ENABLE-S3

ENABLE-S3
ECSEL EU PROJECT

Formal Modelling Techniques for Efficient Development of Railway Control Products

Michael Butler, Dana Dghaym, Thai Son Hoang, **<u>Colin Snook</u>**,

Tomas Fischer, Klaus Reichl, Peter Tummeltshammer

UNIVERSITY OF Southampton
School of Electronics
and Computer Science

THALES

Enable-S3 - Horizon 2020 project

Objective:

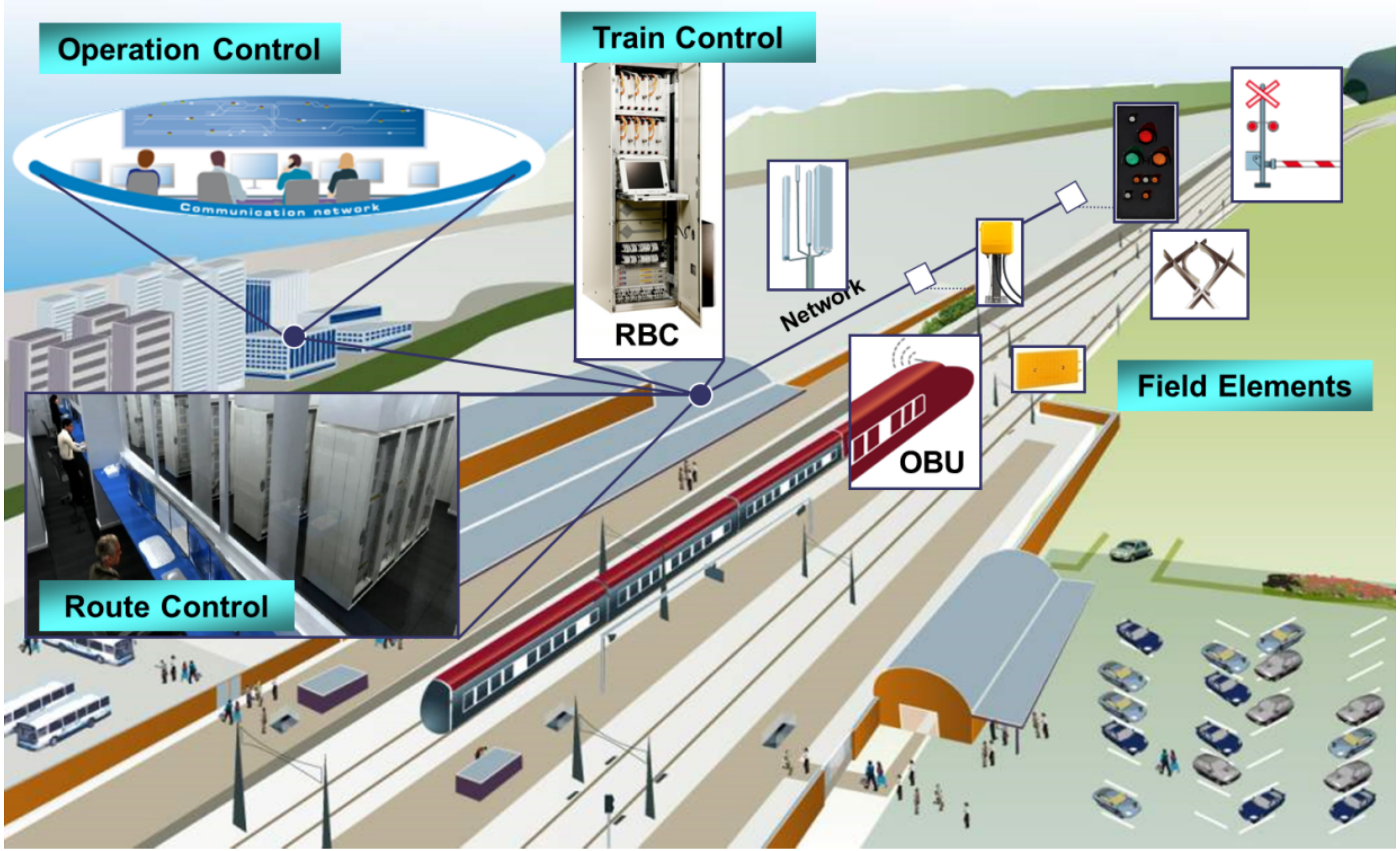
establish cost-efficient cross-domain virtual and semi-virtual V&V platforms and methods for ACPS

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 programme and Austria, Denmark, Germany, Finland, Czech Republic, Italy, Spain, Portugal, Poland, Ireland, Belgium, France, Netherlands, United Kingdom, Slovakia, Norway.

Outline

- **Introduction**
- Railground – requirements
- Background – modelling techniques
- Railground - Model
- Summary and Conclusions

Railway products



Issues:

- Commercial
- Safety with variability

Trends:

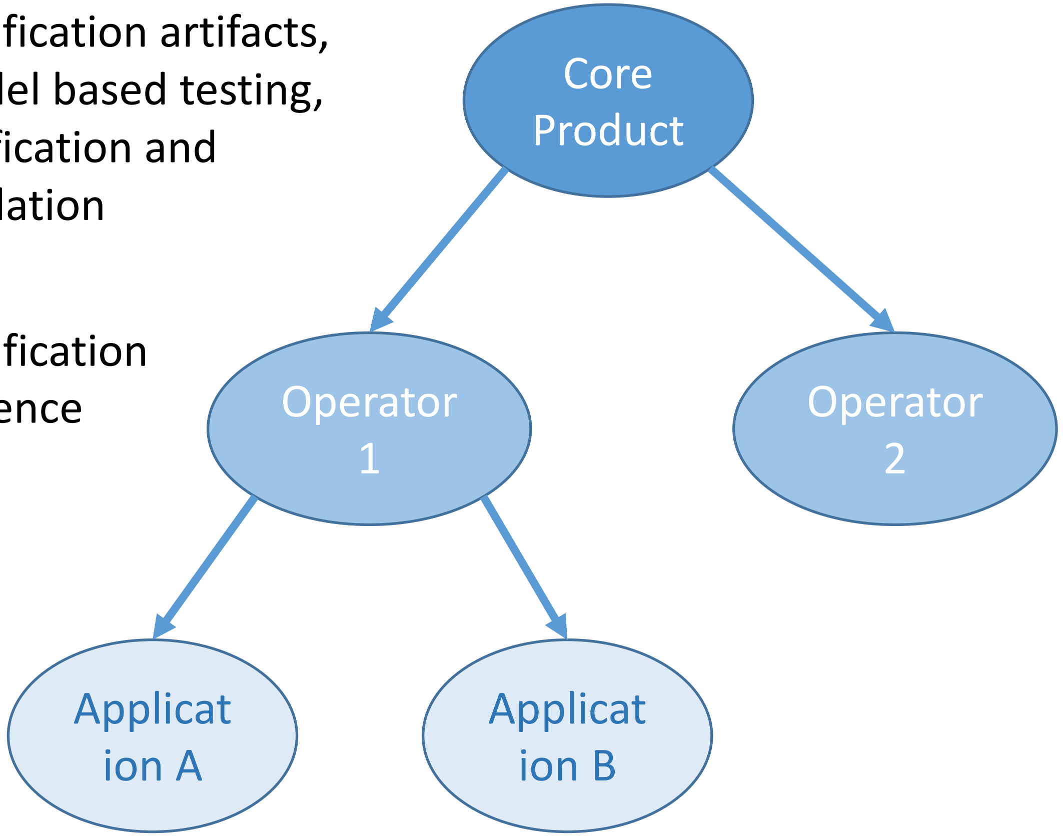
- Removal of some field elements (Signals, TTD, …), replacing by intelligent algorithms.
- Remote movement authority, possibly in clouds
- Central operation centers, split between safety critical and non-safety critical

OBU - Onboard Unit (ETCS compliant)
RBC - Radio Block Center
TTD - Trackside Train Detection

Railway Control Product Line

Re-usable safety certification artifacts, Model based testing, verification and validation

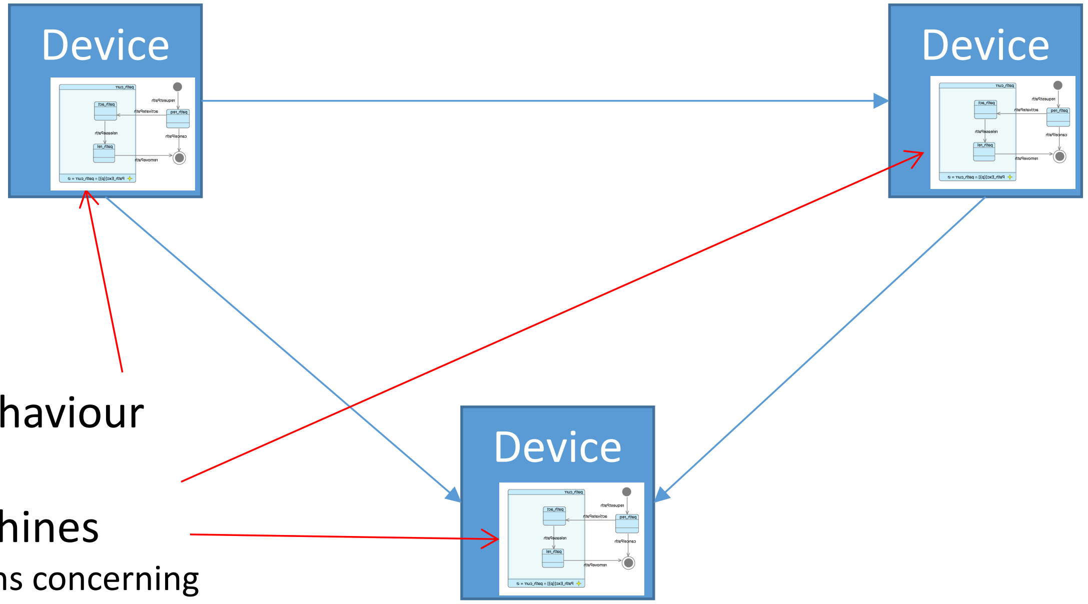
Certification Evidence



Model based testing - Conformity of implementation with model

- Core Product
  - Generic (partially abstract)
  - Strongly verified
  - Formally proven (FM experts)
- Operator Specific Product
  - Customer Rules (e.g. Signals)
  - Implementation technology
  - Refinement of Core model
  - FM aware domain engineers
- Application Specific Instantiation
  - Configuration tools
    - (e.g. Station Layout)
    - Instantiated to meet Axioms
  - Application Engineers (non-FM)

Cyber Physical Systems



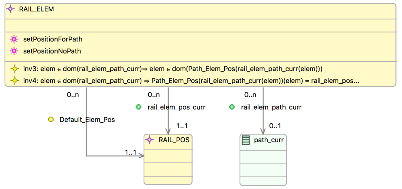
Local state and behaviour

(using)

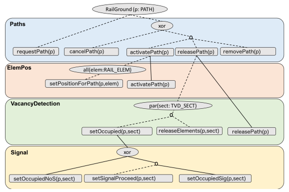
iUML-B State Machines

(with guards and actions concerning attributes and associations)

Interconnected devices
up one meta-level
Entity Relationship diagrams
using
iUML-B Class Diagrams


RAIL_ELEM
setPositionForPath
setPositionNoPath
inv3: elem ∊ dom(rail_elem_path_curr)⇒ elem ∊ dom(Path_Elem_Pos(rail_elem_path_curr(elem)))
inv4: elem ∊ dom(rail_elem_path_curr) ⇒ Path_Elem_Pos(rail_elem_path_curr(elem))(elem) = rail_elem_pos...
0..n
0..n
0..n
rail_elem_pos_curr
rail_elem_path_curr
Default_Elem_Pos
1..1
0..1
1..1
RAIL_POS
path_curr

System Behaviour
consisting of
Sequences of local events
using
Event Refinement Diagrams



We need to analyse the holistic system behaviour… Not enough to verify individual devices

Outline

- Introduction
- **Railground – requirements**
- Background – modelling techniques
- Railground - Model
- Summary and Conclusions

Railground - Topology

Element (e.g. T, P1, P2)

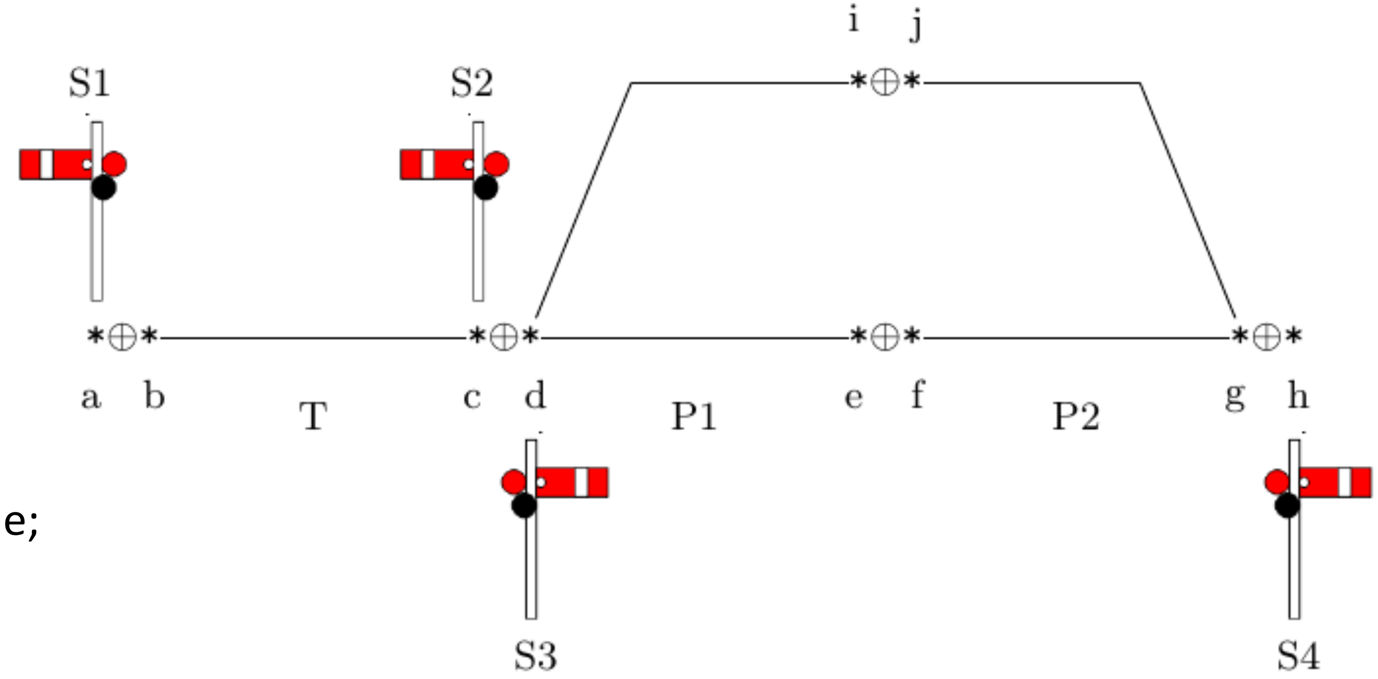
Connector (*) - a port of an Element
(e.g. P1 has 3 connectors: d,i,e)

Connectors are connected ⊕
to Connectors of other Elements
(e.g. c is connected to d)

Segment - a direction that an Element can provide;
given by an ordered pair of its connectors:
(e.g. P1 has 4 segments:- di, id, de, ed)

Position – a distinct subset of an Element's
segments that it can provide in a particular state.
(e.g. P1 has three positions:

1) 'Left' = {di,id}
2) 'Right' = {de, ed}
3) 'in transition (or broken)' = {}



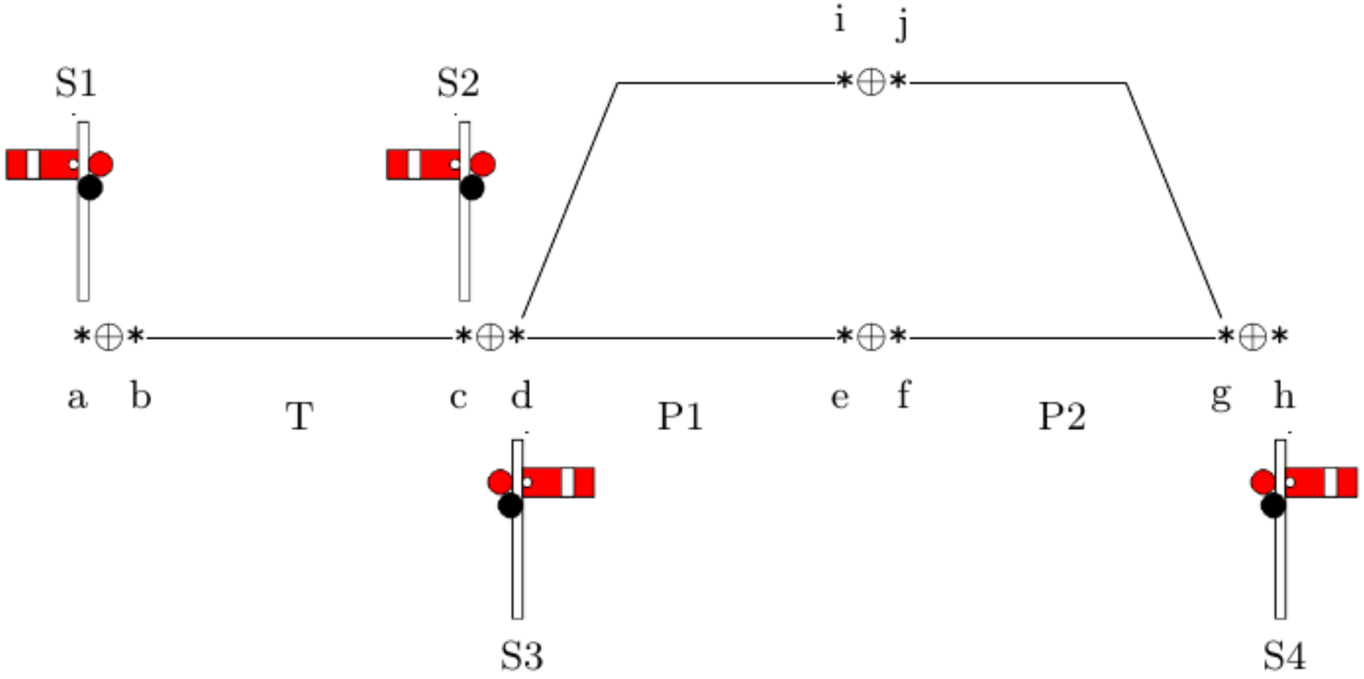
Railground - Paths

Path – a sequence of **Segment**s.

A path cannot contain two segments from the same Element.

The elements of a path are released from it as soon as the train passes through that segment

States of a path:
 Requested => elements are being positioned
 Active => trains can use the path (and its elements cannot be moved)
 Released => all elements have been released from the path

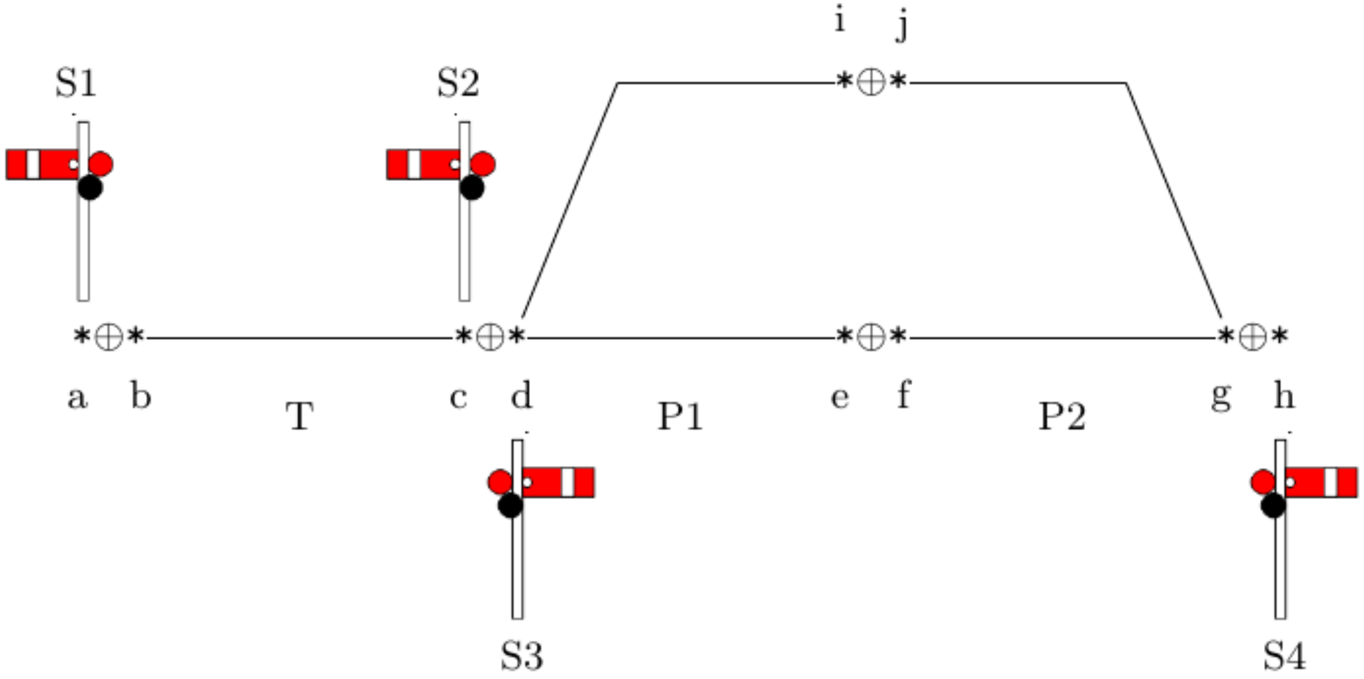


Railground – Vacancy Detection

TVD section
(Track Vacancy Detection)
A set of segments that belong to a train detection device.

State = vacant | occupied

Each segment belongs to exactly one TVD

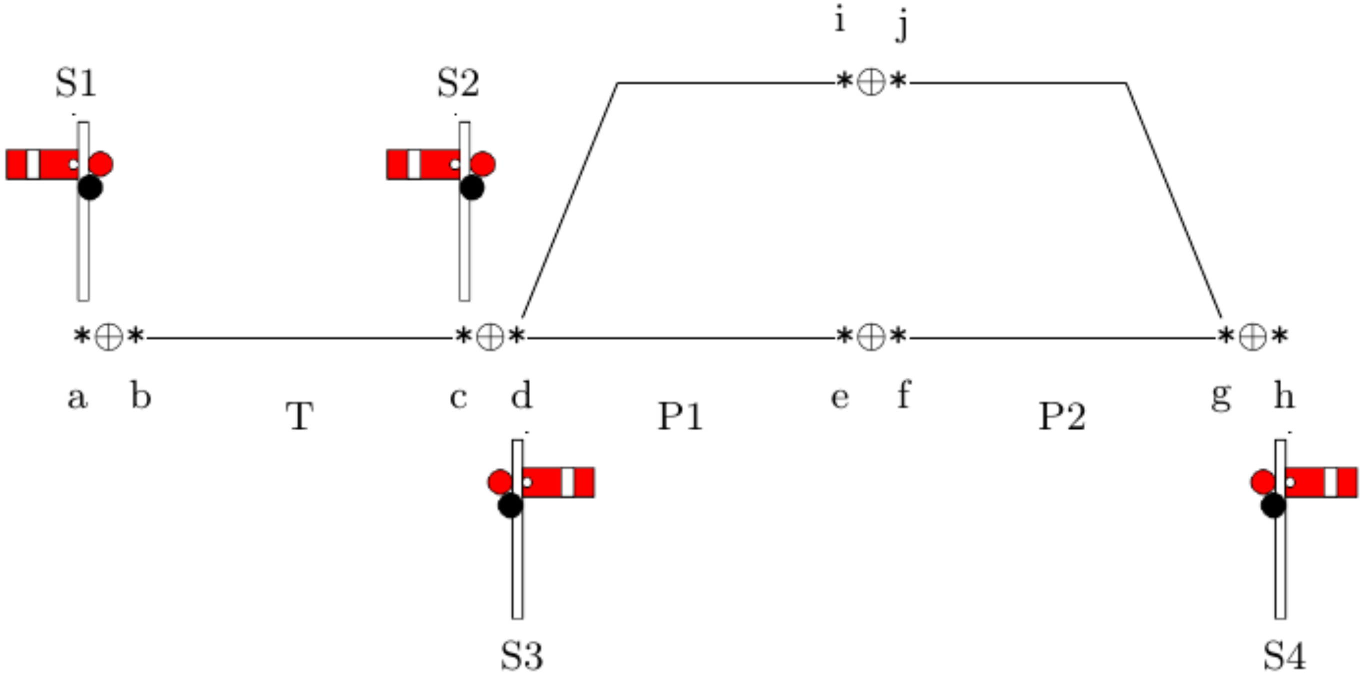


Railground

Signal – signals are attached to connectors and control trains from using the segments that follow that connector

States = default | not default

Signal aspect is abstracted to default (stop) or not default (proceed). This can be refined with different operator requirements when producing a customer specific product

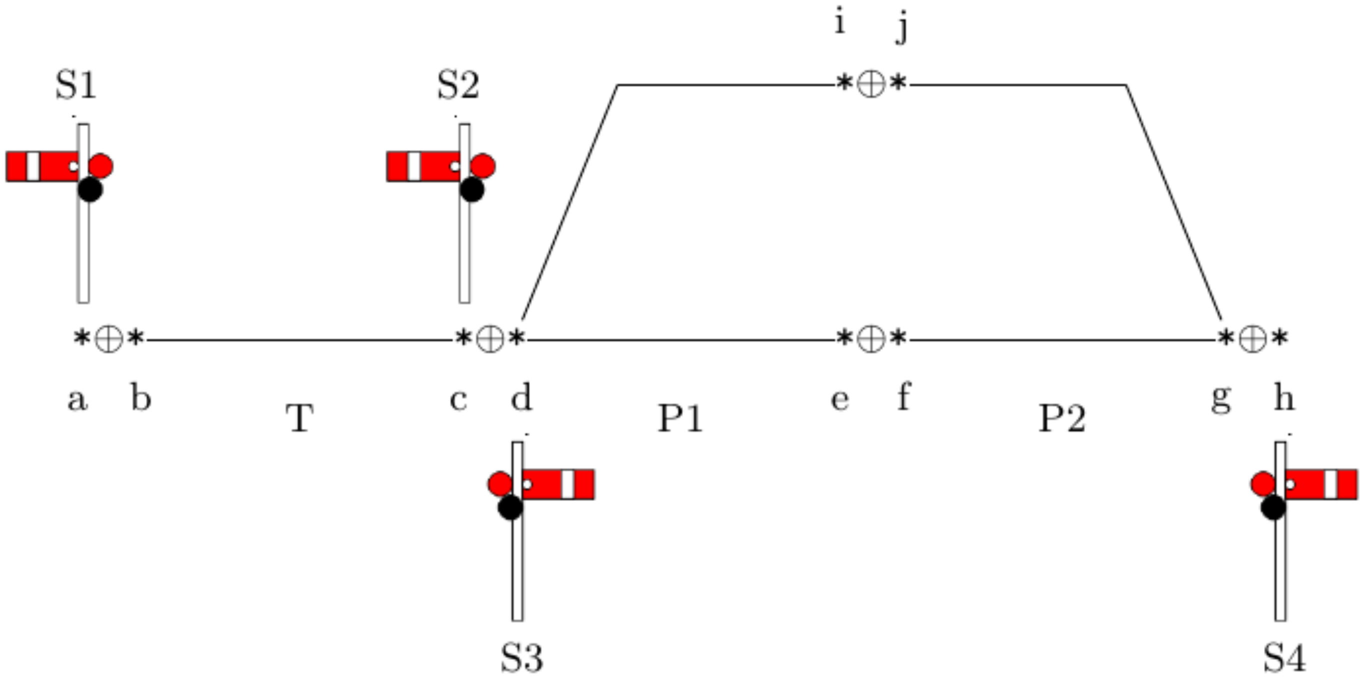


Railground – Safety Requirements

- Two active paths cannot overlap (avoid collisions)

- An active path must have all its elements in the correct positions (avoid derailment)

- A path can only be requested if it is disjoint from other active or requested paths <*lower level design condition*>



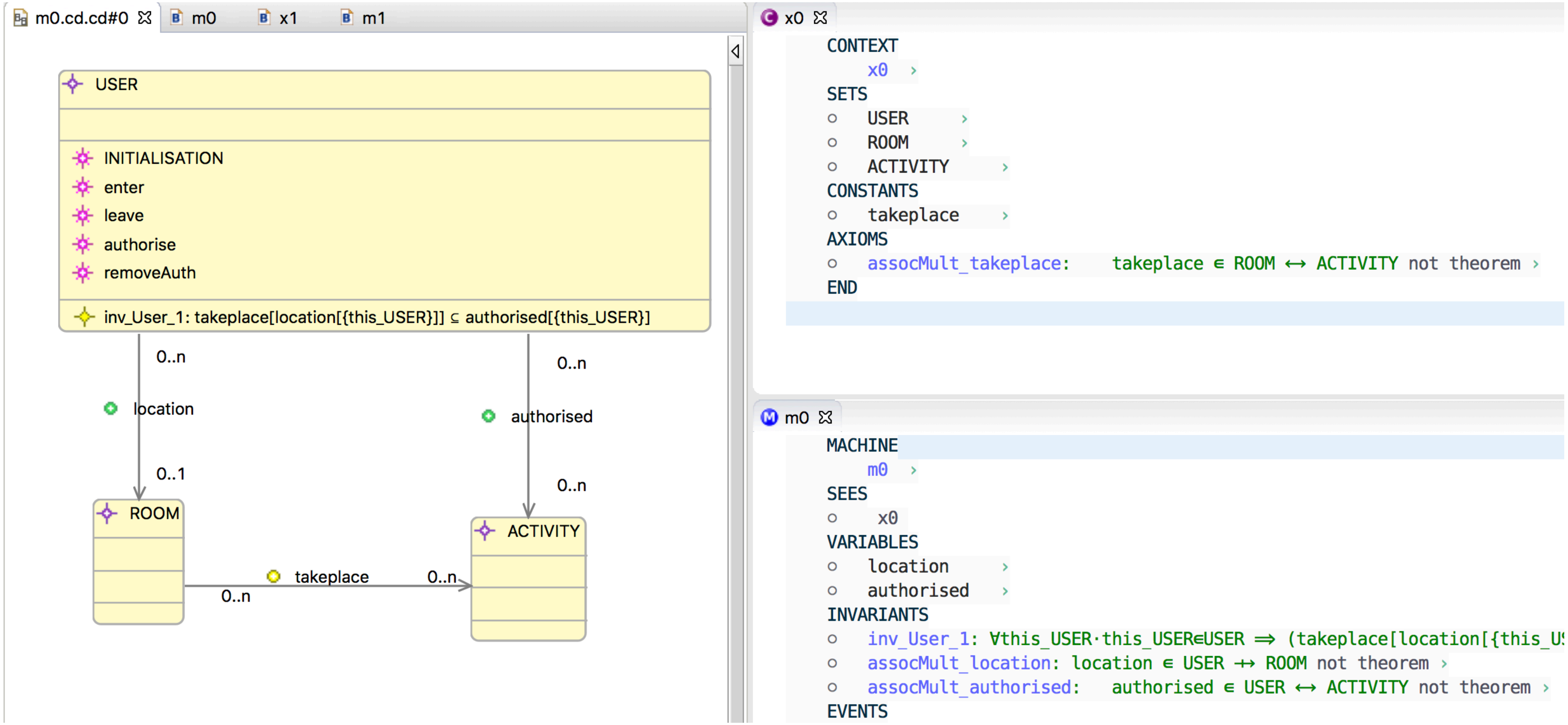
Outline

- Introduction
- Railground – requirements
- **Background – modelling techniques**
- Railground - Model
- Summary and Conclusions

Event-B

- Systems modelling language
  - State - Represented by Typed Variables
  - Guarded events that alter the state
  - Refinement - more detailed state reveals more detailed events
- Formal modelling environment and toolset **(Rodin)**
  - Static checker
  - Automatic Proof tools
- Model Checker (ProB)
  - Invariant violations, refinement simulation errors
  - Animation mode for validation

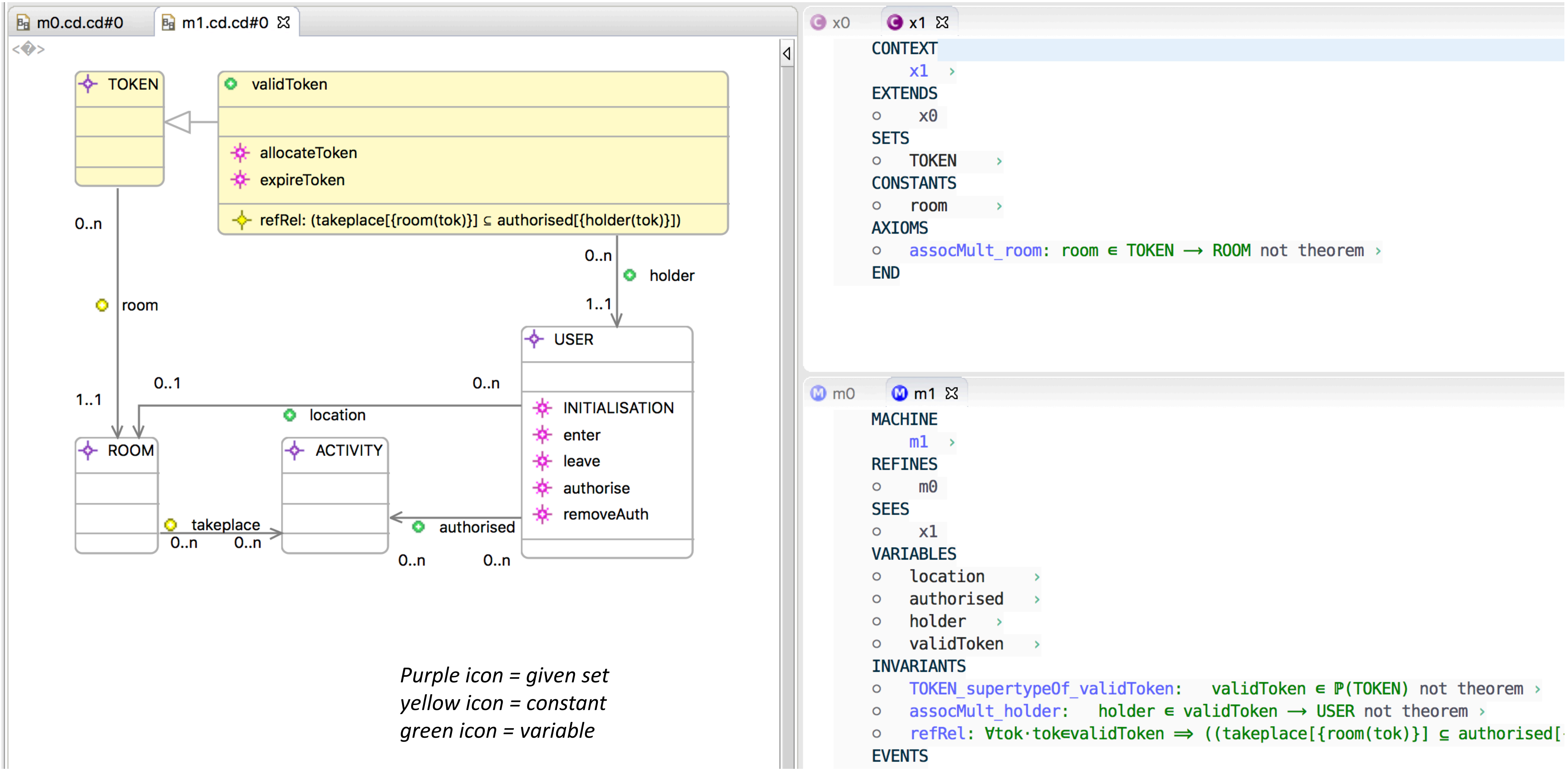
Overview of iUML-B Class Diagrams



```
CONTEXT
    x0 ›
SETS
○   USER ›
○   ROOM ›
○   ACTIVITY ›
CONSTANTS
○   takeplace ›
AXIOMS
○   assocMult_takeplace:   takeplace ∈ ROOM ↔ ACTIVITY not theorem ›
END
```

```
MACHINE
    m0 ›
SEES
○    x0
VARIABLES
○   location ›
○   authorised ›
INVARIANTS
○   inv_User_1: ∀this_USER·this_USER∈USER ⇒ (takeplace[location[{this_U
○   assocMult_location: location ∈ USER ⇸ ROOM not theorem ›
○   assocMult_authorised:   authorised ∈ USER ↔ ACTIVITY not theorem ›
EVENTS
```

Overview of iUML-B Class Diagrams

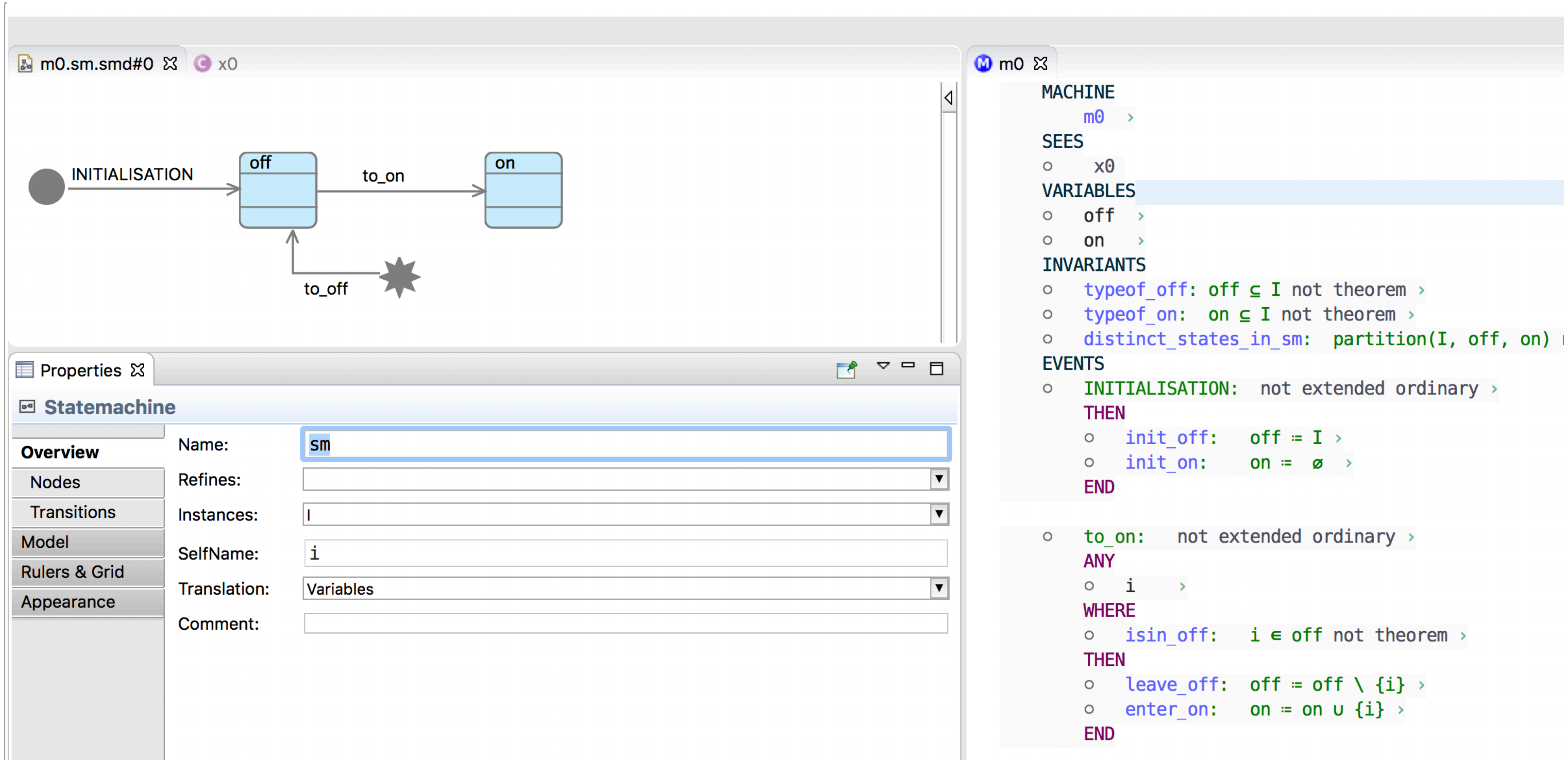


```
CONTEXT
    x1 ›
EXTENDS
  ○  x0
SETS
  ○  TOKEN ›
CONSTANTS
  ○  room ›
AXIOMS
  ○  assocMult_room: room ∈ TOKEN ⟶ ROOM not theorem ›
END
```

```
MACHINE
    m1 ›
REFINES
  ○  m0
SEES
  ○  x1
VARIABLES
  ○  location ›
  ○  authorised ›
  ○  holder ›
  ○  validToken ›
INVARIANTS
  ○  TOKEN_supertypeOf_validToken:   validToken ∈ ℙ(TOKEN) not theorem ›
  ○  assocMult_holder:   holder ∈ validToken ⟶ USER not theorem ›
  ○  refRel: ∀tok·tok∈validToken ⇒ ((takeplace[{room(tok)}] ⊆ authorised[
EVENTS
```

Purple icon = given set
yellow icon = constant
green icon = variable

Overview of iUML-B State-machines



Name: sm

Refines:

Instances: I

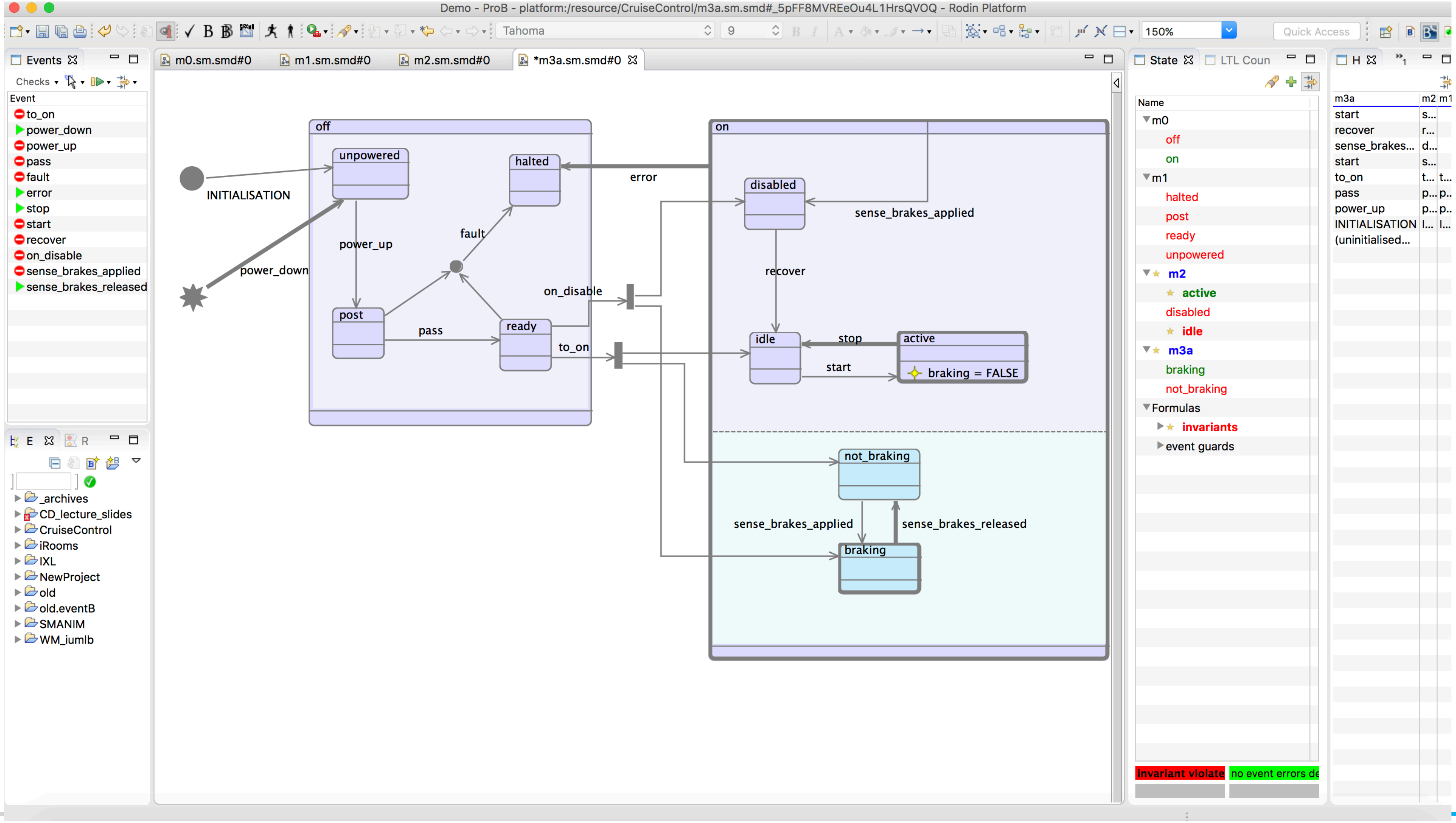
SelfName: i

Translation: Variables

Comment:

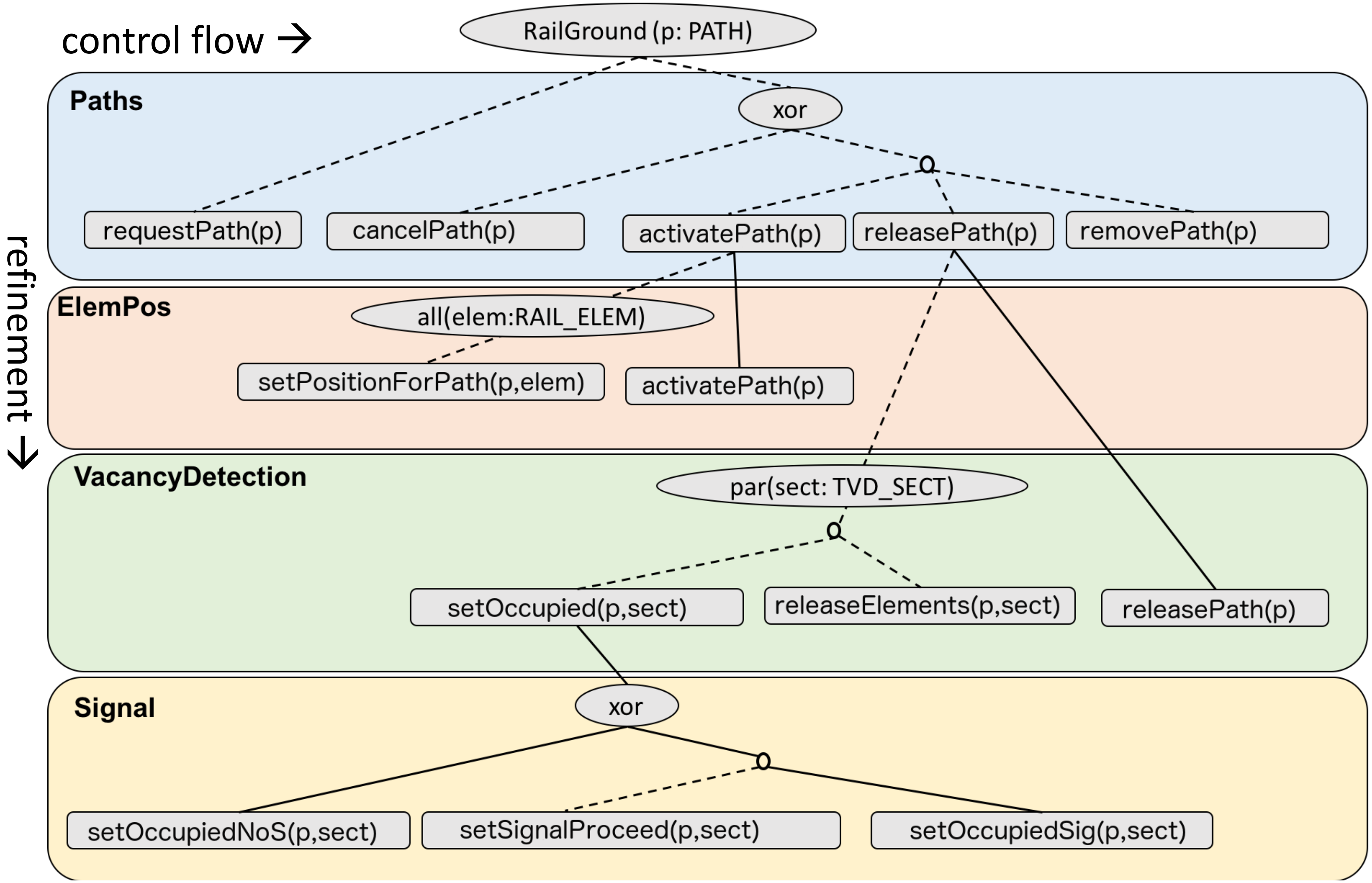
```
MACHINE
    m0  ›
SEES
○    x0
VARIABLES
○   off  ›
○   on   ›
INVARIANTS
○   typeof_off: off ⊆ I not theorem ›
○   typeof_on:  on ⊆ I not theorem ›
○   distinct_states_in_sm:  partition(I, off, on)
EVENTS
○   INITIALISATION:  not extended ordinary ›
    THEN
    ○   init_off:   off ≔ I ›
    ○   init_on:    on ≔  ∅  ›
    END

○   to_on:    not extended ordinary ›
    ANY
    ○   i     ›
    WHERE
    ○   isin_off:   i ∈ off not theorem ›
    THEN
    ○   leave_off:  off ≔ off \ {i} ›
    ○   enter_on:   on ≔ on ∪ {i} ›
    END
```

State-machine animation



Event Refinement Structures (ERS)

- In Event-B refinements we decompose events into new and refining events
- ERS visualises this event refinement
  - Solid line => event refinement
- Also shows control flow of events
  - Read from left to right
  - Combinator operators for iteration, choice, interleaving
- Inspired by Jackson Structured Diagrams (JSD)

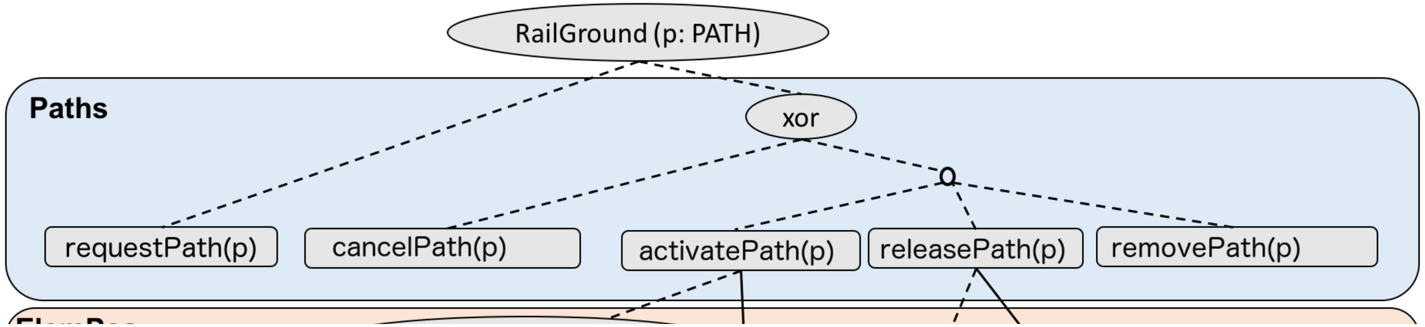


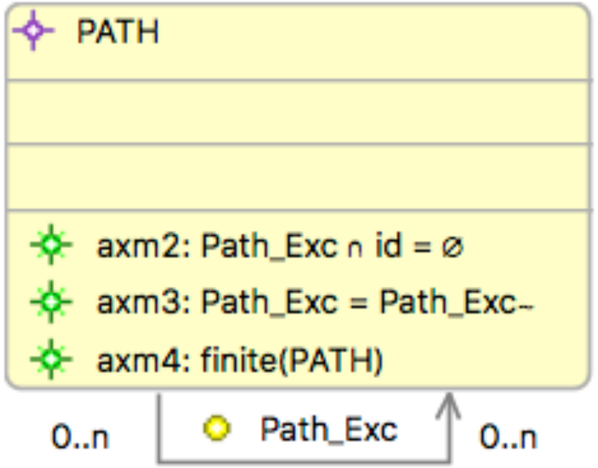
Outline

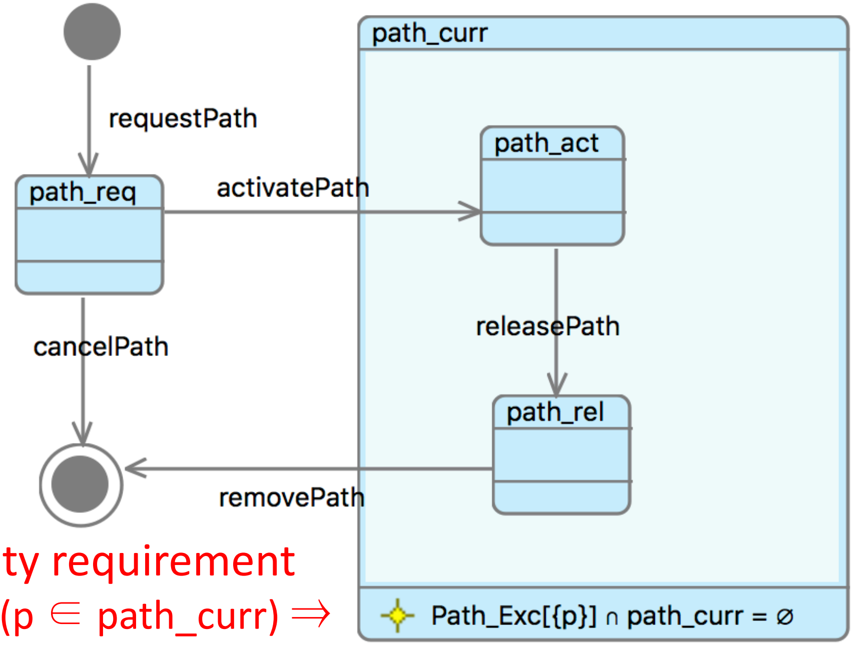
- Introduction
- Railground – requirements
- Background – modelling techniques
- **Railground - Model**
- Summary and Conclusions

Abstract Model - Paths

- We start with paths since they feature prominently in the safety requirements
- Association Path_Exc models conflicts between Paths
    - Paths cannot conflict with themselves
    - Conflict is symmetric
- iUML-B State-machine shows the lifecycle of a path
- State invariant ensures that if a path,p, is current, none of its conflicting paths are current





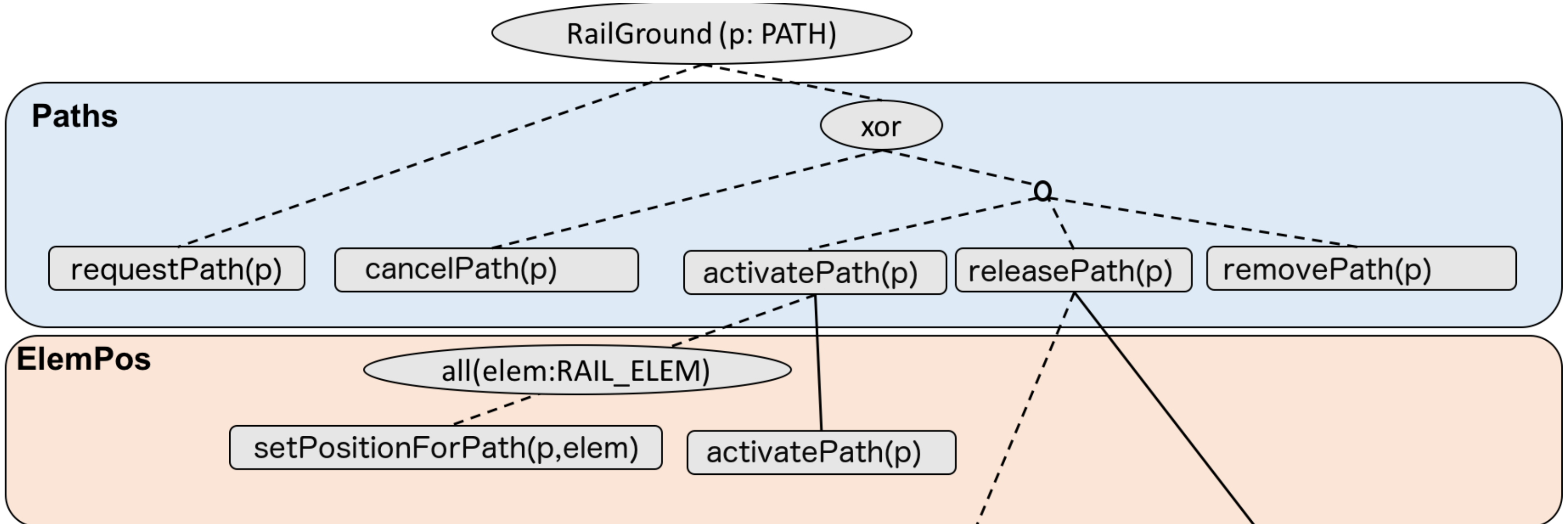


safety requirement
$\forall p \cdot (p \in path\_curr) \Rightarrow$

First Refinement – Elements and position

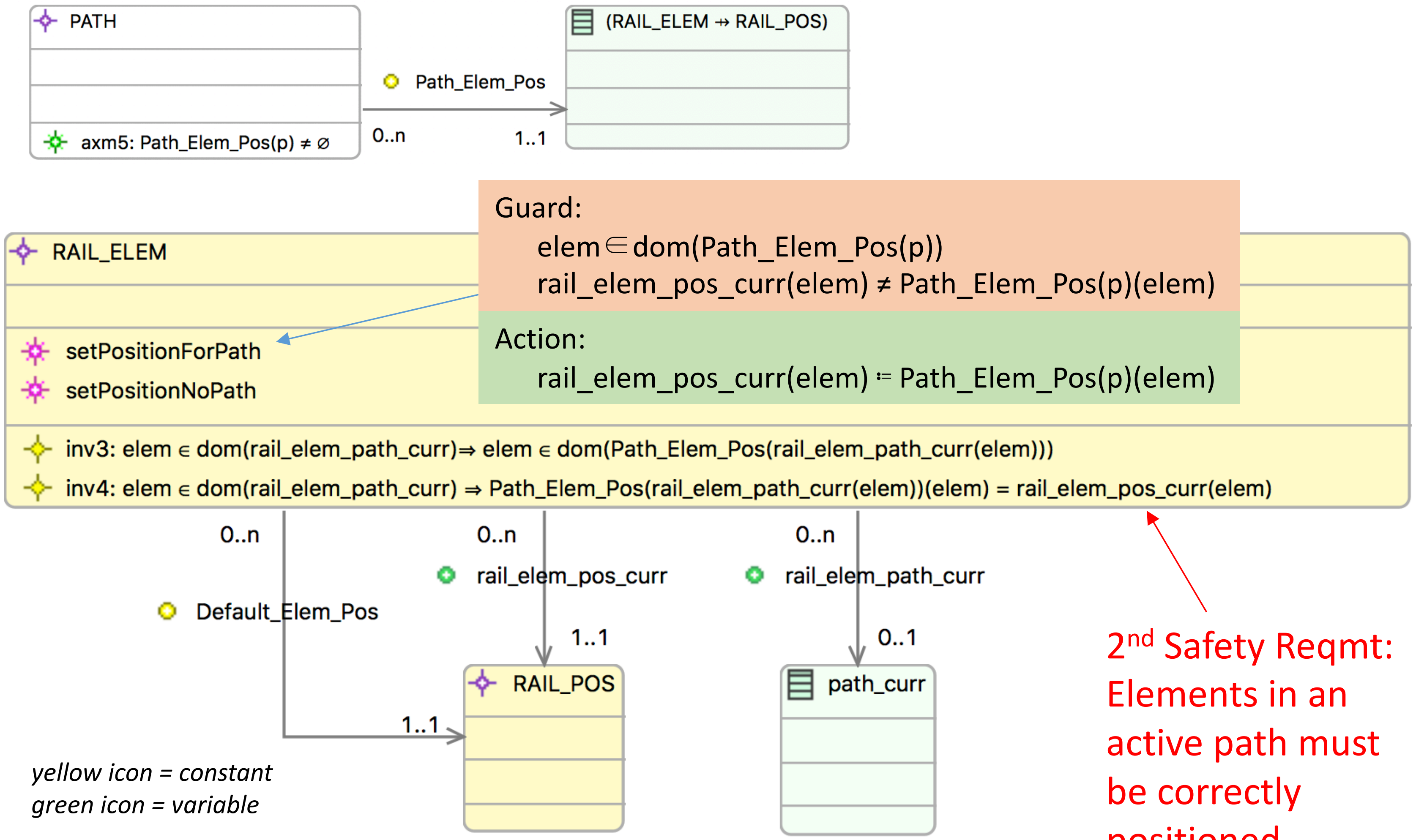
- setPositionForPath is a preliminary step to activatePath.

- It must be completed for all elements in the path before the refined event is enabled.



First Refinement – Elements and position

- Path_Elem_Pos - element positions for a path.
- rail_Elem_Pos current position
  - Set by setPositionForpath
- rail_elem_path_curr the path of an element (if any)
  - set/cleared by the path requestPath/path releasePath transitions



Guard:
elem ∈ dom(Path_Elem_Pos(p))
rail_elem_pos_curr(elem) ≠ Path_Elem_Pos(p)(elem)

Action:
rail_elem_pos_curr(elem) ≔ Path_Elem_Pos(p)(elem)

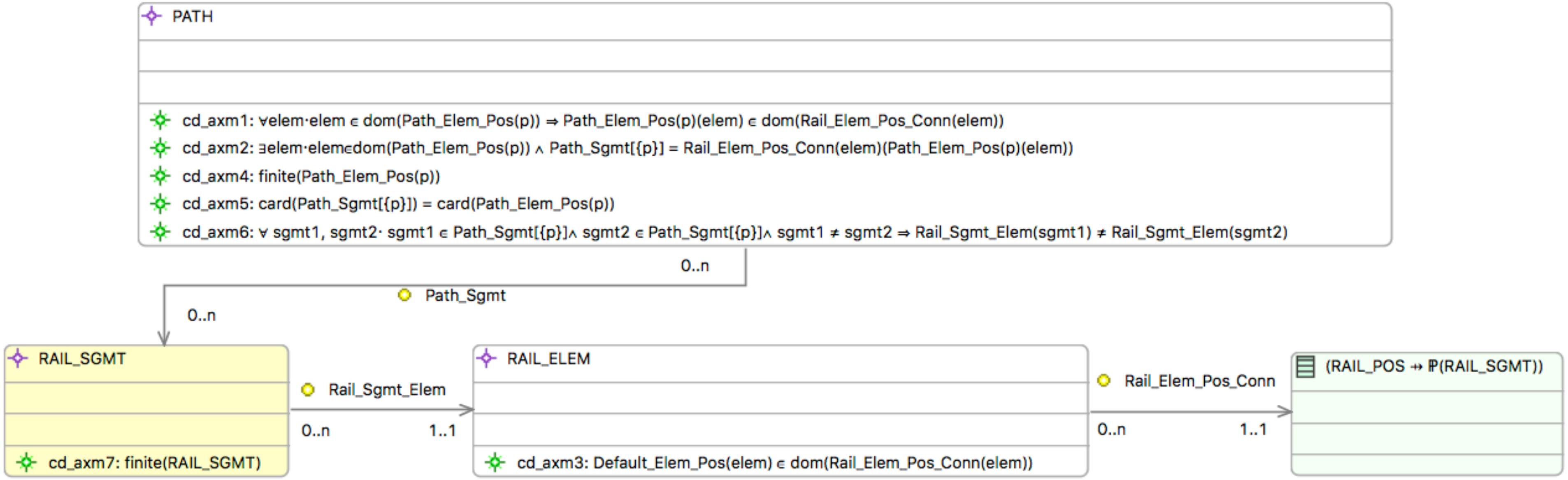
inv3: elem ∈ dom(rail_elem_path_curr)⇒ elem ∈ dom(Path_Elem_Pos(rail_elem_path_curr(elem)))

inv4: elem ∈ dom(rail_elem_path_curr) ⇒ Path_Elem_Pos(rail_elem_path_curr(elem))(elem) = rail_elem_pos_curr(elem)

yellow icon = constant
green icon = variable

2nd Safety Reqmt: Elements in an active path must be correctly positioned

Second Refinement - Segments



cd_axm1: ∀elem·elem ∈ dom(Path_Elem_Pos(p)) ⇒ Path_Elem_Pos(p)(elem) ∈ dom(Rail_Elem_Pos_Conn(elem))

cd_axm2: ∃elem·elem∈dom(Path_Elem_Pos(p)) ∧ Path_Sgmt[{p}] = Rail_Elem_Pos_Conn(elem)(Path_Elem_Pos(p)(elem))

cd_axm4: finite(Path_Elem_Pos(p))

cd_axm5: card(Path_Sgmt[{p}]) = card(Path_Elem_Pos(p))

cd_axm6: ∀ sgmt1, sgmt2· sgmt1 ∈ Path_Sgmt[{p}]∧ sgmt2 ∈ Path_Sgmt[{p}]∧ sgmt1 ≠ sgmt2 ⇒ Rail_Sgmt_Elem(sgmt1) ≠ Rail_Sgmt_Elem(sgmt2)

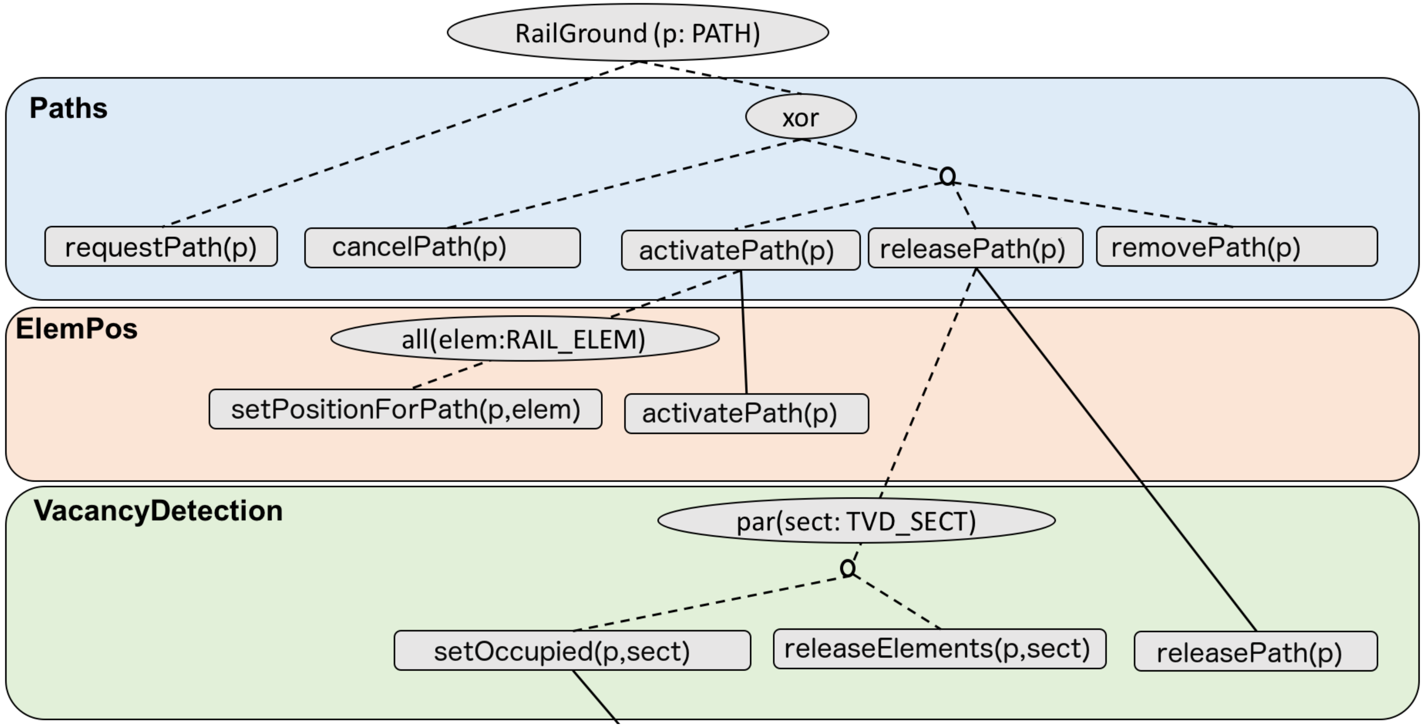
cd_axm7: finite(RAIL_SGMT)

cd_axm3: Default_Elem_Pos(elem) ∈ dom(Rail_Elem_Pos_Conn(elem))

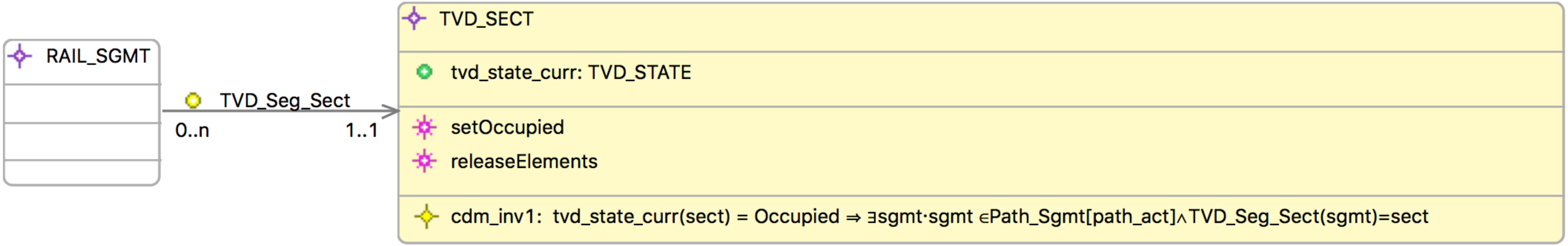
- In this refinement we introduce the segments that make up a path (Path_Sgmt).
- Each segment is associated with exactly one Element (Rail_Sgmt_Elem).
- There is no event refinement, only strengthening of some existing guards and ac tions
- Hence there is no statemachine or ERS to consider

Third Refinement – Vacancy Detection

- The *par* operator permits interleaving of instances of TVD_SECT in the sequence setOccupied;releasePath
- This is a preliminary step for releasePath



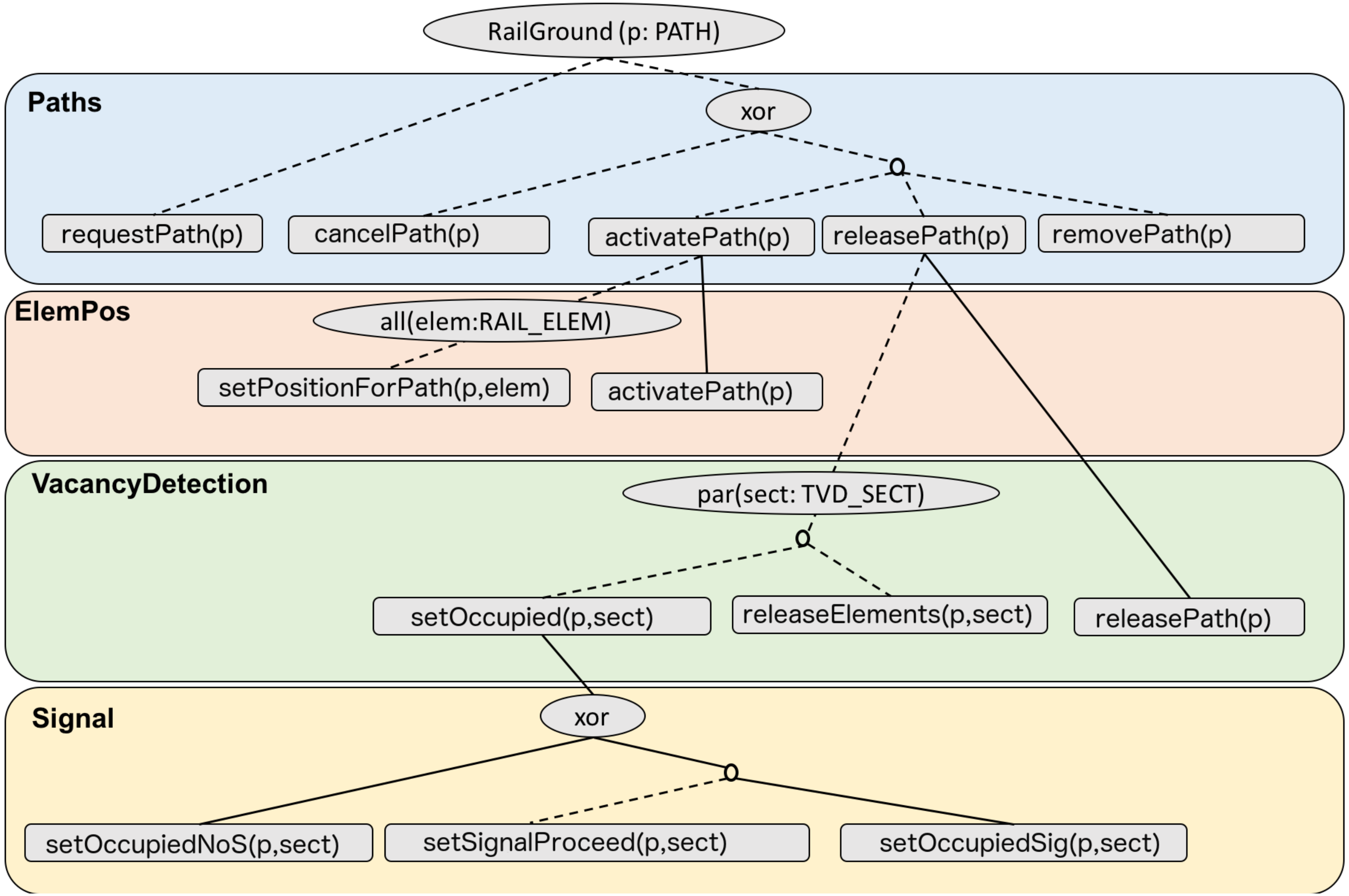
Third Refinement – Vacancy Detection


RAIL_SGMT
TVD_Seg_Sect
0..n
1..1
TVD_SECT
tvd_state_curr: TVD_STATE
setOccupied
releaseElements
cdm_inv1: tvd_state_curr(sect) = Occupied ⇒ ∃sgmt·sgmt ∊Path_Sgmt[path_act]∧TVD_Seg_Sect(sgmt)=sect

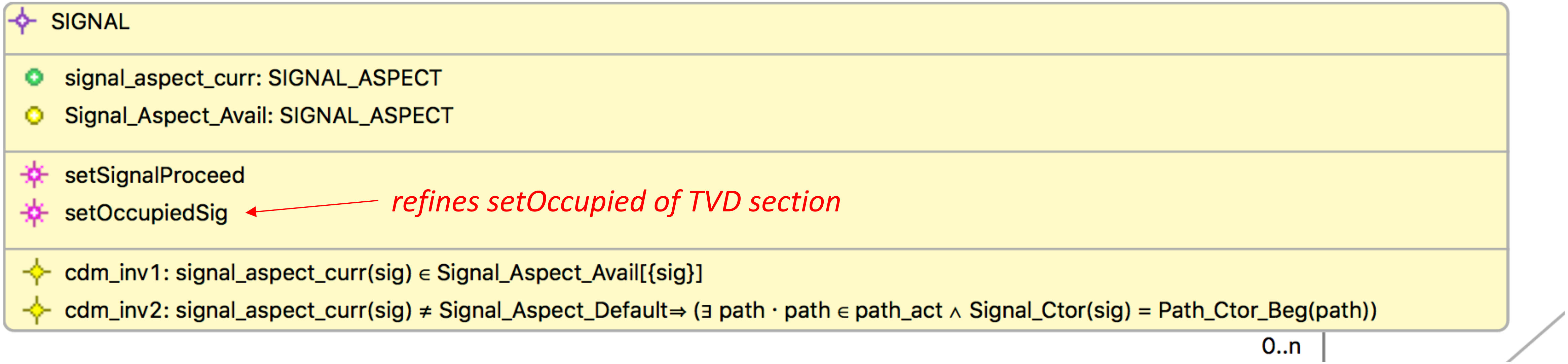
- Every Segment has a TVD section (TVD_Seg_Sect).
- TVD sections have a variable attribute (tvd_state_curr) to record whether the track is occupied or not.
- The variable is set to occupied by method setOccupied and cleared by releaseElements

Fourth Refinement - Signals

- setOccupied is split into two cases in this refinement.
- 1) setOccupiedNoS - the TVD section is part of a path but not protected by a signal.
- 2) setOccupiedSig - the section is protected by a signal and can only occur once the signal has been set to proceed by SetSignalProceed
- Note that, in this second case we model a constraint on the environment (i.e. trains do not pass default signals) to represent an assumption



Fourth Refinement - Signals



SIGNAL

signal_aspect_curr: SIGNAL_ASPECT

Signal_Aspect_Avail: SIGNAL_ASPECT

setSignalProceed

setOccupiedSig

refines setOccupied of TVD section

cdm_inv1: signal_aspect_curr(sig) ∈ Signal_Aspect_Avail[{sig}]

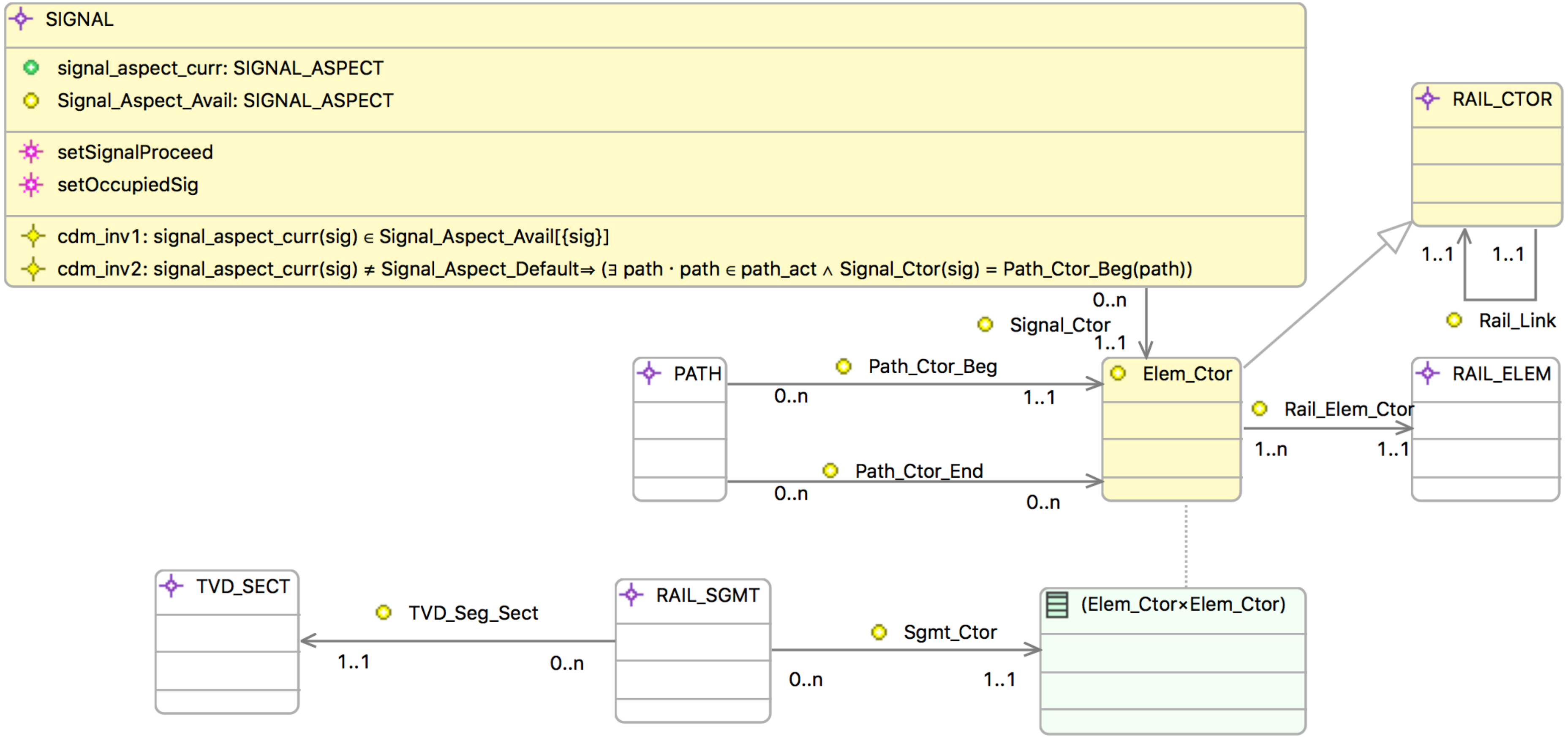
cdm_inv2: signal_aspect_curr(sig) ≠ Signal_Aspect_Default⇒ (∃ path · path ∈ path_act ∧ Signal_Ctor(sig) = Path_Ctor_Beg(path))

- The current aspect is given by attribute signal_aspect_curr
- The possible values for aspect are given by Signal_Aspect_Avail
- Currently only default is specified

- setSignalProceed sets the aspect to ‘not default’ (i.e. proceed)
- setOccupiedSig sets the aspect to default as the corresponding connected section becomes occupied

Fourth Refinement - Signals

- Signals are related to the other devices (classes) in the model via connectors



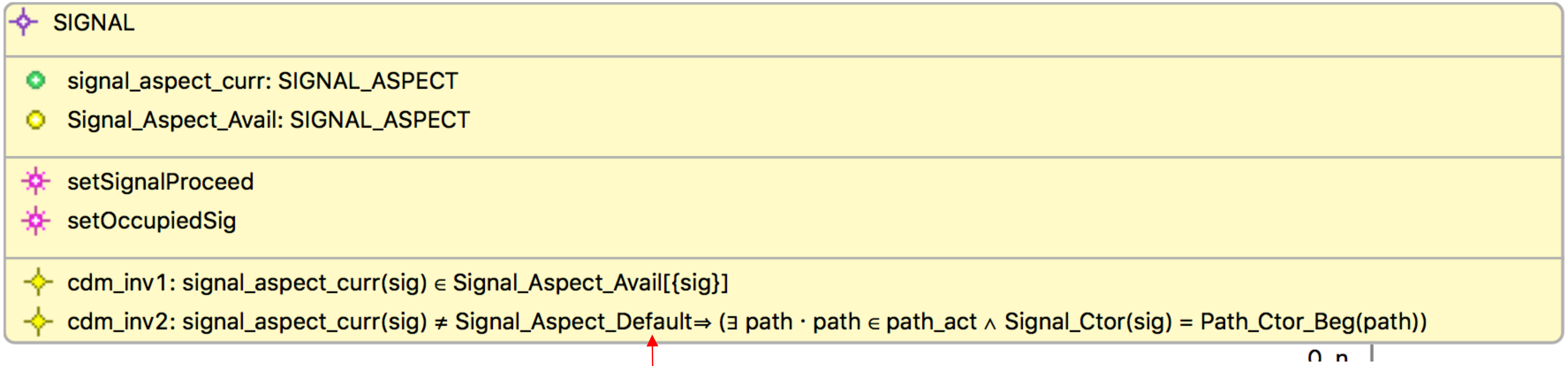
SIGNAL

signal_aspect_curr: SIGNAL_ASPECT
Signal_Aspect_Avail: SIGNAL_ASPECT

setSignalProceed
setOccupiedSig

cdm_inv1: signal_aspect_curr(sig) ∈ Signal_Aspect_Avail[{sig}]
cdm_inv2: signal_aspect_curr(sig) ≠ Signal_Aspect_Default⇒ (∃ path · path ∈ path_act ∧ Signal_Ctor(sig) = Path_Ctor_Beg(path))

Fourth Refinement - Signals



SIGNAL

signal_aspect_curr: SIGNAL_ASPECT

Signal_Aspect_Avail: SIGNAL_ASPECT

setSignalProceed

setOccupiedSig

cdm_inv1: signal_aspect_curr(sig) ∈ Signal_Aspect_Avail[{sig}]

cdm_inv2: signal_aspect_curr(sig) ≠ Signal_Aspect_Default⇒ (∃ path · path ∈ path_act ∧ Signal_Ctor(sig) = Path_Ctor_Beg(path))

Signal Reqmt: A signal is only set to proceed (‘not default’) when there is an active path at rear of the signal

Ensured by guard of setSignalProceed

Model

The Event-B model is available as a dataset here:

https://doi.org/10.5258/SOTON/D0184.

You will need to install Rodin 3.3 from sourceforge and then install plug-ins from within Rodin. (see 'readme' file within the dataset).

Outline

- Introduction
- Railground – requirements
- Background – modelling techniques
- Railground - Model
- **Summary and Conclusions**

Summary - Efficiency and Clarity

- Efficiency
  - ‘Amplification’ obtained from Diagrams
  - abstraction and re-usable v&v models
- Clarity - Separation of Concerns
  - Entity local level v System level
  - Refinement – introduce one entity at a time
  - Visualisation of..
    - entity relationships,
    - entity state & behaviour,
    - system process,
  - More understandable – e.g. by non-FM system engineers
  - Easier to communicate – e.g. to customers
  - Easier to Maintain

Future directions

- Integrate ERS tooling with iUML-B
    - Currently we are only using it to draw diagrams by hand
    - There is an ERS Event-B generator but it conflicts with iUML-B

- Composition/Inclusion mechanism
    - Devices s.a. signals can be modelled in a separate module
        - .. including as a refinement chain
    - Supports Customer Variations with pluggable modules
    - Hoang, Thai Son, Dghaym, Dana, Snook, Colin and Butler, Michael (2017) A composition mechanism for refinement-based methods, At 22nd International Conference on Engineering of Complex Computer Systems, Fukuoka, Japan. 05 - 08 Nov 2017. 10 pp.

- Add or integrate DSLs for Customer Specific variations
    - To generate the specific rules for a pluggable module

ABZ 2018 Case study :–
Hybrid ERTMS/ETCS Level 3 standard

- ABZ 2018 Conference
  - Conference for cross fertilization of state-based formal methods:
  - ASM, Alloy, B, TLA, VDM and Z
  - ABZ2018 in Southampton, UK

- Common case study for 2018:
  - Hybrid ERTMS/ETCS Level 3 standard (virtual fixed block)
  - https://www.southampton.ac.uk/abz2018/information/case-study.page
  - Authors are invited to model the Hybrid ERTMS/ETCS standard in one of the above languages
  - Submission 29th Jan 18

THANK YOU

ECSEL JU

The research leading to these results has received funding from the H2020-ECSEL-2015-2-IA-two-stage-Master/H2020-ECSEL-2015-2-IA-two-stage
for ENABLE-S3 – European Initiative to Enable Validation for Highly Automated Safe and Secure Systems Joint Undertaking under grant agreement n° 692455 and from specific national programs and / or funding authorities.