THALES

Formal Model Validation through Acceptance Tests

Tomas Fischer, Thales Austria GmbH
Dana Dghaym, University of Southampton, UK
Chenyang Zhu, University of Southampton, UK

ENABLE-S3

This project has received funding from the ECSEL Joint Undertaking under grant agreement No 692455. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation program and Austria, Denmark, Germany, Finland, Czech Republic, Italy, Spain, Portugal, Poland, Ireland, Belgium, France, Netherlands, United Kingdom, Slovakia, Norway.

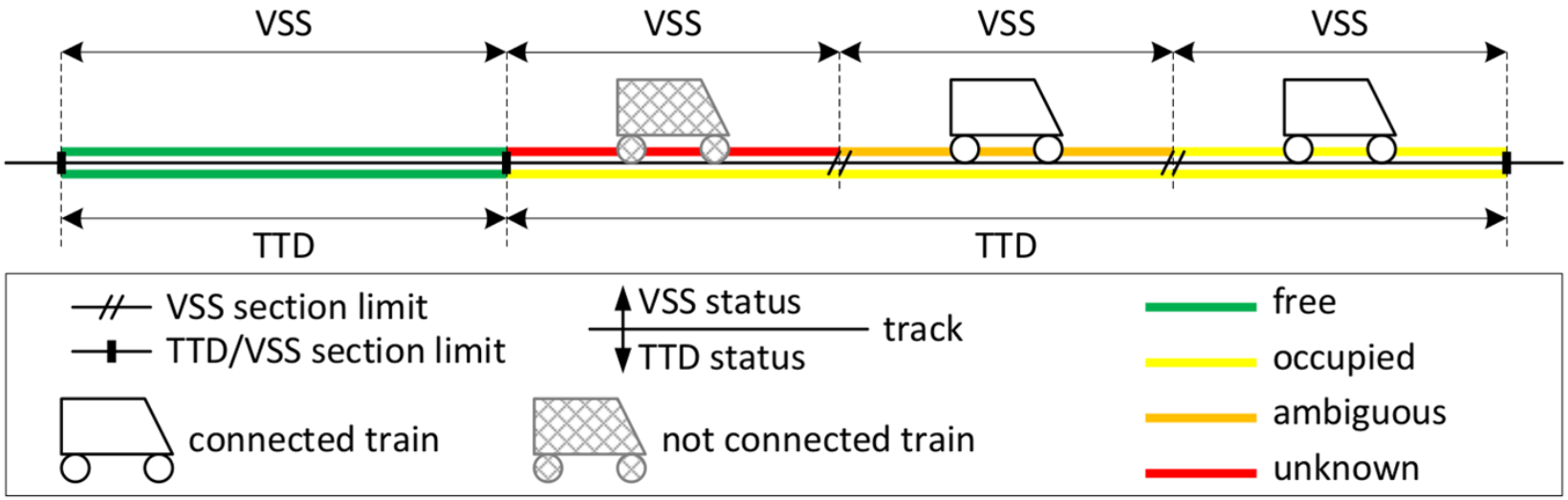
Use Case – Hybrid ERTMS/ETCS Level 3

- ETCS level 1 uses trackside radio beacons to pick up signal via signal adapters and transmit them to trains as Movement Authority (MA) to allow trains to cross one or more fixed block sections.
- ETCS level 2 uses GSM-R to transmit MA to trains from the Radio Block Center (RBC).
- ETCS level 3 departs from classic operation with fixed blocks. The MA is given on the information relating to the position, based on the actual distance of a train from next.
- In this level, trains do not have to stop to wait for the block while the previous train is moving, which ensures greater exploitation of the capacity of the line as it reduces the granularity of the spacing between trains

Use Case – Hybrid ERTMS/ETCS Level 3

- ETCS level 3 fully depends on the condition that RBC knows the position and integrity status of each train for all times while this condition cannot be fulfilled under current GSM-R communication as GSM-R communication is not reliable.
- The System need to be compatible with ETCS level 2
- Hybrid ETCS level 3 concept was developed to utilize the advantages of high performance of ETCS level3 and keep safety properties of the whole system by using trackside detection equipment to avoid collision under poor GSM-R communication

Use Case – Hybrid ERTMS/ETCS Level 3

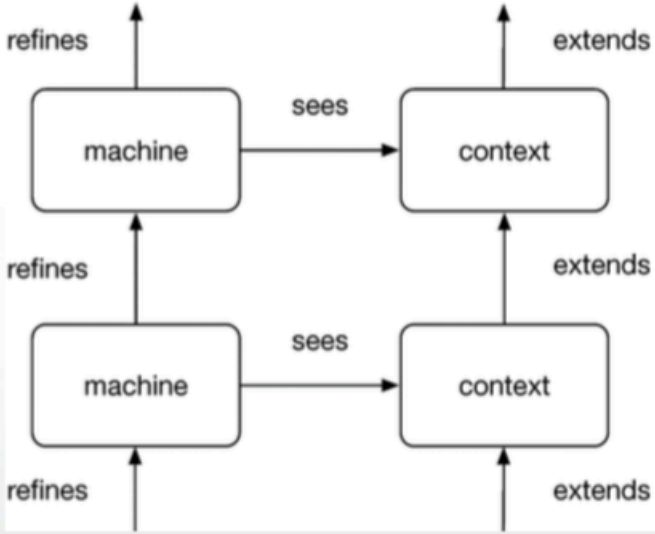


HL3 uses fixed virtual blocks for the separation of trains which are fitted with a train integrity monitoring system (TIMS), while a limited installation of trackside train detection is used for the separation of trains without TIMS, as well as for the handling of degraded situations.

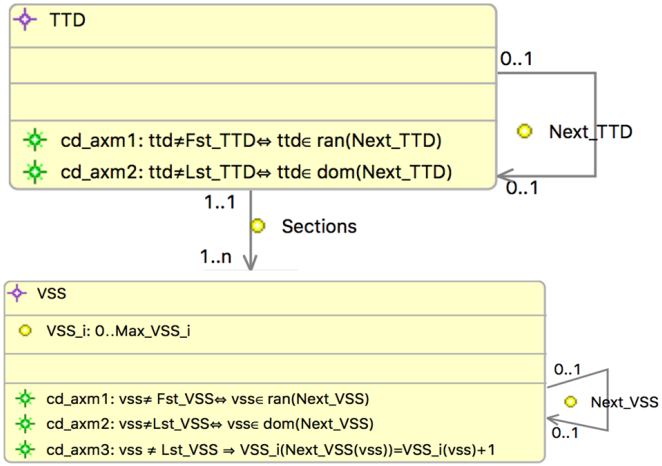
From: EEIG ERTMS Users Group: Principles: Hybrid ERTMS/ETCS Level 3

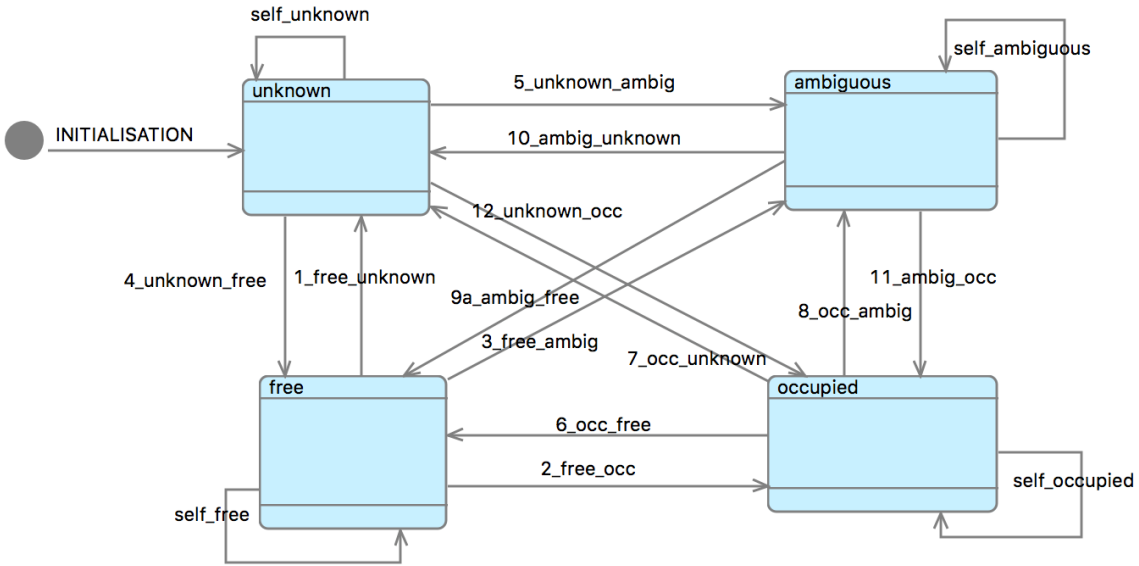
Event-B Model – iUML-B Class Diagrams and State Machines

- Formal method that develop mathematical models of discrete transition systems
- Abstraction/ Refinement
- Theorem proving
  - Invariant Preservation(INV)
  - Feasibility(FIS)
  - Guard Strengthening(GRD)
  - Numeric Variant(NAT)
  - Theorem(THM)
  - Well-definedness(WD)
- Model Checking (ProB)



Event-B Model – iUML-B Class Diagrams and State Machines





invariants

partition(VSS, unknown, occupied, ambiguous, free)

$\forall vss \cdot (vss \in free) \Rightarrow (free = availableVSS)$

Event-B Model – iUML-B → Event-B

```
event 4_unknown_free
 any vss
 where
  @isin_unknown: vss ∈ unknown
  @grd1: vss ∉ ran(reportedPosition) ∨ Sections~(vss) ∉ occupiedTTD
  @grd2: startVSSUpdate = TRUE
  @grd3: vss ∉ updatedVSS
 then
  @act1: updatedVSS ≔ updatedVSS ∪ {vss}
  @leave_unknown: unknown ≔ unknown ∖ {vss}
  @enter_free: free ≔ free ∪ {vss}
  @act_disconnectProp: disconnectPropagationTimer(vss) ≔ Idle
  @act_integProp: integrityLossPropagationTimer(vss) ≔ Idle
end
```

Motivation

What is the value of a fully proven model?

- 4_unknown_free
  - 4_unknown_free/grd1/WD
  - 4_unknown_free/distinct_states_in_VSS_STATE/INV
  - 4_unknown_free/free_invariant1/INV
  - 4_unknown_free/distinct_states_in_vss_sm/INV
  - 4_unknown_free/inv_gluing_available/INV
  - 4_unknown_free/inv_disconnect_timer/INV
  - 4_unknown_free/inv_integrity_timer/INV
  - 4_unknown_free/instanceType_vss/GRD
  - 4_unknown_free/act1/SIM

Element Name	Tot.	Auto	Man.	Rev.	Und.
Total	**69**	**67**	**2**	**0**	**0**
1_free_unknown	10	10	0	0	0
2_free_occ	7	7	0	0	0
3_free_ambig	7	7	0	0	0
4_unknown_free	9	9	0	0	0
5_unknown_ambig	4	4	0	0	0
6_occ_free	7	7	0	0	0
7_occ_unknown	2	2	0	0	0
8_occ_ambig	2	2	0	0	0
9_ambig_free	8	8	0	0	0
10_ambig_unknown	4	4	0	0	0
11_ambig_occ	4	3	1	0	0
12_unknown_occ	5	4	1	0	0

Maybe none…

…if it doesn't represent customer needs

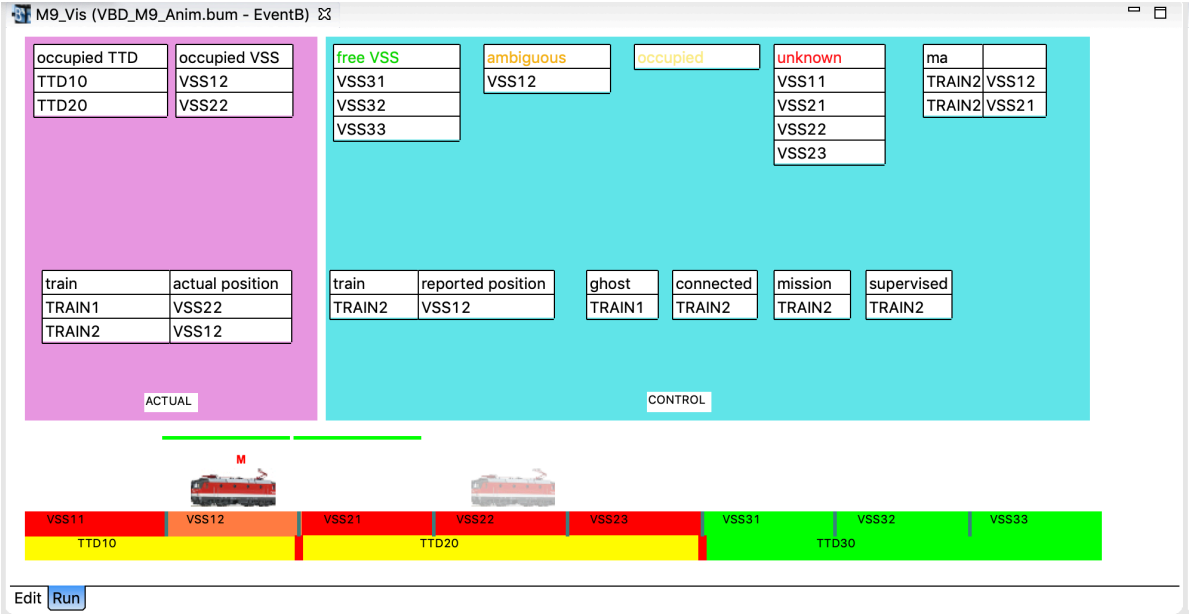
Verification – Validation

Verification – “build the thing right”

- Proofs
- Model checking

Validation – “build the right thing”

- Review
- Visualization, Animation
- **Acceptance tests**



OPEN

THALES

Behavior Driven Development (BDD)

Acceptance tests serve as Requirements

- Common language for domain experts and developers (single source of truth)
  - Gherkin/Cucumber is state-of-the-art for BDD
- Human readable, yet still automatically executable

Language gap

- Domain model
  - Entities with attributes and relationship; state machines and activities
  - #4: *Integer train reconnects within the same session* **or** *TTD is free*
- Formal model
  - Set theory and 1st order predicate logic
  - *@grd1: vss ∉ ran(reportedPosition) ∨ Sections~(vss) ∉ occupiedTTD*

Gherkin – Cucumber

Feature – a piece of business value

- Narrative description: **As a** «role» **I want** «feature» **so that** «business value»

Scenario – one use case

- Examples may separate particular test data from the list of steps

Steps

- **Given** «precondition» **When** «interaction» **Then** «postcondition» **And/But** …

Cucumber for Event-B – execute Gherkin scenarios on an Event-B model

- Collection of step definitions for Event-B state space traversal
- Developed by Thales Austria GmbH, released under EPL 2.0
  https://github.com/tofische/cucumber-event-b

Cucumber for Event-B

Plain Event-B

Given is enabled event "«Name»"
When fire event "«Name»" with "«Formula»"
Then is TRUE formula "«Formula»"

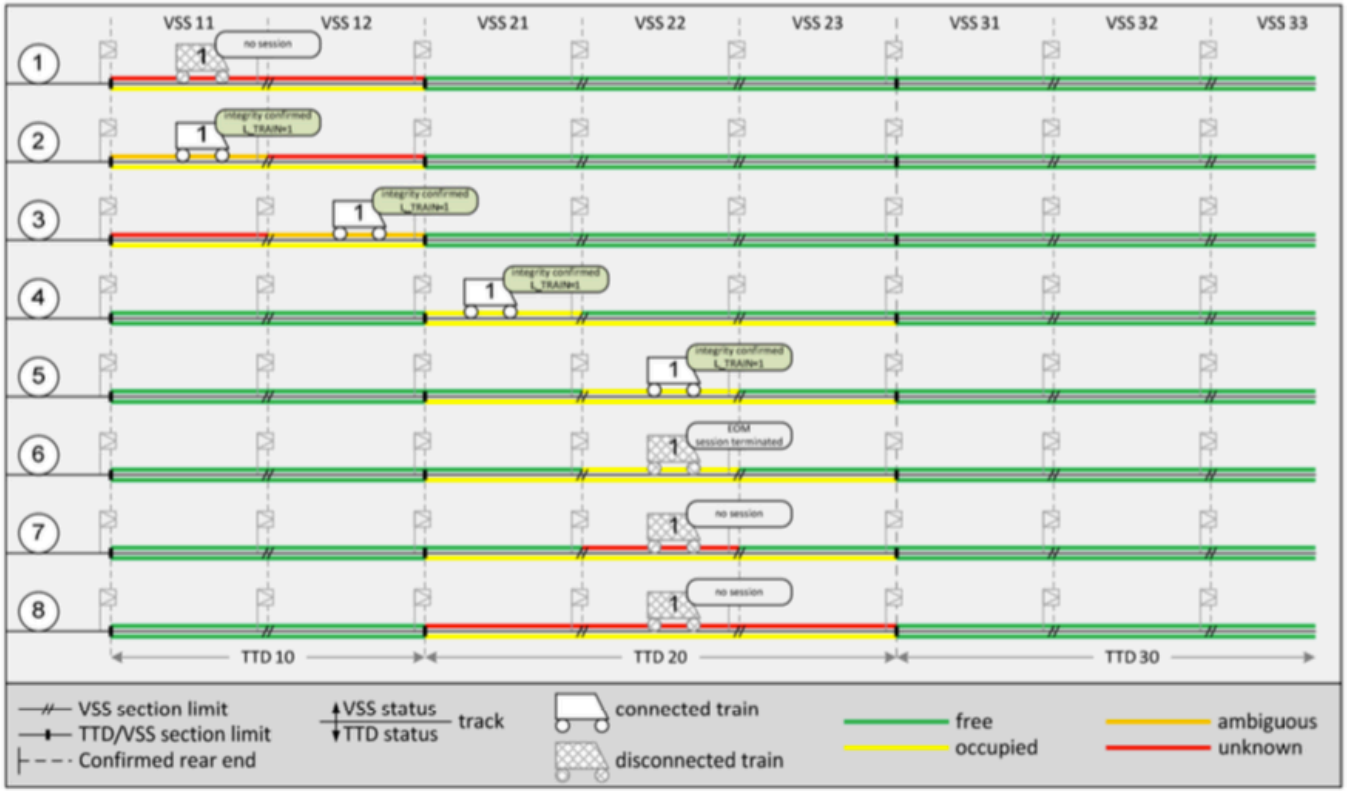
iUML-B Class Diagrams

Given is enabled method "«Name»"
When call method "«Name»"
Then attribute "«Name»" is "«Value»"

iUML-B State Machines

Given is enabled transition "«Name»"
When trigger transition "«Name»"
Then is in state "«State»"

Cucumber for iUML-B – Example



THALES

Cucumber for iUML-B – Example

Scenario: Enter HL3 area

When fire event "ENV_enter_HL3_area" with "tr=TRAIN1"

And fire event "VBD_start_vss_update"

And fire event "1A_free_unknown" with "ttd=TTD10 & vss=VSS11"

And fire event "1A_free_unknown" with "ttd=TTD10 & vss=VSS12"

And fire event "self_free" with "vss=VSS31"

…

And fire event "VBD_vss_update_complete"

Then is TRUE formula "free = {VSS21,VSS22,VSS31,VSS32,VSS33}"

Then is TRUE formula "occupied = {}"

Then is TRUE formula "ambiguous = {}"

Then is TRUE formula "unknown = {VSS11,VSS12}"

Domain specific data description

Different viewpoints

- Class instance: row by row
  - One variable contains the value of all attributes for one instance
- Set theory: column by column
  - One variable contains the value of one attribute for all instances

Implicit constants and/or variables

- State machine states (enumeration or variables translation)
- Associations (with different cardinality)

Future work

- Complex data types
- External data sources (e.g. RailML for Topology data)

Challenges and Limitations

Quality of acceptance tests

- Verification and validation of tests themselves – easier due to domain language
- Adequate test coverage – according to which measures?
- Realistic test data

Focus on functional behaviour

- Safety properties
- Liveness properties

Wrong generalisation of scenarios

- All provided example succeed – but the model is still wrong
- *General TDD / BDD problem*

Summary

Conclusion

- Demonstrated applicability of this approach to a real-word use case
- Confirmed benefits of validating a formal model using acceptance tests
- Pointed out how to close the language gap
- Proposed measures to mitigate identified drawbacks

Recommendation

- Adopt the BDD methodology already during the requirement elicitation

Further work

- Show the conformity of the implementation with respect to the formal model
  - Execute the same acceptance tests on an implementation
  - Augment by model generated test cases for corner cases