Correct-by-Construction Development of Dependable Systems

Alexei Iliasov*, Linas Laibinis**, Elena Troubitsyna** and Alexander Romanovsky*

*Newcastle University, UK       **Åbo Akademi University, Finland
Structure of the tutorial

- Four parts (sessions), each split into presentation and interactive sections:
  - Part I – Introduction into formal methods and Event-B;
  - Part II – Model development by refinement;
  - Part III – Modelling and verification of system dependability properties;
  - Part IV – The Event-B ecosystem.
Part I – Introduction into formal methods and Event-B

- Advantages of formal modelling;
- History of Event-B;
- Event-B basics;
- Description of the running example.
Formal versus traditional development

- Traditional development
  - Process-based assurance

- Formal development
  - Product-based assurance

- Proofs give solid evidence of dependability;

- Formal modelling allows us to uncover design problems earlier;
  - Cheaper to fix, more robust architecture.
Formal development by refinement

- Abstract modelling
  - Helps to cope with complexity;
  - Focus on stating requirements and assumptions;
  - Allows us to spot requirements ambiguities and contradictions.

- Refinement
  - Elaboration on abstract models;
  - Structuring of requirements;
  - (Automated) proof of adherence to the abstract model.
Historical note

- The B Method: invented in 1990-s by J.-R. Abrial to formally specify and develop sequential systems correct by construction;

- 1990-s: The Action Systems formalism by R.Back and K.Sere;


- from 2007: The Rodin Platform – free industrial-strength tool support for Event-B.

- Wide use of Event-B in the railway domain.
The dynamic system behaviour is described in terms of guarded commands (events):

- Stimulus $\rightarrow$ response.

General form of an event:

\[
\text{WHEN guard THEN action END}
\]

where

- \textit{guard} is a state predicate defining when an event is enabled;
- \textit{action} is (possibly non-deterministic) update of state variables.
Overall system behaviour: a (potentially) infinite loop of system events:

```
forever do
  Event1 or
  Event2 or
  Event3 or ...
end
```

- **Model invariant** defines a set of allowed (safe) states;
  - Each event should preserve the invariant;
  - We should verify this by proofs.
A system model in Event-B

Machines contain the dynamic structure of the system (variables, invariants, events)

Contexts contain the static structure (constants and axioms)

**Machine**
- variables
- invariants
- theorems
- events
- variant

**Context**
- carrier sets
- constants
- axioms
- theorems

A machine "sees" its contexts
Proof-based semantics: consider all possible executions at once;

A model is converted into a number of proof obligations;

A proof obligation is a mathematical theorem;

Every proof obligation must be proven correct.
Invariant preservation

- An invariant property is assumed to hold before every event;

- Each event must re-establish it:

\[ I(s, c, v) \land G(s, c, v) \land R(s, c, v, v') \implies I(s, c, v') \]

where

- \( I(s, c, v) \) – the model invariant;
- \( G(s, c, v) \) – the event guard;
- \( R(s, c, v, v') \) – the event action;
- \( v \) – old states;
- \( v' \) – new states.
The Rodin platform – tool support for Event-B

- Automates incremental development by refinement;

- Supports strong interplay between modelling and proof;
  - Reactive: analysis tools are automatically invoked in the background whenever a change is made.

- The platform is extendable by plug-ins that
  - extend the Event-B language and proving techniques;
  - bridge the platform with various model-checkers, theorem provers, animators, modelling notations (e.g., UML), etc.
Interactive session
Sluice connects areas with dramatically different pressures;
It is unsafe to open a door unless the pressure is levelled between the connected areas;
The purpose of the system is to operate doors safely by adjusting the pressure in the room.
Requirements: system

**SYS** the purpose of the system is to allow a user to safely travel between inside or outside areas;

**ENV1** the system has three locations - outside, middle and inside;

**ENV2** the system has two doors - door 1, connecting outside and middle, and door 2, connecting middle and inside;

**ENV3** a pump is located in the middle area.
Requirements: pressure

LOC1 pressure in the inside area is always LOW;
LOC2 pressure in the outside area is always HIGH;
SENS1 middle area has a pressure sensor reporting (without a delay) the current pressure.
Requirements: safety

**SAF1** a door may be open only if the pressures in the locations it connects are equalised;

**SAF2** at most one door is open at any moment;

**SAF3** pressure may only be changed when the doors are closed.
**Assumptions: user**

**USR1** a human operator positioned in the inside area is willing to get to the outside area;

**USR2** a human operator positioned in the outside area is willing to get to the inside area;

**USR3** a human operator promptly travels through an open door but only in the direction of travel.
Requirements: operational goals

**OPR1** when a human operator is outside, he eventually gets inside;

**OPR2** when a human operator is inside, he eventually gets outside.
Refinement
Part II – The notion of model refinement

- Intuition: gradual elaboration on the system behaviour and data structures;

- A top-down, correct-by-construction development of a system;

- Refined behaviours and data structures should be consistent (non-contradictory) with more abstract ones;

- Helps to structure the system requirements;

- A way of handling of the system complexity and structuring of the proof effort;
The notion of model refinement (cont.)

- Essential property: transitivity. Allows us to build a refinement chain of gradual development (unfolding) of the system;

- Mathematically:
  \[ M_1 \sqsubseteq M_2 \sqsubseteq \ldots \sqsubseteq M_n \]
  \[ M_1 \sqsubseteq M_n \]

- Refinement preserves externally observable behaviour;

- Many formalisations based on the idea of model refinement, e.g., Refinement Calculus, the B Method, ...
Refinement in Event-B

- Defined separately for a context and a machine;

- For a context component, it is called extension;

- Context extension allows
  - introducing new data structures (sets and constants), as well as
  - adding more constraints (axioms) for already defined ones.
For a machine component, there are several possible kinds of refinement:

- simple extension of an abstract model by new variables and events (*superposition refinement*);

- constraining the behaviour of an abstract model (*refinement by reducing model non-determinism*);

- replacing some abstract variables by their concrete counterparts (*data refinement*);

- a mixture of those.
Superposition refinement

- Adding new variables and events;
- Reading and updating new variables in old event guards and actions;
- Interrelating new and old variables by new invariants;
- **Restriction**: the old variables cannot be updated in new events!
Refinement of non-determinism

- Focuses on the old (abstract) model events:
  - Strengthening the guards;
  - Providing several versions of the same event;
  - Refining non-deterministic actions (assignments).

```
evt =
  WHEN g THEN
    detected :∈ BOOL
  END

evt1 refines evt =
  WHEN g ∧ g' THEN
    detected := TRUE
  END

evt2 refines evt =
  WHEN g ∧ g'' THEN
    detected := FALSE
  END
```
Data refinement

- Replacing some old variables by their concrete counterparts;

- A part of concrete invariant, *gluing invariant*, describes the logical relationships between the old and new variables:

- The gluing invariant is used in all proofs to show the correctness of such a replacement.

\[(\text{comm\_failure} = \text{TRUE}) \iff (\text{msg\_sent} = \text{FALSE} \lor (\text{msg\_sent} = \text{TRUE} \land \text{msg\_lost} = \text{TRUE}))\]
Refinement proof obligations

- As an abstract model, a refined model should satisfy invariant preservation properties;

- In addition, we should show that
  - guards of the old events are strengthened (or remain the same);
  - actions of the old events *simulate* those of the abstract ones – each refined model transition (execution step) is allowed by the abstract model;

- In all POs, the gluing invariant is used to relate the old and new model states.
Sluice example: a control system

The sluice system is an instance of a control system.

The general structure of control systems:
The control systems are cyclic:
- get inputs from the sensors,
- process them;
- output new values to the actuators.

The overall behaviour of the system is an alternation between the events modelling plant evolution and controller reaction.
Fault tolerance

- Safety cannot be achieved without fault tolerance (FT);
- Main goal of FT: prevent propagation of a fault to system boundaries (and potentially jeopardise safety);
- Steps of fault tolerance: error detection and error recovery;
- General principle of error detection: find a discrepancy between the expected state of a fault-free system and the observed state.
Specifying a controller

To ensure fault tolerance, its cyclic execution is often split into three steps:

- error detection based on read sensor values;
- update of its internal state and decision on possible control actions (based on both the sensor values and error detection results);
- prediction of the expected sensor values for the next cycle.
Phases of a controller

To ensure fault tolerance, its cyclic execution is often split into three steps:

- error detection based on read sensor values;
- based on the sensor values and error detection results, update of its internal state and decide on control actions;
- prediction of the expected sensor values for the next cycle.
Sluice example: refinement plan

Five small incremental refinement steps:

- Introducing feedback loop of a control system (m1);
- Elaborating on the environment part and adding sensors (m2);
- Data refining failure modes (m3);
- Elaborating on error detection (m4);
- Introducing actuators and refining error prediction (m5).
Sluice example: refinement plan

Five small incremental refinement steps:

- Introducing feedback loop of a control system (m1);
- Elaborating on the environment part and adding sensors (m2);
- Data refining failure modes (m3);
- Elaborating on error detection (m4);
- Introducing actuators and refining error prediction (m5).
The goal – to abstractly model the feedback loop of a control system. An example of superposition refinement:

- Introduce a new type \( \text{PHASE} = \{ \text{ENV}, \text{DET}, \text{CONT}, \text{PRED} \} \);
- Add new variables \( \text{phase} \in \text{PHASE} \) and \( \text{failure} \in \text{BOOL} \);
- Introduce new events Environment, Detection and Prediction with the corresponding guards;
- Strengthen guards of the events \( \text{open1} \) and \( \text{close1} \);
- Introduce new events \( \text{stop} \) and \( \text{other\_control} \).
The goal – to introduce positioning sensors for door1. An example of superposition refinement:

- Add new variables door1\_open\_sen ∈ BOOL, door1\_closed\_sen ∈ BOOL and door1\_pos\_sen ∈ 0..100;
- Model reading of sensor values in Environment;
- Strengthen guards of the events open1 and close1 to only react to specific sensor values.
The goal – to replace the abstract representation of system failure with a more concrete one. An example of data refinement:

- Remove the variable \textit{failure} from the model;
- Add new variables \textit{door1\_sen\_failure} $\in$ BOOL and \textit{other\_failures} $\in$ BOOL;
- Add a gluing invariant linking \textit{failure} with \textit{door1\_sen\_failure} and \textit{other\_failures};
- Modify the events, replacing all the occurrences of \textit{failure} accordingly.
The bool() operator

- Evaluates a logical expression and returns a boolean value, e.g., TRUE or FALSE;

- Using in an invariant or guard:

  \[ v = \text{bool}(expr) \]

  is the same as

  \[ (v = \text{TRUE}) \iff (expr = \text{TRUE}) \]

- Using in an event action allows us to avoid having two events, one for the case when \( expr = \text{TRUE} \), and the other one for the case when \( expr = \text{FALSE} \).
Sluice example: the fourth refinement

The goal – to elaborate on the error detection procedure. An example of both non-determinism refinement and superposition refinement:

- Add a new variable \textit{retries};
- Refine the event \textit{Detection} into several events covering different detection procedures: \textit{Detection\_open1}, \textit{Detection\_closed1}, and \textit{Detection\_pos\_range};
- Add a new action \textit{retries} := 0 into the events \textit{open1} and \textit{close1};
- Refine the event \textit{other\_control} into \textit{retry} to model retrying of sensor readings.
Sluice example: the fifth refinement

The goal – to elaborate on error prediction and detection as well as introduce system actuators. An example of both non-determinism refinement and superposition refinement:

- Introduce a new type \textit{MOTOR} and function constants \textit{min\_exp\_pos}, \textit{max\_exp\_pos};
- Add new variables \textit{motor1}, \textit{min\_pos1}, and \textit{max\_pos1};
- Refine the event \textit{other\_control} into the events \textit{motor\_open1} and \textit{motor\_close1} to model setting of actuators;
- Modify the events \textit{open1} and \textit{close1} accordingly;
- Refine the event \textit{Prediction} to calculate the expected range of door1 position values: \textit{min\_exp\_pos} \ldots \textit{max\_exp\_pos};
- To introduce a new detection event \textit{Detection\_not\_expected} as a refinement of \textit{Detection}.
Modelling and verifying dependability
Ensuring dependability of complex control systems is challenging;

Formal modelling and refinement in Event B helps to structure complex requirements and develop systems that are correct and safe by construction;

How to capture the results of safety analysis in a formal specification?

Need methods that bridge (informal) safety analysis techniques with formal specification.
FMEA is a well-known inductive safety analysis technique; for each system component it defines its possible failure modes, local and system effect of component failures, as well as detection and recovery procedures;

FMEA information is presented in a tabular format.
FMEA table fields:

- **Component** – name of a component;
- **Failure mode** – possible failure modes;
- **Possible cause** – possible cause of a failure;
- **Local effects** – caused changes in the component behaviour;
- **System effect** – caused changes in the system behaviour;
- **Detection** – determination of the failure;
- **Remedial action** – actions to tolerate the failure.
The failure mode "Faulty sensing of door position" of the Door1 component:

<table>
<thead>
<tr>
<th>Component</th>
<th>Door1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure mode</td>
<td>Faulty sensing of door position</td>
</tr>
<tr>
<td>Possible cause</td>
<td>Loss of precision of door switch-type or value-type positioning sensors</td>
</tr>
<tr>
<td>Local effects</td>
<td>Mismatch between door sensor readings</td>
</tr>
<tr>
<td>System effects</td>
<td>Switch to degraded or manual mode or shut down</td>
</tr>
<tr>
<td>Detection</td>
<td>Mismatch between door sensor readings</td>
</tr>
<tr>
<td>Remedial action</td>
<td>Retry three times. If failure persists, switch to degraded mode and raise alarm.</td>
</tr>
</tbody>
</table>
The failure mode ”Door positioning out of expected range” of the Door1 component:

<table>
<thead>
<tr>
<th>Component</th>
<th>Door1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure mode</td>
<td>Door positioning out of expected range</td>
</tr>
<tr>
<td>Possible cause</td>
<td>Loss of precision of door positioning</td>
</tr>
<tr>
<td>Local effects</td>
<td>Sensor reading is out of the expected range</td>
</tr>
<tr>
<td>System effects</td>
<td>Switch to degraded or manual mode or shut down</td>
</tr>
<tr>
<td>Detection</td>
<td>Comparison of the received value with the predicted one</td>
</tr>
<tr>
<td>Remedial action</td>
<td>Retry three times. If failure persists, switch to degraded mode and raise alarm.</td>
</tr>
</tbody>
</table>
Experience in modelling fault tolerance

- In the tutorial we have modelled fault tolerance by retrying:
  - a mechanism to tolerate transient faults.

- Fault tolerance requires redundancy (of components, data, time);

- We have done extensive work on modelling various mechanisms of fault tolerance.
Fault masking: Triple Modular Redundancy (TMR)

- Relies on (physical) component redundancy;
- Often used to mask sensor faults;
- In Event-B: each module (sensor) can be modelled by the corresponding variables (partitions of state);
- TMR can be then introduced as a part of data refinement step.
Fault tolerance and backward recovery

- Return to some check-point and redo a part of computation;

- Relies on time redundancy (we have to produce the result before the deadline for the output).

```
s1 → s2 → s3 → s4 → s5 → s6[Error detected] → s7
```
Dependability engineering

- Dependability of a computing system is the ability to deliver a service that can be justifiably trusted.

- Dependability attributes: availability, reliability, safety, security, integrity, maintainability;

- Means for dependability:
  - fault prevention (rigorous design);
  - fault removal;
  - fault tolerance;
  - fault forecasting.
Safety analysis techniques

- Techniques for achieving various dependability attributes are tightly connected, e.g., safety analysis techniques allow us to derive fault assumptions;

- Fault tolerance contributes to system safety by making component failures less probable;

- In the tutorial we have considered an inductive technique – Failure Mode and Effect analysis (FMEA);

- Another widely used safety technique is Fault Tree Analysis (a deductive technique).
Fault Tree Analysis (FTA)

- Hazard – an undesirable state that can lead to an accident;
- FTA allows us to identify dependencies between the basic events and the event that constitutes the hazard;
- Steps of FTA: identify a hazard and build a fault tree, which logically connects the hazardous event with the compound events causing it etc.);
- The information analysed: identified hazards, system structure, failures and functional deviations, logical dependencies.
Example: an industrial crane
FTA of a crane

- Damage of equipment
  - Load collides with containers
    - Gantry moves with not lifted load
    - Gantry moves simultaneously with lowering load
      - X or Y movement
      - Z movement
  - Load falls on containers
Safety requirements (goals)

Examples:

- Z (axis) movement must not be performed simultaneously with X,Y movements;

- The load must be in the upper position before X,Y movements can start;

- The load must not be released when X,Y movements are performed.
Safety in Event-B development

- A safety property specifies a proper behaviour of the system;

- Initially, a desired safety property is defined using abstract system variables;

- We unfold it by refinement until it refers to the basic system components;

- At each refinement step we formulate gluing safety invariants that relate the newly introduced variables and abstract variables present in the safety property.
Quantitative safety analysis

- Relies on the probabilistic extension of Event-B;

\[
\text{sensor\_reading} \equiv \text{OK\_reading} \oplus_p \text{failure}
\]

- At the final refinement step, we arrive at a probabilistic model of the behaviour of each component and hence can calculate the probability of safety breach.
Formal modelling and certification of safety-critical systems

- IEC 61508: four safety integrity levels (SILs);
  - SIL 3 requires formal modelling;
  - SIL 4 requires formal verification.

- Safety cases are widely used to justify safety

  \[ \text{Goal} \rightarrow \text{Strategy} \rightarrow \text{Argument} \]

- How Event-B models can facilitate creating a safety case?
Linking Event-B and safety cases

Goals:
- Usually goals correspond to system safety requirements;
  - All target requirements should be represented in the model.

Argument:
- A technique showing how the goal is achieved;
  - Each requirement should be verified.

We propose
- A taxonomy of requirements;
- Define how they should be reflected in the model;
- Define the verification means.
Static safety requirements
- Modelled as invariants as well as guard strengthening in refinements;
  - Verified by the invariant-related proofs.

Dynamic safety requirements
- Modelled as a certain sequence of events;
  - Verified deadlock freedom (by model checking);
  - Verified the desired event order (by flow verification).
Rodin Ecosystem
Rodin Ecosystem

Rodin Platform is a modelling environment

- theorem proving is the main validation technology
- Event-B is its principal formal *notation*

It is possible and often necessary to change or extend

- the notation
- means of verification
Rodin Ecosystem: notations

- Offer an alternative notation
- Augment Event-B with new notational elements
  - make models more concise in for a certain problem domain
  - improve notation expressiveness
Rodin Ecosystem: notations

- Offer an alternative notation
  - UML-B, Event-B/SLP
- Augment Event-B with new notational elements
  - modal view, decomposition, state diagram, flow, model instantiation, qualitative probability, records, ...
Rodin Ecosystem: verification

- provers and solvers
  - AtelierB provers, Isabelle connection, SMT connection, ProB disprover
- animation and model checking
  - ProB, AnimB
Rodin Ecosystem: tool chain

Rodin may be integrated into a domain-specific development tool chain.

- Requirement tracing: ProR
- Distributed development: team work plug in, decomposition methods
- Automatic code generation: to Ada, C, Java, VHDL
Rodin Ecosystem

Rodin is an open platform.

- Anybody may contribute
- Code is written in Java or Scala
- Model editing tools are introspective
- It is easy to learn by examples and plenty of active developers to help
A closer look at

- ProB
- Flow
ProB
Use cases
The meaning of flow POs

The core theorems generated by the tool characterise the relation between the after-state of one event and the guard of another.

In the figure above, two events, \( f \) and \( g \), are characterised by their domains (guards) and ranges (after-states of actions). The right-hand side part is the flow diagram notation.
Enabling (FENA)

The enabling relation states that the first event enables the second event. Hence, when the first event happens it is always true that the second may happen next. Formally, the range of the first event is fully contained in the domain of the second event.

On a diagram this is denoted by a solid arrow connecting two events. Corresponding proof obligations end with \textit{FENA} suffix.
Possible usecase

*at all times, a user located inside may get to outside area*
Experience in Event-B Modelling
Experience in Event-B Modelling

- **RODIN EC project (2004-2007):**  
  http://rodin.cs.ncl.ac.uk

- **DEPLOY EC project (2008-2012):**  
  http://www.deploy-project.eu/  
  (4 Deployment Partners and 3 DEPLOY Associates)

- **Outside and after DEPLOY:**  
• Formal approaches to protocol engineering service-oriented architectures for telecomm (Nokia)
• Engine failure management systems (ATEC)
• Formal techniques within an MDA context (Nokia)
• CCF Display and Information System - CDIS - computer-based system that provides airport and flight information (Praxis)
• Ambient campus (Newcastle and Aabo)
Cruise-control system

Engine start-stop system

A significant progress was made in formal development of industry-size automotive applications. Several traceable approaches were developed and applied to support disciplined and structured mapping of the informal (natural language) requirements into abstract formal models; these were based on Problem Frames and RSML.
Communication-based Train Control (CBTC)

At the time when DEPLOY started Siemens had had considerable experience in using formal methods (but not the Rodin/Event-B technologies). The company is now prepared to move to their wider use provided they are further improved to allow them to be applied in the development of safety critical applications (e.g. SIL4).
Bepi Colombo (ESAs first mission to Mercury)

Attitude and Orbit Control System (AOCS)

- The focus was on formal modelling of behavioural requirements: modelling modal, distributed and reconfigurable systems and on fault tolerance modelling based on linking Event-B with the FMEA analysis. The achieved results meet the initial expectations of Space System Finland and form a solid base for further application of technologies within the company.
Service choreography modelling

BPMN translation to Event-B

A substantial progress was made in understanding how to achieve the right interplay between applying proofs and model-checking, how to ensure consistency among different modelling layers and how to select invariants that a model must preserve during runtime. Achieving highest possible degree of automation played a crucial role in this work.
• Critical Software Technologies
  • Integrated Secondary Flight Display, used on-board commercial and military aircraft

• Grupo AuS
  • dead man control and door control
  • safety critical hardware verification

• XMOS
  • Xcore micro-processor instruction set
Outside and after DEPLOY

- QNX
  - design of software for a simple medical device to support a safety case and to help in the approval process

- Battelle and ClearSy
  - automation of the Flushing and Culver metro lines in New York

- ADVANCE project
  - Dynamic Trusted Railway Interlocking (Alstom)
  - Smart Energy Grids (Critical Software Technologies)

- Defence system modelling and certification in the UK and France
Outside and after DEPLOY

- Systerel
  - modeling of interlocking, communication-based train control (CBTC), CBTC train tracking and security barrier
  - validation of flight software configuration data
- STMicroelectronics modelling and generation of a VHDL code for a smartcard-based microcircuit
- The Dependable Systems Forum (DSF) project involves several Japanese Companies namely NTT-Data, Fujitsu, Hitachi, NEC, Toshiba, and SCSK. The DSF project applied several formal methods including Event-B and Rodin to an industrial development.
More Information

- RODIN public deliverables and papers
- DEPLOY public deliverables and papers
- Event-B site


The End
Thank you!

Send us your comments/feedback:

- Alexei Iliasov alex@iliasov.org
- Elena Troubitsyna Elena.Troubitsyna@abo.fi
- Linas Laibinis Linas.Laibinis@abo.fi
- Alexander Romanovsky alexander.romanovsky@newcastle.ac.uk

Visit:

- http://www.event-b.org/ (community website)
- http://sf.net/projects/rodin-b-sharp/ (platform download)
- http://rodintools.org/ (Rodin Tools company)