

# Verification and tools in Event-B modelling

Mike Poppleton  
[users.ecs.soton.ac.uk/mrp](mailto:users.ecs.soton.ac.uk/mrp)

Slides adapted from Prof. Michael Butler,  
Marktoberdorf Summer School 2012

## Overview

- Abstraction & refinement  
validation & verification
- Proof obligations in Event-B
- Rodin tool features

## Problem Abstraction

- Abstraction can be viewed as a process of **simplifying** our understanding of a system.
- The simplification should
  - **focus** on the **intended purpose** of the system
  - **ignore** details of **how** that purpose is achieved.
- The modeller/analyst should make **judgements** about what they believe to be the **key features** of the system.

1st Oct 2012

SAICSIT: Event-B/2

3

## Abstraction (continued)

- If the purpose is to provide some **service**, then
  - model **what** a system does from the perspective of the service users
  - ‘users’ might be computing agents as well as humans.
- If the purpose is to **control**, **monitor** or **protect** some **phenomenon**, then
  - the abstraction should **focus** on those phenomenon
  - in **what** way should they be controlled, monitored or protected?

1st Oct 2012

SAICSIT: Event-B/2

4

## Refinement

- Refinement is a process of **enriching** or **modifying** a model in order to
  - **augment** the functionality being modelled, **or**
  - **explain** how some purpose is achieved
- Facilitates abstraction: we can **postpone** treatment of some system features **to later** refinement steps
- Event-B provides a notion of **consistency** of a refinement:
  - Use proof to **verify the consistency** of a refinement step
  - **Failing proof** can help us identify **inconsistencies**

1st Oct 2012

SAICSIT: Event-B/2

5

## Validation and verification

- **Requirements validation:**
  - The extent to which (informal) requirements satisfy the needs of the stakeholders
- **Model validation:**
  - The extent to which (formal) model accurately captures the (informal) requirements
- **Model verification:**
  - The extent to which a model correctly maintains invariants or refines another (more abstract) model
    - Measured, e.g., by degree of validity of proof obligations

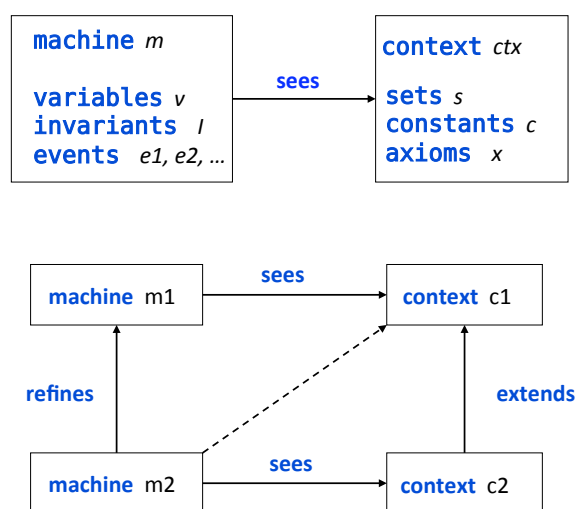
1st Oct 2012

SAICSIT: Event-B/2

6

## Event-B verification and tools

### Event-B modelling components



1st Oct 2012

SAICSIT: Event-B/2

8

## Event structure

```

E =                                     \\ event name
  any
    x1, x2, ...                       \\ event parameters
  where
    G1                                \\ event guards (predicates)
    G2
    ...
  then
    v1 := exp1                       \\ event actions
    v2 := exp2
    ...
  end

```

1st Oct 2012

SAICSIT: Event-B/2

9

## Role of Event Parameters

- Most generally, parameters represent nondeterministically chosen values, e.g.,

```

NonDetInc =
  any d where v+d ≤ MAX then v:=v+d end

```

- Event parameters can also be used to model **input** and **output** values of an event

- Can also have nondeterministic actions:

```

when v < MAX then v :| v < v' ≤ MAX end

```

1st Oct 2012

SAICSIT: Event-B/2

10

- A refined machine has two kinds of events:
  - Refined events that refine some event of the abstract machine
  - New events that refine *skip*
- Verification of event refinement uses
  - *gluing* invariants linking abstract and concrete variables
  - *witnesses* for abstract parameters

11

[illegible]

12

## Proof obligations in Event-B

- **Well-definedness (WD)**
  - e.g, avoid division by zero, partial function application
- **Invariant preservation (INV) \*\*\***
  - each event maintains invariants
- **Guard strengthening (GRD) \*\*\***
  - Refined event only possible when abstract event possible
- **Simulation (SIM) \*\*\***
  - update of abstract variable correctly simulated by update of concrete variable
- **Convergence (VAR)**
  - Ensure convergence of new events using a variant

1st Oct 2012

SAICSIT: Event-B/2

13

## Invariant Preservation

- Assume: variables  $v$  and invariant  $I(v)$
- Deterministic event:
 
$$Ev = \text{when } P(v) \text{ then } v := \text{exp}(v) \text{ end}$$
- To prove  $Ev$  preserves  $I(v)$ :
 

$$\text{INV: } P(v), I(v) \vdash I(\text{exp}(v))$$
- This is a sequent of the form  $\text{Hypotheses} \vdash \text{Goal}$
- The sequent is a **Proof Obligation (PO)** that must be verified

1st Oct 2012

SAICSIT: Event-B/2

14

## Using Event Parameters

- Event has form:

$$Ev = \text{any } x \text{ where } P(x,v) \text{ then } v := \text{exp}(x,v) \text{ end}$$

INV:  $I(v), P(x,v) \vdash I(E(x,v))$

1st Oct 2012

SAICSIT: Event-B/2

15

## Example PO from Rodin

Enter/inv3/INV

☐

$\forall u, r \cdot$   
 $u \in \text{dom}(\text{location}) \wedge$   
 $\text{location}(u) = r$   
 $\Rightarrow$   
 $\text{takeplace}[\{r\}] \subseteq \text{authorised}[\{u\}]$

☐  $u \in \text{USER} \setminus \text{dom}(\text{location})$   
☐  $\text{takeplace}[\{r\}] \subseteq \text{authorised}[\{u\}]$   
☐  $(\text{location}u\{u \mapsto r\})(u0) = r0$   
☐  $u0 \in \text{dom}(\text{location}u\{u \mapsto r\})$   
☐  $\text{takeplace} = \text{ROOM} \times \text{ACTIVITY}$   
☐  $\text{location} \in \text{USER} \rightarrow \text{ROOM}$

Selected Hypotheses

☒ Goal

$\text{takeplace}[\{(\text{location}u\{u \mapsto r\})(u0)\}] \subseteq \text{authorised}[\{u0\}]$

1st Oct 2012

SAICSIT: Event-B/2

16



## How do we know what to prove?

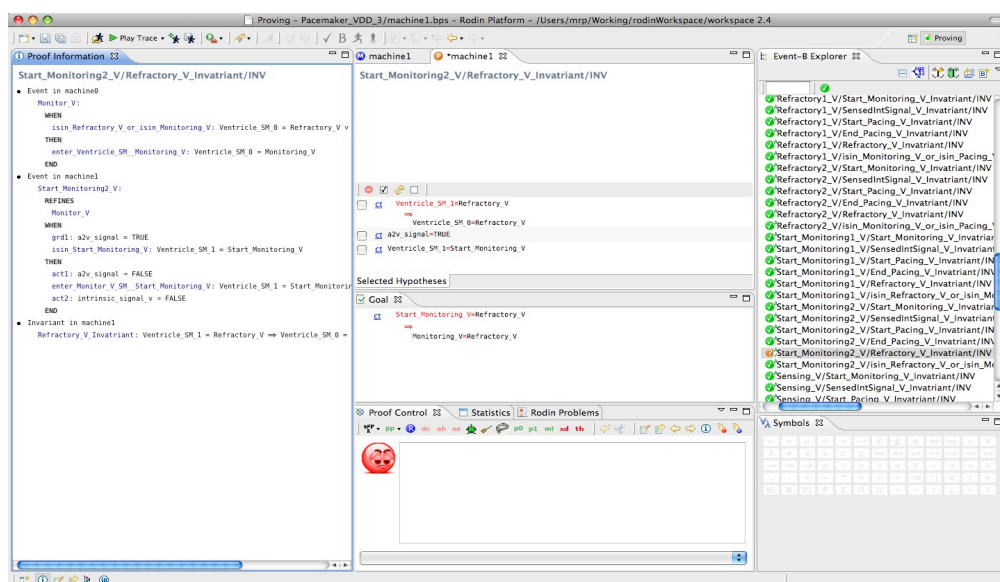
- Need for proofs imposes *proof obligations*
  - the user *does not have to state them*
  - they *are automatically generated* by a *tool*
- Proof obligations serve to
  - *verify properties* of a model

1st Oct 2012

SAICSIT: Event-B/2

17

### Rodin: Proving perspective: proof info, proof & control, PO explorer views:



1st Oct 2012

SAICSIT: Event-B/2

18

## Proof and model checking

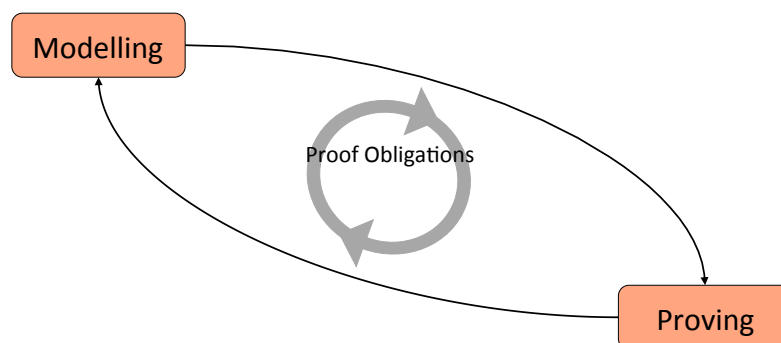
- **Model checking:** force the model to be finite state and explore state space looking for invariant violations
  - completely automatic
  - powerful debugging tool (counter-example)
- **(Semi-)automated proof:** based on logical deduction rules
  - no restrictions on state space
  - leads to discovery of invariants that deepen understanding
  - not completely automatic

1st Oct 2012

SAICSIT: Event-B/2

19

## Models are created and verified iteratively

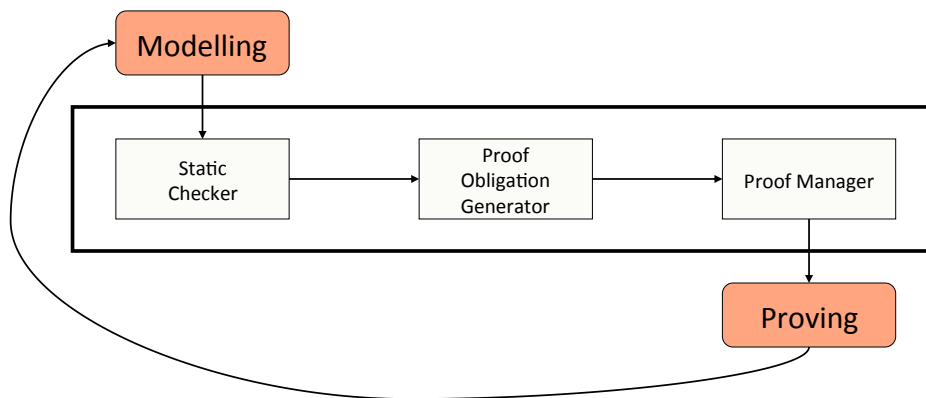


1st Oct 2012

SAICSIT: Event-B/2

20

## Rodin architecture



1st Oct 2012

SAICSIT: Event-B/2

21

## Rodin Architecture

- Extension of Eclipse IDE
- Repository of structured modelling elements
- Rodin Eclipse Builder manages:
  - Well-formedness + type checker
  - Consistency/refinement PO generator
  - Proof manager
  - Propagation of changes
- Extension points to support plug-ins

1st Oct 2012

SAICSIT: Event-B/2

22

## Differential proving in Rodin

- Models are constantly being changed
- When a model changes, proof impact of changes should be minimised as much as possible:
- Sufficiency comparison of POs
  - In case of success, provers return list of *used hypotheses*
  - Proof valid provided the used hypothesis are in the new version of a PO
- Model refactoring:
  - Identifier renaming applied to models (avoiding name clash)
  - Corresponding POs and proofs automatically renamed

1st Oct 2012

SAICSIT: Event-B/2

23

## Rodin Proof Manager (PM)

- PM constructs *proof tree* for each PO
- Automatic and interactive modes
- PM manages *used hypotheses*
- PM calls *reasoners* to
  - *discharge* goal, or
  - *split* goal into *subgoals*
- Collection of reasoners:
  - *simplifier, rule-based, decision procedures, ...*
- Basic *tactic language* to define PM and reasoners

1st Oct 2012

SAICSIT: Event-B/2

24

## Statistics from Flash-based file development in Event-B

Machines	Total POs	Automatic	Interactive
MCH0	35	22	13
MCH1	57	49	8
MCH2	33	32	1
MCH3	37	34	3
MCH4	26	26	0
MCH5	27	26	1
MCH6	31	30	1
MCH7	109	97	12
MCH_FL0	8	8	0
MCH_FL1	110	110	0
MCH_FL2	57	57	0
MCH_FL3	9	9	0
Overall	540	501 (93%)	39 (7%)

1st Oct 2012

SAICSIT: Event-B/2

25

## Range of Automated Provers

- **Built-in:** tactic language, simplifiers, decision procedures
- **AtelierB plug-in** for Rodin (ClearSy, FR)
- **SMT plug-in** (Systeme, FR)
- **Isabelle plug-in** (Schmalz, ETHZ)

1st Oct 2012

SAICSIT: Event-B/2

26

## Validation/verification offered by ProB

- Animation: show behaviour of model in clear terms
- Model Checking
- Refinement Checking
- Graphical Domain Specific Visualization
- Visualization of State Space



1st Oct 2012

SAICSIT: Event-B/2

27

## ProB

- Animator and model checker
  - search for **invariant violations**
  - search for **deadlocks**
  - search for **proof obligation violations**
- Implementation uses constraint logic programming
  - makes all types **finite**
  - exploits **symmetries** in B types

1st Oct 2012

SAICSIT: Event-B/2

28

## Rodin: ProB perspective: model text, ProB views:

The screenshot displays the Rodin ProB perspective with the following components:

- Events View:** Lists events such as Refractory1\_A, Refractory2\_A, Start\_Monitoring\_A, Sensing\_A(x2), SenseIntSignal\_A, Refractory1\_V, Refractory2\_V, Start\_Monitoring1\_V, Start\_Monitoring2\_V, Sensing\_V(x2), SenseIntSignal\_V, and StartPacing\_V.
- Event-B Editor:** Shows the Rodin ProB model text, including a `where` block for `P1sin_SensedIntSignal_A` and an `event Refractory2A refines Refractory_A` block.
- State View:** Displays the state of the model, including variables like `monitoring_implicitContext`, `monitoring_A`, `monitoring_V`, `pacing_V`, `refractory_A`, `refractory_V`, `endPacing_V`, `sensedIntSignal_A`, `sensedIntSignal_V`, `startMonitoring_A`, `startMonitoring_V`, `startPacing_V`, `atrium_SM_0`, `atrium_SM_1`, `ventricle_SM_0`, `ventricle_SM_1`, `atrium_SM_1`, `ventricle_SM_1`, `atrium_signal`, `intrinsic_signal_a`, `intrinsic_signal_v`, `pace_signal_v`, and `v2e_signal`.
- Properties View:** Shows the properties of the selected event, including `id`, `bounds`, `x`, `y`, `width`, `height`, and `image`.

1st Oct 2012 SAICSIT: Event-B/2 29

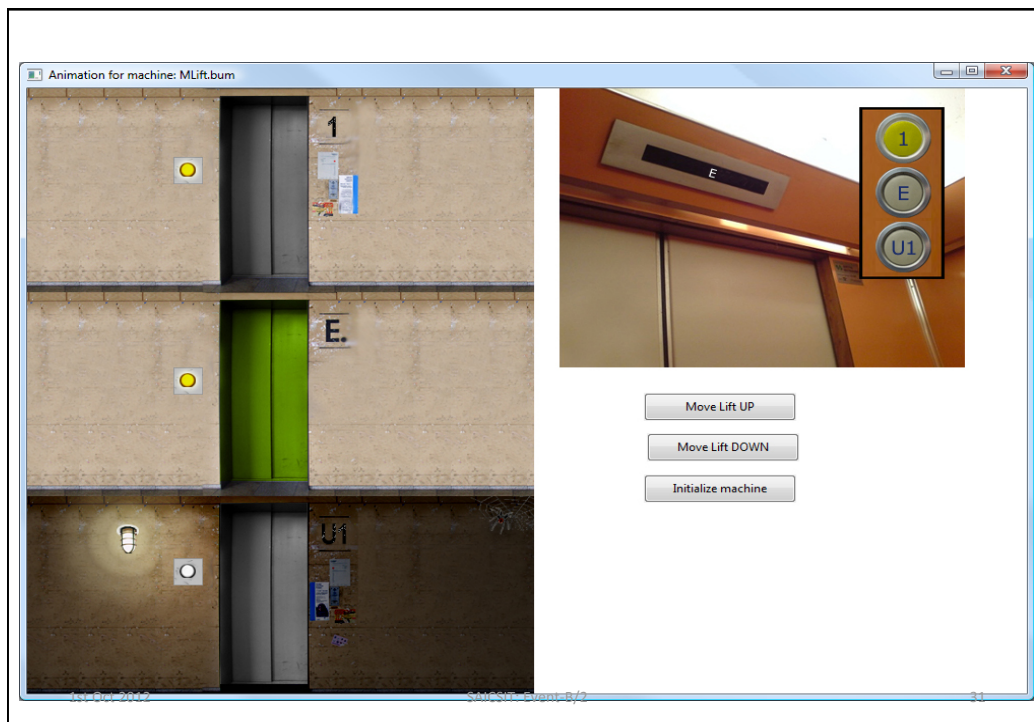
## BMotionStudio



The screenshot displays the BMotionStudio interface with the following components:

- Tool bar:** Contains icons for various tools and actions.
- WYSIWYG editor:** Shows a 3D model of a control panel with buttons and a screen.
- Outline View:** Lists the objects in the scene, including `background`, `background2`, `background3`, `background4`, `background5`, `background6`, `background7`, `background8`, `background9`, `background10`, `background11`, `background12`, `background13`, `background14`, `background15`, `background16`, `background17`, `background18`, `background19`, `background20`, `background21`, `background22`, `background23`, `background24`, `background25`, `background26`, `background27`, `background28`, `background29`, `background30`, `background31`, `background32`, `background33`, `background34`, `background35`, `background36`, `background37`, `background38`, `background39`, `background40`, `background41`, `background42`, `background43`, `background44`, `background45`, `background46`, `background47`, `background48`, `background49`, `background50`, `background51`, `background52`, `background53`, `background54`, `background55`, `background56`, `background57`, `background58`, `background59`, `background60`, `background61`, `background62`, `background63`, `background64`, `background65`, `background66`, `background67`, `background68`, `background69`, `background70`, `background71`, `background72`, `background73`, `background74`, `background75`, `background76`, `background77`, `background78`, `background79`, `background80`, `background81`, `background82`, `background83`, `background84`, `background85`, `background86`, `background87`, `background88`, `background89`, `background90`, `background91`, `background92`, `background93`, `background94`, `background95`, `background96`, `background97`, `background98`, `background99`, `background100`.
- Properties View:** Shows the properties of the selected object, including `id`, `bounds`, `x`, `y`, `width`, `height`, and `image`.

1st Oct 2012 Properties View/2 [Ladenberger et al., FMICS'09]



## Proof and model checking

- **Model checking:** force the model to be finite state and explore state space looking for invariant violations
  - ☺ completely automatic
  - ☺ powerful debugging tool (counter-examples)
  - ☹ state-space explosion
- **(Semi-)automated proof:** based on deduction rules
  - ☹ not completely automatic
  - ☺ leads to discovery of invariants - deepen understanding
  - ☺ no restrictions on state space



## Some references

- Full introduction to modelling and verification in Event-B, to advanced level (including definition of proof obligations):
  - Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press 2010
- Abrial, Butler, Hallerstede, Hoang, Mehta and Voisin
  - *Rodin: An Open Toolset for Modelling and Reasoning in Event-B*.
  - International Journal on Software Tools for Technology Transfer (STTT), 12 (6), 2010.
- Leuschel and Butler
  - *ProB: An Automated Analysis Toolset for the B Method*. *International Journal on Software Tools for Technology Transfer*, 10, (2), 185-203, 2008.

1st Oct 2012

SAICSIT: Event-B/2

33

## Rodin and its plug-ins: read about and install via [www.event-b.org](http://www.event-b.org)

- ProB model checker:
  - consistency and refinement checking
- External provers:
  - AtelierB plug-in for Rodin (ClearSy, FR)
  - SMT plug-in (Systerel, FR)
  - Isabelle plug-in (Schmalz, ETHZ)
- Theory plug-in – user-defined mathematical theories
- UML-B: Linking UML and Event-B
- Graphical model animation
  - ProB, AnimB, B-Motion Studio
- Requirements management (ProR)
- Team-based development
- Decomposition
- Code generation
- ...

1st Oct 2012

SAICSIT: Event-B/2

34

END