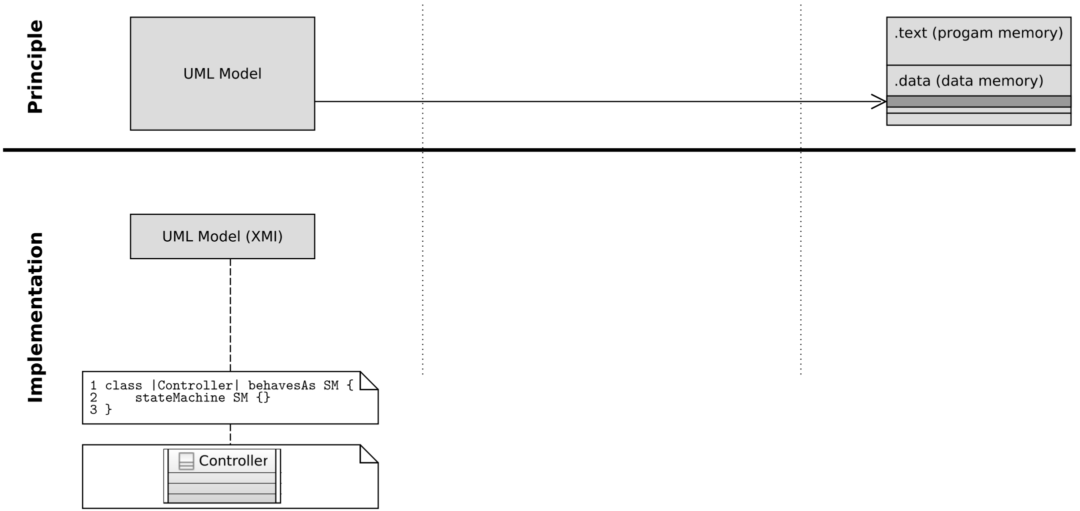


Loading the Runtime Model at Compile-Time

Principle

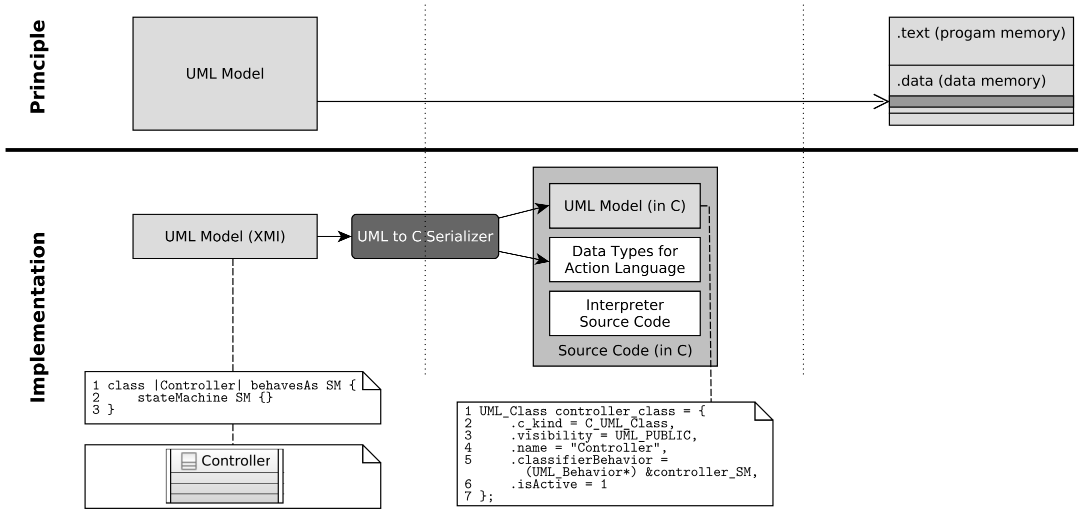


Loading the Runtime Model at Compile-Time



```
class |Controller| behavesAs SM {
    stateMachine SM {}
}
```

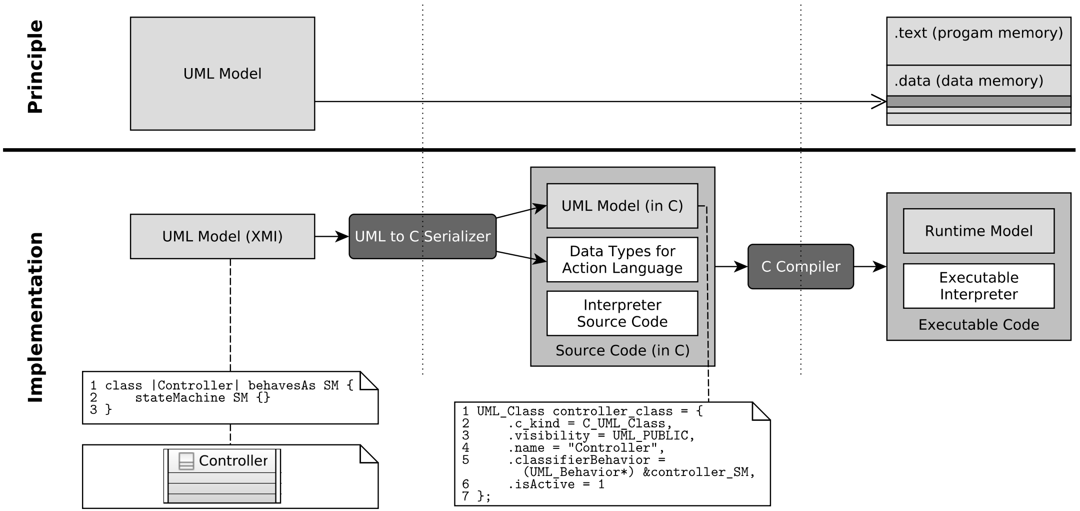
Loading the Runtime Model at Compile-Time



```
class |Controller| behavesAs SM {
    stateMachine SM {}
}
```

```
UML_Class controller_class = {
    .c_kind = C_UML_Class,
    .visibility = UML_PUBLIC,
    .name = "Controller",
    .classifierBehavior =
      (UML_Behavior*) &controller_SM,
    .isActive = 1
};
```

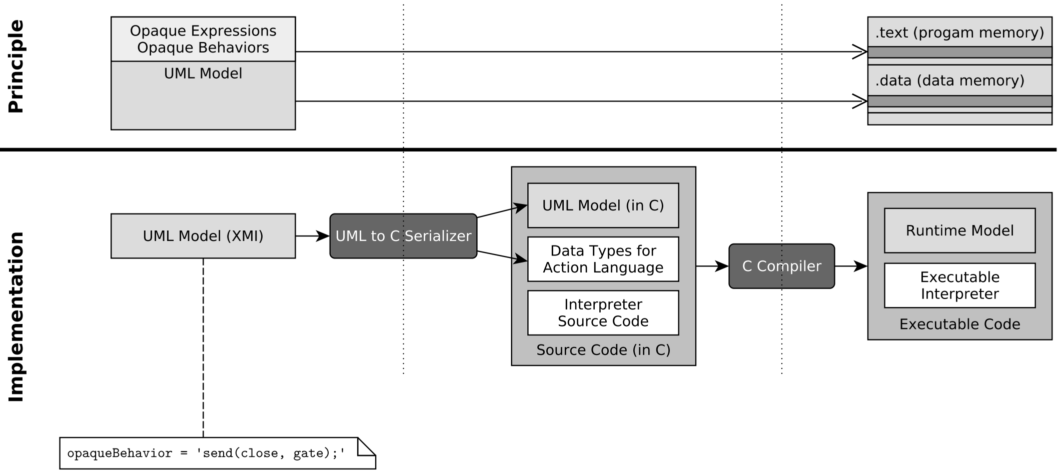
Loading the Runtime Model at Compile-Time



```
class |Controller| behavesAs SM {
    stateMachine SM {}
}
```

```
UML_Class controller_class = {
    .c_kind = C_UML_Class,
    .visibility = UML_PUBLIC,
    .name = "Controller",
    .classifierBehavior =
      (UML_Behavior*) &controller_SM,
    .isActive = 1
};
```

Loading the Runtime Model at Compile-Time



```
opaqueBehavior = 'send(close, gate);'
```

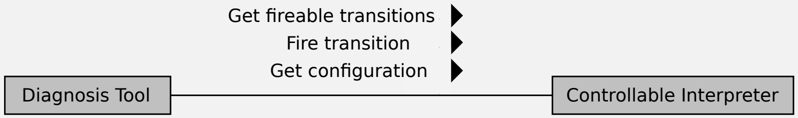
Table of Contents

UML Model Diagnosis: Goals and Requirements

Our goals

- Simulate the model

Requirements to achieve these goals

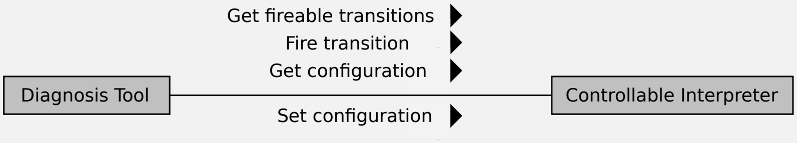


UML Model Diagnosis: Goals and Requirements

Our goals

- Simulate the model (with rollback for back-in-time simulation)
- Explore the model state-space
- Detect deadlocks

Requirements to achieve these goals

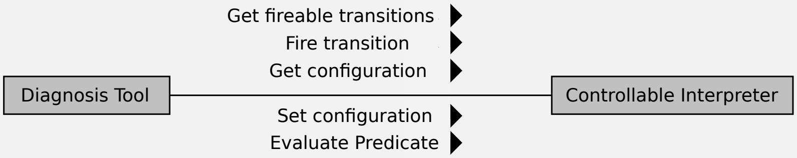


UML Model Diagnosis: Goals and Requirements

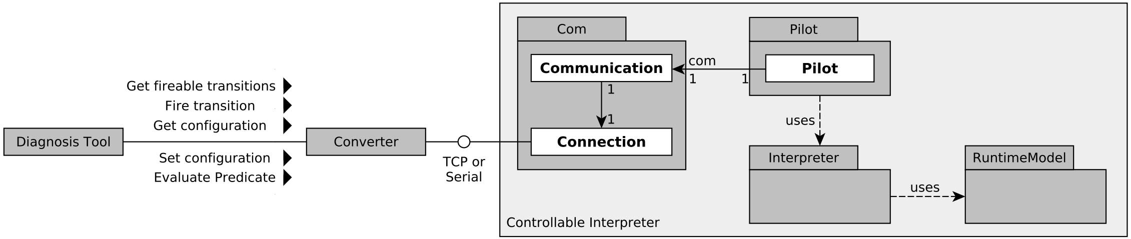
Our goals

- Simulate the model (with rollback for back-in-time simulation)
- Explore the model state-space
- Detect deadlocks
- Verify formal properties via model-checking

Requirements to achieve these goals



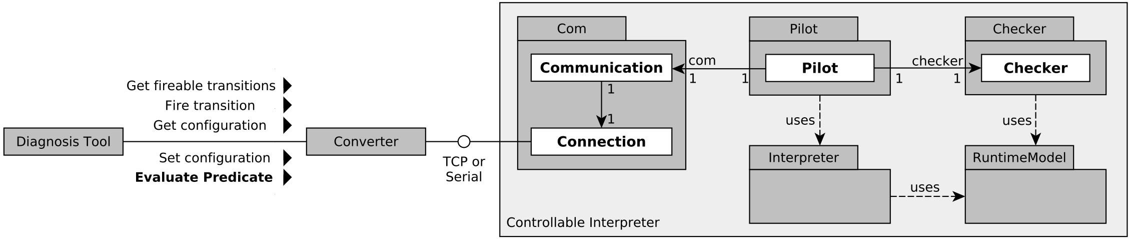
Diagnosis Design



Design of an application layer protocol over:

- TCP connection
- Serial connection (e.g., UART, USB)

Diagnosis Design

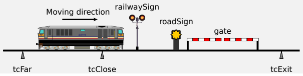


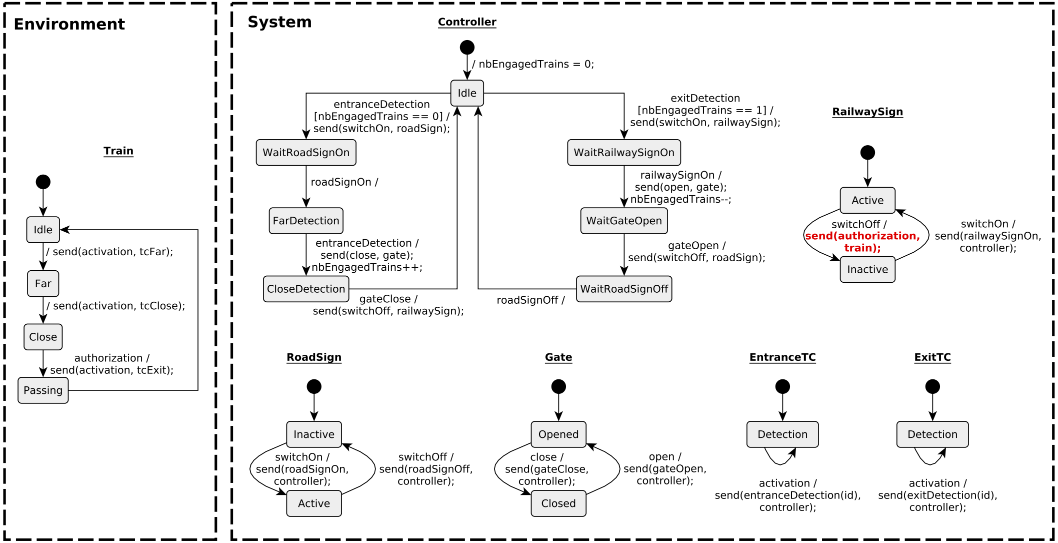
A formal property consists of:

- Atomic propositions (i.e., predicates related to model concepts)
  - $\rightarrow$ Compiled into executable code by the converter
  - $\rightarrow$ Evaluated by the controllable interpreter
- Logical operators used to link atomic propositions together
  - $\rightarrow$ Evaluated by the diagnosis tool (model-checker)

Table of Contents

Level Crossing Model Under Verification





Model-Checking of the Level Crossing Model

Expression of Properties into LTL

1. `[] !(trainIsPassing && gateIsOpen)`
2. `[] !(trainIsPassing && roadSignIsOff)`
3. `[] (gateIsClosed -> <> gateIsOpen)`
4. `[] (roadSignIsOn -> <> roadSignIsOff)`

Model-Checking of the Level Crossing Model

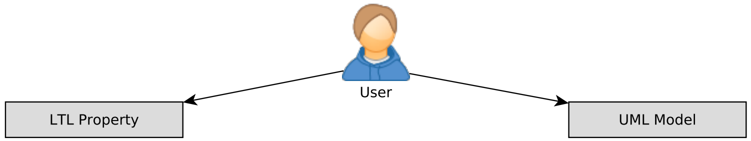
Expression of Properties into LTL

1. `[] !(trainIsPassing && gateIsOpen)`
2. `[] !(trainIsPassing && roadSignIsOff)`
3. `[] (gateIsClosed -> <> gateIsOpen)`
4. `[] (roadSignIsOn -> <> roadSignIsOff)`

Expression of Atomic Propositions

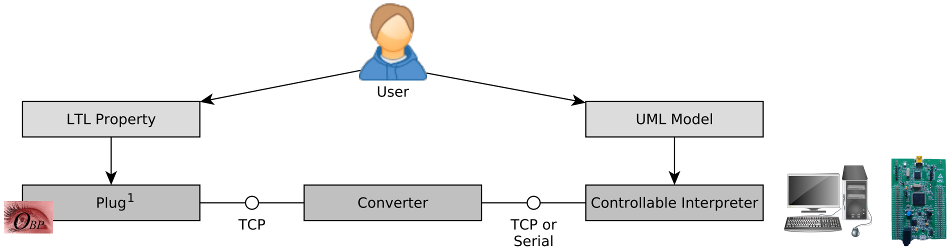
- `trainIsPassing = |train.state == PASSING|`
- `gateIsClosed = |gate.state == CLOSED|`
- `gateIsOpen = |gate.state == OPEN|`
- `roadSignIsOn = |roadSign.state == ACTIVE|`
- `roadSignIsOff = |roadSign.state == INACTIVE|`

Experiments



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

Experiments



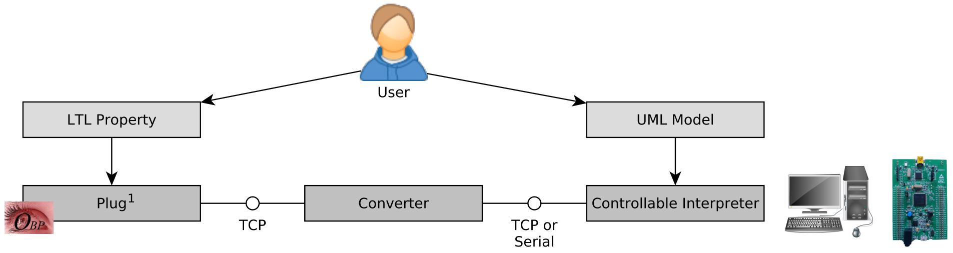
Experiments

Diagnosis of the level-crossing model on:

- Desktop computer
- STM32 discovery

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

Experiments



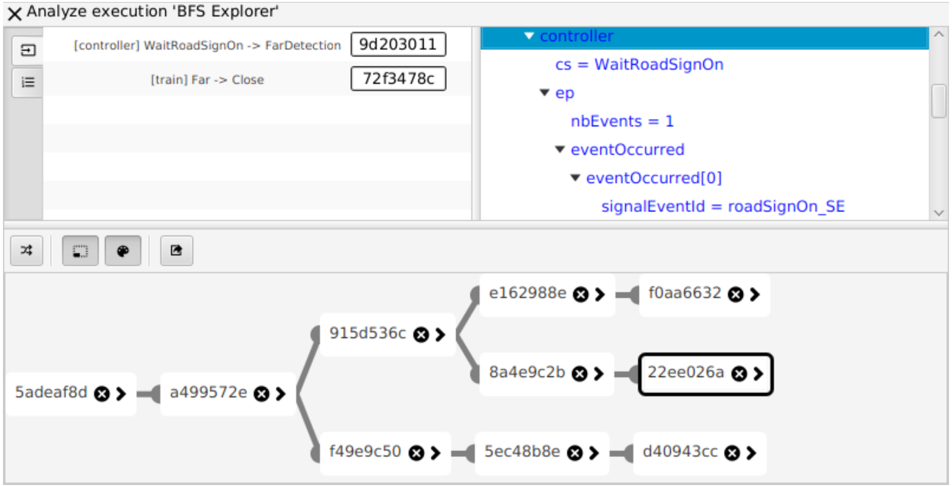
Experiments

Using the two implementations of the event pool

- the FIFO implementation that drops ignored events
- the ordered list implementation that defers ignored events

[1] `https://plug-obp.github.io/`

Results - Simulation



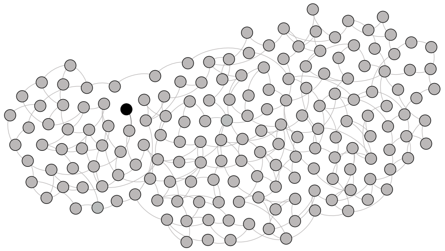
Analyze execution 'BFS Explorer'

[controller] WaitRoadSignOn -> FarDetection	9d203011
[train] Far -> Close	72f3478c

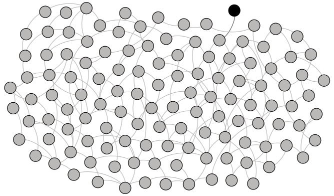
```
▼ controller
    cs = WaitRoadSignOn
  ▼ ep
      nbEvents = 1
    ▼ eventOccurred
      ▼ eventOccurred[0]
          signalEventId = roadSignOn_SE
```

Results - State-space Exploration

	FIFO (drops)	OrderedList (defers)
Nb configurations	173	122
Nb transitions	276	209



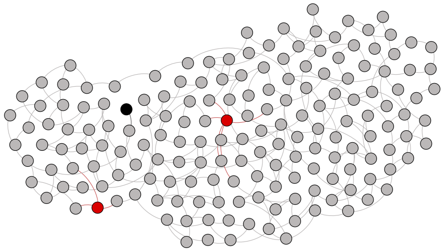
State-space graph with FIFO

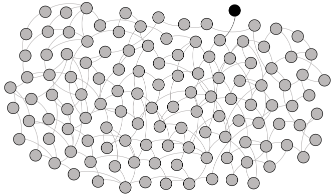


State-space graph with OrderedList

Results - Deadlock Detection

	FIFO (drops)	OrderedList (defers)
Nb configurations	173	122
Nb transitions	276	209
Nb deadlocks	2	0


State-space graph with FIFO


State-space graph with OrderedList

Results - LTL Model-checking

	FIFO (drops)	OrderedList (defers)
`[] !(trainIsPassing && gateIsOpen)`	✓	✓
`[] !(trainIsPassing && roadSignIsOff)`	✓	✓
`[] (gateIsClosed -> <> gateIsOpen)`	✗	✓
`[] (roadSignIsOn -> <> roadSignIsOff)`	✓	✓

✓: Property verified ✗: Property violated

Execution performance

Verification of the 4 properties on a desktop computer[1] in 1.71 seconds

[1] Intel® Core™ i7-8550U CPU at 1.80GHz with 4 cores, 16GB DDR4 2400MHz RAM, running a Linux OS

Table of Contents

Conclusion

Our contribution

- Use the same operational semantics implementation for execution and LTL verification
- What is checked during model diagnosis is what is executed at runtime

[1]Preliminary study: `https://plug-obp.github.io/experiments/`

Conclusion

Our contribution

- Use the same operational semantics implementation for execution and LTL verification
- What is checked during model diagnosis is what is executed at runtime

Limitations

- No support for UML activities
- No evaluation of the resource overhead of the interpreter

[1]Preliminary study: `https://plug-obp.github.io/experiments/`

Conclusion

Our contribution

- Use the same operational semantics implementation for execution and LTL verification
- What is checked during model diagnosis is what is executed at runtime

Limitations

- No support for UML activities
- No evaluation of the resource overhead of the interpreter

Perspectives

- Support multiple action languages (e.g., UML activities / Alf)
- Integrate the tool with UML modelers (e.g., Papyrus) [1]
- Apply this approach to other domain-specific languages (e.g., Capella in Eclipse PolarSys)

[1] Preliminary study: `https://plug-obp.github.io/experiments/`

Thank you for your attention

Bibliography I

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

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

Erwan Bousse, Thomas Degueule, Didier Vojtisek, Tanja Mayerhofer, Julien Deantoni, and Benoit Combemale.
Execution Framework of the GEMOC Studio (Tool Demo).
In *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*, SLE 2016, pages 84–89, New York, NY, USA, 2016. ACM.

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

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

Bibliography II

Alexandre Duret-Lutz and Denis Poitrenaud.
Spot: An extensible model checking library using transition-based generalized büchi automata.
In *Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, MASCOTS '04, pages 76–83, Washington, DC, USA, 2004. IEEE Computer Society.

Andreas Gaiser and Stefan Schwoon.
Comparison of Algorithms for Checking Emptiness on Büchi Automata.
In Petr Hlinený, Václav Matyáš, and Tomáš Vojnar, editors, *Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'09)*, volume 13 of *OpenAccess Series in Informatics (OASIcs)*, pages 18–26, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Frédéric Jouault, Ciprian Teodorov, Jérôme Delatour, Luka Le Roux, and Philippe Dhaussy.
Transformation de modèles UML vers Fiacre, via les langages intermédiaires tUML et ABCD.
Génie logiciel, 109, June 2014.

Gijs Kant, Alfons Laarman, Jeroen Meijer, Jaco Pol, Stefan Blom, and Tom Dijk.
Ltsmin: High-performance language-independent model checking.
In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 9035*, pages 692–707, New York, NY, USA, 2015. Springer-Verlag New York, Inc.

Agnes Lanusse, Yann Tanguy, Huascar Espinoza, Chokri Mraidha, Sebastien Gerard, Patrick Tessier, Remi Schnekenburger, Hubert Dubois, and François Terrier.
Papyrus UML: an open source toolset for MDA.
In *Proceedings of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009)*, pages 1–4, 2009.

Bibliography III

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

OMG.
Unified Modeling Language, December 2017.

Sebastien Revol, Géry Delog, Arnaud Cuccurru, and Jérémie Tatibouët.
Papyrus: Moka Overview, 2018.

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

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

Markus Voelter, Daniel Ratiu, Bernhard Schaetz, and Bernd Kolb.
Mbeddr: An Extensible C-based Programming Language and IDE for Embedded Systems.
In *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, SPLASH '12, pages 121–140, New York, NY, USA, 2012. ACM.

Interpreter Execution

- At startup: initialization of all active objects
- At each interpretation step:
  - Compute the set of fireable transitions by
    - Evaluating transitions guards
    - Checking if transitions triggers are satisfied
  - Fire the first fireable transition of each active object by:
    - Executing transitions effects
    - Updating current states of active objects

Verification of Formal Properties

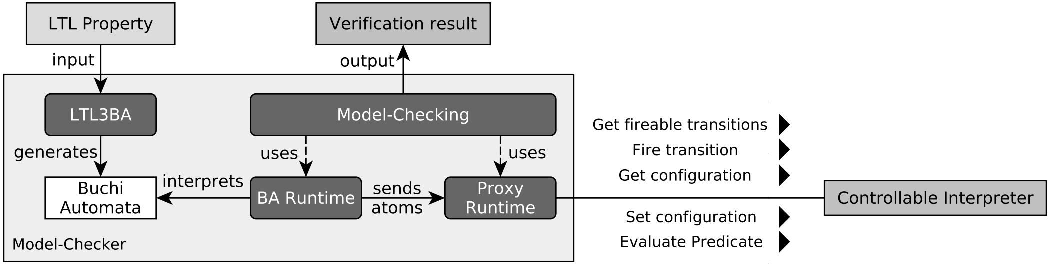
A formal property consists of:

- Atomic propositions (i.e., predicates related to model concepts)
  $\Rightarrow$ Evaluated by the interpreter
- Logical operators link atomic propositions together
  $\Rightarrow$ Evaluated by the model-checker

Process to Evaluate an Atomic Proposition

- Express atomic proposition using the action language of the interpreter
- Generate the corresponding C function (on Converter)
- Compile the C function into a dynamic library (on Converter)
- Load dynamically the library into the interpreter
- Request evaluation of the atomic proposition by the **Checker**

Application to Model-Checking



Connection to Plug (`https://plug-obp.github.io/`), an extension of the OBP model-checker:

- ProxyRuntime implements a TCP client and the application layer protocol
- Nested DFS algorithm [7]
- Verification of LTL properties