Formal Modelling of Railway Interlockings
Using Event-B and the Rodin Tool-chain

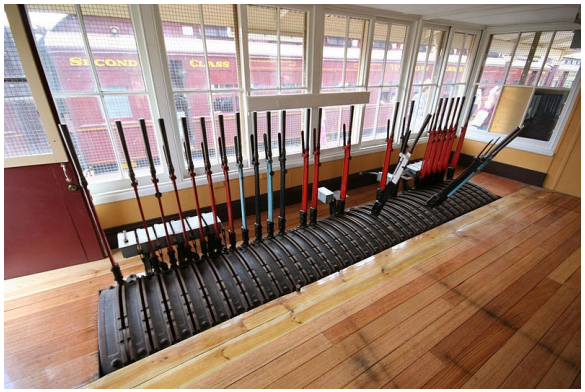
Klaus Reichl, Thales Austria GmbH
Luis Diaz, Thales España Grp, S.A.U.

THALES

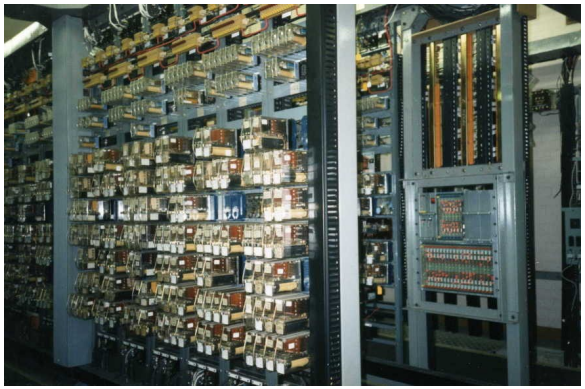
Overview

1. What is an Interlocking, Anyway?
2. Approaching the Interlocking Problem
3. Introducing Variation Points
4. The Model Talks Back
5. What We Like and What We'd Like to See in Rodin

What is an Interlocking ...



Mechanical Interlocking

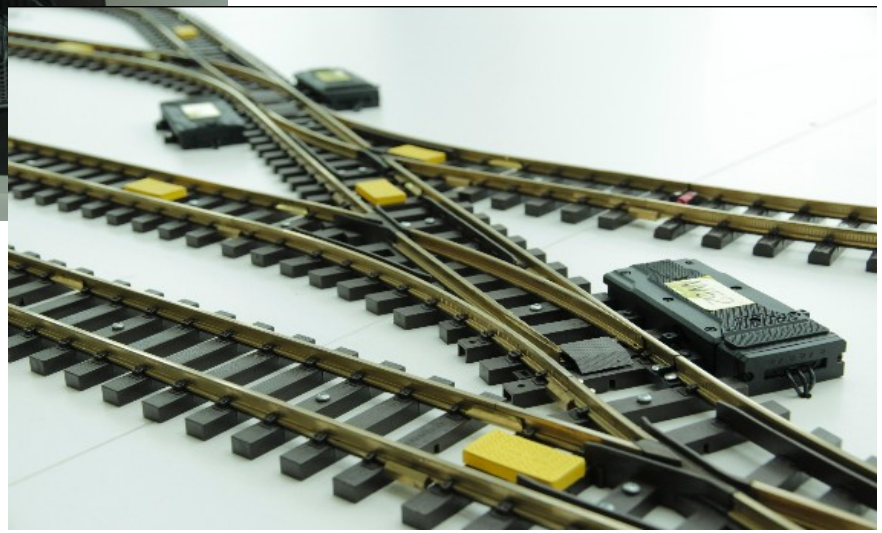


Relay Interlocking

Electronic Interlocking

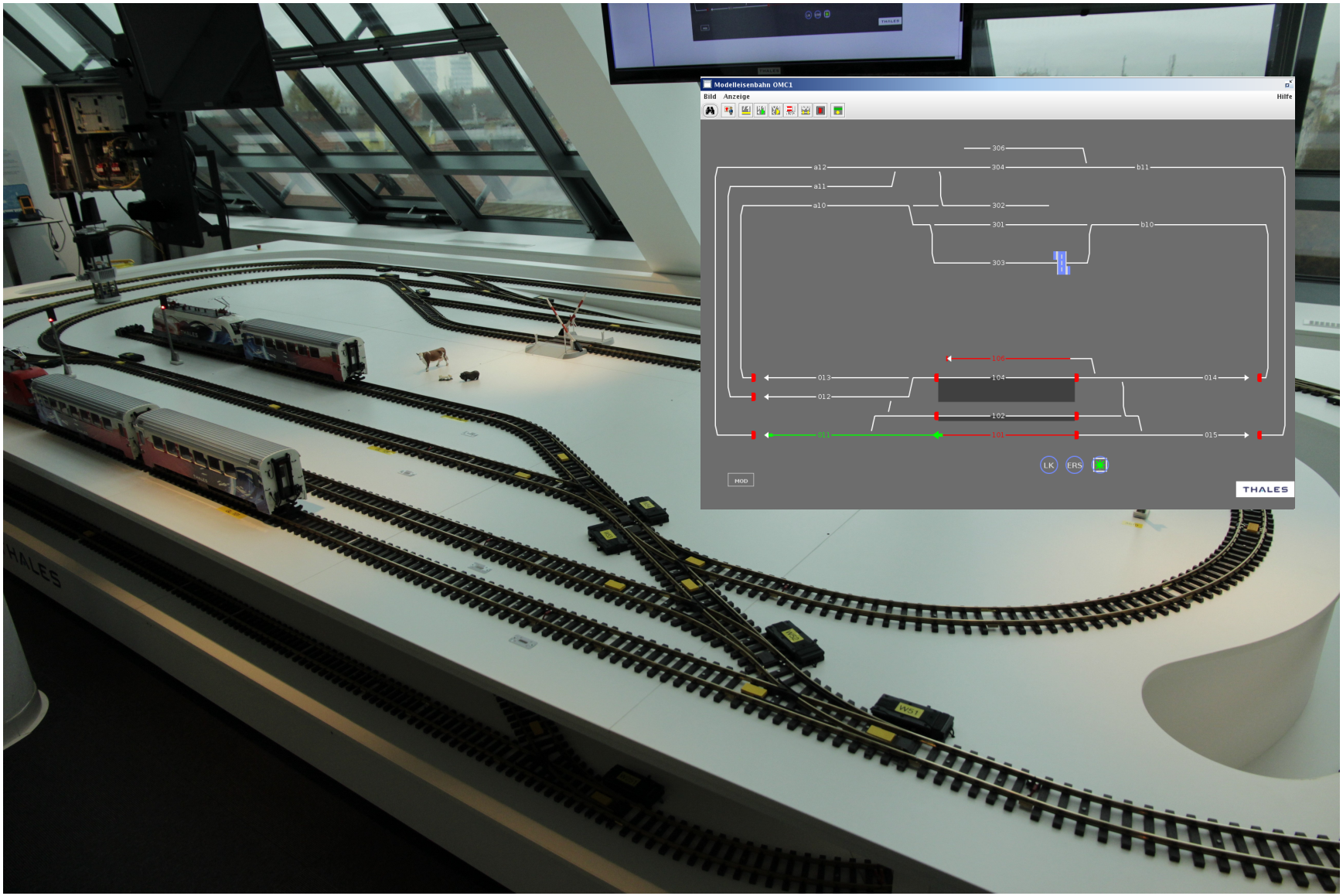
THALES

... and an Interlocking Model



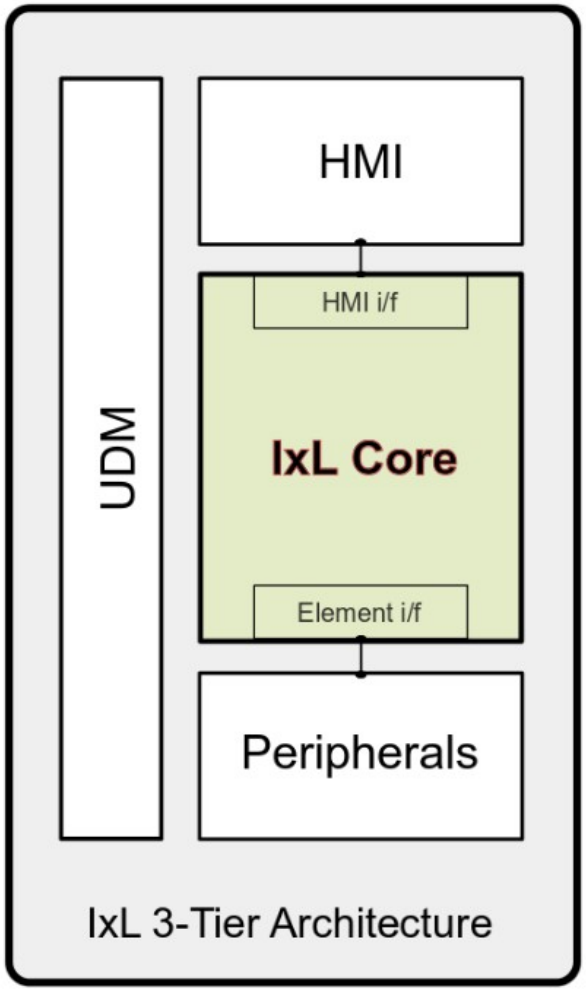
THALES

Overview of Operations

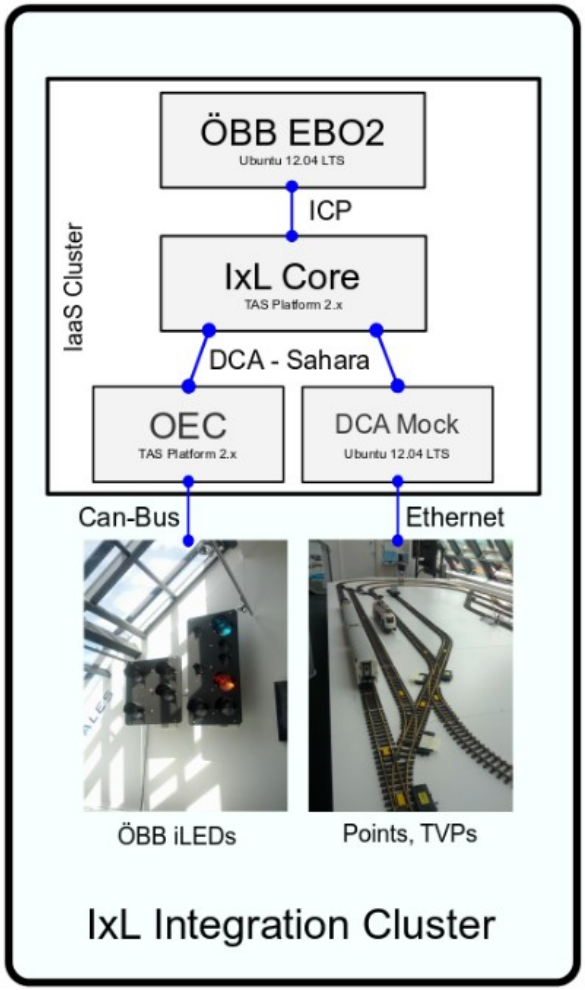


THALES

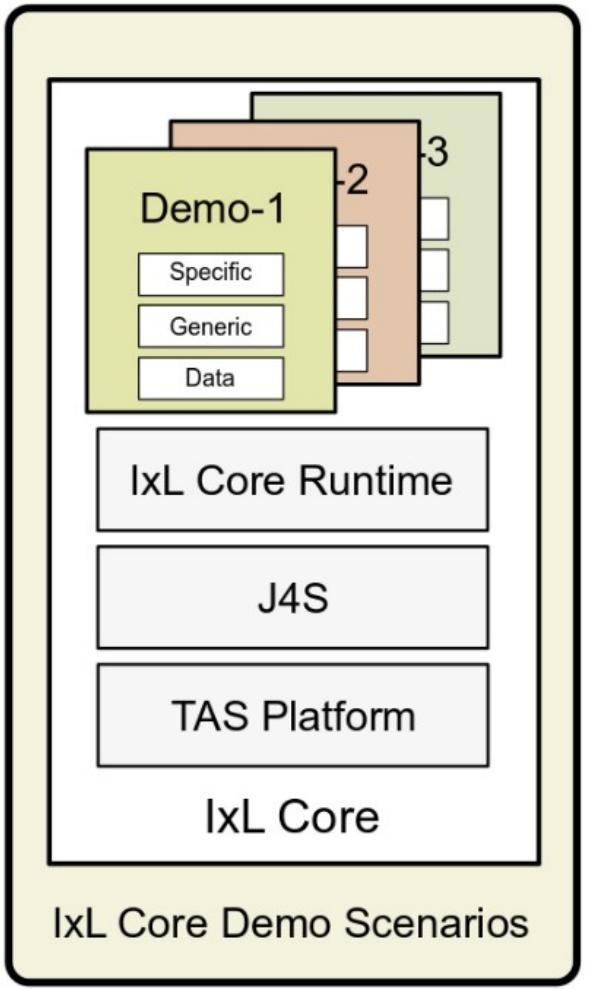
Demonstrator Architecture



IxL 3-Tier Architecture



IxL Integration Cluster

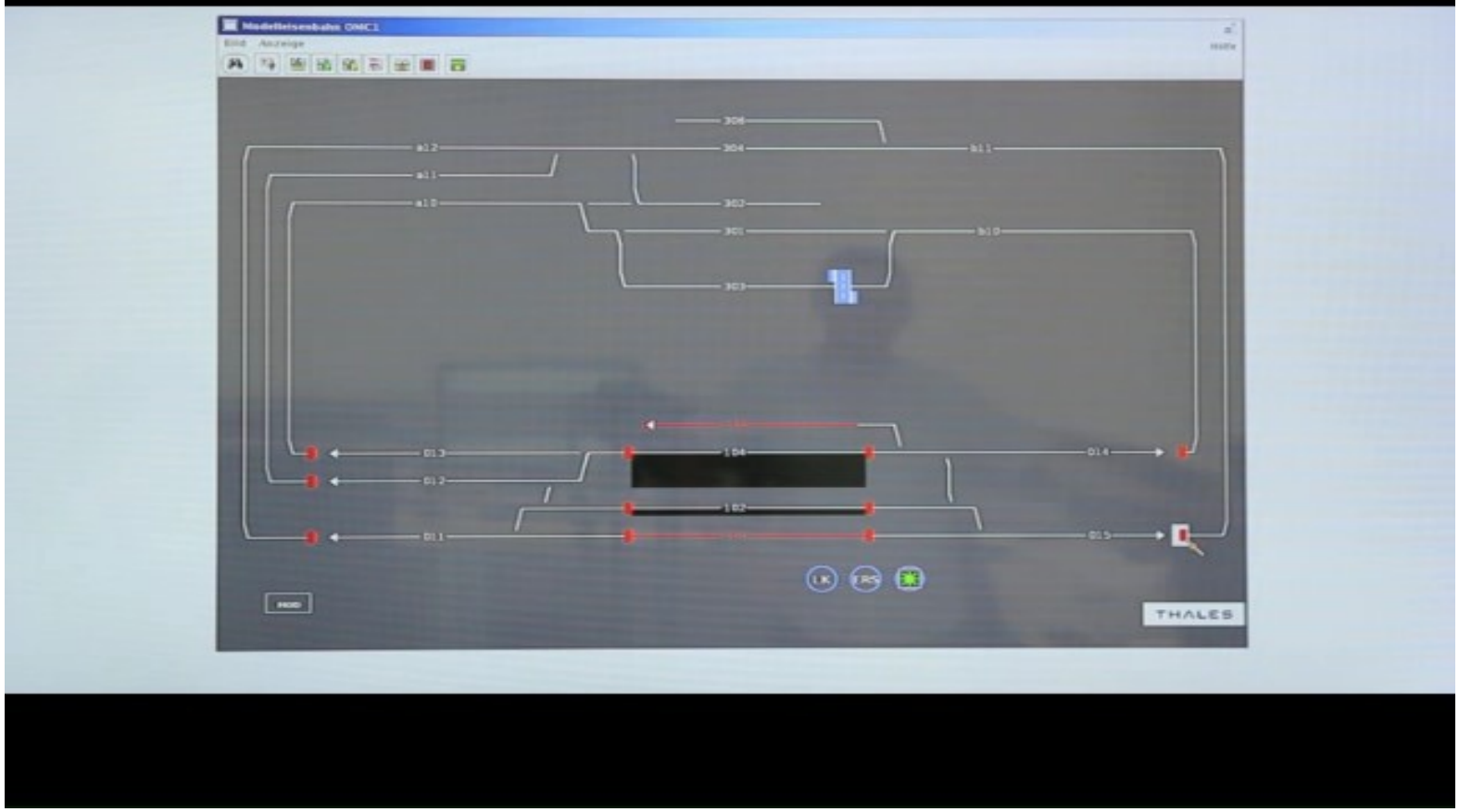


IxL Core Demo Scenarios

THALES

Overview of Operations

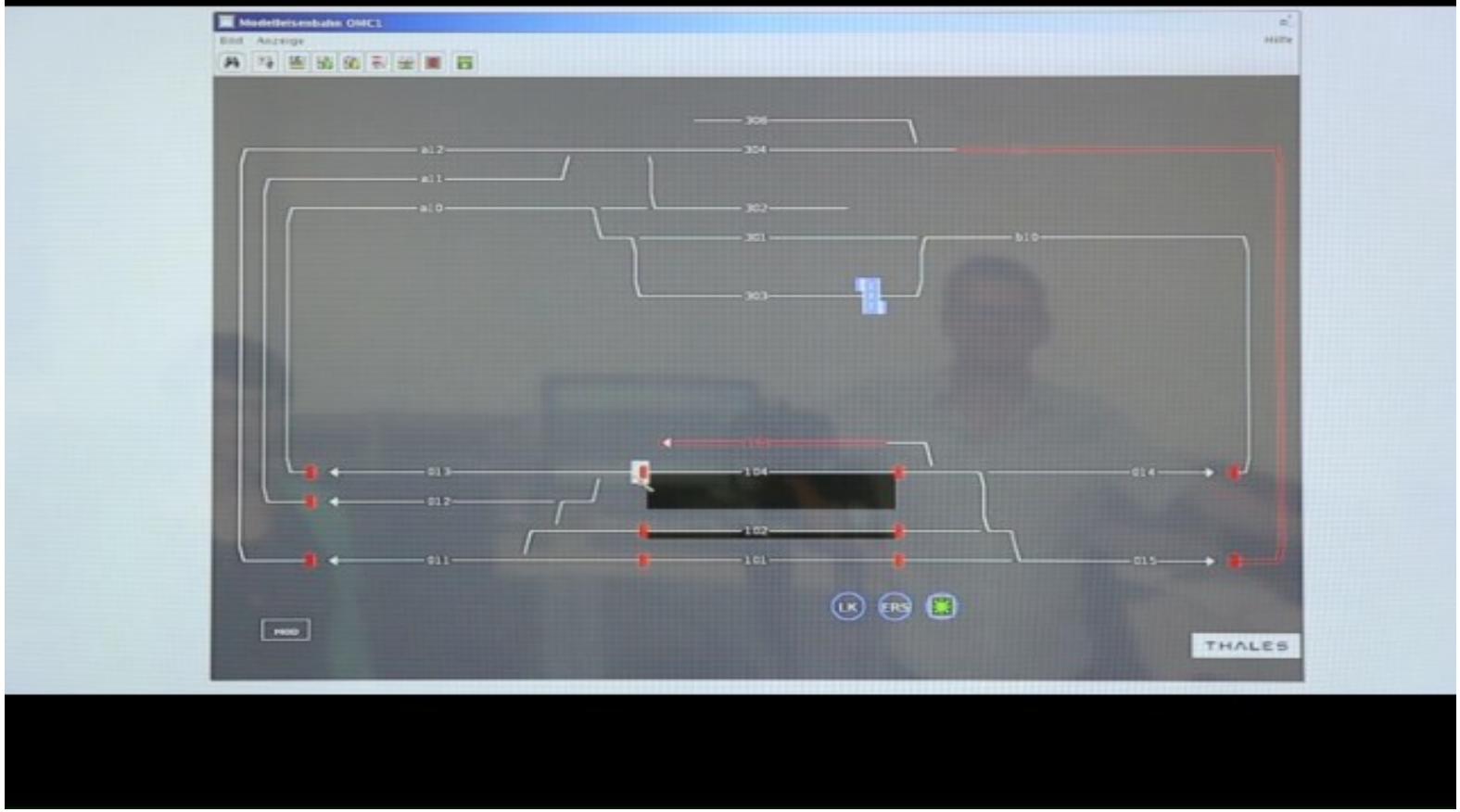
If slide doesn't play look for 05-movie-move-in.ogg



THALES

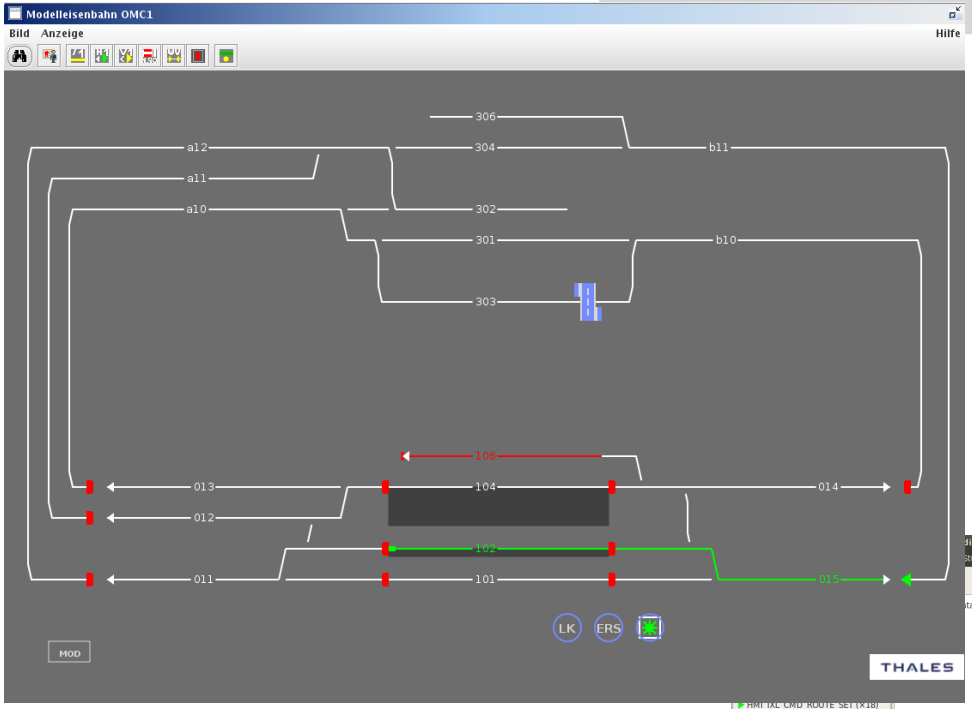
Overview of Operations

If slide doesn't play look for 05-movie-move-out.ogg

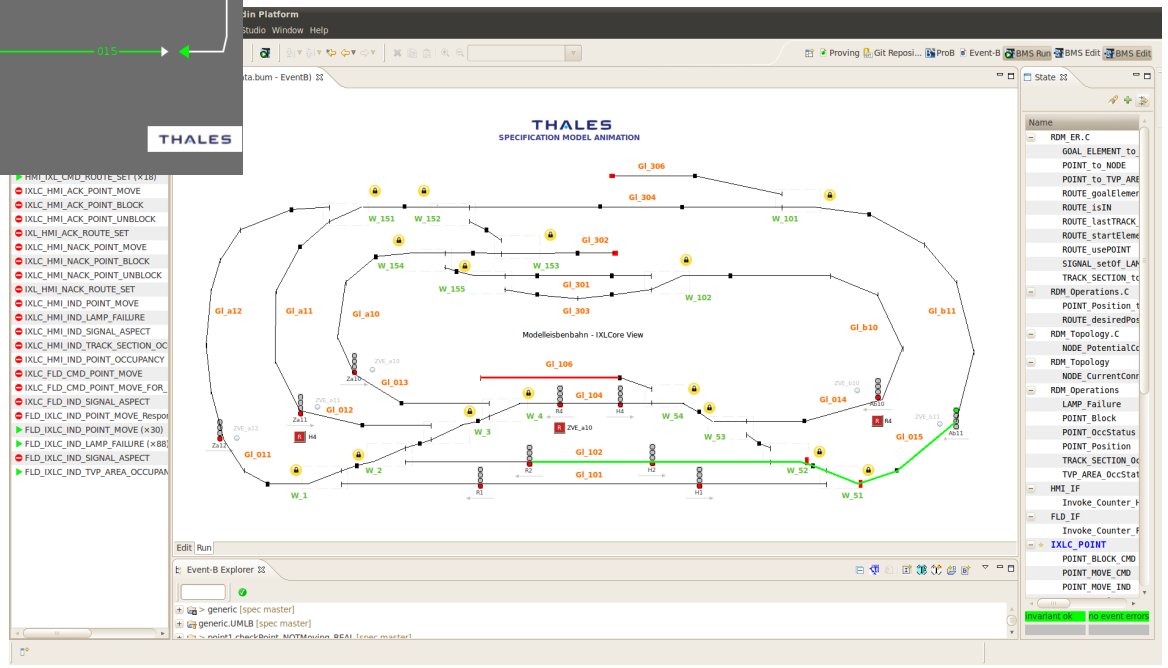


THALES

… now the Formal Model



Implementation –
Austrian OeBB



Specification Model -
Event-B + ProB

THALES

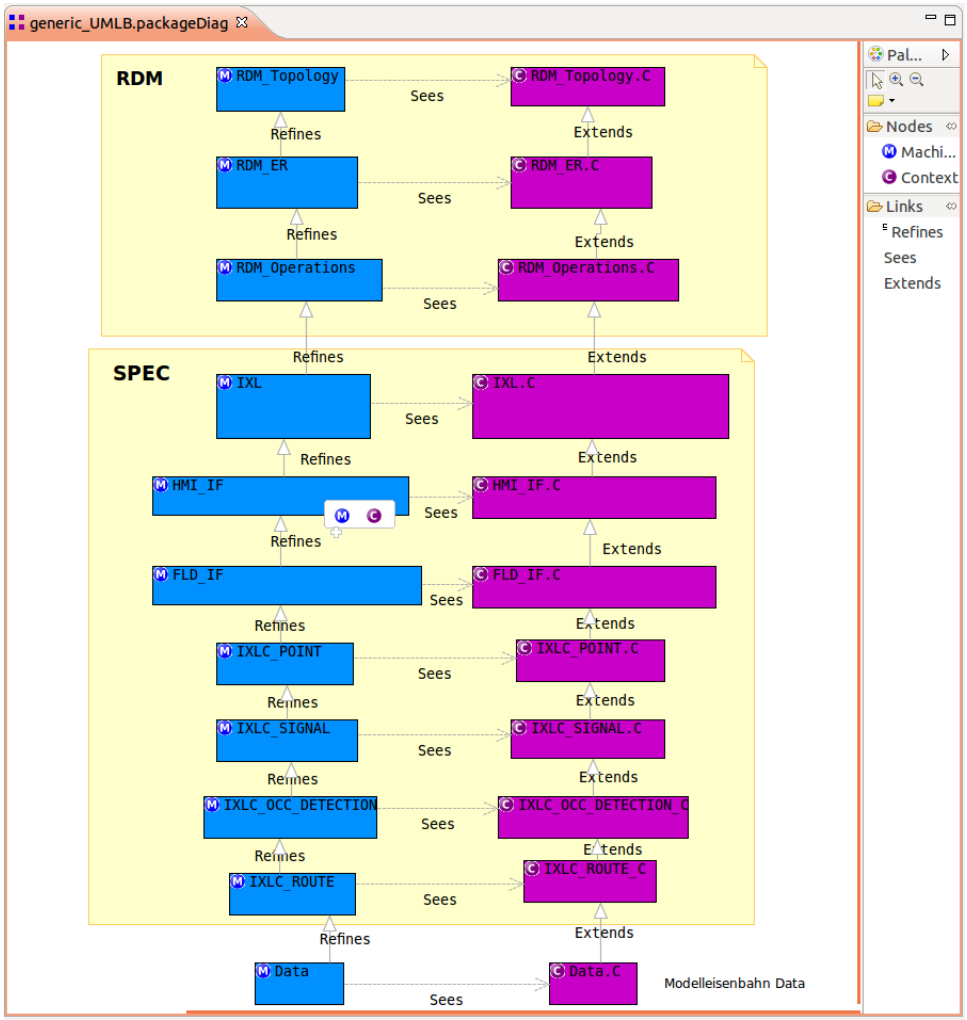
Live Demos

1. What is an Interlocking, Anyway?
2. Approaching the Interlocking Problem
3. Introducing Variation Points
4. The Model Talks Back
5. What We Like and What We'd Like to See in Rodin

Live Demos

- UML-B and Event-B Models
- Html talks about
  - IxL a la carte
  - Inconsitent Product
- ProB and B Motion Studie Demo

UML-B Refinement Design



Even-B Refinement Design

- Problem Space:
  Railway Domain Model – RDM
- Solution Space:
  IxL Specification - SPEC
- Configuration:
  Station Data and Configuration

THALES

Summary

1. What is an Interlocking, Anyway?
2. Approaching the Interlocking Problem
3. Introducing Variation Points
4. The Model Talks Back
5. What We Like and What We'd Like to See in Rodin

What We Like and What We'd Like to See in Rodin

Industrialization - Systematic Technology Readiness Level

- HIGH TRL – Ready for Industrial Use
  - Event-B core including POG, selected Provers, Theories
  - Selected Plug-ins (UML-B, ProB)
- MEDIUM TRL – Methodology and/or Tool still to be completed
  - iUML-B, Modularization, Instantiation
- LOW TRL – Methodology and/or Tool needs Research
  - Test Case Generation, Code Generation

Technical Needs – Behavior Driven Development & Domain Driven Design

- Domain Vocabularies / Ontologies
- Name Spaces and Separated Models

Tools and Plug-ins

- Support for Multi-user / Multi-site Development and Review
  - Text Only Editor and Representation – SCC (git)
  - Model Diff and Merge – Incremental Review and Merging
- M2M, M2T – Interface and Documentation Generation

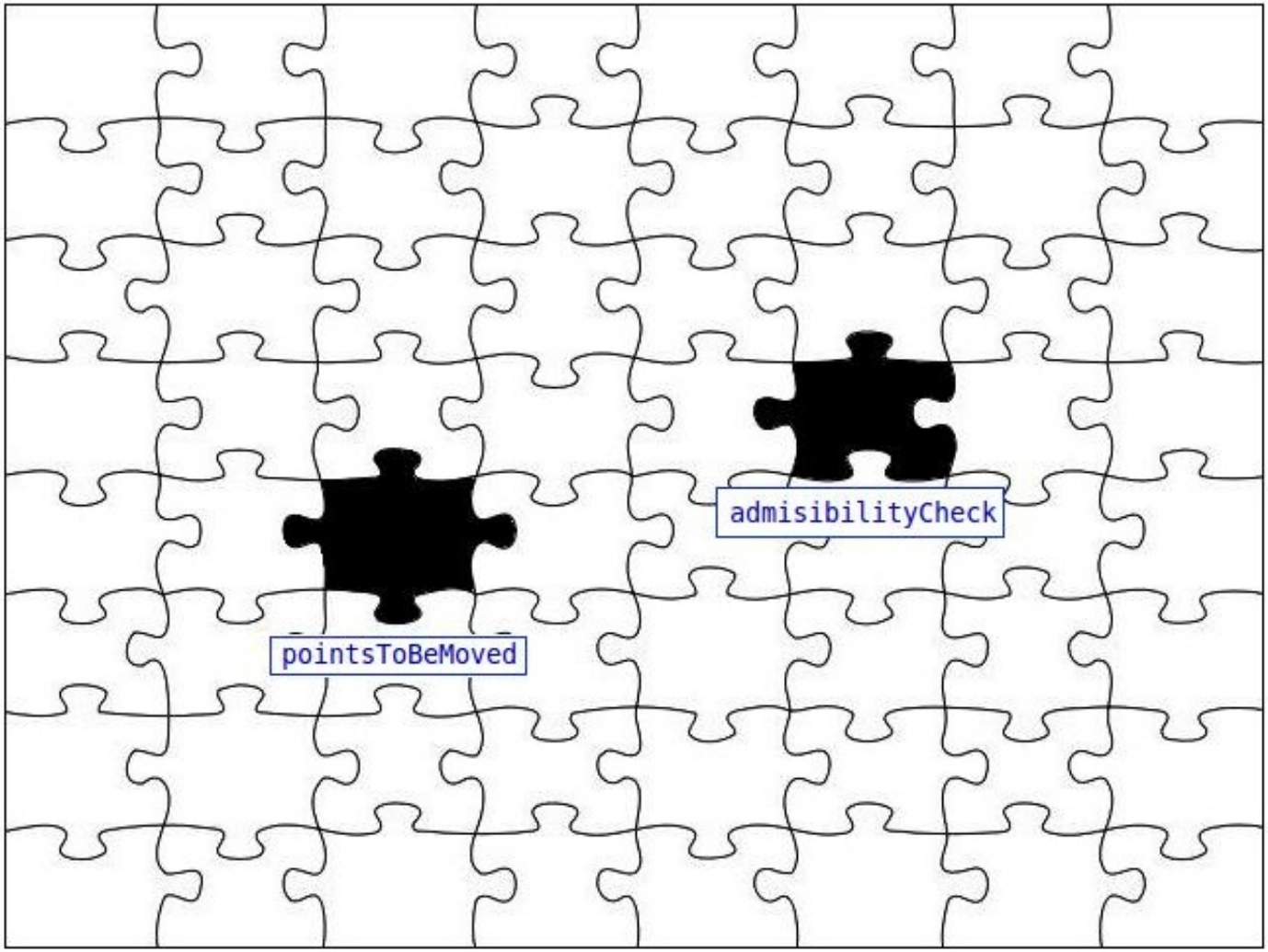
The idea of GIK as a Product Line is to be able to build Interlocking à la carte. Different Core Assets are combined to create an Interlocking with the desired behavior for a specific customer.

This page shows how this idea is supported by the Specification Model.

Specific customer solution in Event-B

The main event-B model is called "Generic". This model contains all the common IXL Core behavior**.** This Generic model is the Event-B framework that allows to instance a complete Model.

IXL CORE



In case some parts need to be plugged in to the Generic part in order to complete the 100% of the specific customer solution, the variable features **have to be** selected. The resulting model maintains the main properties of the Formal Methods:

(1) Model Checking:

Station data needed.

Automatic but inefficient (does not scale)

The system is then modelled as a set of states together with a set of transitions between states that describe how the system moves from one state to another in response to internal or external stimuli. Model checking tools are then used to verify that desired properties (expressed in some assertion language) hold in the system. Model checking if fully automateed because propositional logic is decidable. If the increase in size is big enough, then propositional decision procedures take an infeasible amount of time.

(2) Theorem Proving:

No data needed.

Precise but interactive (requires user interaction)

Proof obligations are generated to guarantee that certain properties of the modeled system are valid.

Theorem provers do not need to exhaustively visit the system state space to verify properties. Consequently, a theorem prover approach can reason about infinite state spaces and state spaces involving complex datatypes and recursion. This can be achieved because a theorem prover reasons about contraints on states, not instances of states. Theorem provers search for proofs in the syntactic domain, which is typically much smaller than the semantic domain searched by model checkers. Consequently, theorem provers are well-suited for reasoning about "data-intesive" systems with complex data structures but simple information flow.

(3) Animation

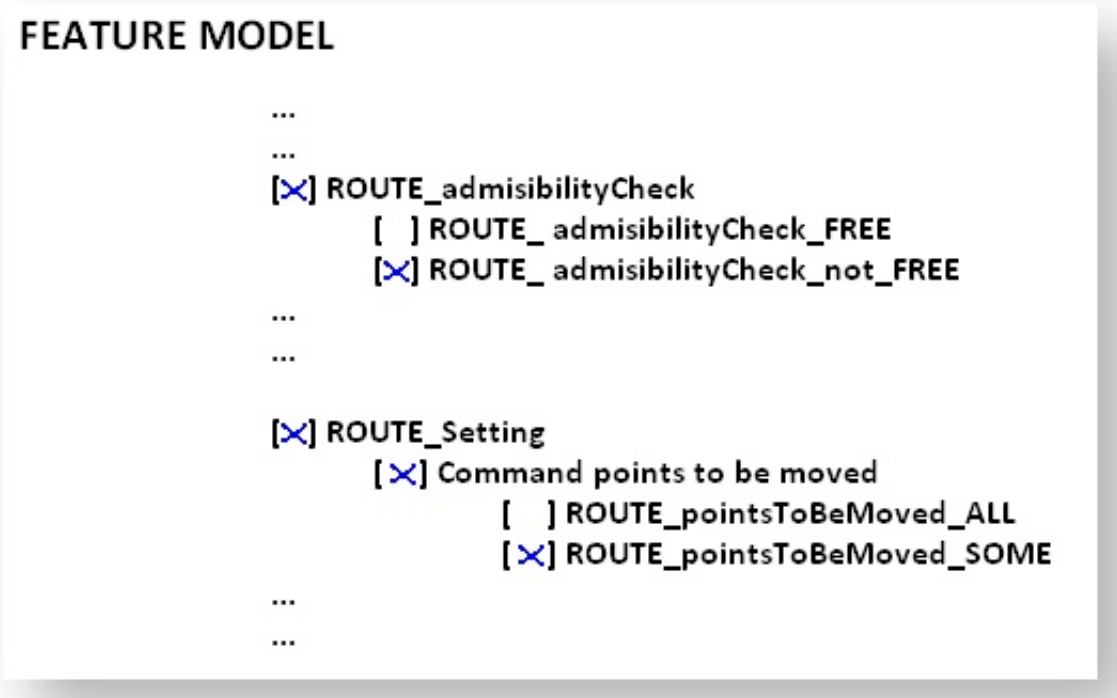
(4)Test Case Generation

In the Demonstrator scope we have two variability points:

- **admisibilityCheck** (SC-0021): different criteria to accept a route command in terms of track occupation
    - admisibilityCheck_FREE: all points and track sections of the route body must be in the state FREE (Demo1)
    - admisibilityCheck_NOT_FREE: the route is accepted despite an occupation (Demo2)

- **pointsToBeMoved** (Invented):number of points commanded to move by the route
    - pointsToBeMoved_ALL: all points of the route will be commanded to the field to move, even if some of them are already in the desired position for the route. (DCA will stop it)
    - pointsToBeMoved_SOME: only the points of the route which are not in the desired position for the route, are commanded to the field to move.

Having this variability points, in order to create a new Interlocking for a specific customer, the right features are selected.

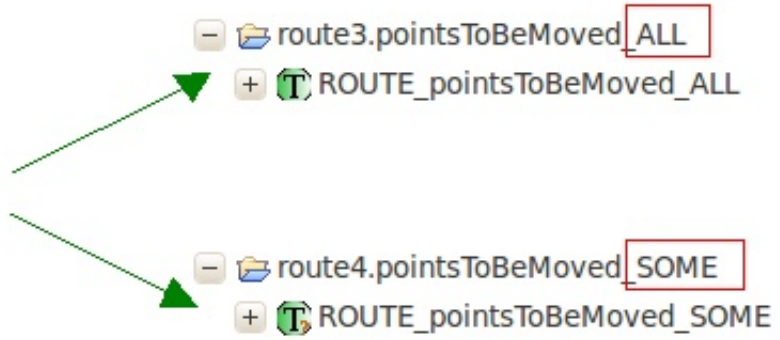


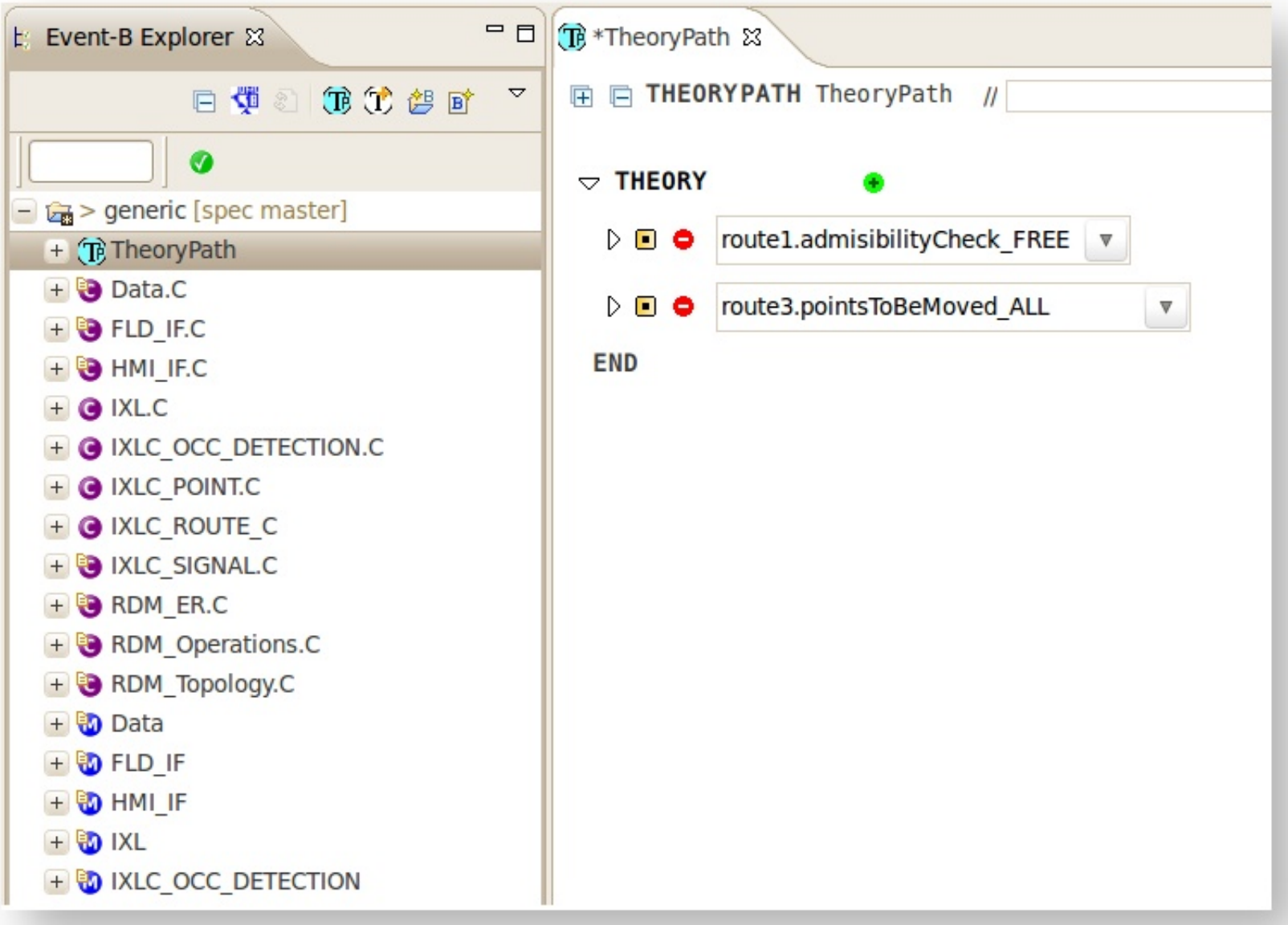
In Event-B there is one project for each Core Asset. Each one contains a specific behavior that match with in the Generic model:
There is a template (TheoryPath), inside of the Generic model (framework), that force you to select between the different possibilities before get the final model.

Remark: in order to create the final model, not Event-B knowledge is needed (see Right-Side picture). If the trace between Features, Requirements and Event-B is proper, it is only required to know which features are needed for this customer.

```
admisibilityCheck(
                    ROUTE_SET_CMD(invoke),
                    ROUTE_isIN,
                    ROUTE_usePOINT,
                    TRACK_SECTION_OccStatus,
                    POINT_OccStatus,
                    OCCUPANCY_FREE      )
```


```
pointsToBeMoved(
                    ROUTE_SET_CMD(invoke),
                    ROUTE_usePOINT,
                    ROUTE_desiredPosition,
                    POINT_Position
                )
```





The benefits of this mechanism is:

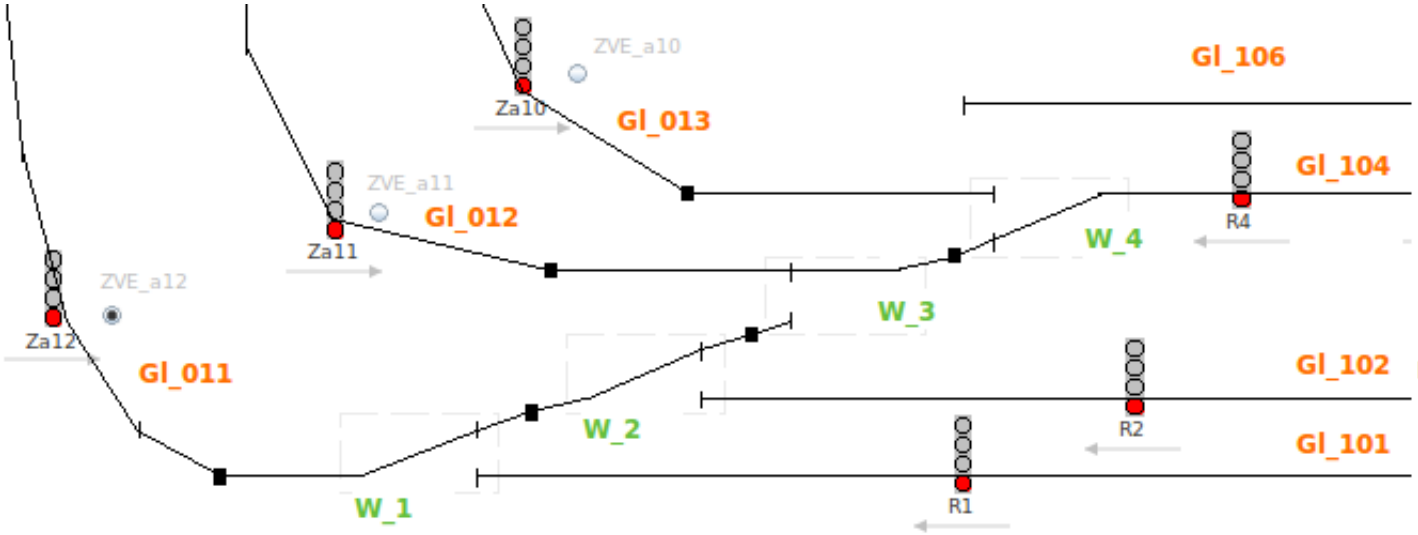
- Lack of functionality is detected. Model can not be built if the puzzle is not complete.

- (**FORMAL PROOFS**) Incompatibles between different selections are detected. A final model will not be formally proved if the added pieces are incompatible.

Final model is proved

Final model can be animated to test the behavior

Specific Test Cases are generated



This page is the continuation of the page (6) Interlocking a la carte**.**

Problem:

GIK allows to combine different Core Asset to create a specific customer solution. Also, new Core Asset can be added when a new customer desires a new behavior in his product.

But not all the possible combinations between our Core Asset works in a consistent way together, several combinations can be simply **incompatible**. Even a new Feature desired for a customer can be inconsistent with out generic behavior.

Solution:

Formal Methods allows to detect these kinds of incompatibilities by using Formal Proofs or Model Checking.

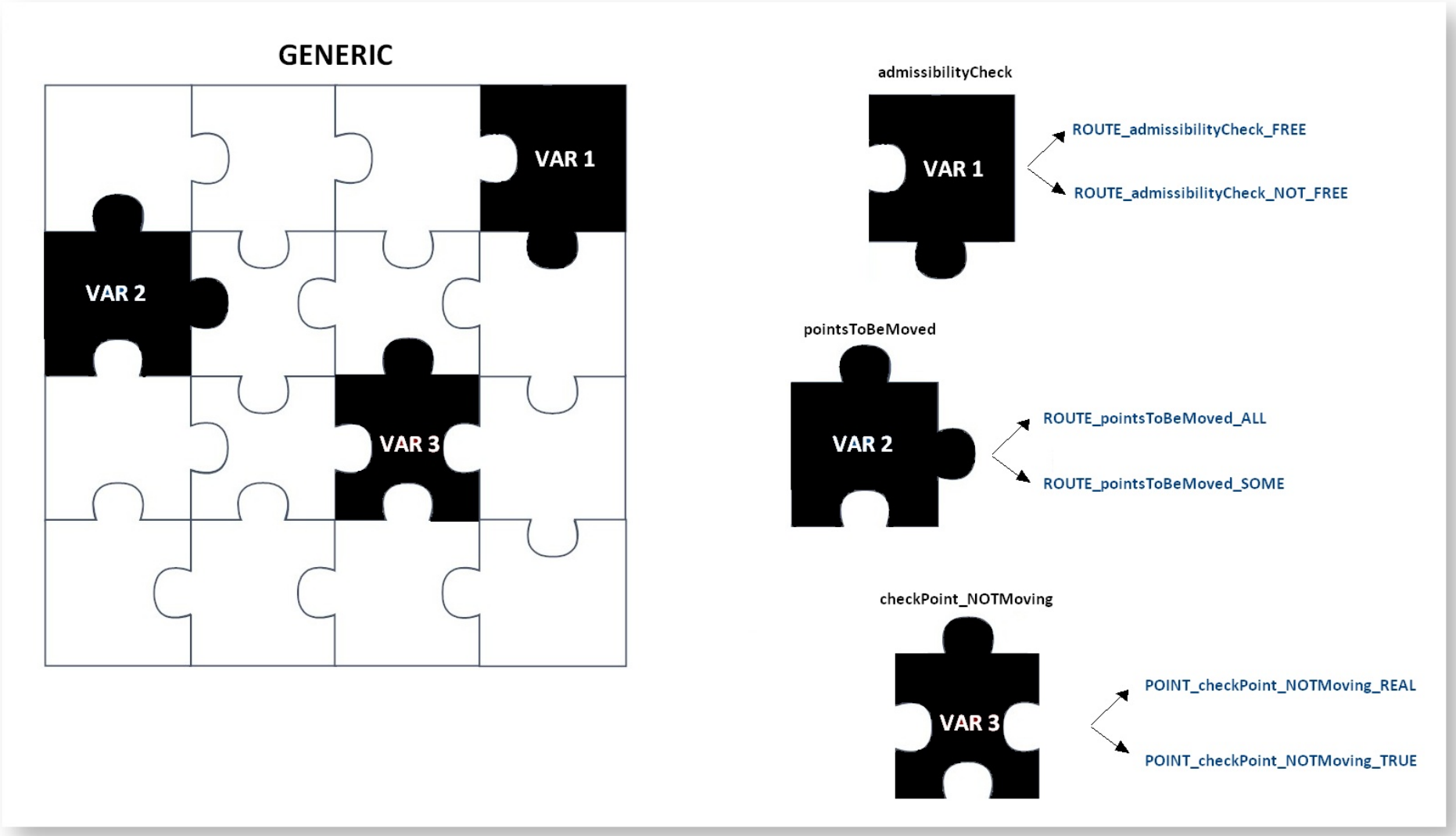
This page shows an example:

INCOMPATIBLE CUSTOMER BEHAVIOR

Our Specification is composed by a Generic project that contains the common behavior of all our customers. Some parts of the behavior of this Generic model is not completed; To complete these parts we need to select between all our Variability Core Assets.

This example shows 2 different customers selecting a different combination of variability. One customer makes a valid selection but Customer Y not.

The different Core Assets defined in the Specification Model are:



3 points of variability has to be selected to be able to create a concrete **IXLCore** solution.Rodin tool forces you to complete the selection, then lack of functionality is detected.

Next picture shows an example of two different selection.

CUSTOMER DEMO X

THEORY

- generic
- route1.admisibilityCheck_FREE
- route4.pointsToBeMoved_SOME
- point1.checkPoint_NOTMoving_REAL

END

CUSTOMER DEMO Y

THEORY

- generic
- route1.admisibilityCheck_FREE
- route4.pointsToBeMoved_SOME

diff

- point2.checkPoint_NOTMoving_TRUE

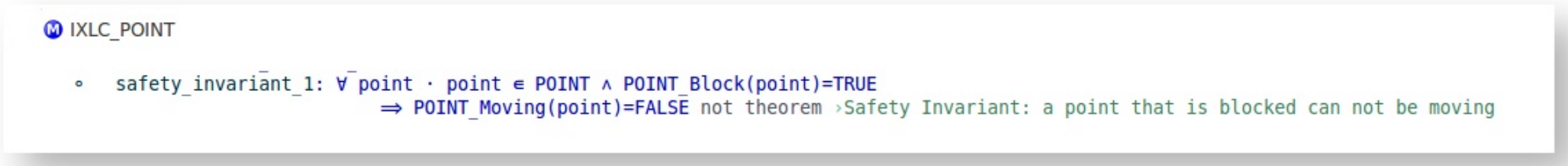
END

The only difference between these two selections are the last variability point "checkPoint_NOTMoving", this variability point is a condition of the BLOCK_POINT command.

- Customer DEMO X: this customer has selected the Core Asset that **check if the point to manually block is moving**.
- Customer DEMO Y: this customer has selected the Core Asset that **do not check if the point to manually block is moving**.

After the selection, both customer will be proved by formal proofs and validated using the model animation.

It is important to remark that the Generic model contains the next Safety Invariant:



All the invariants defined in the system has to be preserved, and in this case this invariant means "**If a point is blocked, the point can not be moving at the same time**"

FORMAL PROOFS - Proving the Invariant

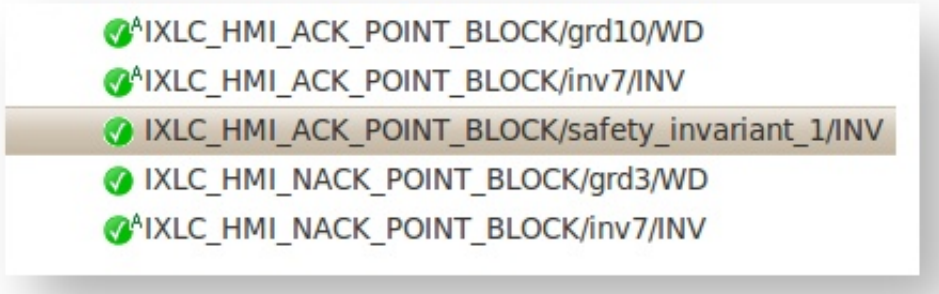
There is an event "**HMI_IXLC_CMD_POINT_BLOCK**" that change the status of a point to the status Block. Some conditions have to be checked to be able to execute this event. One of this condition is "**checkPointNOTMoving**", the variability point that both customers selected differently.

Without using any data, the consistency between our model behavior can be checked by the Formal Proofs.

CASE Customer DEMO X:

This customer did a selection of a Core Asset "**checkPointNOTMoving_REAL**" that checks if the point is not moving. If this condition is not fulfilled the Event to block the point will not be enabled.

One proof obligation is automatically generated to check if the Event "**HMI_IXLC_CMD_POINT_BLOCK**" preserves the Safety Invariant.

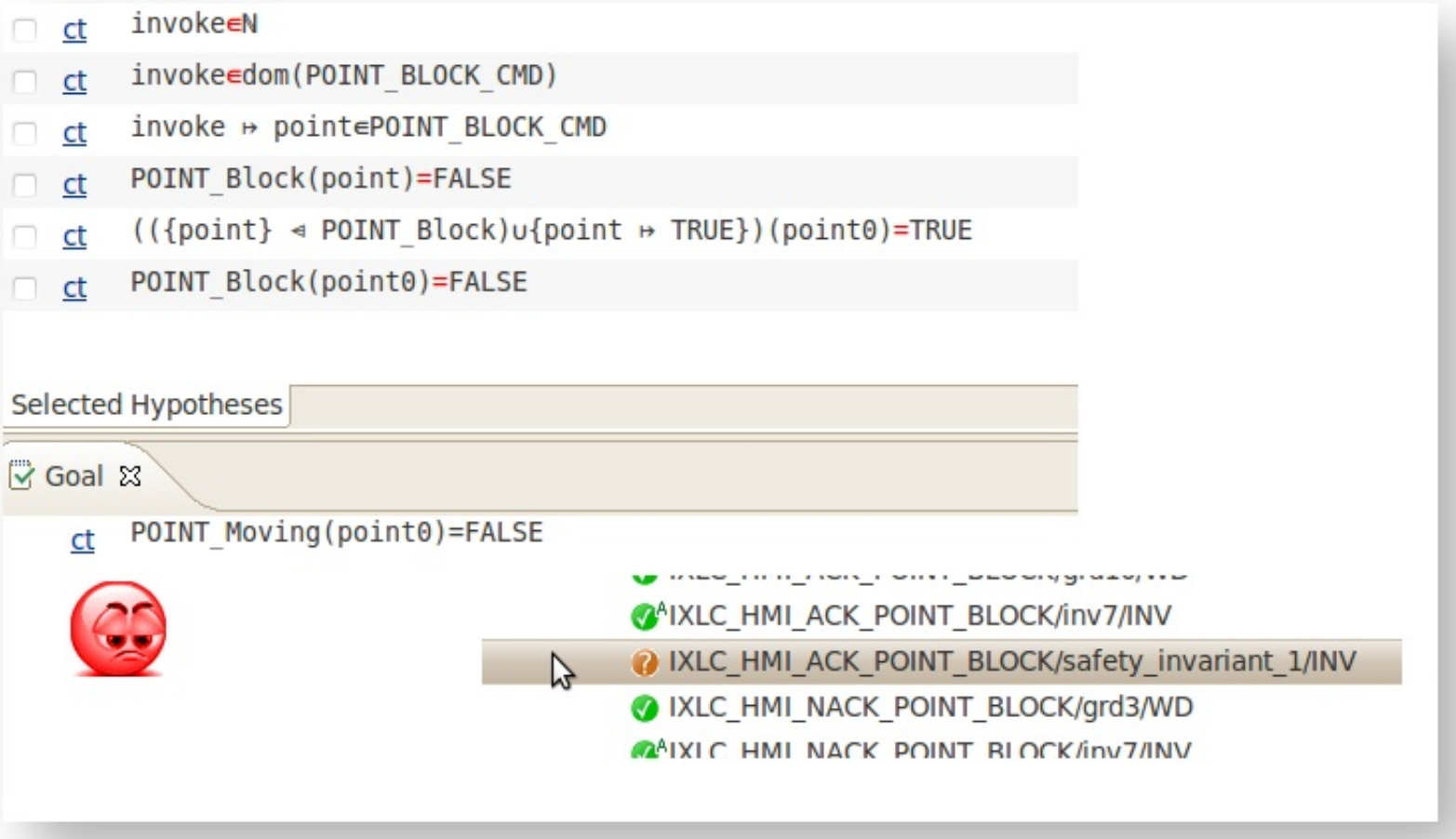


In this case, we did succeed with the proof confirming that the selection of our customer for this variability point was correct. The behavior respects our Invariants.

CASE Customer DEMO Y:

This customer did a selection of a Core Asset "**checkPointNOTMoving_TRUE**" that does NOT check if the point is moving, the predicate of this condition is always TRUE.

As this conditions is always TRUE (equivalent to don't check it), the Event to block the point will be enabled, even if the point is moving.

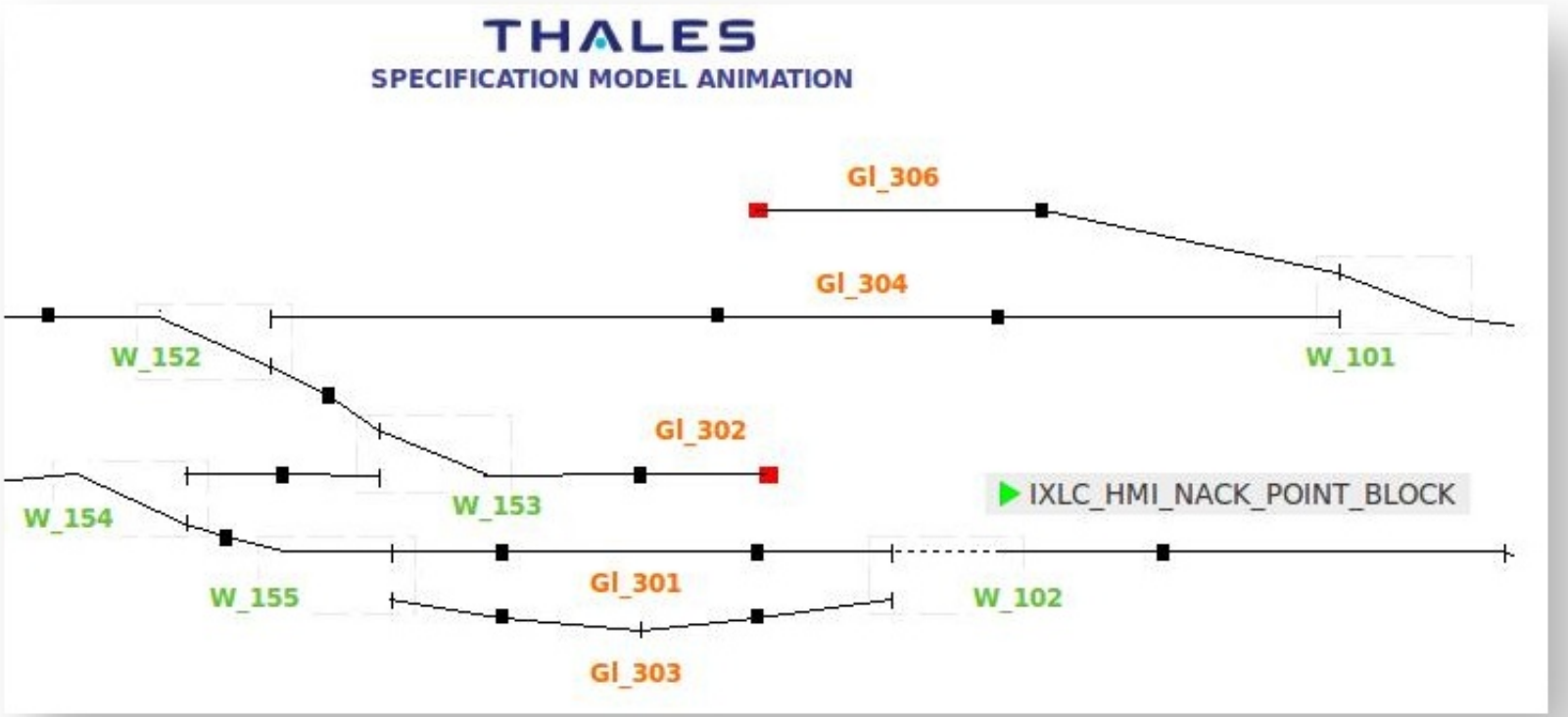


In this case this Proof Obligation can not be proved since we can not prove that the point is not moving when we execute a Block command, as the Invariant requires.

MODEL CHECKING & ANIMATION

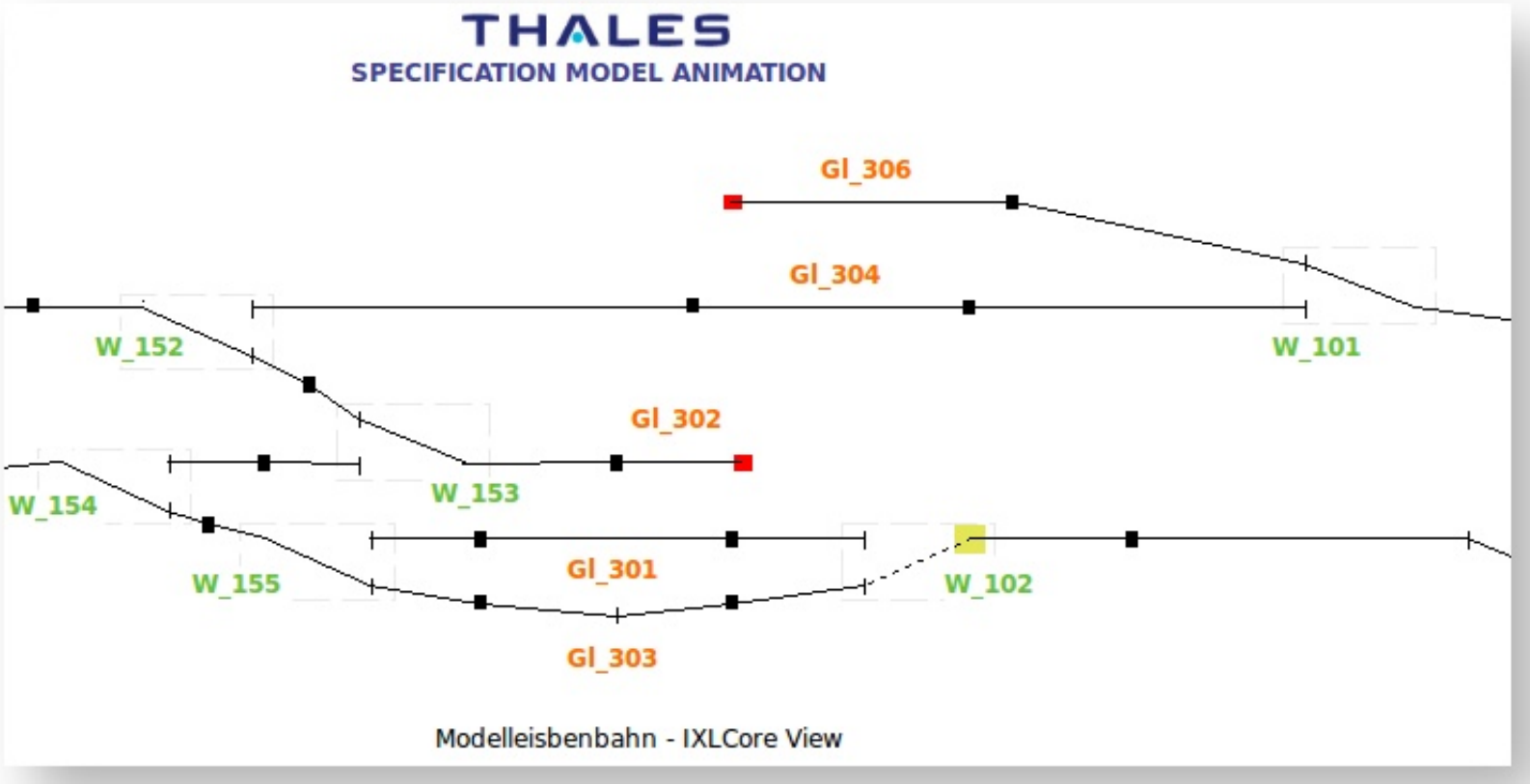
This inconsistency in the behavior can be detected using the Station Data and doing model checking and animation of the model.

CASE Customer DEMO X: as we can see in the picture, if we try to manually block a point that is moving (W_102), we receive a NACK command.

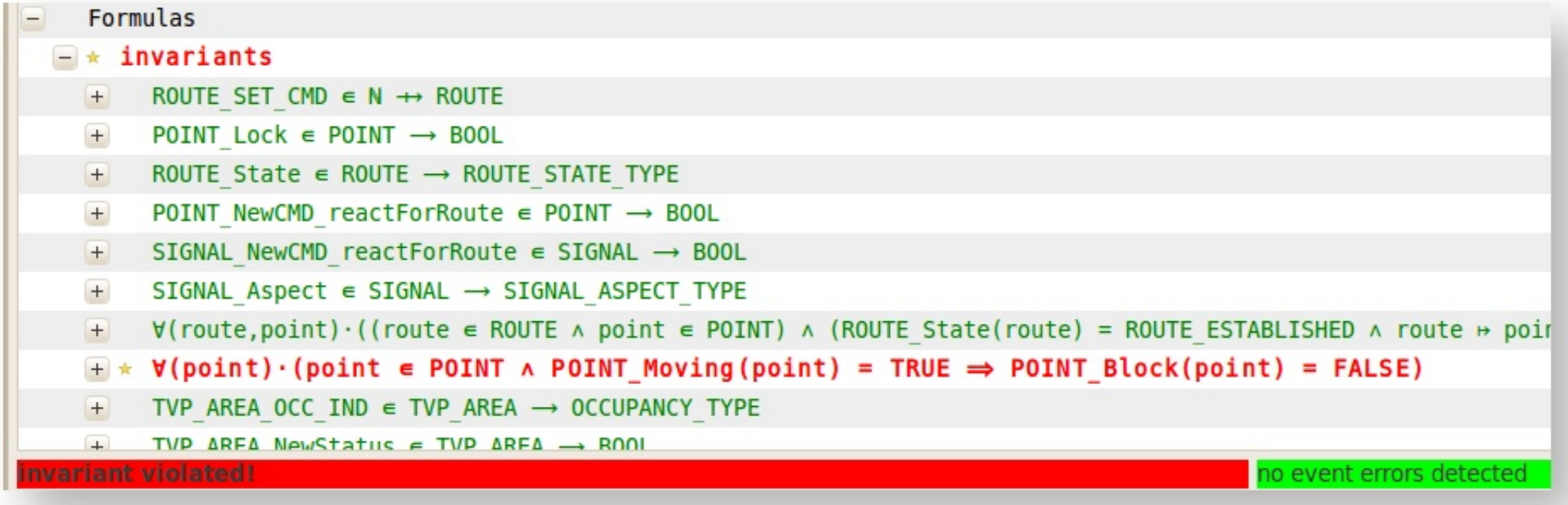


CASE Customer DEMO Y: as we can see in the picture, if we try to manually block a point that is moving (W_102), the command is accepted.

In this case we the system is in a state with a point moving and blocked at the same time.



In this moment the Invariant is not TRUE anymore and the tool detect this forbidden behavior.



CONCLUSION:

We have techniques using the Formal Methods to find these inconsistences in the behavior. Since the Product Lines allows the creation of multiple products by reusing and combining components, even more now is needed this kind of detection in the very early phases of the process (Specification).