Verification and tools in Event-B modelling

Mike Poppleton
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.

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?

Refinement

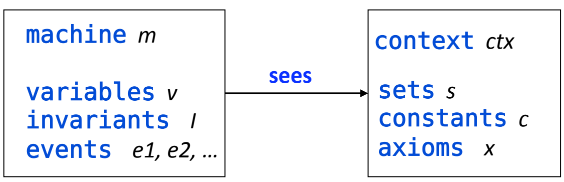
- Refinement is a process of enriching or modifying a model in order to
  - augment the functionality being modelled, or
  - explain how some purpose is achieved
- Facilitates abstraction: we can postpone treatment of some system features to later refinement steps
- Event-B provides a notion of consistency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

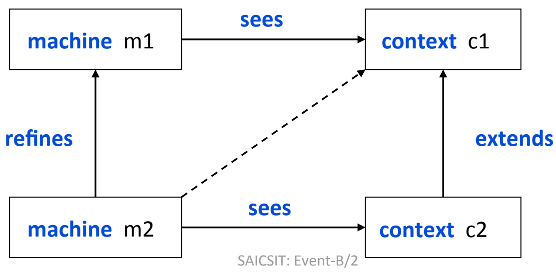
Validation and verification

- 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

Event-B verification and tools

Event-B modelling components





Event structure

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


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

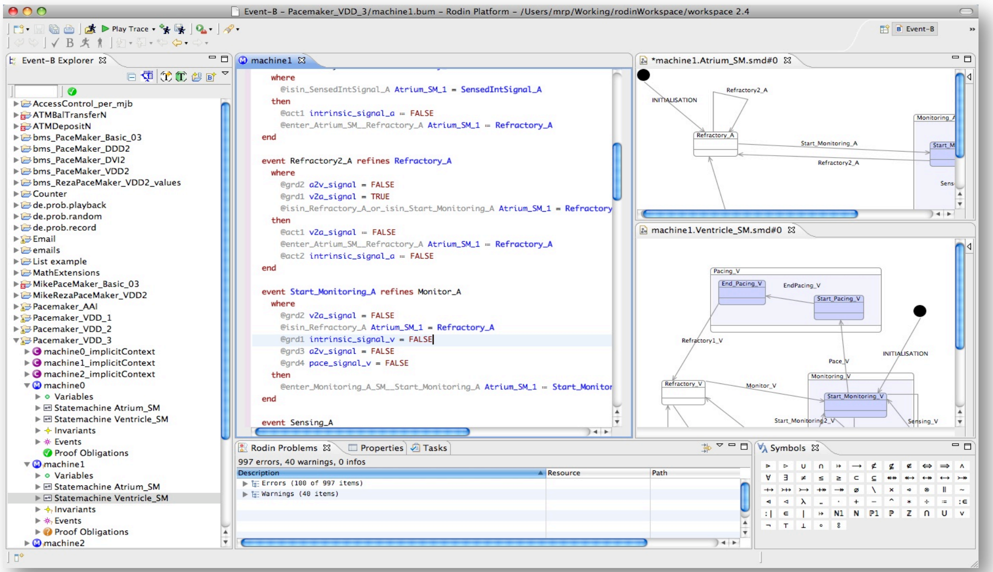
- Can also have nondeterministic actions:

  when v<MAX **then** v :| v < v’ ≤ MAX **end**

Refinement for events

- 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

Rodin: Event-B perspective: model text, explorer, statemachine views:



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

Invariant Preservation

- Assume: variables v and invariant I(v)
- Deterministic event:

  Ev = **when** P(v) **then** v := exp(v) **end**
- To prove Ev preserves I(v):

INV:	P(v), I(v) ⊢ I(exp(v))

- This is a sequent of the form Hypotheses ⊢ Goal
- The sequent is a Proof Obligation (PO) that must be verified

Using Event Parameters

- Event has form:

  Ev = **any** x **where** P(x,v) **then** v := exp(x,v) **end**

INV: I(v), P(x,v) ⊢ I(E(x,v))

Example PO from Rodin

Enter/inv3/INV

```
∀ u, r ·
    u∈dom(location) ∧
    location(u)=r
  ⇒
    takeplace[{r}]⊆authorised[{u}]
u∈USER \ dom(location)
takeplace[{r}]⊆authorised[{u}]
(location∪{u ↦ r})(u0)=r0
u0∈dom(location∪{u ↦ r})
takeplace=ROOM × ACTIVITY
location∈USER ⇸ ROOM
```

Selected Hypotheses

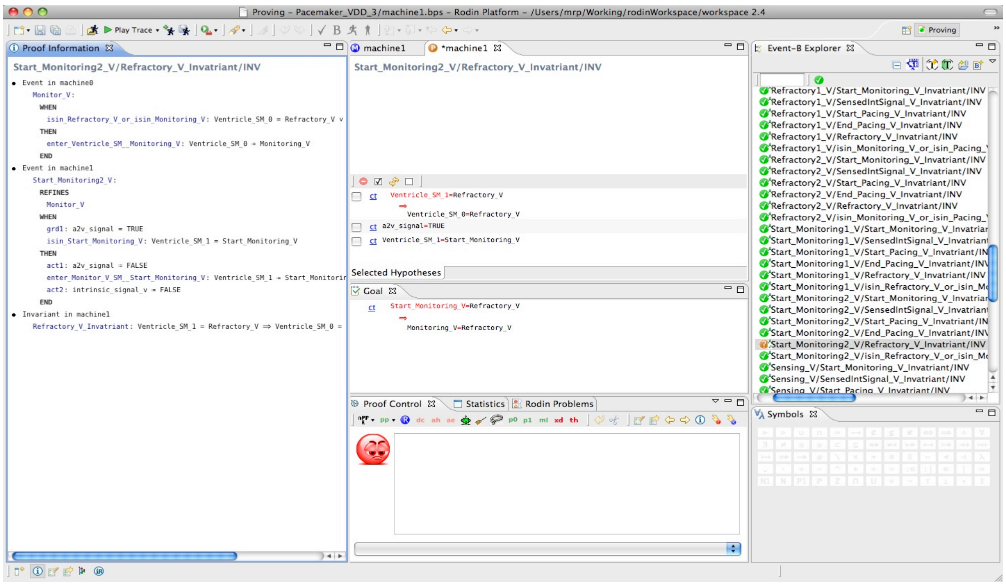
Goal

```
takeplace[{(location∪{u ↦ r})(u0)}]⊆authorised[{u0}]
```

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

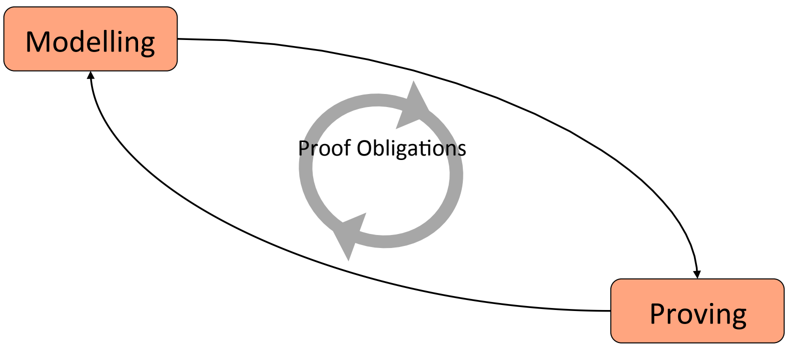
Rodin: Proving perspective: proof info, proof & control, PO explorer views:



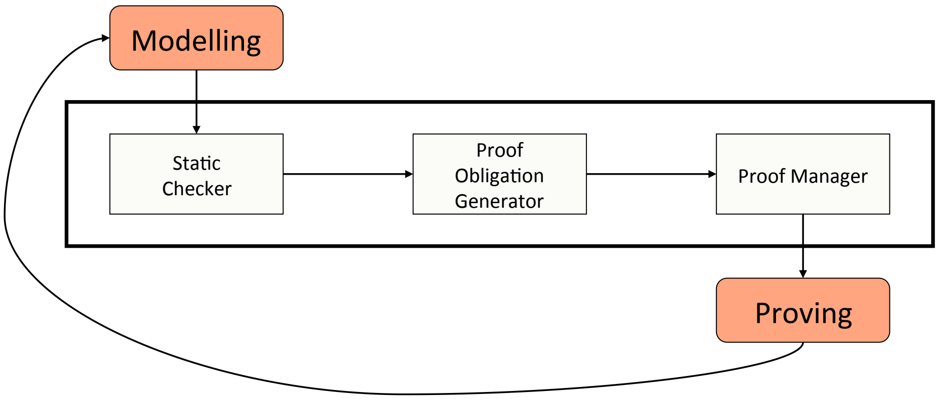
Proof and model checking

- 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

Models are created and verified iteratively



Rodin architecture



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

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

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

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%)

Range of Automated Provers

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

Validation/verification offered by ProB

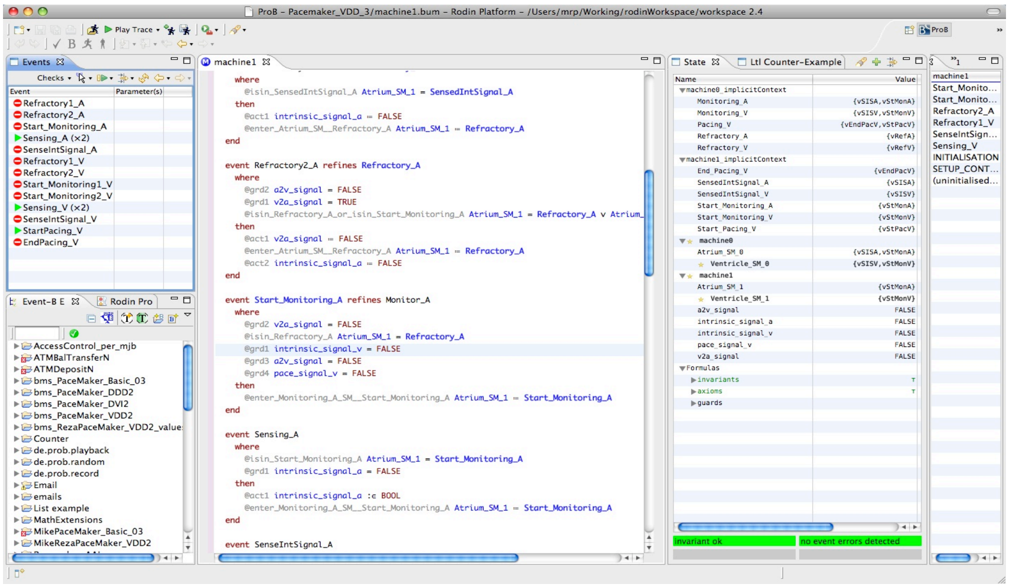
- Animation: show behaviour of model in clear terms
- Model Checking
- Refinement Checking
- Graphical Domain Specific Visualization
- Visualization of State Space

ProB

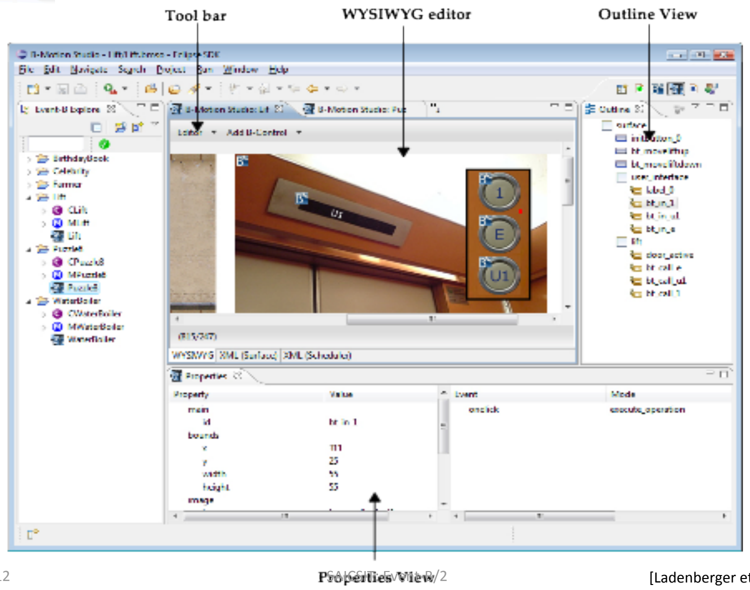
- Animator and model checker
  - search for invariant violations
  - search for deadlocks
  - search for proof obligation violations

- Implementation uses constraint logic programming
  - makes all types finite
  - exploits symmetries in B types

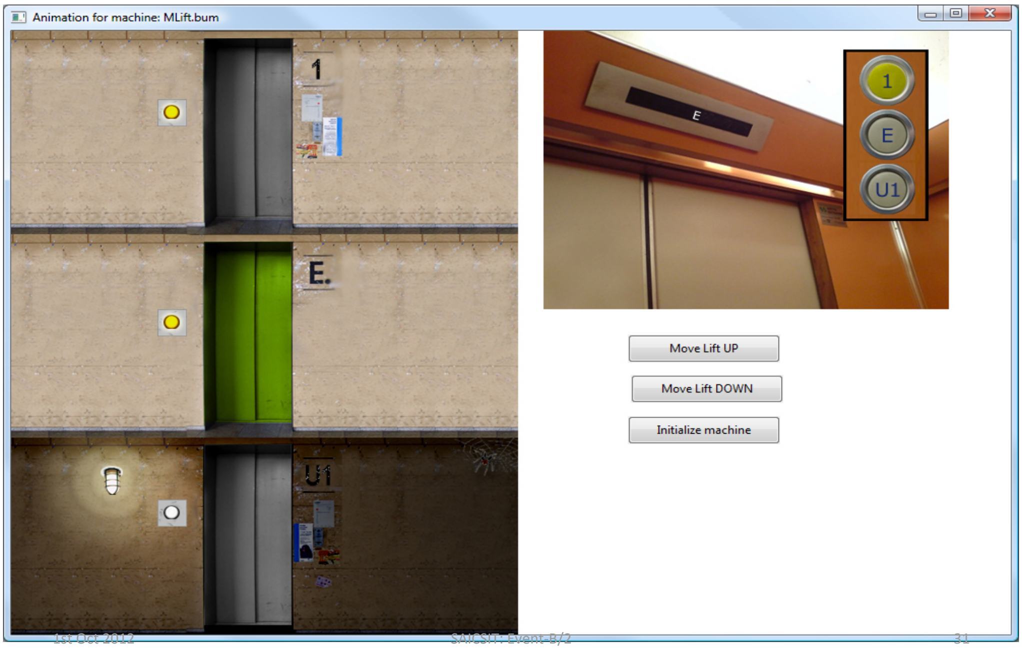
Rodin: ProB perspective: model text, ProB views:



BMotionStudio



[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.

Rodin and its plug-ins:
read about and install via 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
- …

END