

An Introduction to The Formal Development and Verification of Software with Event-B/ RODIN

Mike Poppleton

users.ecs.soton.ac.uk/mrp

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

Session 1: Problem Abstraction and Model Refinement - An Overview

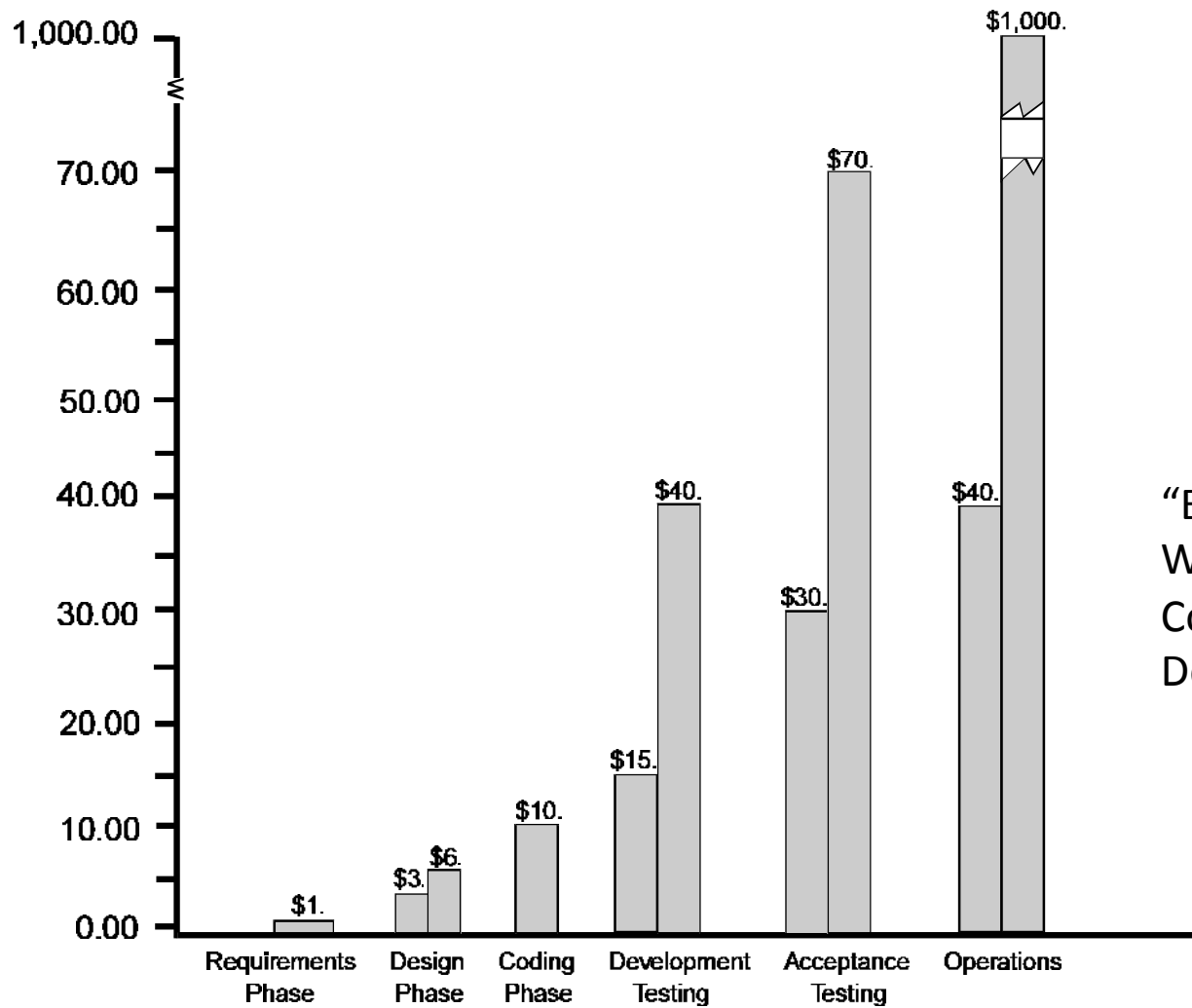
This afternoon:

- Session 2: Verification and tools in Event-B modelling
- Session 3: Case study: the cardiac pacemaker

Overview

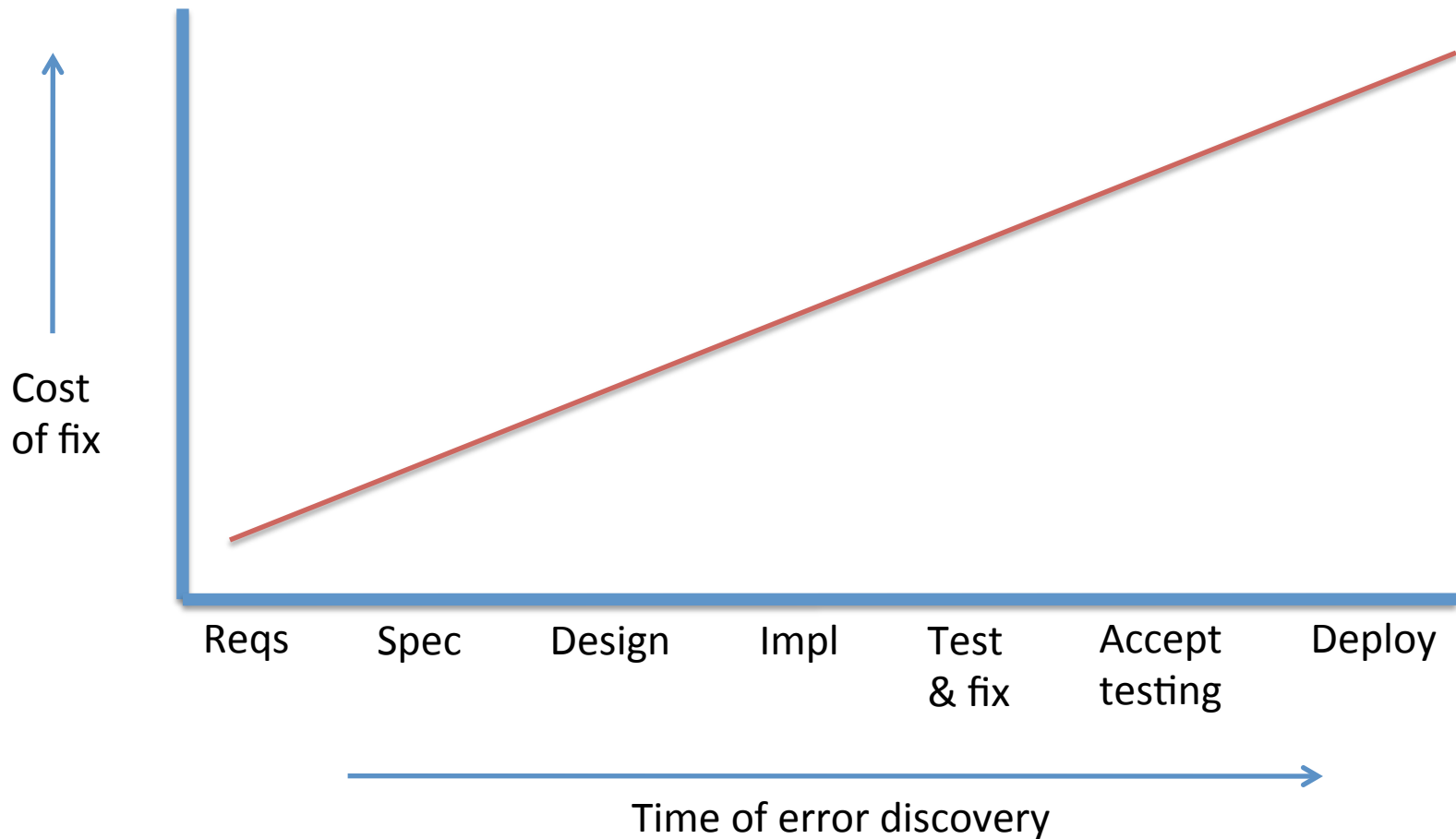
- Motivation
 - difficulty of discovering errors / cost of fixing errors
- Small pedagogical example (access control)
 - abstraction
 - refinement
 - automated analysis
- Background on Event-B formal method
- Methodological considerations

Cost of fixing requirements errors

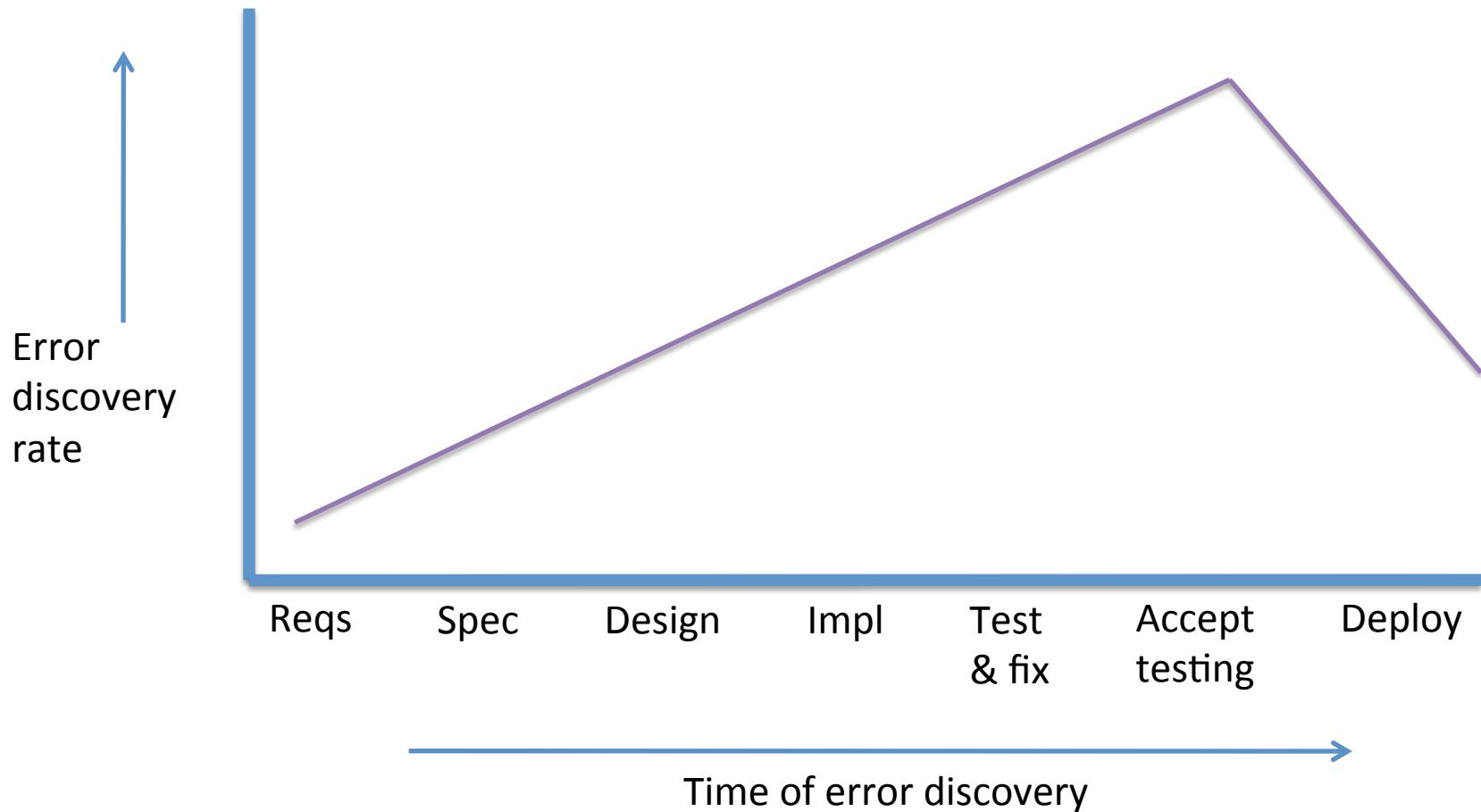


“Extra Time Saves Money”
Warren Kuffel
Computer Language
December 1990

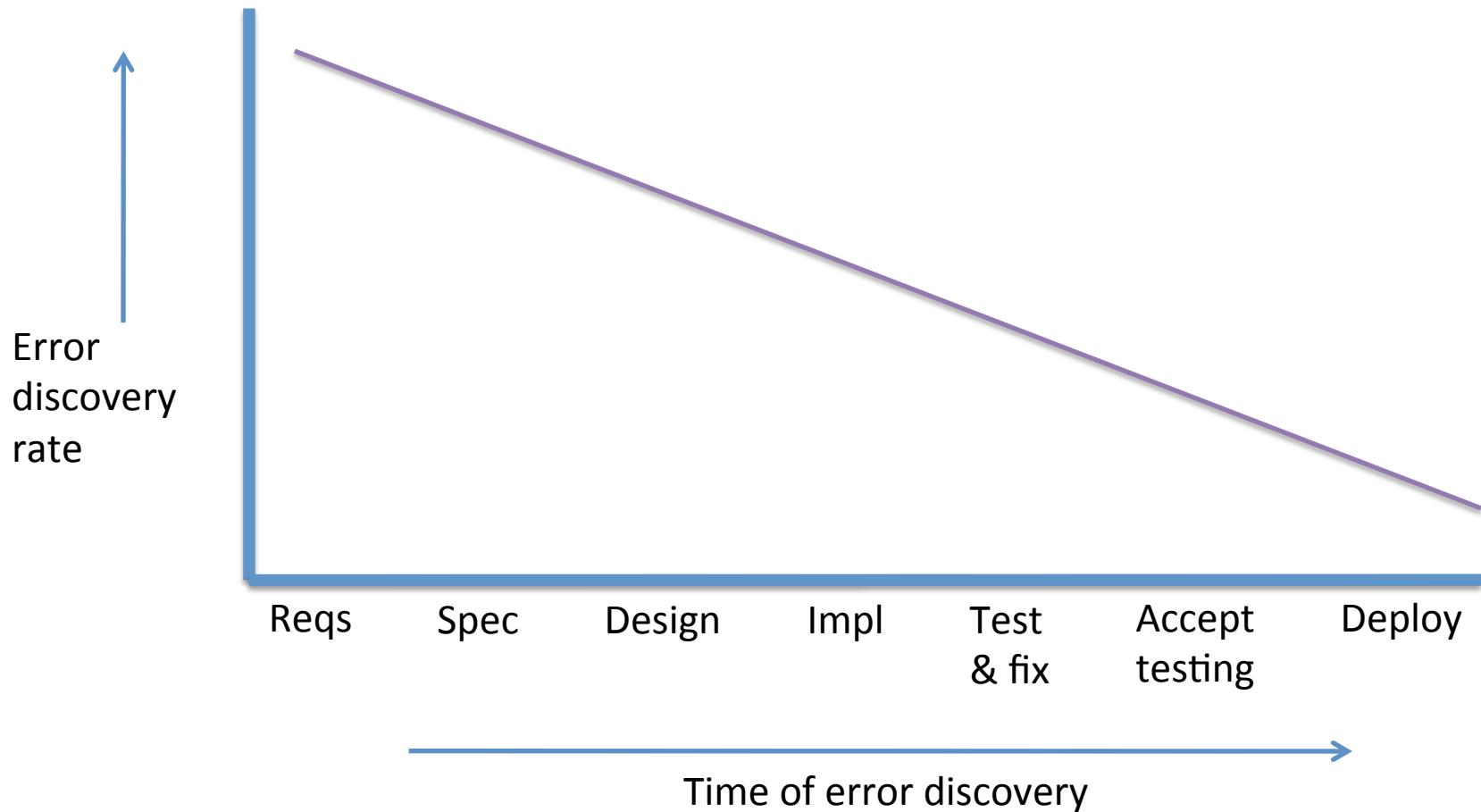
Cost of error fixes grows - difficult to change this



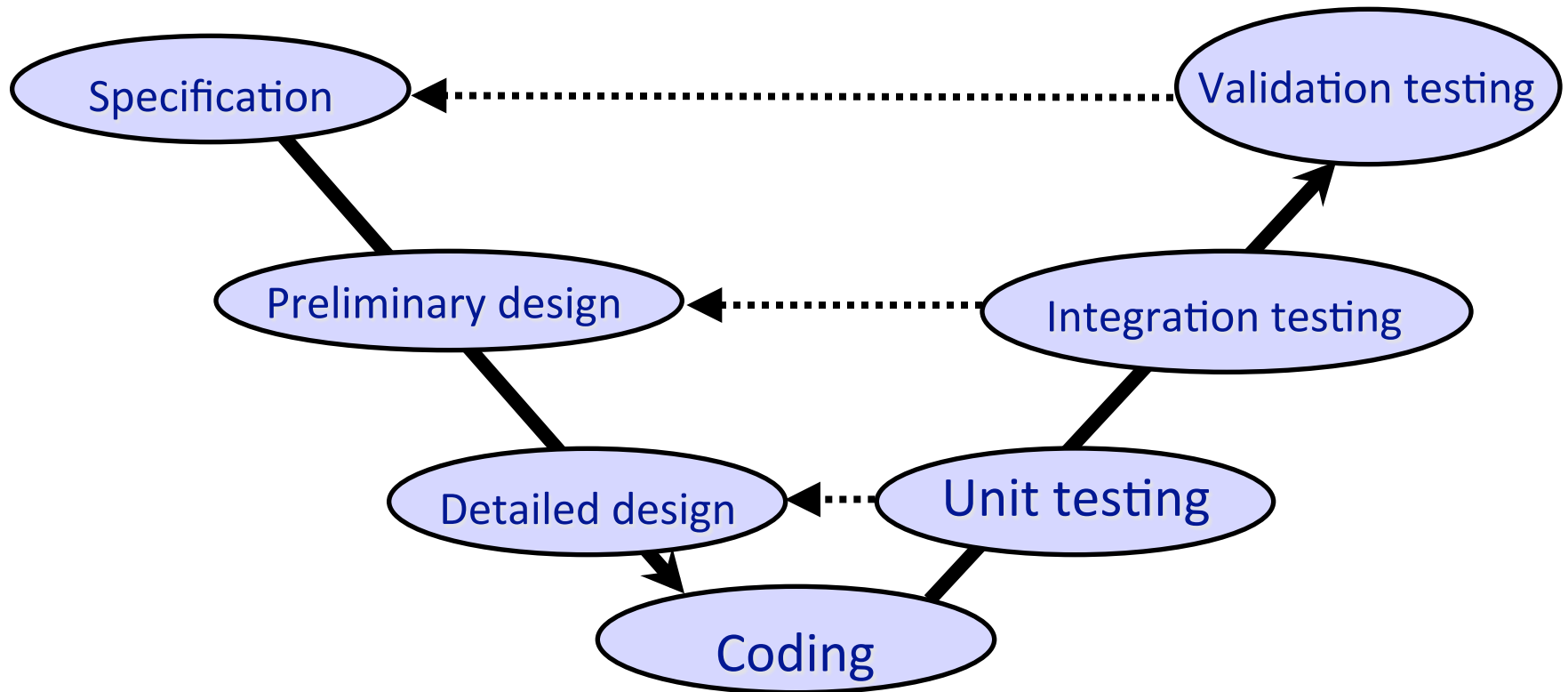
Rate of error discovery



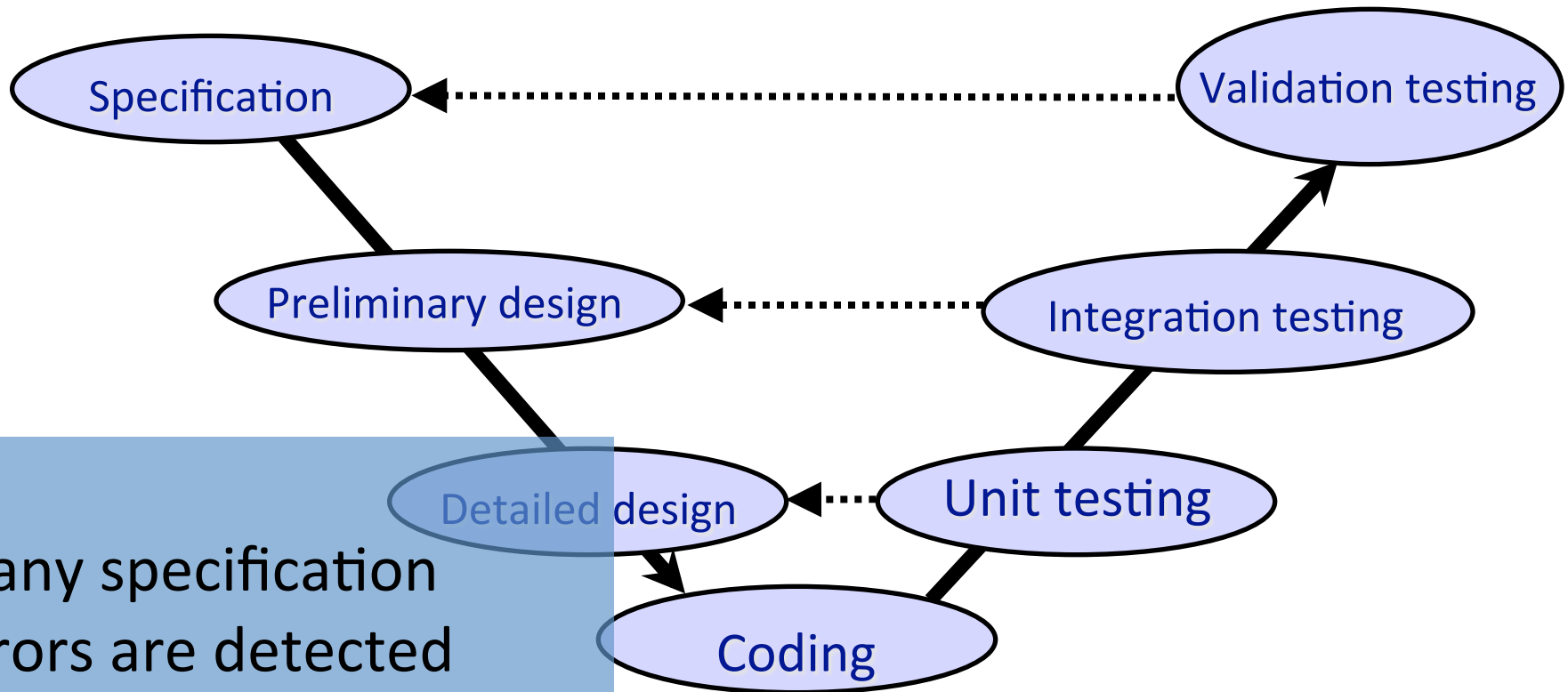
Invert error identification rate?



So, what's wrong with the V model?



So, what's wrong with the V model?



Many specification errors are detected only after a lot of development has been undertaken

Why is it difficult to identify errors?

- Lack of precision
 - ambiguities
 - inconsistencies
- Too much complexity
 - complexity of requirements
 - complexity of operating environment
 - complexity of designs

Need for precise models/blueprints

- Early stage analysis
 - Precise descriptions of intent
 - Amenable to analysis by tools
 - Identify and fix ambiguities and inconsistencies as early as possible
- Mastering complexity
 - Encourage abstraction
 - Focus on *what* a system does
 - Early focus on *key / critical* features
 - Incremental analysis and design: separation of concerns

Correctness-by-construction using Formal Methods

- Mathematical techniques for formulation and analysis of systems
- Formal methods facilitate:
 - Clear specifications (contract)
 - Rigorous *validation* and *verification*

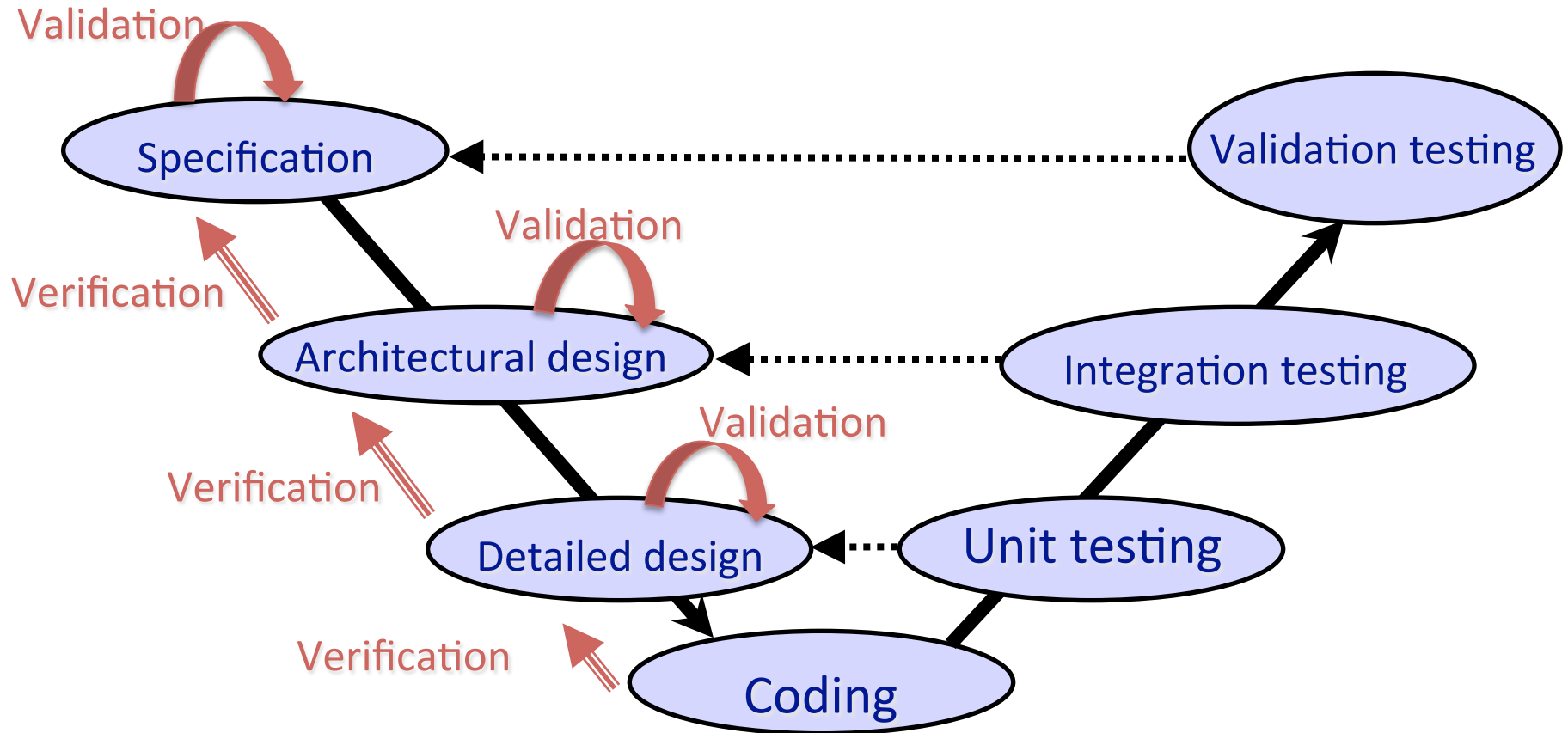
Validation: does the contract specify the right system?

- answered through judgement

Verification: does the finished product satisfy the contract?

- can be answered formally

Early stage analysis



Rapid prototyping *versus* modelling

- **Rapid prototyping:** provides early stage feedback on system functionality
 - Plays an important role in getting **user feedback**
 - and in understanding some design constraints
 - But we will see that formal modelling and proof provide a **deep understanding** that is hard to achieve with rapid prototyping
- **Advice: use any approach that improves design process!**

Rational design, by example

- Example: access control system
- Example intended to give a feeling for:
 - problem abstraction
 - modelling language
 - model refinement
 - role of verification and Rodin tool

Important distinction

- **Program** Abstraction:
 - **Automated** process based on a **formal** artifact (program)
 - Purpose is to reduce complexity of automated verification
- **Problem** Abstraction:
 - **Creative** process based on **informal** requirements
 - Purpose is to increase understanding of problem

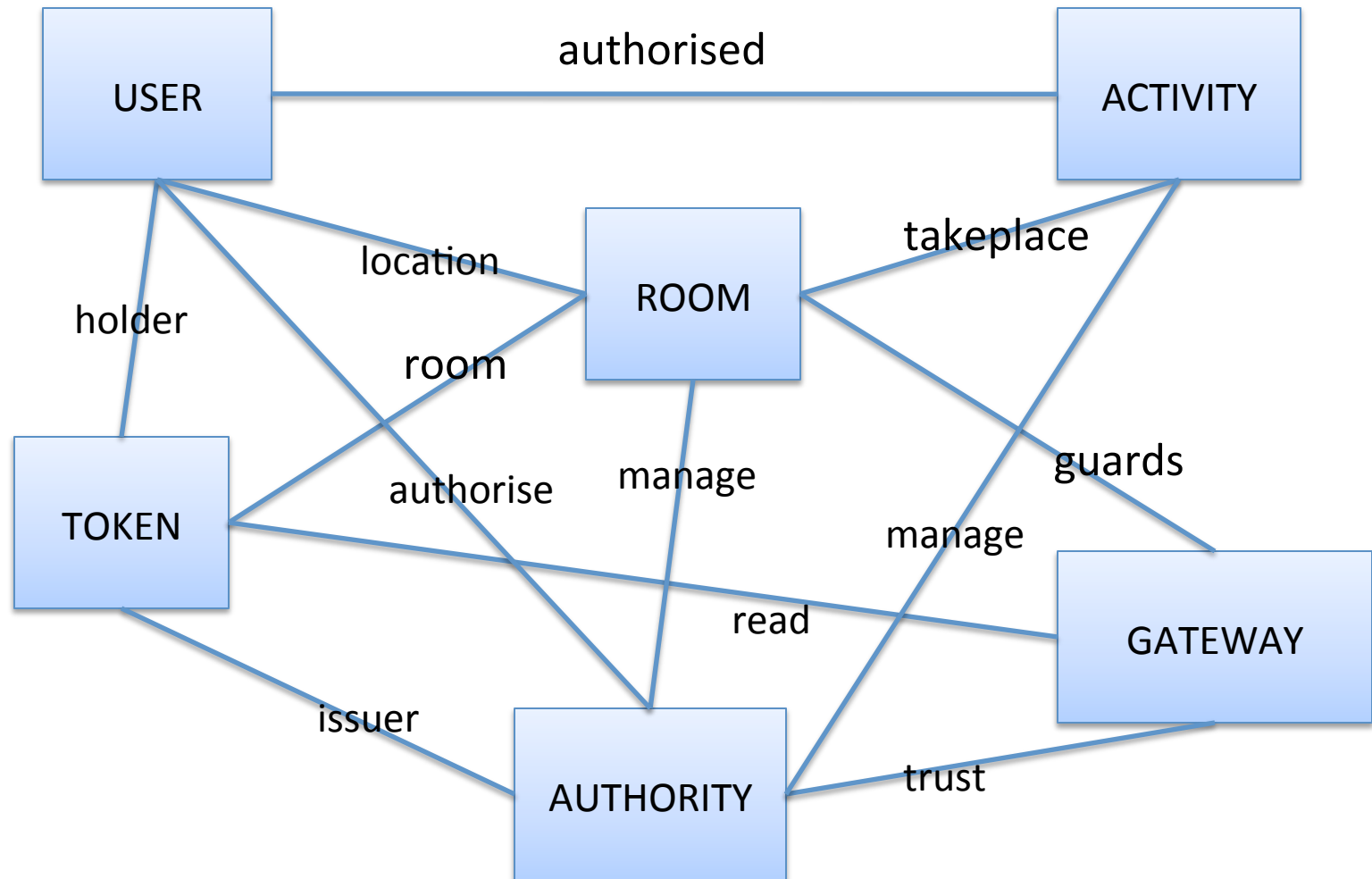
Access control requirements

1. Users are authorised to engage in activities
2. User authorisation may be added or revoked
3. Activities take place in rooms
4. Users gain access to a room using a one-time token provided they have authority to engage in the room activities
5. Tokens are issued by a central authority
6. Tokens are time stamped
7. A room gateway allows access with a token provided the token is valid

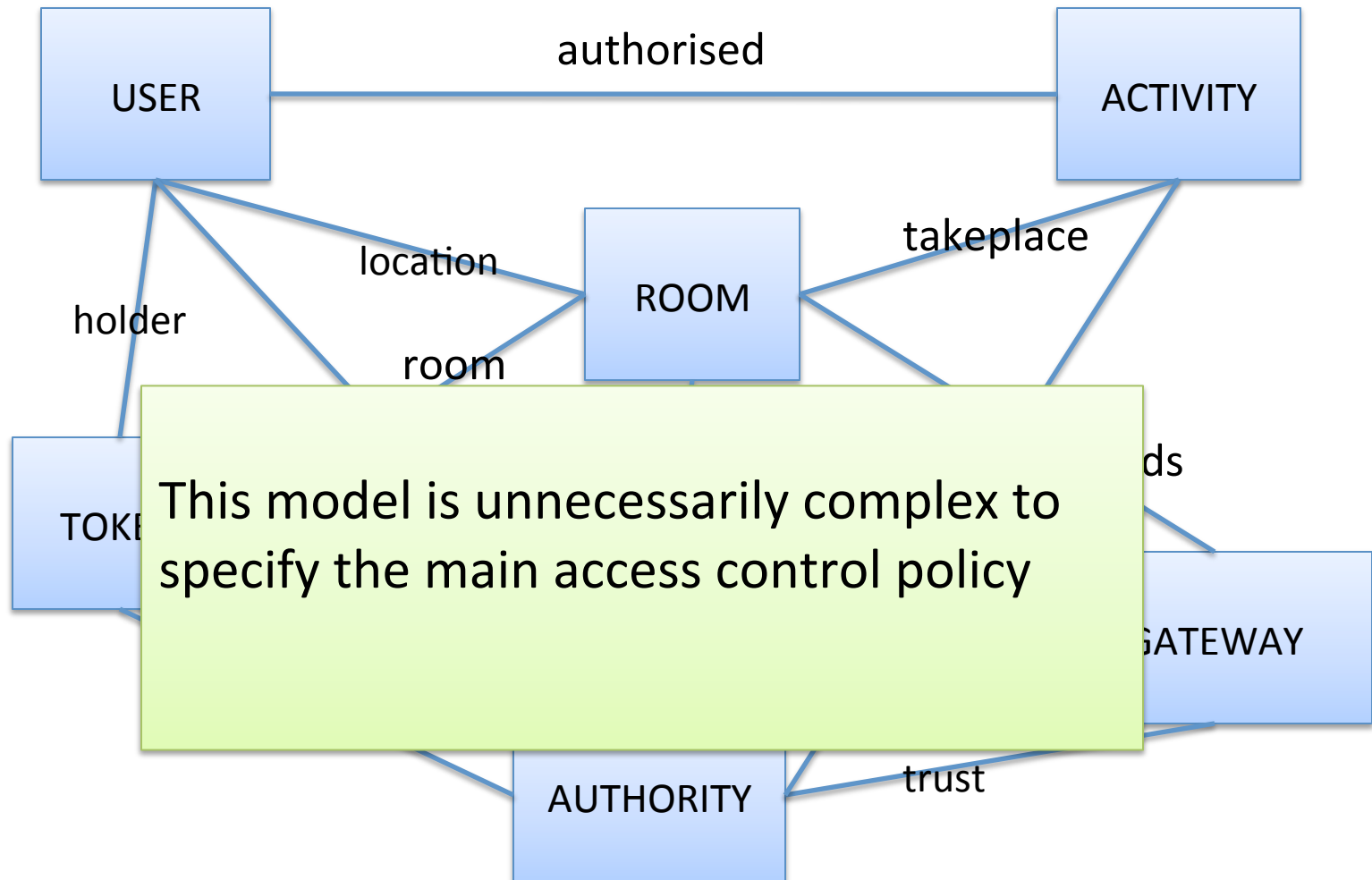
Access control requirements

1. Users are authorised to engage in activities
2. User authorisation may be added or revoked
3. Activities take place in rooms
4. Users gain access to a room using a one-time token provided they have authority to engage in the room activities
5. Tokens are issued by a central authority
6. Tokens are time stamped
7. A room gateway allows access with a token provided the token is valid

Entities and relationships



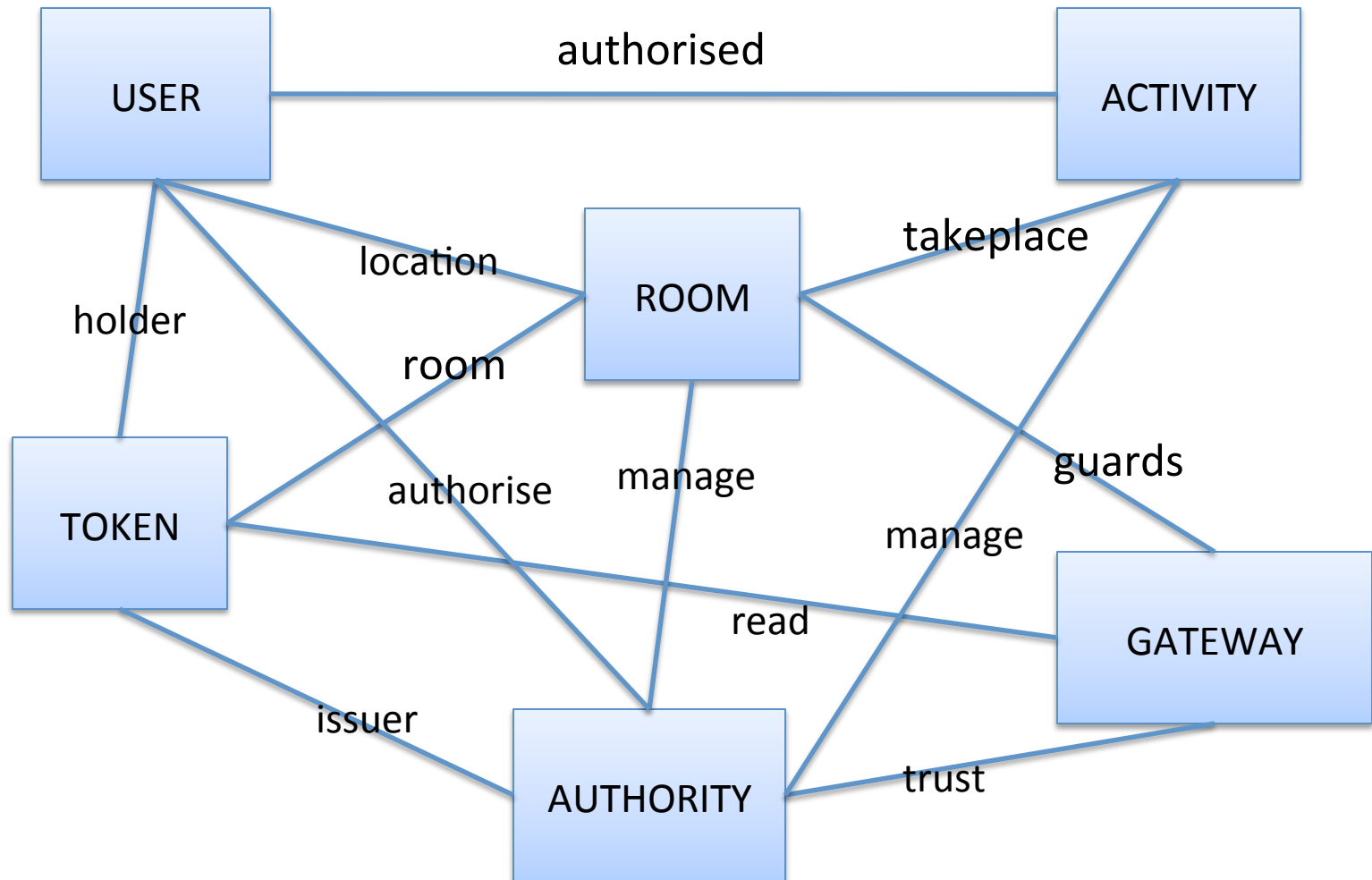
Entities and relationships



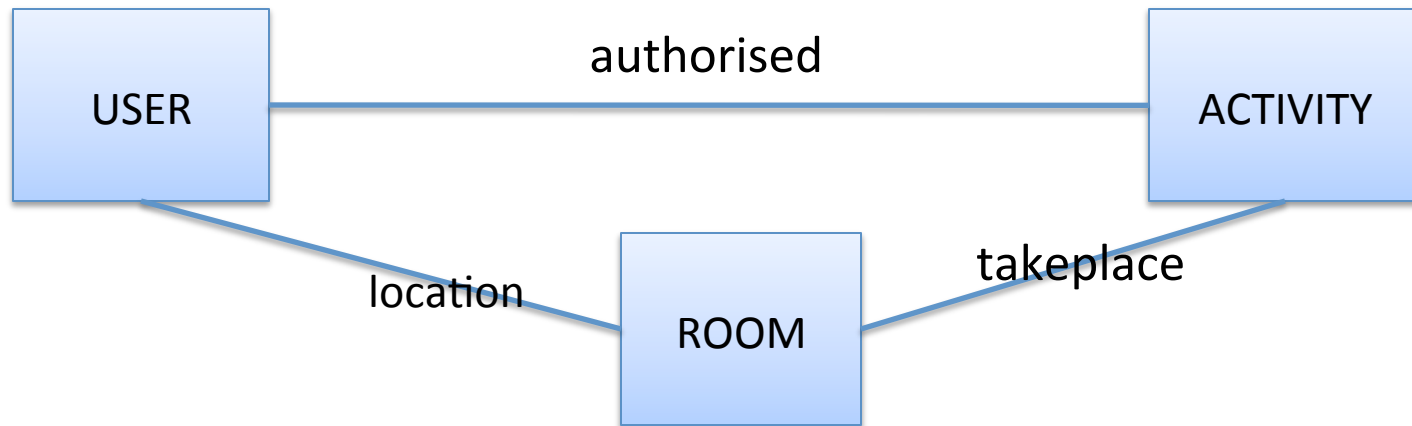
Extracting the essence

- **Purpose** of our system is to enforce an access control policy
- **Access Control Policy**: *Users may only be in a room if they are authorised to engage in all activities that may take place in that room*
- To express this we only require **Users**, **Rooms**, **Activities** and **relationships** between them
- **Abstraction**: focus on key entities in the problem domain related to the purpose of the system

Entities and relationships



Abstract by removing entities



Relationships represented in Event-B

$\text{authorised} \in \text{USER} \leftrightarrow \text{ACTIVITY}$ // relation

$\text{takeplace} \in \text{ROOM} \leftrightarrow \text{ACTIVITY}$ // relation

$\text{location} \in \text{USER} \rightarrow \text{ROOM}$ // partial

function

Access control invariant

$$\begin{aligned} \forall u, r . \quad & u \in \text{dom}(\text{location}) \quad \wedge \\ & \text{location}(u) = r \\ & \Rightarrow \\ & \text{takeplace}[r] \subseteq \text{authorised}[u] \end{aligned}$$

if user u is in room r ,
then u must be authorised to engaged in
all activities that can take place in r

State snapshot as tables

USER	ACTIVITY
u1	a1
u1	a2
u2	a2

authorised

ROOM	ACTIVITY
r1	a1
r1	a2
r2	a1
r2	a2

takeplace

USER	ROOM
u1	r2
u2	r1
u3	

location

Event for entering a room

Enter(u,r) \triangleq

when

grd1 : $u \in \text{USER}$

grd2 : $r \in \text{ROOM}$

grd3 : $\text{takeplace}[r] \subseteq \text{authorised}[u]$

then

act1 : $\text{location}(u) := r$

end

Does this event maintain the access control invariant?

Role of invariants and guards

- **Invariants**: specify properties of model variables that should *always* remain true
 - violation of invariant is undesirable (*safety*)
 - use (automated) proof to verify invariant preservation
- **Guards**: specify *enabling conditions* under which events may occur
 - should be strong enough to ensure invariants are maintained by event actions
 - but not so strong that they prevent desirable behaviour (*liveness*)

Remove authorisation

RemoveAuth(u,a) $\hat{=}$

when

grd1 : $u \in \text{USER}$

grd2 : $a \in \text{ACTIVITY}$

grd3 : $u \mapsto a \in \text{authorised}$

then

act1 : $\text{authorised} := \text{authorised} \setminus \{ u \mapsto a \}$

end

Does this event maintain the access control invariant?

Counter-example from model checking

The screenshot displays the ProB model checker interface. The main window is titled 'State' and shows a table with two columns: 'Name' and 'Value'. The table lists the state of a model M1, including variables 'authorised', 'location', and 'takeplace' with their corresponding values. The 'authorised' variable is highlighted in red, indicating a violation of an invariant. The 'location' variable is set to '{(User1|->Room2)}' and the 'takeplace' variable is set to '{(Room1|->Activity1),(Room1|->Activity2),(Room2|->Activity1),(Room2|->Activity2)}'.

Name	Value
▼ M1	
authorised	{{User1 ->Activity1),(User2 ->Activity2)}
location	{{User1 ->Room2}}
takeplace	{{Room1 ->Activity1),(Room1 ->Activity2),(Room2 ->Activity1),(Room2 ->Activity2)}

The right-hand pane, titled 'History', shows a sequence of operations performed during the model checking process, including 'RemAuth(Activity2,User1)', 'Enter(Room2,User1)', 'AddAuth(Activity2,User2)', 'AddAuth(Activity2,User1)', 'AddAuth(Activity1,User1)', '\$initialise_machine({},{},{z}', '\$setup_constants()', and '(root)'. The 'RemAuth' operation is highlighted in red.

invariant violated!

State

Name	
M1	
authorised	
location	
takeplace	{{Room

invariant violated!

History

Operations

RemAuth(Activity2,User1)

Enter(Room2,User1)

AddAuth(Activity2,User2)

AddAuth(Activity2,User1)

AddAuth(Activity1,User1)

\$initialise_machine({},{}},{z

\$setup_constants()

(root)

oB Proving Event-B

History

Operations

RemAuth(Activity2,User1)

Enter(Room2,User1)

AddAuth(Activity2,User2)

AddAuth(Activity2,User1)

AddAuth(Activity1,User1)

\$initialise_machine({},{}},{z

\$setup_constants()

(root)

Failing proof

Event-B - Rooms1/M1.bum - Rodin Platform - /Users/mjb/Rodin/workspace1.0

Resource ProB Proving Event-B

Event-B Explorer

- Event-B Explorer
 - Variables
 - Invariants
 - Events
 - Proof Obligations
 - INITIALISATION/inv1/INV
 - INITIALISATION/inv2/INV
 - INITIALISATION/inv3/INV
 - INITIALISATION/inv4/INV
 - AddAuth/inv1/INV
 - AddAuth/inv4/INV
 - Enter/inv3/INV
 - Enter/inv4/INV
 - Leave/inv3/INV
 - Leave/inv4/INV
 - RemAuth/inv1/INV
 - RemAuth/inv4/INV

*M1

```
event Enter
  any u r
  where
    @grd1 u ∈ User\dom(location)
    @grd2 r ∈ Room
    @grd3 takeplace[{r}] ⊆ authorised[{u}]
  then
    @act1 location := location ∪ { u ↦ r }
  end

event Leave
  any u r
  where
    @grd1 u ↦ r ∈ location
  then
    @act1 location := location \ { u ↦ r }
  end

event RemAuth
  any u a
  where
    @grd3 u ↦ a ∈ authorised
  then
    @act1 authorised := authorised \ { u ↦ a }
  end

end
```

SAICSIT: Event-B/1

31

Strengthen guard of *RemAuth*

The screenshot displays the Rodin Platform interface for the Event-B model 'Rooms1/M1.bum'. The left pane, 'Event-B Explorer', shows a tree of proof obligations, all of which are marked as solved (green checkmarks). The right pane shows the Event-B code for the 'RemAuth' event, with the guard condition highlighted in blue.

Event-B Explorer (Left Pane):

- Events
- Proof Obligations
 - INITIALISATION/inv1/INV
 - INITIALISATION/inv2/INV
 - INITIALISATION/inv3/INV
 - INITIALISATION/inv4/INV
 - AddAuth/inv1/INV
 - AddAuth/inv4/INV
 - Enter/inv3/INV
 - Enter/inv4/INV
 - Leave/inv3/INV
 - Leave/inv4/INV
 - RemAuth/inv1/INV
 - RemAuth/inv4/INV
- M2
 - Rules
 - SecureDB
 - Shadow
 - SharedBuffers20081008
 - Ships
 - SignalEvaluationCruiseControl
 - SSF1

Event-B Code (Right Pane):

```
@grd2  $r \in \text{Room}$ 
@grd3  $\text{takeplace}[\{r\}] \subseteq \text{authorised}[\{u\}]$ 
then
  @act1  $\text{location} := \text{location} \cup \{ u \mapsto r \}$ 
end

event Leave
  any  $u \ r$ 
  where
    @grd1  $u \mapsto r \in \text{location}$ 
  then
    @act1  $\text{location} := \text{location} \setminus \{ u \mapsto r \}$ 
  end
end

event RemAuth
  any  $u \ a$ 
  where
    @grd3  $u \mapsto a \in \text{authorised}$ 
    @grd4  $\forall rr. u \mapsto rr \in \text{location} \Rightarrow rr \mapsto a \notin \text{takeplace}$ 
  then
    @act1  $\text{authorised} := \text{authorised} \setminus \{ u \mapsto a \}$ 
  end
end
```

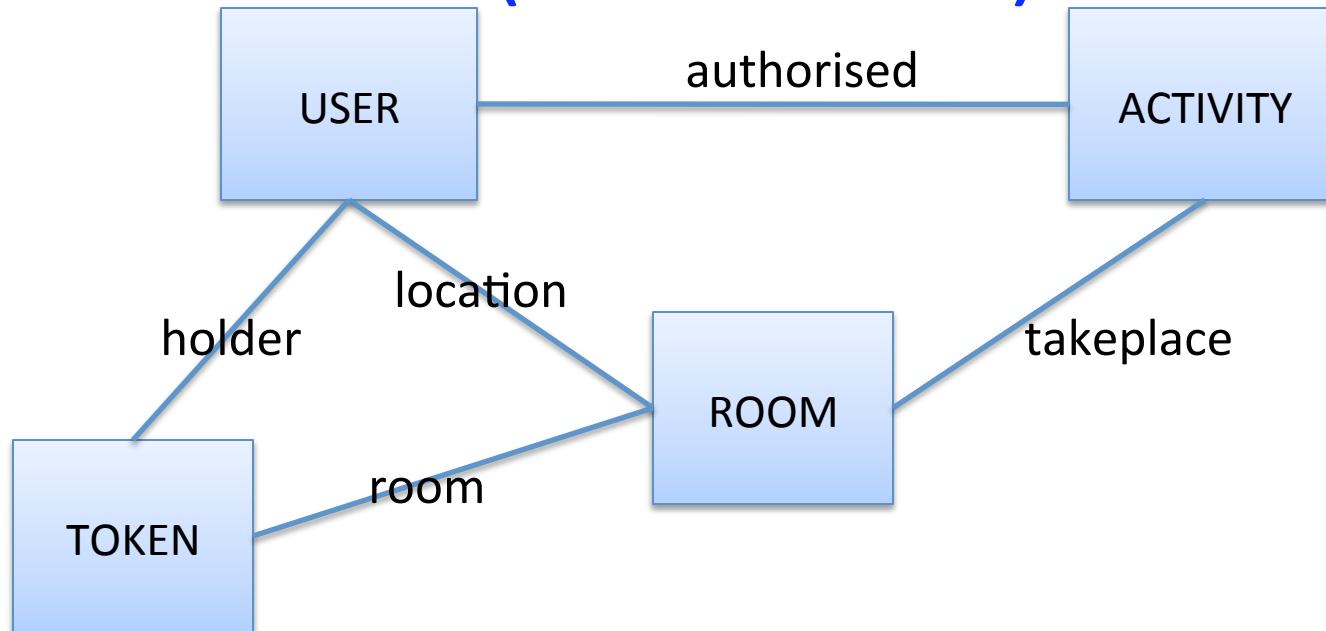
Page-Footer:

1st Oct 2012
SAICSIT: Event-B/1

Early stage analysis

- We constructed a simple **abstract** model
- Already using verification technology we were able to **identify errors** in our conceptual model of the desired behaviour
 - we found a solution to these early on
 - verified the “correctness” of the solution
- Now, lets proceed to another **stage** of analysis...

We construct a new model (refinement)



Guard of abstract Enter event:

grd3: $\text{takeplace}[r] \subseteq \text{authorised}[u]$

is replaced by a guard on a token:

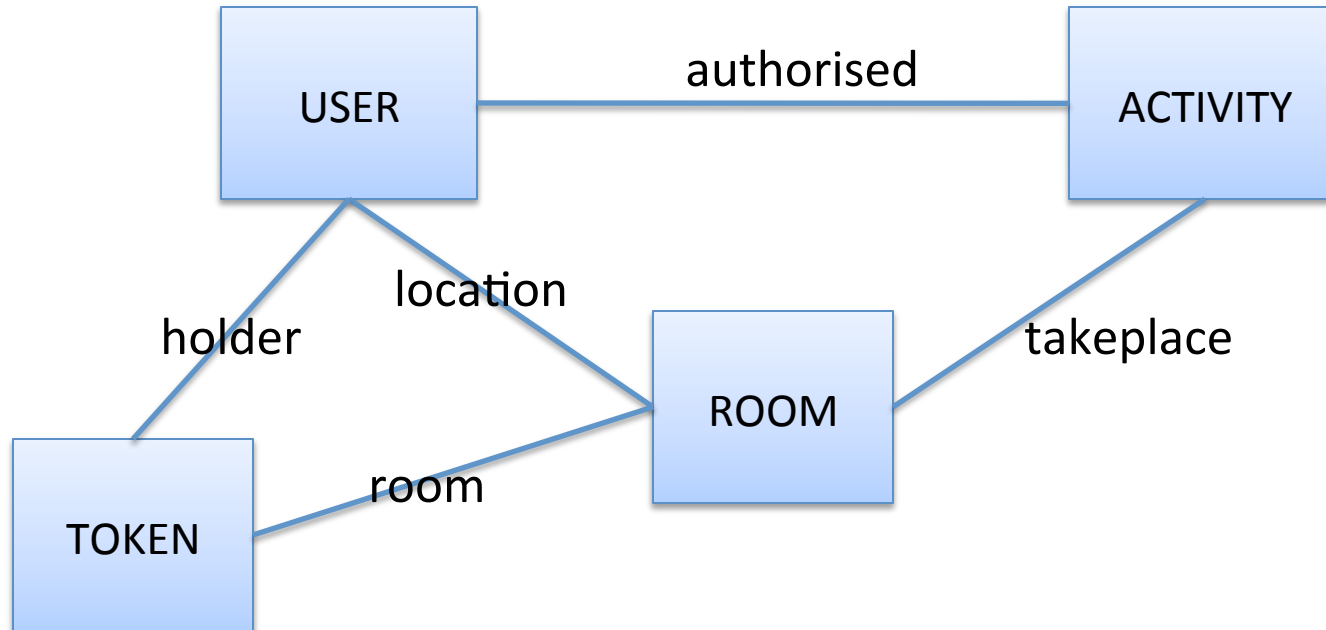
grd3b: $t \in \text{valid} \wedge \text{room}(t) = r \wedge \text{holder}(t) = u$

Failing refinement proof

The screenshot shows a theorem prover interface with the following components:

- Top toolbar:** Contains icons for undo (red circle with minus), redo (checkmark), copy (yellow arrows), and paste (empty square).
- Hypotheses list:** A list of three hypotheses, each with an unchecked checkbox, a blue underlined 'ct' label, and a text expression:
 - `t=validToks`
 - `r=room(t)`
 - `u=holder(t)`
- Selected Hypotheses:** A label for the selected hypotheses section.
- Goal:** A section with a green checkmark icon, the word 'Goal', and a close button (X). Below it, the goal expression is displayed:
`ct takeplace[{room(t)}] ⊆ authorised[{holder(t)}]`
- Bottom panel:** Includes a scroll bar and a keyboard shortcuts bar with keys: N1, N, P1, P, Z, O, U, V, ~, T, L, °, 8.

Gluing invariant



To ensure consistency of the refinement we need **invariant**:

inv 6: $t \in \text{valid}$

\Rightarrow

$\text{takeplace} [\text{room}(t)] \subseteq \text{authorised} [\text{holder}(t)]$

Invariant enables PO discharge

The screenshot displays the Rodin Platform Proving interface, titled "Proving - Rooms1/M2.bps - Rodin Platform - /Users/mjb/Rodin/workspace1.0". The interface is divided into several panes:

- Left Pane:** Contains a "Proof T" and "Proof I" section with a green checkmark and a blue "ML" label.
- Center Pane:** Shows a list of variables under the heading "Enter/grd3/GRD". Each variable is preceded by a green checkmark and a "ct" label:
 - ☐ **ct** ueUser \ dom(location)
 - ☐ **ct** tetok
 - ☐ **ct** r=rtok(t)
 - ☐ **ct** u=utok(t)Below this list is a "State" section and a "Goal" section containing the expression `takeplace[{r}] ⊆ authorised[{u}]`.
- Right Pane:** Displays a tree view of the proof obligations. The "Proof Obligations" section is expanded, showing a list of obligations, each with a green checkmark and a "A" label:
 - inv2 /WD
 - INITIALISATION /inv2 /INV
 - AddAuth /inv2 /INV
 - CreateToken /grd5 /WD
 - CreateToken /grd6 /WD
 - CreateToken /inv2 /INV
 - Enter /grd3 /WD
 - Enter /grd4 /WD
 - Enter /inv2 /INV
 - Enter /grd3 /GRD
 - RescindToken /inv2 /INV
 - RemAuth /grd5 /WD
 - RemAuth /inv2 /INV
 - RemAuth /grd4 /GRD
- Bottom Pane:** Contains a "Proof Cont" section, a "Statistics" section, and a "Rodin Prob" section. It also features a green smiley face icon and the text "SAICSIT: Event-B/1".

But get new failing PO

The screenshot displays the Rodin Platform interface for proving a property. The main window is titled "Proving - Rooms1/M2.bps - Rodin Platform - /Users/mjb/Rodin/workspace1.0". The interface is divided into several panes:

- Proof Tree:** Shows the current proof state. The root node is "RemAuth/inv2/INV". It contains two events: "Event in M1" and "Event in M2". The "Event in M2" node is expanded, showing a "RemAuth" node with a "REFINES" block. The "REFINES" block contains a "RemAuth" node and a "WHERE" block. The "WHERE" block contains three goals: "grd3: $u \mapsto a \in \text{authorised}$ ", "grd4: $\forall rr. u \mapsto rr \in \text{location} \Rightarrow rr \mapsto a \notin \text{takeplace}$ ", and "grd5: $u \in \text{dom}(\text{location}) \Rightarrow \text{location}(u) \mapsto a \notin \text{takeplace}$ ". The "Event in M2" node also contains a "THEN" block with a goal "act1: $\text{authorised} = \text{authorised} \setminus \{ u \mapsto a \}$ ".
- State:** Shows the current state of the proof. It contains a "Goal" block with a goal "takepla" and a "ct" (checked) status.
- Goal:** Shows the current goal. It contains a goal "takepla" and a "ct" (checked) status.
- Proof Obligations:** A list of proof obligations. The list includes: "inv2/WD", "INITIALISATION/inv2", "AddAuth/inv2/INV", "CreateToken/grd5/", "CreateToken/grd6/", "CreateToken/inv2/II", "Enter/grd3/WD", "Enter/grd4/WD", "Enter/inv2/INV", "Enter/grd3/GRD", "RescindToken/inv2/", "RemAuth/grd5/WD", "RemAuth/inv2/INV" (highlighted with a red question mark), and "RemAuth/grd4/GRD".

The bottom of the interface shows a "Rules" pane with a list of rules: "SecureDB", "Shadow", and "Shadow-Buffer-20001000". The bottom right corner displays the page number "38".

Strengthen guard of refined *RemAuth*

The screenshot displays the Rodin Platform interface for proving a refinement. The main window shows the Rodin script for the `RemAuth` event, which refines the `RemAuth` event. The script includes guards for authorization and location, and an action to update the authorized set. The `@grd6` guard is highlighted, showing a strengthened condition.

```
event RemAuth refines RemAuth
  any u a
  where
    @grd3  $u \mapsto a \in \text{authorised}$ 
    @grd5  $u \in \text{dom}(\text{location}) \Rightarrow \text{location}(u) \mapsto a \notin \text{takeplace}$ 
    @grd6  $\forall t. t \in \text{tok} \wedge u = \text{utok}(t) \Rightarrow \text{rtok}(t) \mapsto a \notin \text{takeplace}$ 
  then
    @act1  $\text{authorised} = \text{authorised} \setminus \{ u \mapsto a \}$ 
  end
end
```

The `Goal` window shows the goal to be proved:

```
 $\forall t. t \in \text{tok} \Rightarrow \text{takeplace}[\{\text{rtok}(t)\}] \subseteq (\text{authorised} \setminus \{ u \mapsto a \})[\{\text{utok}(t)\}]$ 
```

The `Proof Control` window shows the progress of the proof, with a list of proof obligations on the right. The `RemAuth/in` obligation is highlighted.

1st Oct 2012 SAICSIT: Event-B/1 39

Requirements revisited

1. Users are authorised to engage in activities
2. User authorisation may be added or revoked
3. Activities take place in rooms
4. ...

Question: was it obvious initially that revocation of authorisation was going to be problematic?

Rational design – what, how, why

- *What* does it achieve?
 - if user u is in room r ,
 - then u must be authorised to engaged in
all activities that can take place in r
- *How* does it work?
 - Check that a user has a valid token
- *Why* does it work?
 - For any valid token t , the holder of t must be authorised to
engage in all activities that can take place in the room
associated with t

What, how, why written in Event-B

- *What* does it achieve?

inv1: $u \in \text{dom}(\text{location}) \wedge \text{location}(u) = r$
 \Rightarrow
 $\text{takeplace}[r] \subseteq \text{authorised}[u]$

- *How* does it work?

grd3b: $t \in \text{valid} \wedge r = \text{room}(t) \wedge u = \text{holder}(t)$

- *Why* does it work?

inv2: $t \in \text{valid}$
 \Rightarrow
 $\text{takeplace}[\text{room}(t)] \subseteq \text{authorised}[\text{holder}(t)]$

B Method (Abrial, from 1990s)

- *Model* using set theory and logic
- *Analyse models* using proof, model checking, animation
- Refinement-based development
 - verify conformance between higher-level and lower-level models
 - chain of refinements
- Code generation from low-level models
- Commercial tools, :
 - *Atelier-B* (ClearSy, FR) - used mainly in railway industry
 - *B-Toolkit* (B-Core, UK, Ib Sorensen)

B evolves to Event-B (from 2004)

- B Method was designed for *software* development
- Realisation that it is important to reason about *system* behaviour, not just software
- Event-B is intended for modelling and refining system behaviour
- Refinement notion is more flexible than B
 - Same set theory and logic
- Rodin tool for Event-B (V1.0 2007)
 - Open source, Eclipse based, open architecture
 - Range of plug-in tools

System level reasoning

- Examples of systems modelled in Event-B:
 - Train signalling system
 - Mechanical press system
 - Access control system
 - Air traffic information system
 - Electronic purse system
 - Distributed database system
 - Cruise control system
 - Processor Instruction Set Architecture
 - ...
- System level reasoning:
 - Involves abstractions of *overall* system not just software components

Other Lectures

- Verification and tools in Event-B modelling
- Case study: the cardiac pacemaker

Rodin Demo

Access Control Example

END