

Overview of ADVANCE Process and Tools

John Colley, Michael Butler
ADVANCE Industry Day 2014

Outline

- ADVANCE Project Overview
- Activities supported by ADVANCE Tools
- ADVANCE Process

ADVANCE(287563)

Advanced Design and Verification Environment for Cyber-Physical System Engineering www.advance-ict.eu

- Cyber-Physical Systems
- Key Innovation
- Technical Approach
- Demonstration and Use

Cyber-Physical Systems

- Integrations of Computing and Physical Mechanisms
 - provide physical services
 - Transportation
 - Energy Distribution
 - Medical Care
 - Manufacturing
 - with increased
 - Adaptability
 - Autonomy
 - Efficiency
 - Safety

Cyber-Physical System Challenges

“... the lack of temporal semantics and adequate concurrency models in computing, and today’s “best effort” networking technologies make predictable and reliable real-time performance difficult, at best. ”

Cyber-Physical Systems - Are Computing Foundations Adequate?

Edward A. Lee, EECS, UC Berkeley, 2006

Verifying Cyber-Physical Systems

- Most Traditional Embedded Systems are *Closed Boxes*
 - amenable to *Bench Testing*
- Cyber-Physical Systems
 - are typically networked
 - can have complex interactions with their physical environment
 - pose a much greater verification challenge
- How can predictable behaviour and timing be achieved?

Cyber-Physical Systems - Are Computing Foundations Adequate?
Edward A. Lee, EECS, UC Berkeley, 2006

Key Innovation of ADVANCE

- Focuses on the key role played by *Modelling* in Cyber-Physical System Engineering
 - Modelling is used at *all stages* of the Development Process
 - From Requirements Analysis to System Acceptance Testing
 - *Augments* Formal, Refinement-based Modelling and Verification with
 - Simulation
 - Testing
- in a Single Design and Verification Environment*

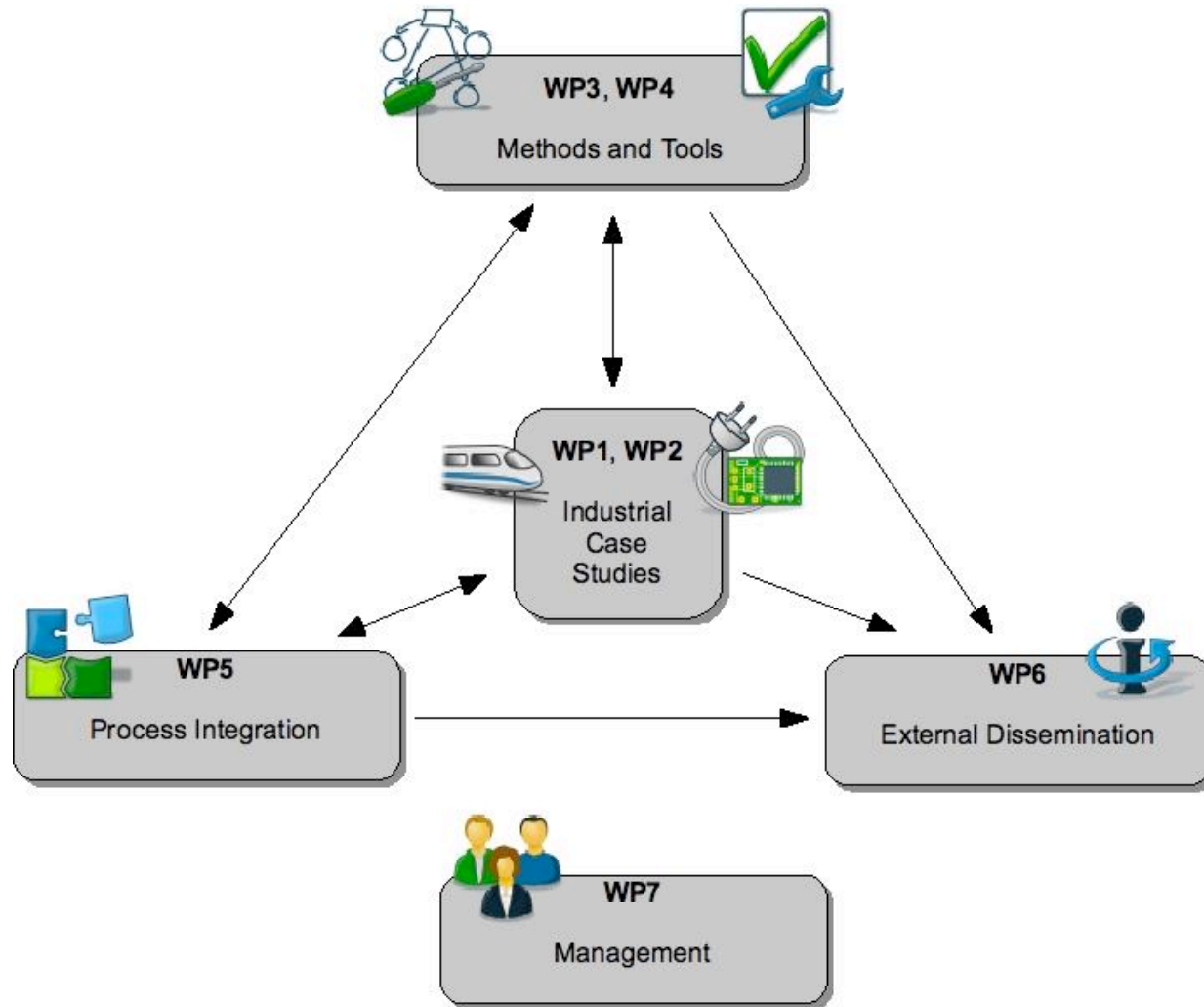
Technical Approach: Overview

- **Formal Modelling** supported by strong Formal Verification Tools to establish deep understanding of Specification and Design
- **Simulation-based Verification** to ensure that the Formal Models exhibit the expected behaviour and timing in the target physical environment
- **Model-based Testing** for the systematic generation of high-coverage test suites








ADVANCE Multi-Simulation Framework

- Different simulation tools are better suited to simulating different parts of a Cyber-physical system:
 - Environments
 - Controllers
 - Physical Plant
- The **ADVANCE Framework** manages the co-operation of multiple simulators to enable effective Cyber-physical system verification

Demonstration and Use



ADVANCE Workpackages

WP1 Dynamic Trusted Railway Interlocking Case Study	Alstom	
WP2 Smart Energy Grids Case Study	Critical/Selex	
WP3 Methods and Tools for Model Construction and Proof	Systerel	
WP4 Methods and Tools for Simulation and Testing	Düsseldorf	
WP5 Process Integration	Southampton	
WP6 External Dissemination and Exploitation	Critical	
WP7 Management	Southampton	

Achieving high assurance is not easy

- **Requirements** are poorly understood and analysed
- No software system is self-contained
 - it operates within a potentially **complex environment**
 - complexity of environment means that **hazards / vulnerabilities** in environment are poorly understood
- Designs are verified only **after** implementation
 - **expensive** to fix
 - verification usually **incomplete** – many undiscovered bugs
 - Ensuring coverage of faults/attacks in testing is difficult

Verified Design with Event-B

- **Formal** modelling at **early stages** to prevent errors in understanding requirements and environment
- **Verify conformance** between high-level specifications and designs using incremental approach
- **Rodin: open source toolset** for modelling, verification and simulation

Safety/security properties in Event-B

- Aircraft landing gear:
Gear=retracting \Rightarrow Door=open
- Railway signalling safety:
 - The signal of a route can only be green when all blocks of that route are unoccupied
sig(r) = GREEN \Rightarrow blocks[r] \cap occupied = \emptyset
- Access control in secure building:
 - **if** user u is in room r , **then** u must have sufficient authority to be in r
location(u) = r \Rightarrow
takeplace[r] \subseteq authorised[u]

Refinement in Event-B

- High level models
 - abstract details, allowing focus on system-level properties
- Refined models
 - introduce more requirements or design details
- Conformance:
 - behaviour exhibited by refined model should be allowed by abstract model
- Example, signalling mechanism as a refinement:
 - System level property:
Gear=retracting \Rightarrow Door=open
 - Design level properties:
Gear=retracting \Rightarrow GearRetractSignal=TRUE
GearRetractSignal=TRUE \Rightarrow Door=open

Main features of the Rodin Toolset

- **Model Verification**
 - Ensure that Event-B models satisfy key properties formulated in a mathematical way
- **Model Validation**
 - Ensure that Event-B models accurately capture the intended behaviour / requirements of a system
- **Model Transformation**
 - Transform models from one representation to another, e.g.,
 - graphical to mathematical representation
 - model to code transformation

Simple Verification Example

Invariant: $x \leq y \leq x+C$ (y is bounded by x)

IncEvent $\hat{=}$ **when** $y < x+C$ **then** $y := y+1$ **end**

- Assume the the system is initialised to a state that satisfies the invariant.
- Can the system ever get into a state in which the invariant is **violated**?
- Formulate the question as a **mathematical** problem
 - Is this **theorem provable**?:
$$x \leq y \leq x+C \wedge y < x+C \Rightarrow x \leq y+1 \leq x+C$$
- NB: theorem and its proof hold for **all values** of x,y,C.

Proof Obligations and Provers

- In Event-B theorems such as these are called **Proof Obligations** (POs)
 - The Rodin tool generates the POs for a model automatically
- The Rodin provers (semi-)automatically construct **mathematical proofs** of the validity of the POs.

Counter examples for invalid models

- Suppose our event had a specification error:

Invariant: $x \leq y \leq x+C$ (y is bounded by x)

IncEvent $\hat{=}$ **when** $y \leq x+C$ **then** $y := y+1$ **end**

- A Model Checker can generate counterexamples that demonstrate the consequence of the error in *IncEvent* :

- **Before**: $x=0, y=2, C=2$ **ok**
- **After**: $x=0, y=3, C=2$ **fault**

- Model checker can also generate error traces from initial states:

- **Init**: $x=0, y=0, C=2$ **ok**
- **IncEvent**: $x=0, y=1, C=2$ **ok**
- **IncEvent**: $x=0, y=2, C=2$ **ok**
- **IncEvent**: $x=0, y=3, C=2$ **fault**

Model Verification in Rodin

- Proof Obligation generation
 - Invariant preservation
 - Refinement checking
- Automated and interactive proof
 - Proof manager uses a range of internal and external plug-in theorem proving tools
 - Customisable through proof tactics

Model Verification in Rodin (continued)

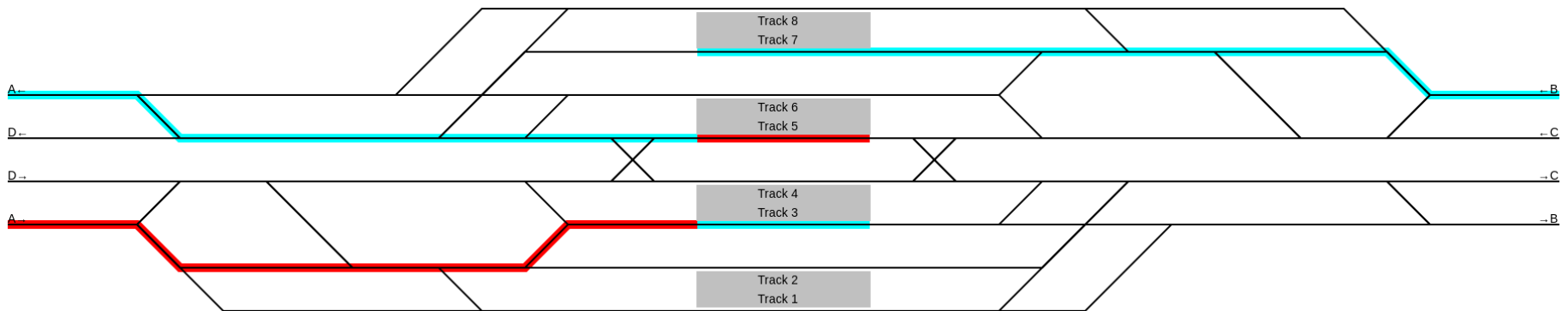
- Model checking with ProB plug-in: automated search for
 - invariant violations
 - refinement violations
 - deadlocks
- Proof Support for Domain-specific theories
 - Tables and operators for data manipulation
 - Hierarchical structures (e.g. file system)
 - Train occupancy as chains on a graph

Model Validation

- Requirements tracing
 - Validating a formal model against (informal) requirements involves human judgements
 - Strong structuring and tracability helps to ensure that the validation is *comprehensive* and *maintainable*
 - Tracing is supported by ProR plug-in
- Graphical animation
 - ProB provides a simulation engine for Event-B
 - BMotionStudio allows interactive graphical animations to be constructed, driven by the simulation engine
 - Very valuable for validating model, especially with domain experts

Graphical animation of *Stuttgart 21* model

- Visualization with ProB2:



- Track diagram clearly visible
- reserved blocks (blue) and occupied blocks (red)
- uncolored tracks are free in the current state

Model Validation (continued)

- Multi-simulation
 - Event-B models **discrete** event systems
 - Some environment variables are best represented as **continuous** quantities
 - E.g., voltage, temperature, speed,...
 - **Rodin multi-simulation** framework allows co-simulation of discrete and continuous models
 - links ProB with external simulation tools, e.g., Simulink, Modelica
 - Co-simulation allows us to **validate** a discrete controller model given certain **assumptions** about the (continuous) **environment** it controls
 - environment variables represented in a continuous model

Continuous / discrete co-simulation

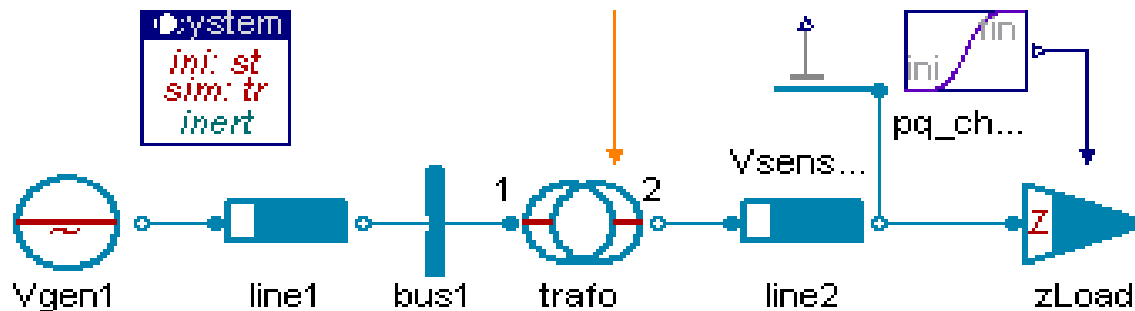


Figure 5. Distribution voltage control system in Modelica

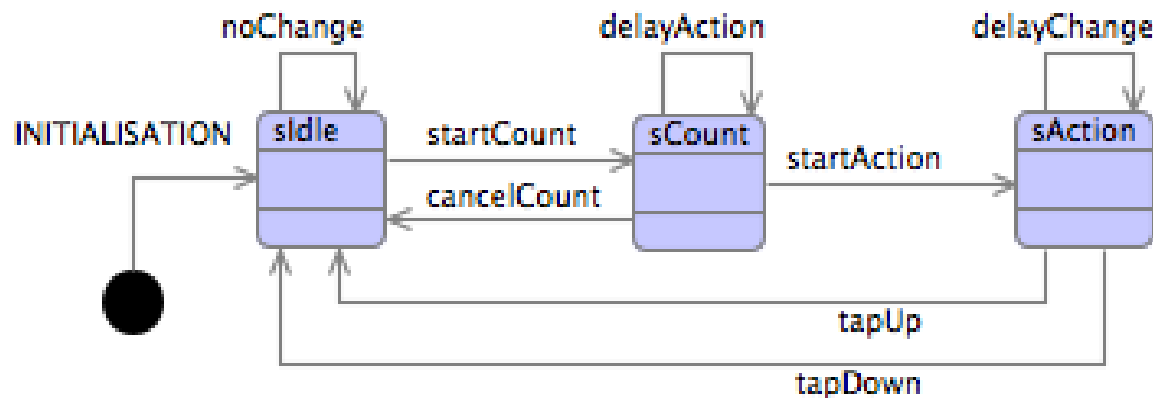


Figure 6. Event-B state machine of the OLTC controller

Co-simulation Results

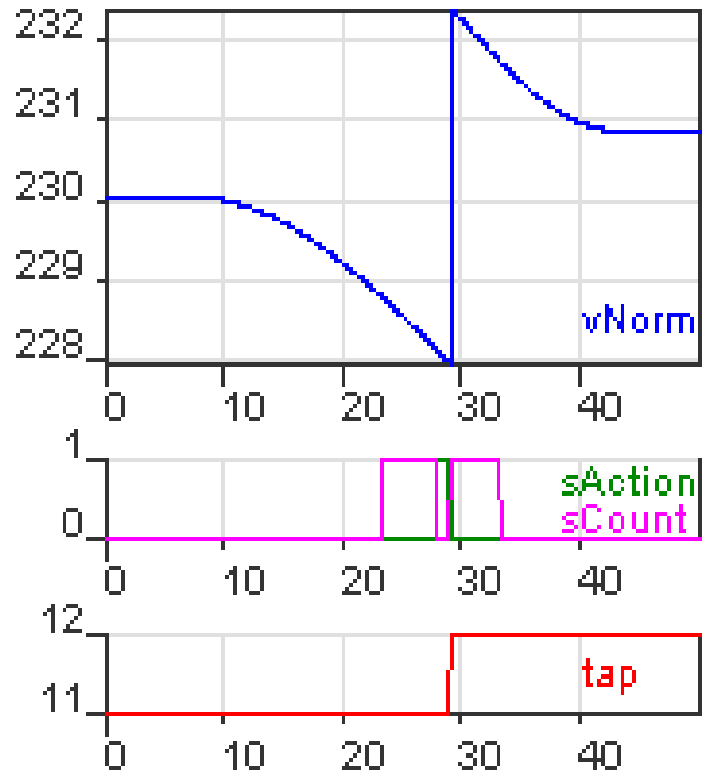
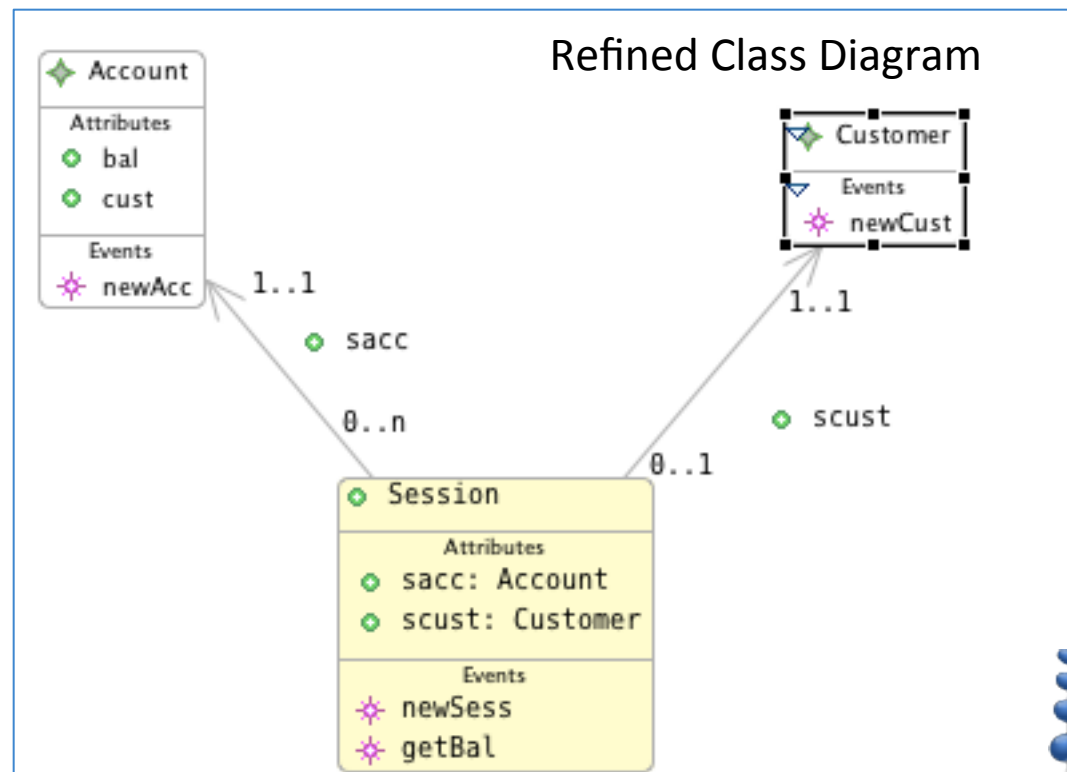
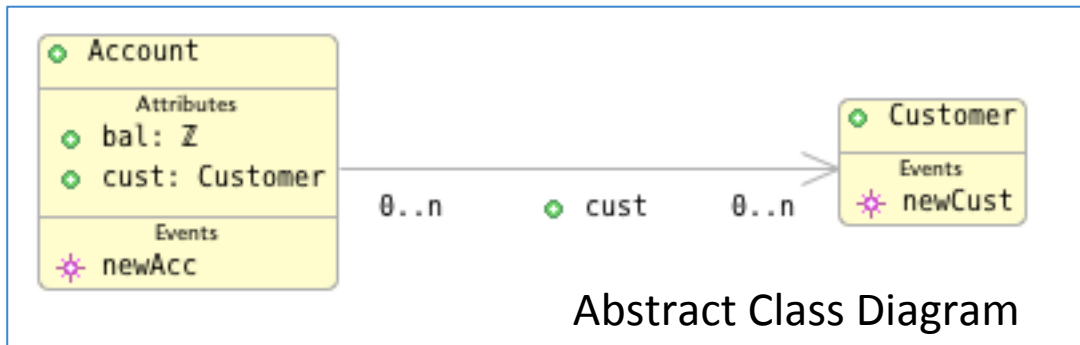


Figure 7. Co-simulation results of the OLTP voltage control (simulation time = 30s, step size = 0.1s)

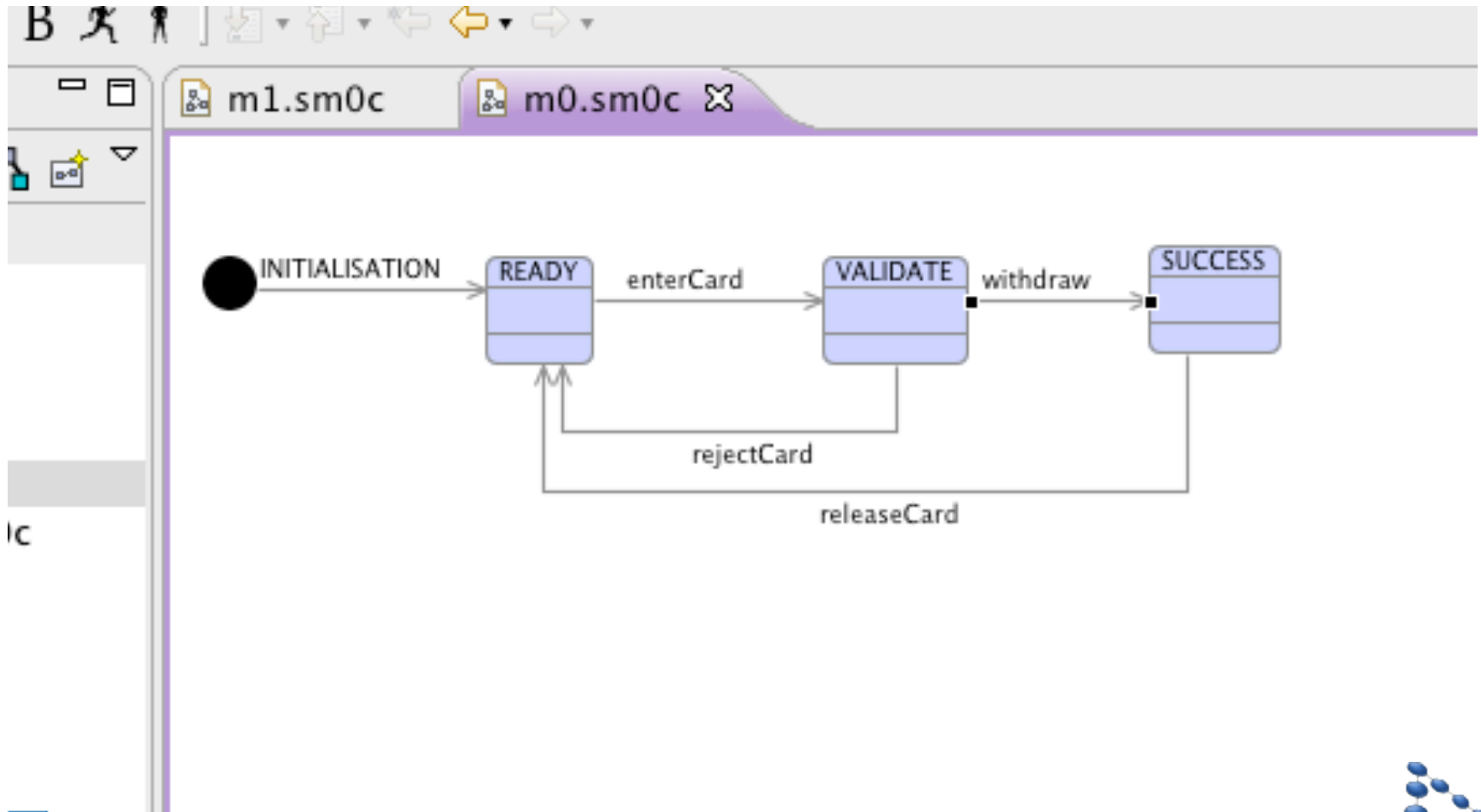
Model Transformation

- UML-B
 - UML-like graphical notation for Event-B
 - Supports class diagrams and statemachines
 - Graphical representation of refinement
- Composition and decomposition
 - Composition: combine models to form larger models
 - Decomposition: split large models into sub-models for further refinement and decomposition
 - Composition and decomposition need to be performed in a disciplined way
- Code generation
 - Generate C/Ada/Java from low-level models
 - Customisable
 - Support for generating multi-tasking implementations

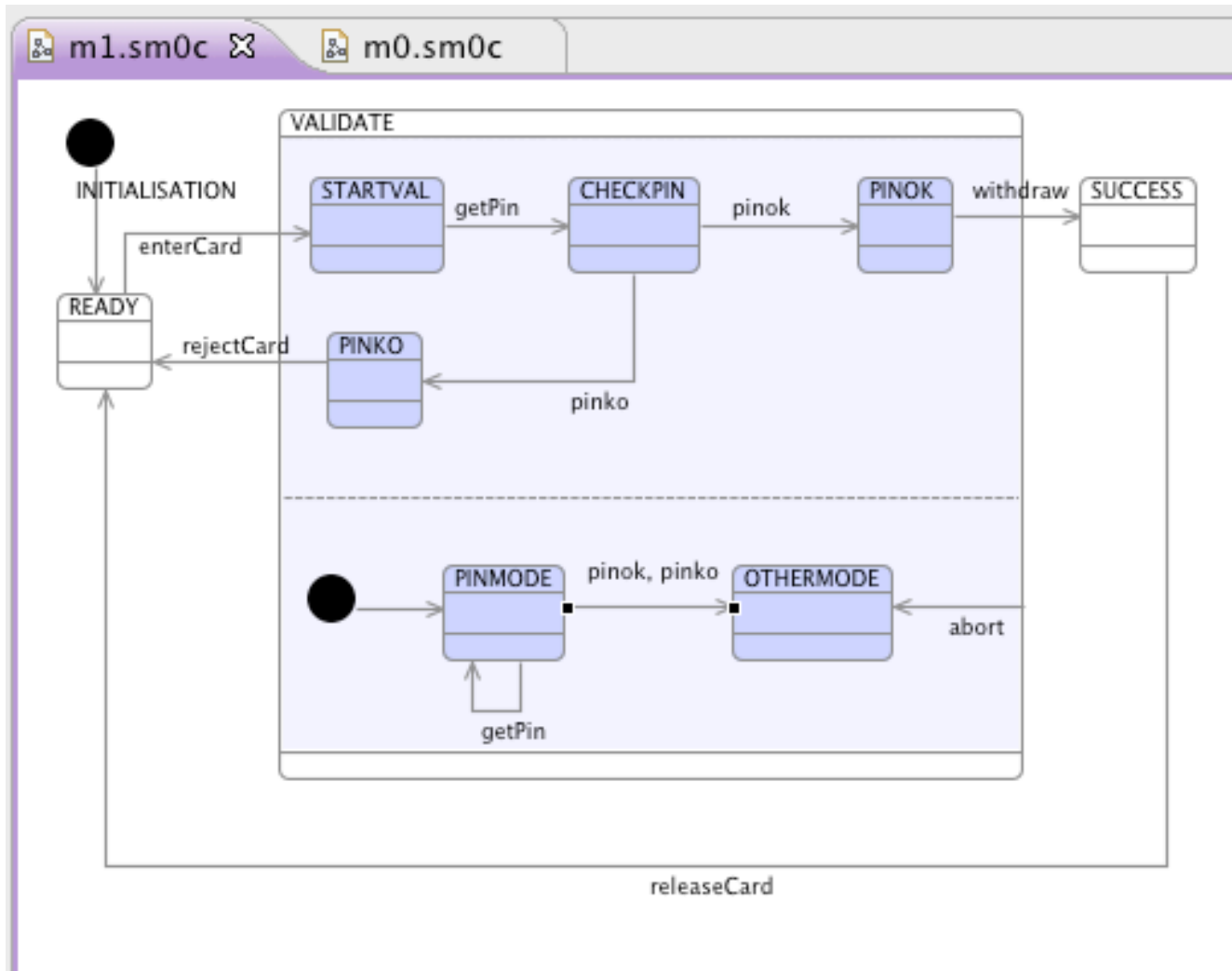
UML-B Class diagrams for Bank Accounts



UML-B State machine for ATM



Refined model of ATM



Rodin Architecture

- Extension of Eclipse Open Source IDE
- Core Rodin Platform manages:
 - Well-formedness + type checker
 - Consistency/refinement PO generator
 - Proof manager
- Extension points to support plug-ins
 - ProB, Bmotion Studio, ProR

The ADVANCE Process

- Deriving the Safety Constraints from the Functional Requirements using STPA
- Modeling the Safety Constraints in Event-B
 - System-level Safety Constraints
- Determining how Unsafe Control Actions could occur
- Documenting the Requirements and Design Decisions with ProR
- Refining the model *and safety constraints* to ensure Control Actions are safe in the presence of Hazards
 - Architecture-level Safety Constraints
- Constraint-based test generation and MC/DC coverage
- Shared Event Decomposition
 - Further refinement/ implementation
 - FMI-based Multi-simulation

The Functional Requirements

- System Overview
- Monitored Phenomena
- *Controlled Phenomena*
- Commanded Phenomena
- Mode Phenomena

Controlled Phenomena

Landing Gear Doors

1. The Controller will *open* the Doors when the Pilot moves the Lever to Extend or Retract the Landing Gear
2. The Controller will then *close* the Doors when the Landing Gear is fully Extended or Retracted
3. The Doors will remain *open* while the Landing Gear is Extending or Retracting

Safety Requirements

“Any controller – human or automated – needs a model of the process being controlled to control it effectively”

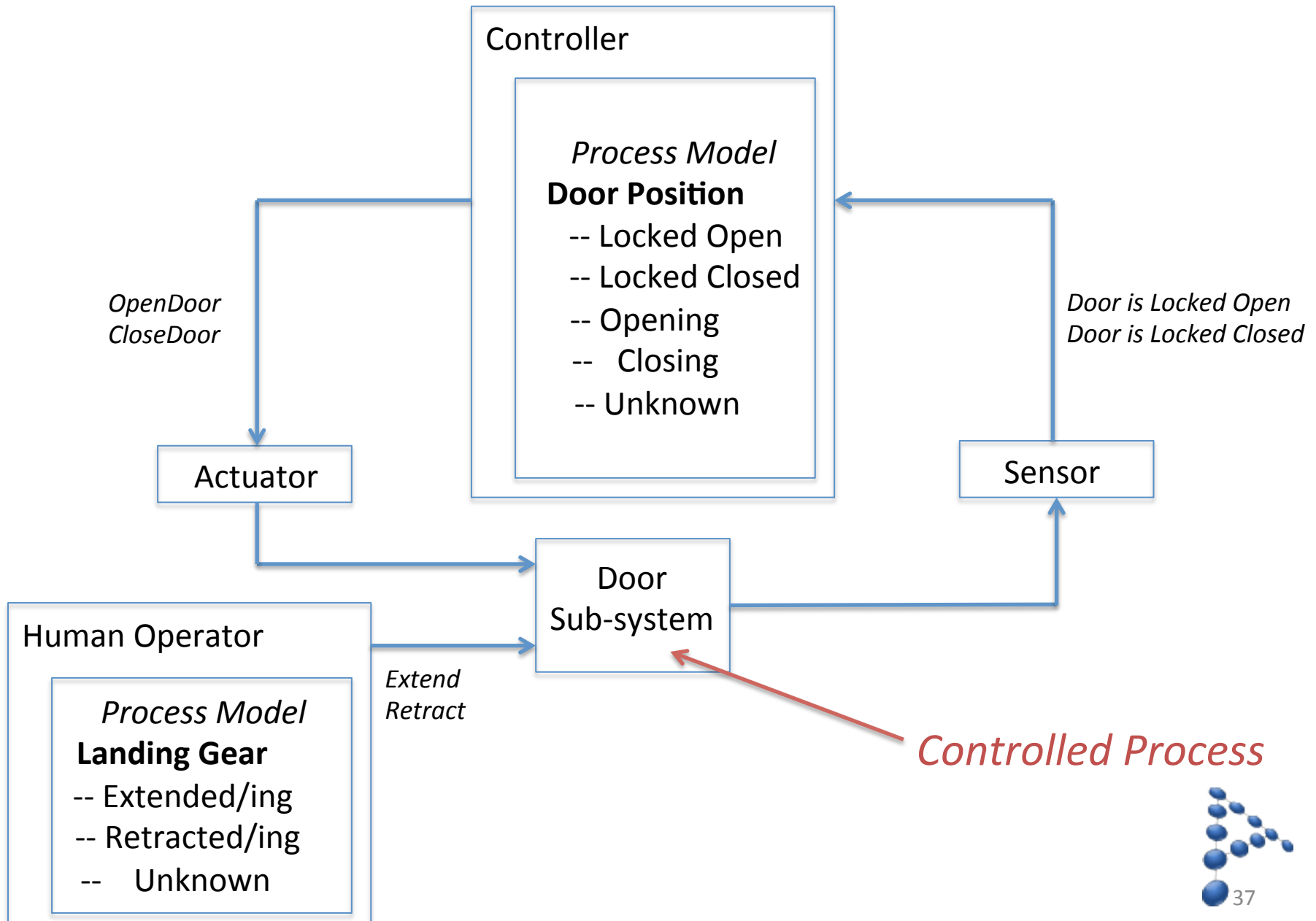
“Accidents can occur when the controller’s process model does not match the state of the system being controlled and the controller issues unsafe commands.”

Engineering a Safer World, Leveson, 2012

System-Theoretic Process Analysis (STPA)

1. Identify Potentially Hazardous Control Actions and derive the Safety Constraints
2. Determine how Unsafe Control Actions could occur

The Door Sub-system Process Models



Step I: Identify Potentially Hazardous Control Actions and Derive Safety Constraints

Controller Action	Not Providing Causes Hazard	Providing Causes Hazard	Wrong Timing or Order Causes Hazard	Stopped too soon/Applied too long
Open Door	Cannot extend Landing Gear for landing	Not Hazardous	Not Hazardous	Damage to Landing Gear/ Not Hazardous
Close Door	Not Hazardous	Damage to Landing Gear	Damage to Landing Gear	Not Hazardous/ Not Hazardous

Safety Constraints

1. If the Landing Gear is Extending, the Door must be Locked Open
2. If the Landing Gear is Retracting, the Door must be Locked Open
3. A “Close Door” command must only be issued if the Landing Gear is Locked Up or Locked Down
4. An “Open Door” command must only be issued if the Landing Gear is Locked Up or Locked Down

Deriving the Formal Safety Constraints

- Natural Language Constraints developed systematically by the *Domain Experts*
 1. If the Landing Gear is Extending, the Door must be Locked Open
 2. If the Landing Gear is Retracting, the Door must be Locked Open
 3. A “Close Door” command must only be issued if the Landing Gear is Locked Up or Locked Down
 4. An “Open Door” command must only be issued if the Landing Gear is Locked Up or Locked Down
- Formal, Event-B Safety Constraints
 - Derived systematically from the Natural Language Descriptions
 - Linked to Requirements
 - ProR

Deriving the Formal Safety Constraints

- Natural Language Constraints developed systematically by the *Domain Experts*

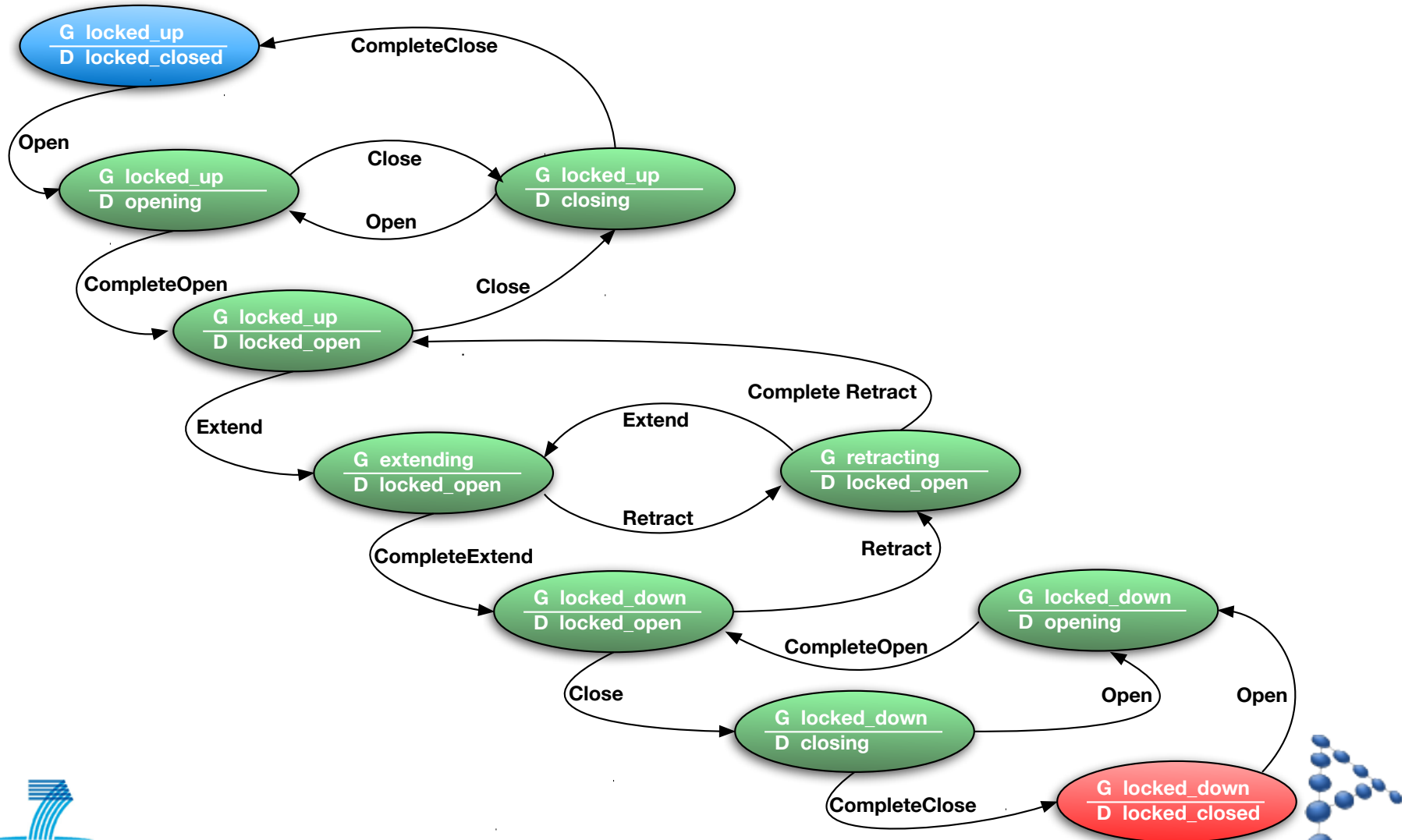
```
gearstate ∈ {locked_down, locked_up} ∨ doorstate = locked_open
```

- Formal, Event-Based

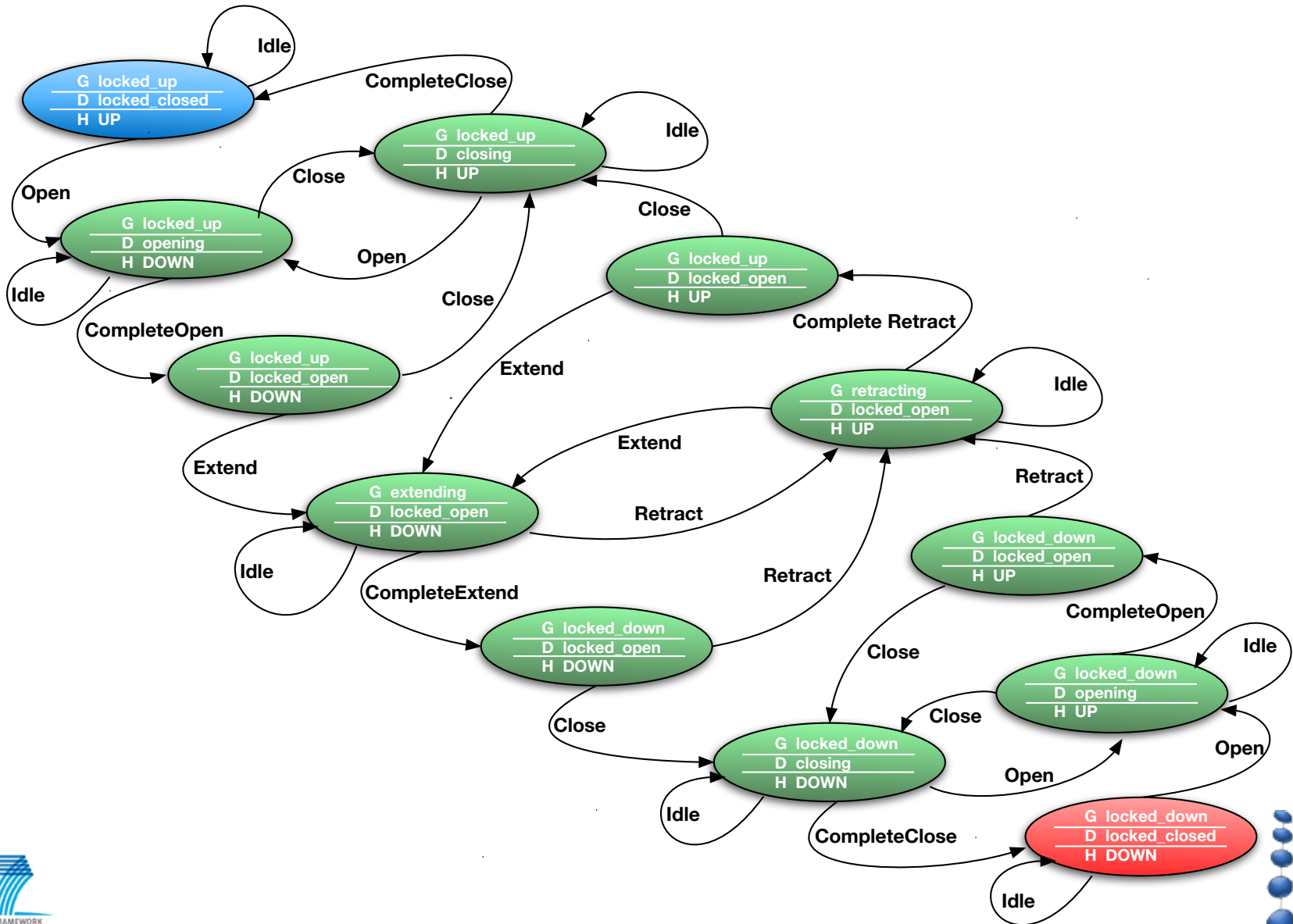
- Derived system
- Linked to Requirements
 - ProR

```
event Close
where
  @grd1 gearstate ∈ {locked_down, locked_up}
  @grd2 doorstate ∈ {opening, locked_open}
then
  @act1 doorstate = closing
end
```

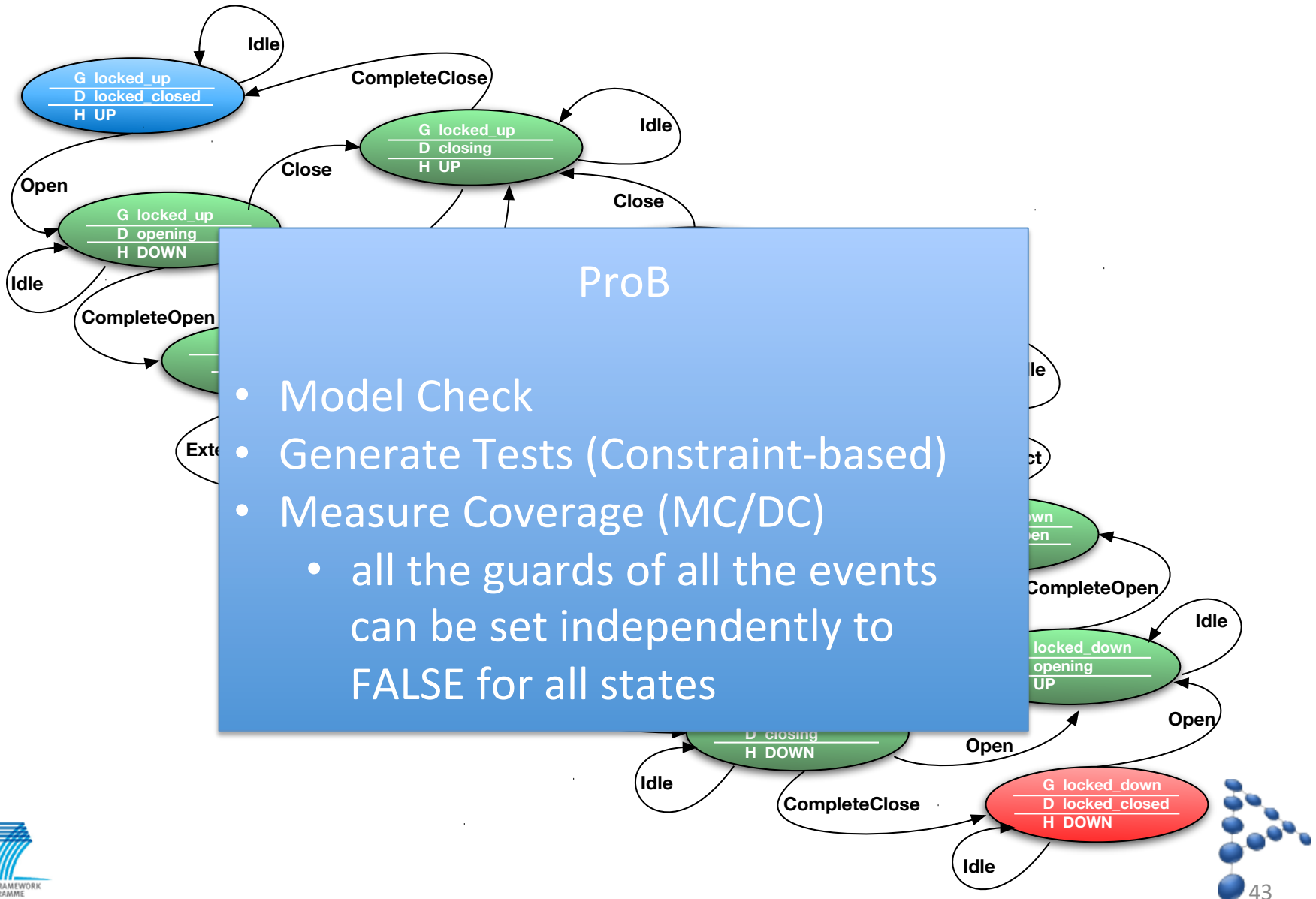

The Model Extended FSM



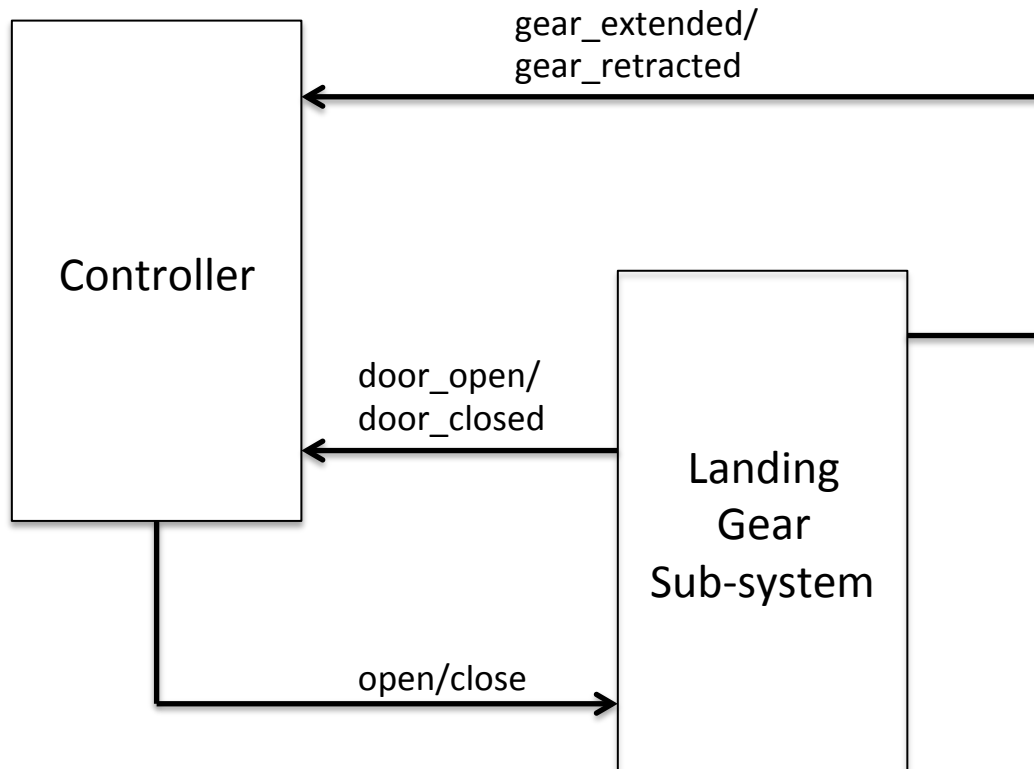
Refinement: Introducing the Handle and Timing



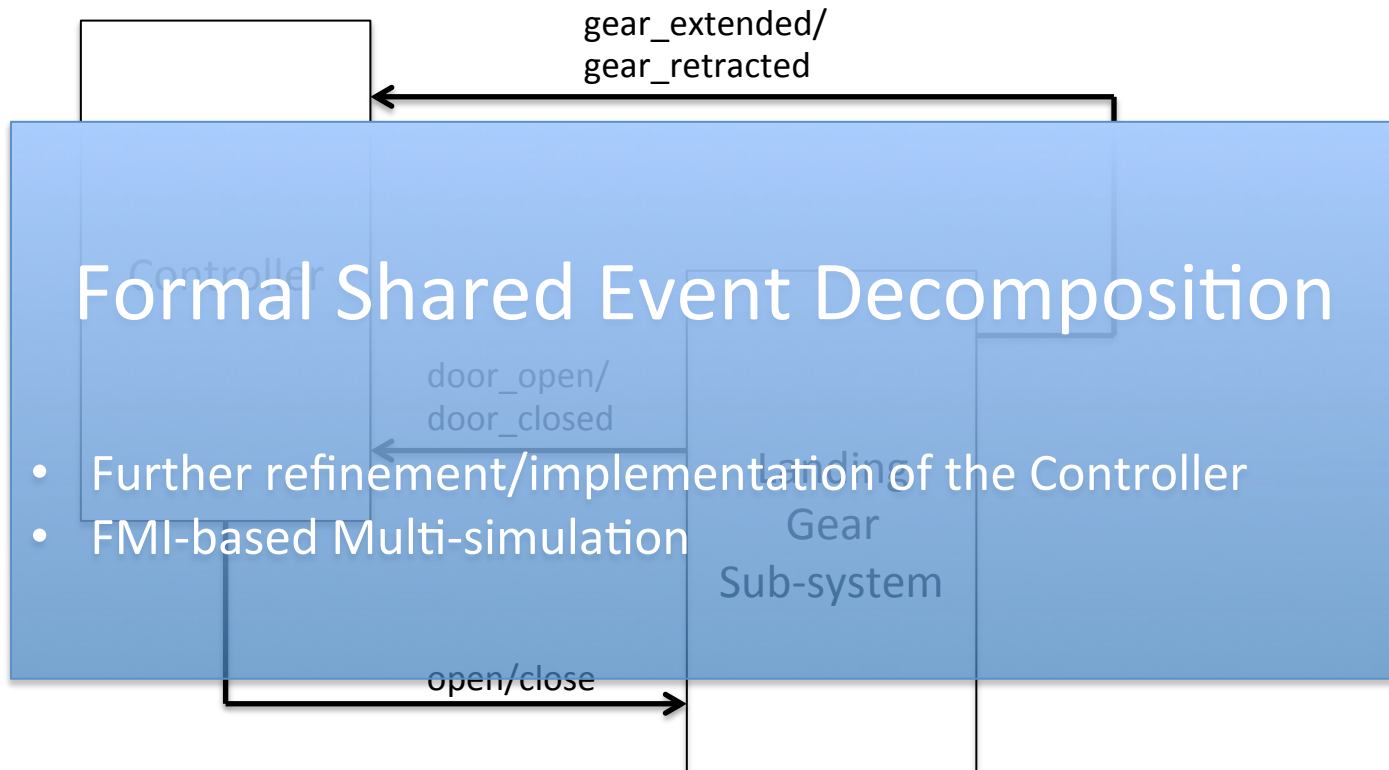
Refinement: Introducing the Handle and Timing



Refinement: The Component View Architecture-Level



Refinement: The Component View Architecture-Level



ADVANCE in Railway and Smart Grids

- WP1: ADVANCE in Railway Interlocking (Alstom)
 - Requirements and hazard analysis
 - Modelling
 - Model visualisation
 - Verification and proof
- WP2: ADVANCE in Smart Grids (Selex, Critical Software)
 - Requirements traceability
 - Modelling
 - Verification and proof
 - Application of FMI-based co-simulation

Role of ADVANCE in Certification

Contributions to stages of EN50129 certification (Rail sector)

- Hazard Analysis:
 - *STPA*
- Safety requirements specification:
 - *Event-B modelling*
- Requirements verification:
 - *Traceability, formal verification, co-simulation*
- Architecture Specification:
 - *Refinement + decomposition*
- Interface hazard analysis:
 - *Verification and logging of decomposition*
- System validation:
 - *Tests derived from models providing **functional** coverage*
 - *Verification and validation reports*

ADVANCE in Aerospace Certification

Supports *DO-254/178C* flows for design assurance of airborne electronic hardware and software

- *DO-331* addressing Model-based development and verification
- *DO-333* addressing Formal Methods to complement testing
- Combines formal and simulation-based verification with MC/DC coverage closure
- Traceable model-based development and verification from conception to certification

Important Messages

- System assurance can be strengthened
 - using **systematic processes** and **verified design**
- Role of **systematic** requirements and safety analysis
 - Structures to focus the analysis
 - Path to formalisation
- Role of **formal modelling** and **refinement**:
 - increase understanding, decrease errors
 - manage complexity through multiple levels of abstraction
- Role of **verification and tools**:
 - improve quality of models (validation + verification)
 - make verification as automatic as possible, pin-pointing errors and even *suggesting* improvements