Experience and Insight from the INESS Project

Helle Hvid Hansen

Eindhoven University of Technology

RSTRC Workshop, 27 Sep 2011, York, UK

The INESS Project (EU FP7)

INtegrated **E**uropean **S**ignalling **S**ystem

Goal:
Develop a harmonized and verified specification of common core functionality for a new generation of European interlockings.

Purpose:
Interoperability between countries, increase competition, faster certification process.

Partners:
- Coordinator: UIC (International Union of Railways)
- Railway operators: ProRail, DB Netz, ...
- Industry: Siemens, Alstrom, Bombardier, Thales, ...
- Universities: Eindhoven, Twente, Southampton, York.

Duration: Oct 2008 - Sep 2011 (extended to Mar 2012).

Task of the Universities

Task:

Formally verify Executable UML model of Common Core functional requirements against a set of safety properties.

xUML lacks tooling for formal verification.

Apply our general-purpose verification technology:

- Model checking: mCRL2, LTSmin (TUE and UT).
- Theorem proving: UML-B/Event-B, Rodin (Southampton)

Strategy:

1. Translate xUML into formal language of our tools.
2. Formalise safety properties (logic formula / xUML).
3. Formally verify that safety properties hold using our tools.

Task of the Universities

Task:

Formally verify Executable UML model of Common Core functional requirements against a set of safety properties.

xUML lacks tooling for formal verification.

Apply our general-purpose verification technology:

- Model checking: mCRL2, LTSmin (TUE and UT).
- Theorem proving: UML-B/Event-B, Rodin (Southampton)

Strategy:

1. Translate xUML into formal language of our tools.
2. Formalise safety properties (logic formula / xUML).
3. Formally verify that safety properties hold using our tools.

Task of the Universities

Task:

Formally verify Executable UML model of Common Core functional requirements against a set of safety properties.

xUML lacks tooling for formal verification.

Apply our general-purpose verification technology:

- Model checking: mCRL2, LTSmin (TUE and UT).
- Theorem proving: UML-B/Event-B, Rodin (Southampton)

Strategy:

1. Translate xUML into formal language of our tools.
2. Formalise safety properties (logic formula / xUML).
3. Formally verify that safety properties hold using our tools.

Model Checking in INESS

Given formal model (of system behaviours) + property P, check that all system behaviours satisfy P.

- Model checking of xUML model is w.r.t. particular instance: model instance = generic xUML model + track layout.
- Formal model is expressed in process algebra mCRL2.
- Symbolic model checking using LTSmin.

Model Checking in INESS

Given formal model (of system behaviours) + property P,
check that all system behaviours satisfy P.

- Model checking of xUML model is w.r.t. particular instance:
  model instance = generic xUML model + track layout.
- Formal model is expressed in process algebra mCRL2.
- Symbolic model checking using LTSmin.

Model Checking Safety Properties

General approach:

- Safety: "*system cannot reach any bad states*".
- Verifying safety: reachability analysis.
- Safety property formalised as logic formula (mu-calculus).

Our approach:

- Express safety property as UML state machine ("observer"): send error signal when bad state is reached.
- Translate observers automatically with xUML model.
- Verification: search for error action.

Advantages:

- Scalability: larger systems can be verified.
- Use of UML facilitates communication with UML modellers.

Model Checking Safety Properties

General approach:

- Safety: "*system cannot reach any bad states*".
- Verifying safety: reachability analysis.
- Safety property formalised as logic formula (mu-calculus).

Our approach:

- Express safety property as UML state machine ("observer"): send error signal when bad state is reached.
- Translate observers automatically with xUML model.
- Verification: search for error action.

Advantages:

- Scalability: larger systems can be verified.
- Use of UML facilitates communication with UML modellers.

Model Checking Safety Properties

General approach:

- Safety: "*system cannot reach any bad states*".
- Verifying safety: reachability analysis.
- Safety property formalised as logic formula (mu-calculus).

Our approach:

- Express safety property as UML state machine ("observer"): send error signal when bad state is reached.
- Translate observers automatically with xUML model.
- Verification: search for error action.

Advantages:

- Scalability: larger systems can be verified.
- Use of UML facilitates communication with UML modellers.

Our first experiment: Micro Interlocking

Common Core was only due late in project timeline. We started with a toy example manufactured by Markus Schacher of KnowGravity.

Purpose of the experiment:

- Learn how to translate xUML constructs into mCRL2
- Stimulate discussions about interpretation of xUML
- Make preliminary assessment of feasibility of our verification strategy
- Serve as a first test case for an automatic translation

Our first experiment: Micro Interlocking

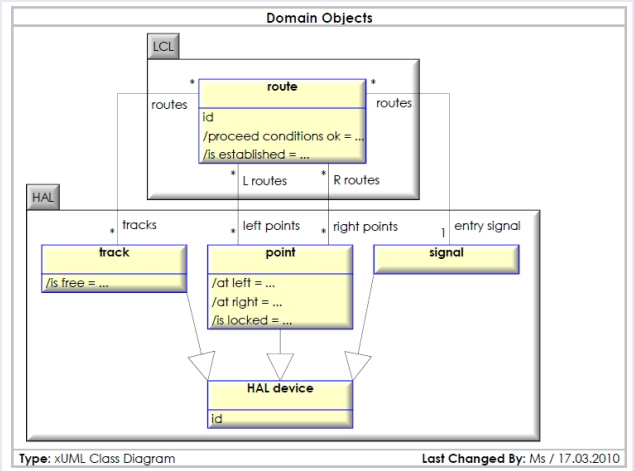
Common Core was only due late in project timeline. We started with a toy example manufactured by Markus Schacher of KnowGravity.

Purpose of the experiment:

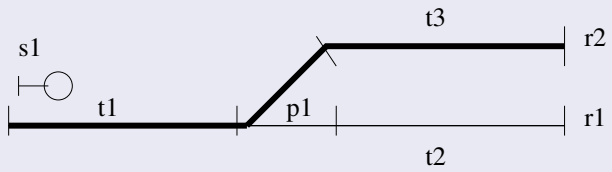
- Learn how to translate xUML constructs into mCRL2
- Stimulate discussions about interpretation of xUML
- Make preliminary assessment of feasibility of our verification strategy
- Serve as a first test case for an automatic translation

So what does Micro Interlocking look like?

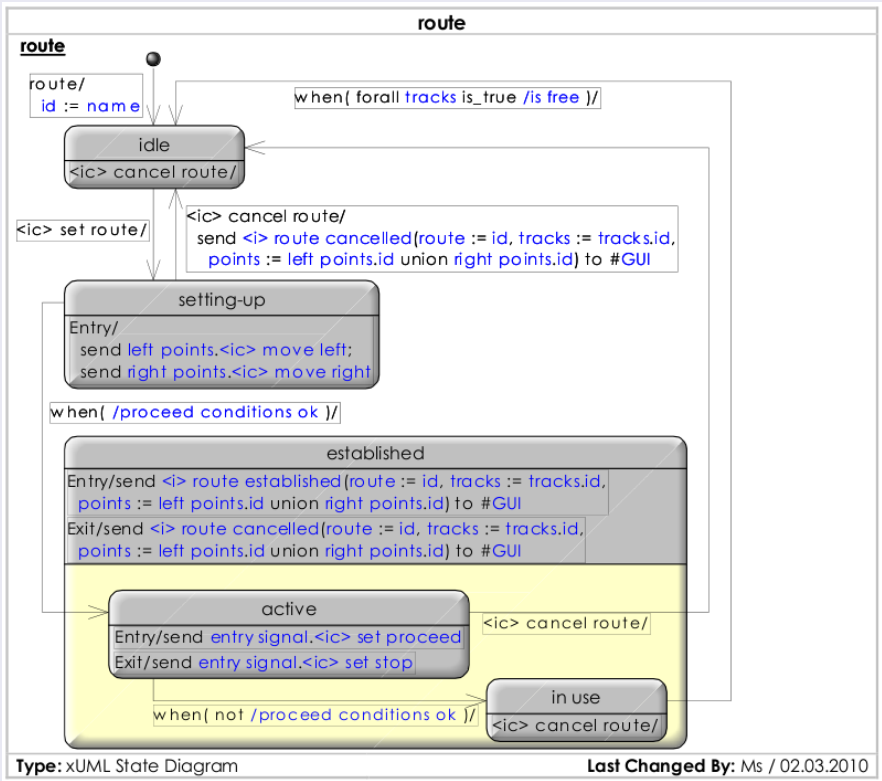
Class diagram:



Track layout:



One of the state diagrams:



Translation from xUML to mCRL2

Interpreting the xUML:

- UML semantics underspecified (what did the modeller mean?)
- Different runtime semantics and event priorities possible.

$\Rightarrow$ Translation must resolve ambiguities.

Expressing xUML in mCRL2:

- UML states translate to process parameters.
- UML events translate to actions.
- ...
- Unbounded object event pools $\Rightarrow$ infinite state space
- Non-local data access $\Rightarrow$ extra communication

Translation from xUML to mCRL2

Interpreting the xUML:

- UML semantics underspecified (what did the modeller mean?)
- Different runtime semantics and event priorities possible.

$\Rightarrow$ Translation must resolve ambiguities.

Expressing xUML in mCRL2:

- UML states translate to process parameters.
- UML events translate to actions.
- ...
- Unbounded object event pools $\Rightarrow$ infinite state space
- Non-local data access $\Rightarrow$ extra communication

Translation from xUML to mCRL2

Interpreting the xUML:

- UML semantics underspecified (what did the modeller mean?)
- Different runtime semantics and event priorities possible.

$\Rightarrow$ Translation must resolve ambiguities.

Expressing xUML in mCRL2:

- UML states translate to process parameters.
- UML events translate to actions.
- ...
- Unbounded object event pools $\Rightarrow$ infinite state space
- Non-local data access $\Rightarrow$ extra communication

Automated Translation from xUML to mCRL2

Implemented using model transformation technology Epsilon, developed in York.

Architecture:

- Parser for action and expression language (non-UML syntax)
- Transformation from UML to intermediate representation (iUML).
- Code generation from iUML to mCRL2.

Input:

- interlocking xUML model
- instance specification (track layout)
- safety property (as xUML model)

Output: mCRL2 specification

Automated Translation from xUML to mCRL2

Implemented using model transformation technology Epsilon, developed in York.

Architecture:

- Parser for action and expression language (non-UML syntax)
- Transformation from UML to intermediate representation (iUML).
- Code generation from iUML to mCRL2.

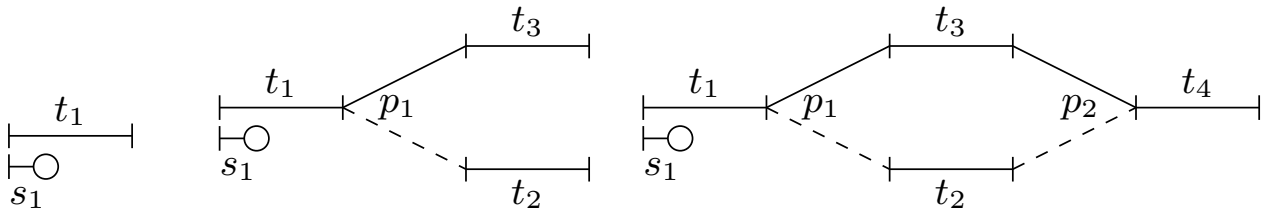
Input:

- interlocking xUML model
- instance specification (track layout)
- safety property (as xUML model)

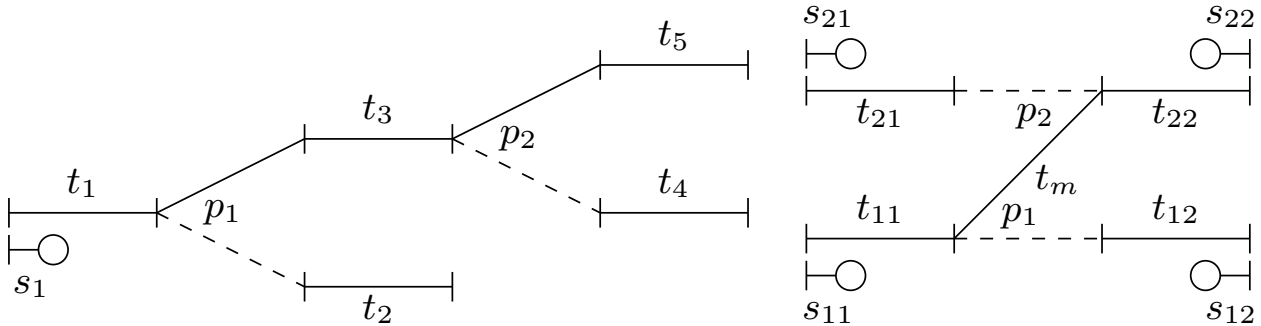
Output: mCRL2 specification

Verification of Micro Interlocking (1)

We instantiated Micro for the following track layouts:

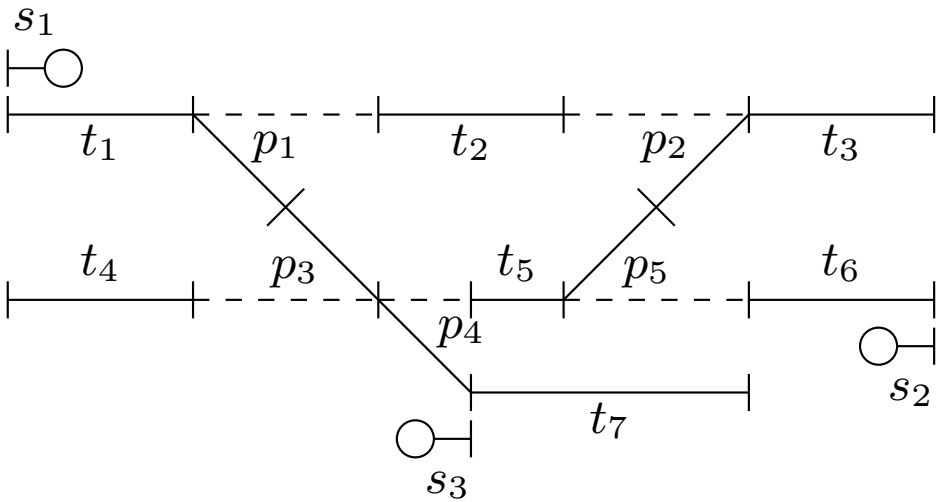


(a) Layout 1 (b) Layout 2 (c) Layout 3



(d) Layout 4 (e) Layout 5

Verification of Micro Interlocking (2)



(f) Layout 6

Verification: feasibility

Resource consumption in state space exploration:

Layout	Elts	Routes	States	Time	Mem (MB)
1	2	1	1.7×10^{4}	0.01 sec	61
2	5	2	1.3×10^{9}	0.25 sec	76
3	7	2	4.9×10^{11}	7.73 sec	86
4	8	3	8.9×10^{13}	19.39 sec	115
5	11	6	6.8×10^{23}	43 mins	3133
6	15	8	7.0×10^{31}	1.7 days	$>$ 32 GB

Safety property:

"A point belonging to an established route should not move."

Found violation already in layout 2.

Verification: feasibility

Resource consumption in state space exploration:

Layout	Elts	Routes	States	Time	Mem (MB)
1	2	1	1.7×10^{4}	0.01 sec	61
2	5	2	1.3×10^{9}	0.25 sec	76
3	7	2	4.9×10^{11}	7.73 sec	86
4	8	3	8.9×10^{13}	19.39 sec	115
5	11	6	6.8×10^{23}	43 mins	3133
6	15	8	7.0×10^{31}	1.7 days	$>$ 32 GB

Safety property:

"A point belonging to an established route should not move."

Found violation already in layout 2.

Verification of Micro Interlocking: Remarks

Micro is too simple to be safe (we expect errors!)

We have demonstrated that

- we can find errors.
- errors can be found in small track layouts.
- error traces can be reported for diagnostics.

We have demonstrated a methodology.

Verification of Micro Interlocking: Remarks

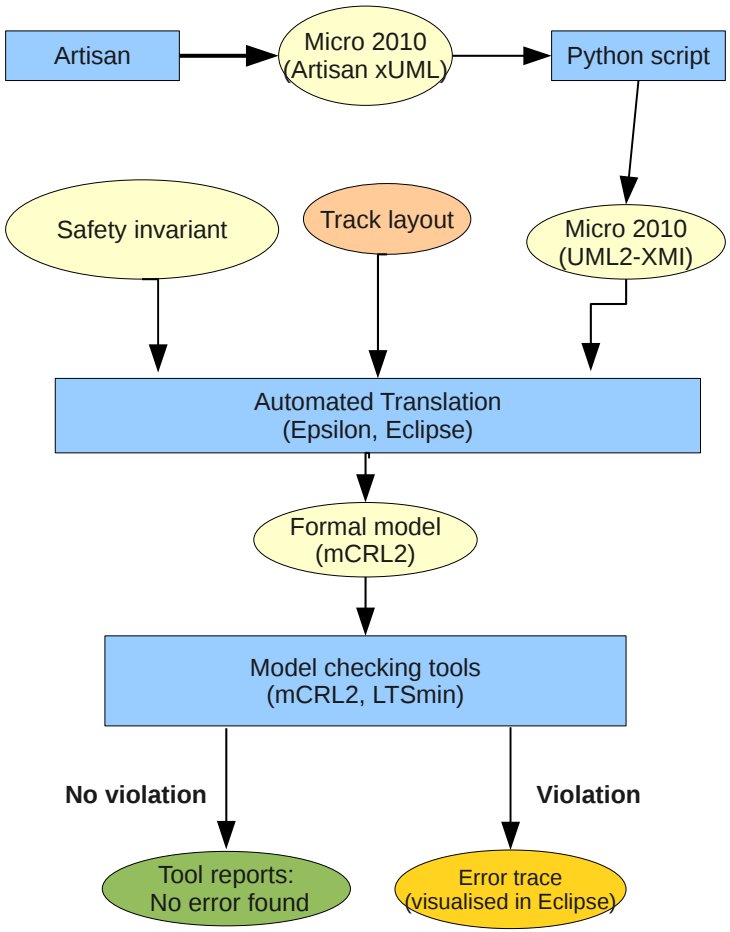
Micro is too simple to be safe (we expect errors!)

We have demonstrated that

- we can find errors.
- errors can be found in small track layouts.
- error traces can be reported for diagnostics.

We have demonstrated a methodology.

Prototype Tool Chain



A generic verification methodology

- Based on Eclipse development platform.
- Works for xUML models similar to INESS example models.
- Safety properties supplied as UML state machines.
- Error trace is visualised as UML sequence diagram.

Conclusion

Challenges encountered:

- xUML semantics is underspecified and ambiguous.
- State space explosion (still, errors may be found in small layouts).
- In theorem proving, abstraction refinement needs much human insight $\Rightarrow$ low degree of automation.

Future work:

- Verify Common Core xUML model (still to be delivered).
- Automatic translation from xUML to Promela (SPIN model checker) using iUML.
- Scalability (improve model checker, compositional verification, abstractions, ...)

Conclusion

Challenges encountered:

- xUML semantics is underspecified and ambiguous.
- State space explosion (still, errors may be found in small layouts).
- In theorem proving, abstraction refinement needs much human insight $\Rightarrow$ low degree of automation.

Future work:

- Verify Common Core xUML model (still to be delivered).
- Automatic translation from xUML to Promela (SPIN model checker) using iUML.
- Scalability (improve model checker, compositional verification, abstractions, ...)

INESS University Teams

Eindhoven (NL):

- Jos Baeten,
- Jan Friso Groote,
- Helle Hvid Hansen (PD)
- Bas Luttik,
- Mohammad Mousavi.

Twente (NL):

- Jeroen Ketema (PD),
- Jaco van de Pol.

(PD = PostDoc)

York (UK):

- Steve King,
- Richard Paige,
- Louis Rose (PD),
- Osmar Marchi dos Santos (PD),
- Jim Woodcock.

Southampton (UK):

- Vitaly Savicks (PD),
- Colin Snook.