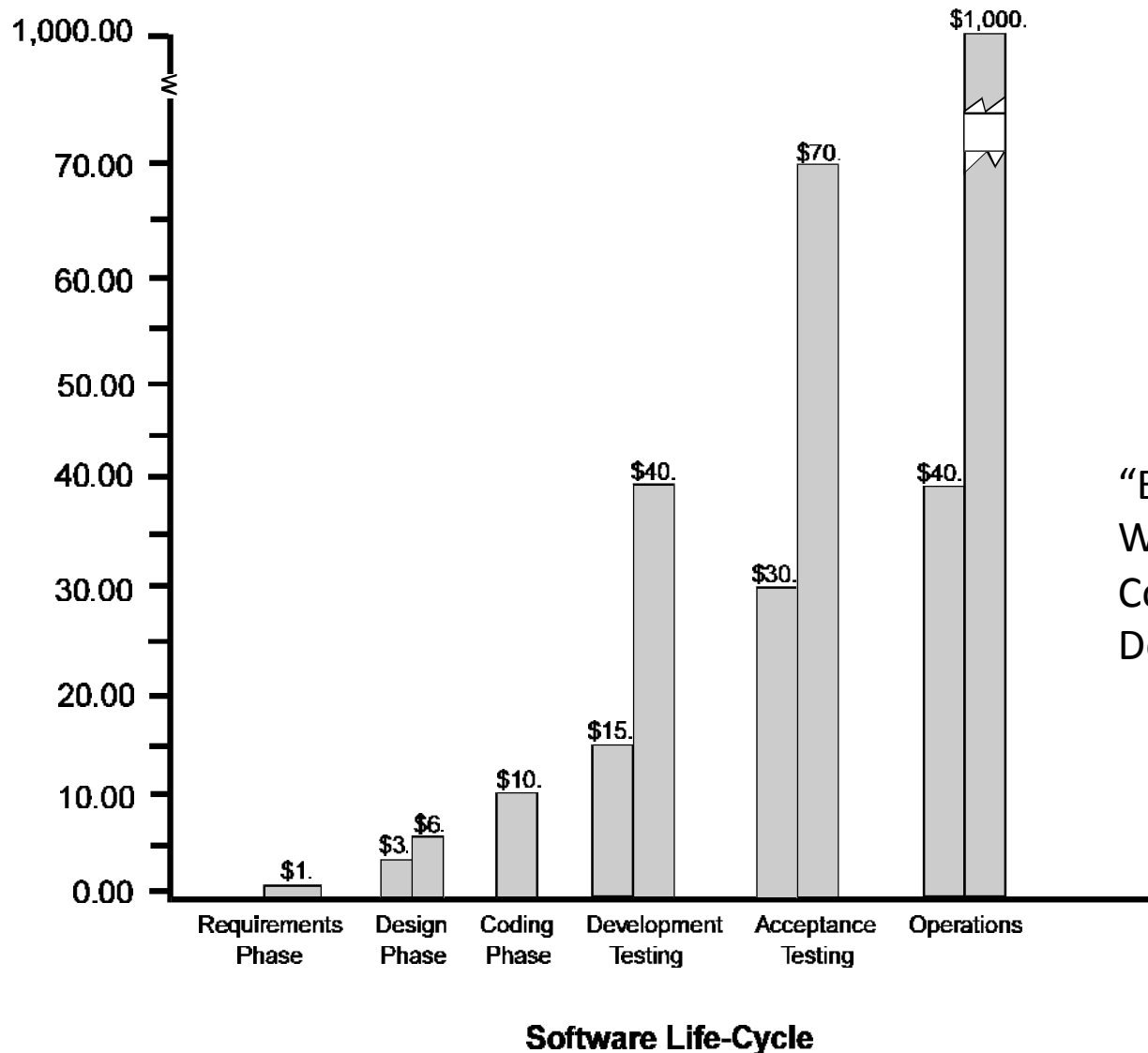


Introduction to Modelling

(with Event-B)

www.event-b.org

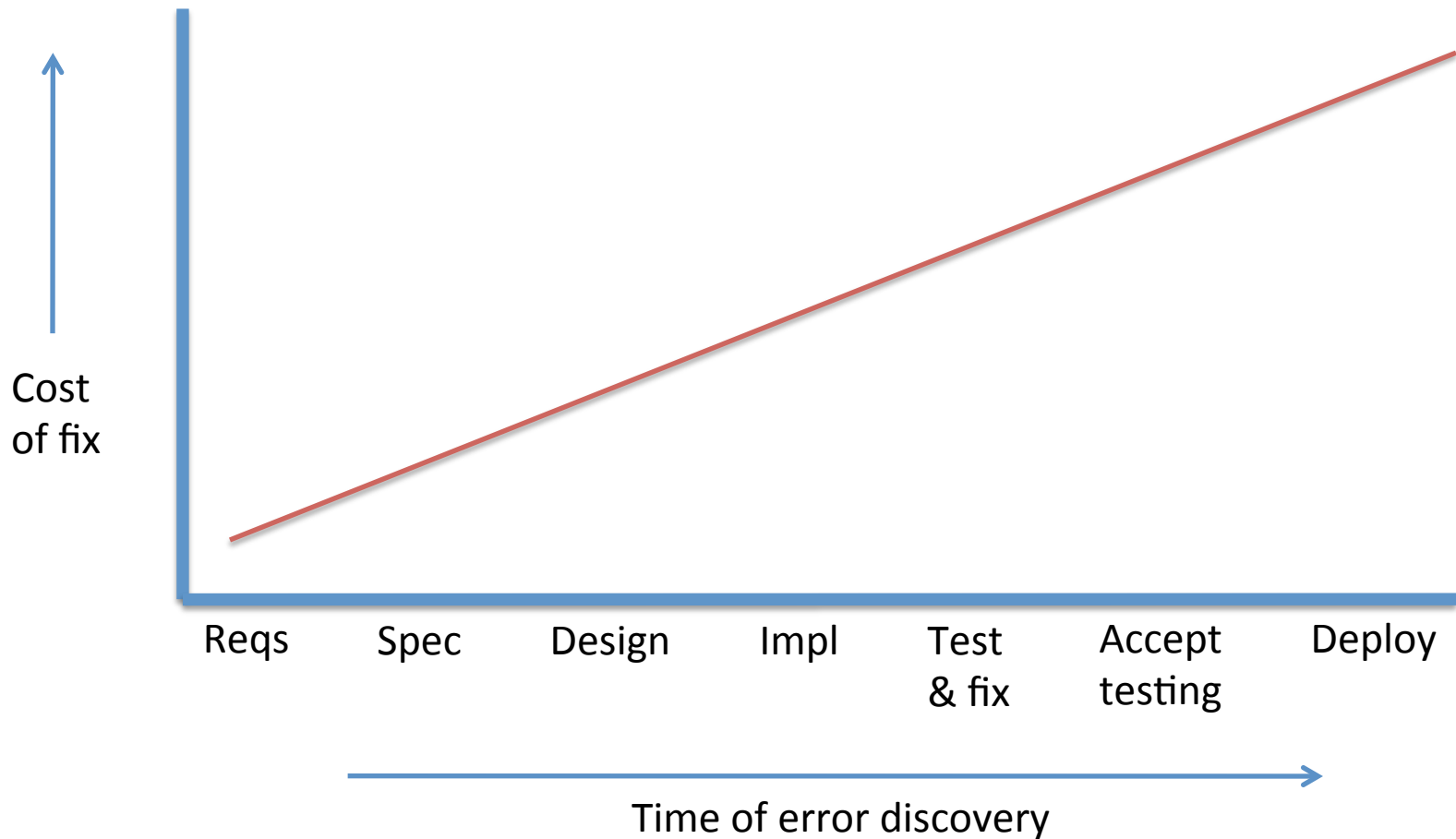
Cost of fixing requirements errors



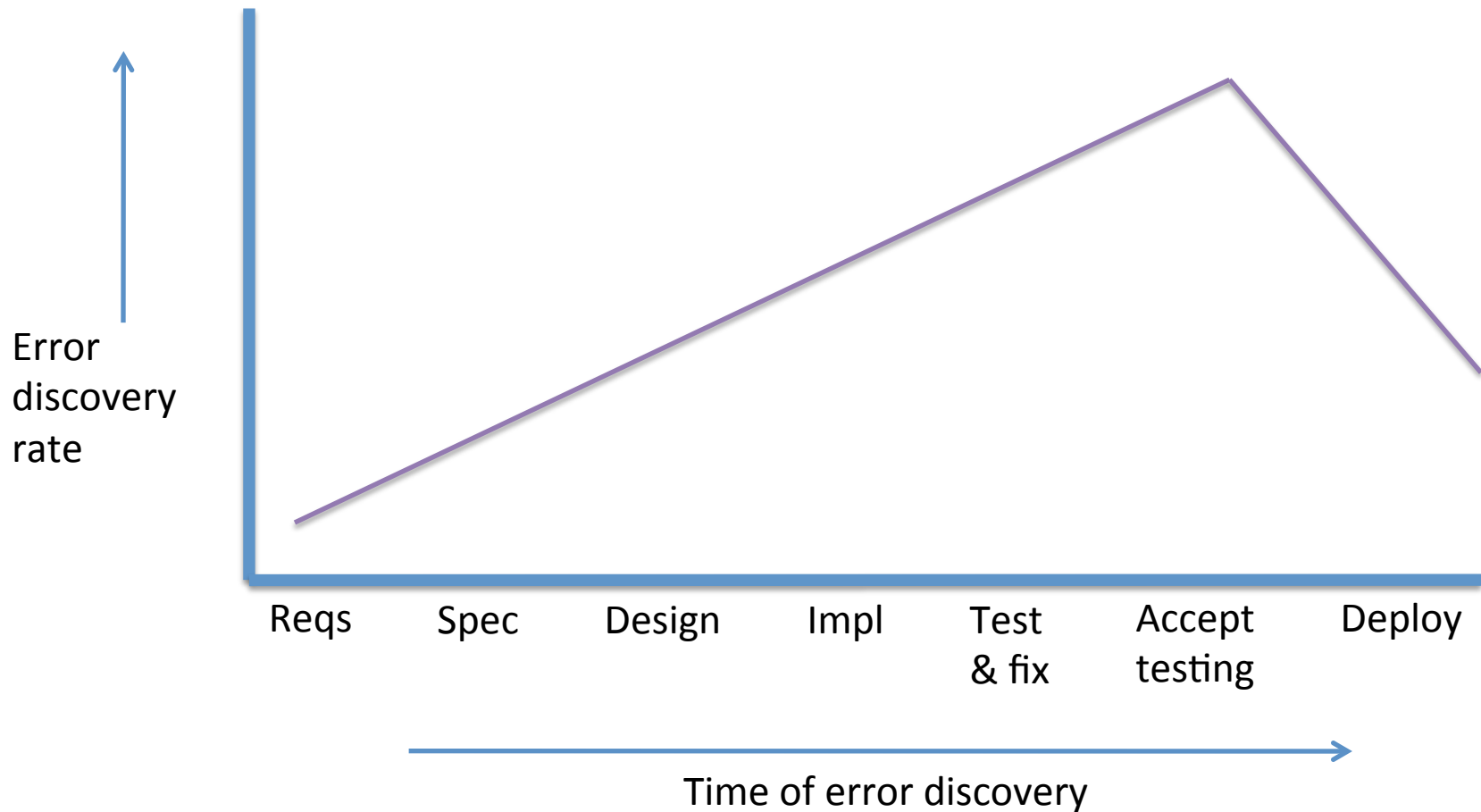
“Extra Time Saves Money”
Warren Kuffel
Computer Language
December 1990

Cost of error fixes grows

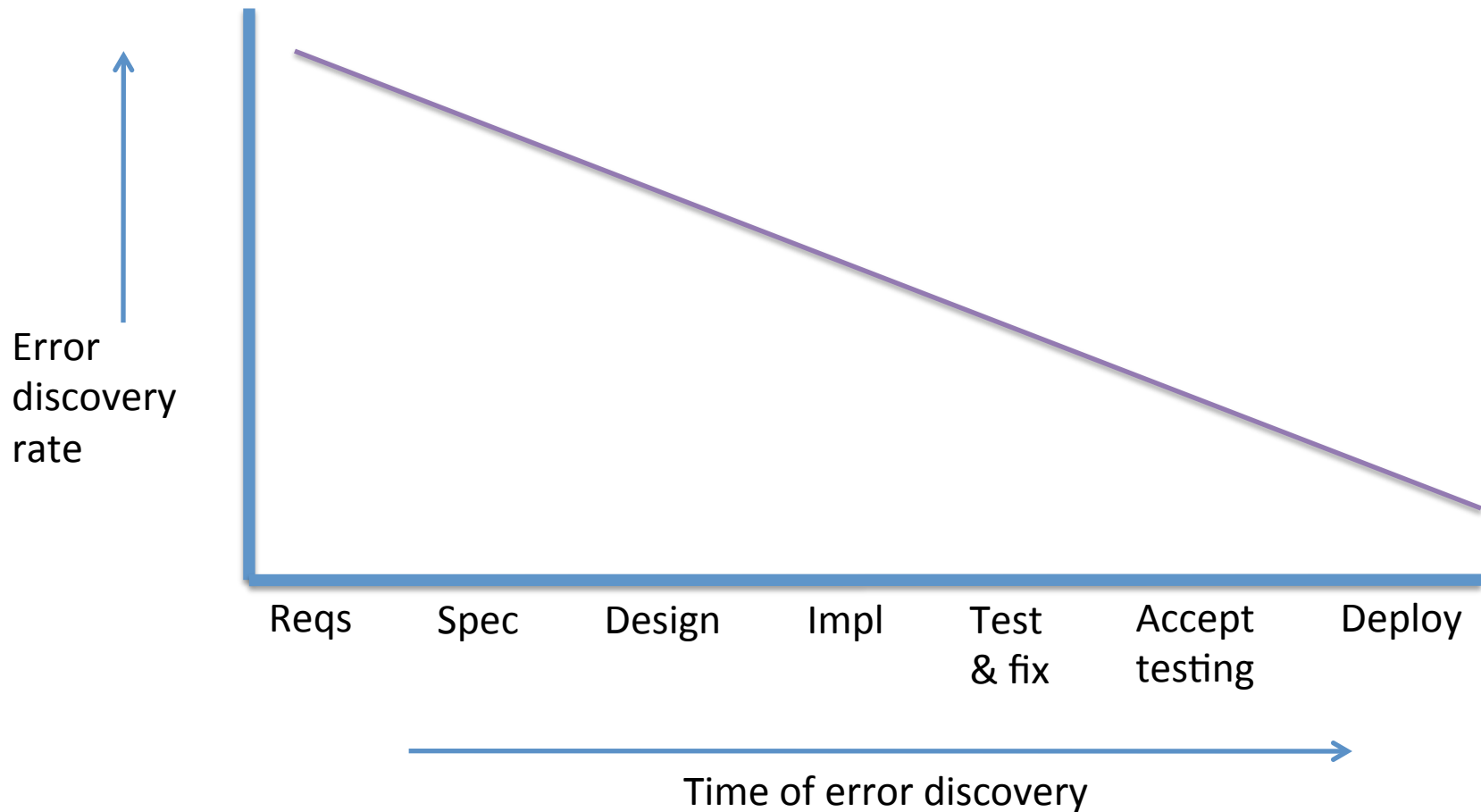
- difficult to change this



Rate of error discovery



Invert error identification rate?



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 precision and abstraction at early stages (front-loading)

- Precision through early stage models
 - Amenable to analysis by tools
 - Identify and fix ambiguities and inconsistencies as early as possible
- Mastering complexity through abstraction
 - Focus on *what* a system does (its purpose)
 - Incremental analysis and design

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

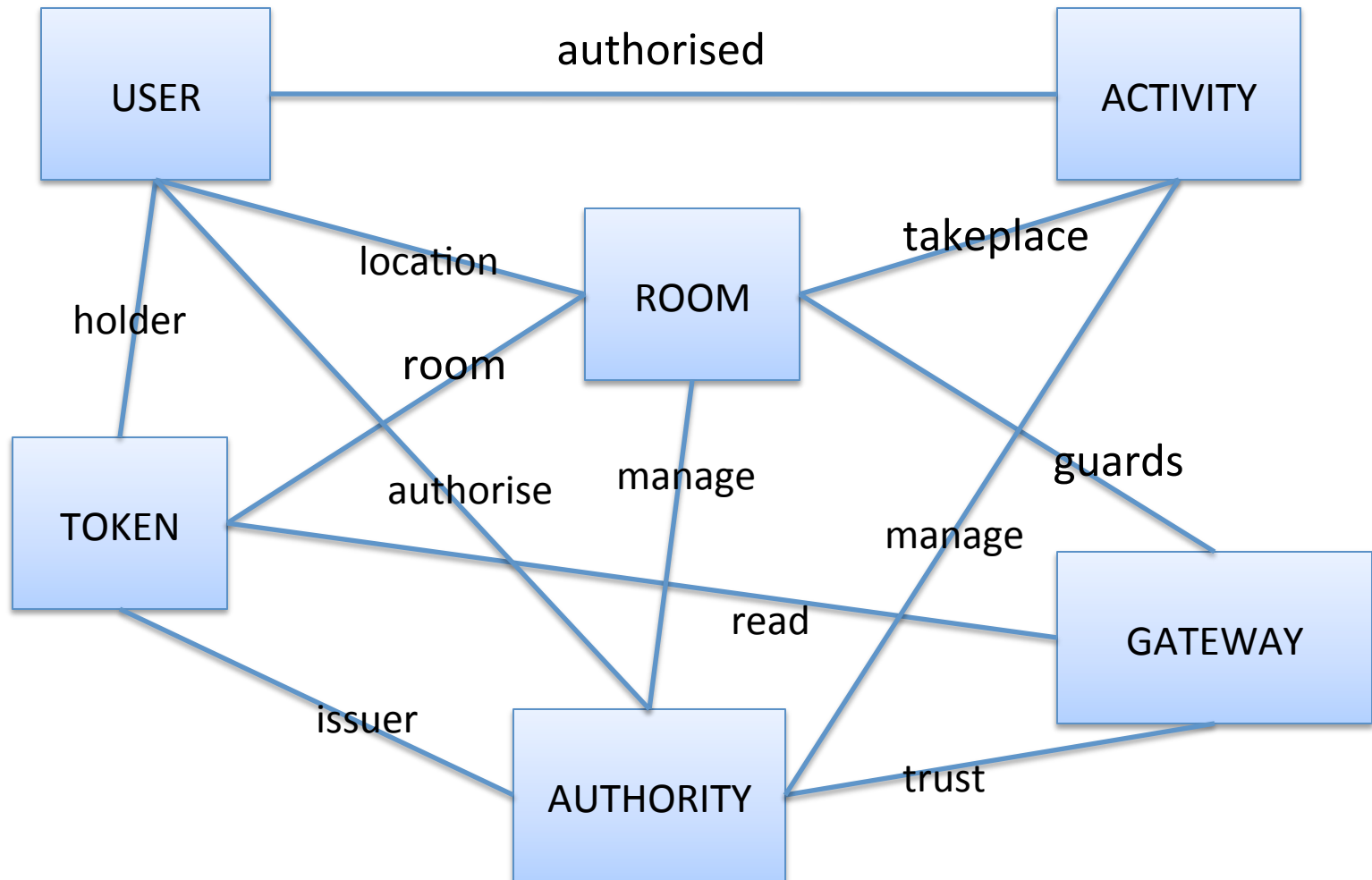
Access control requirements

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

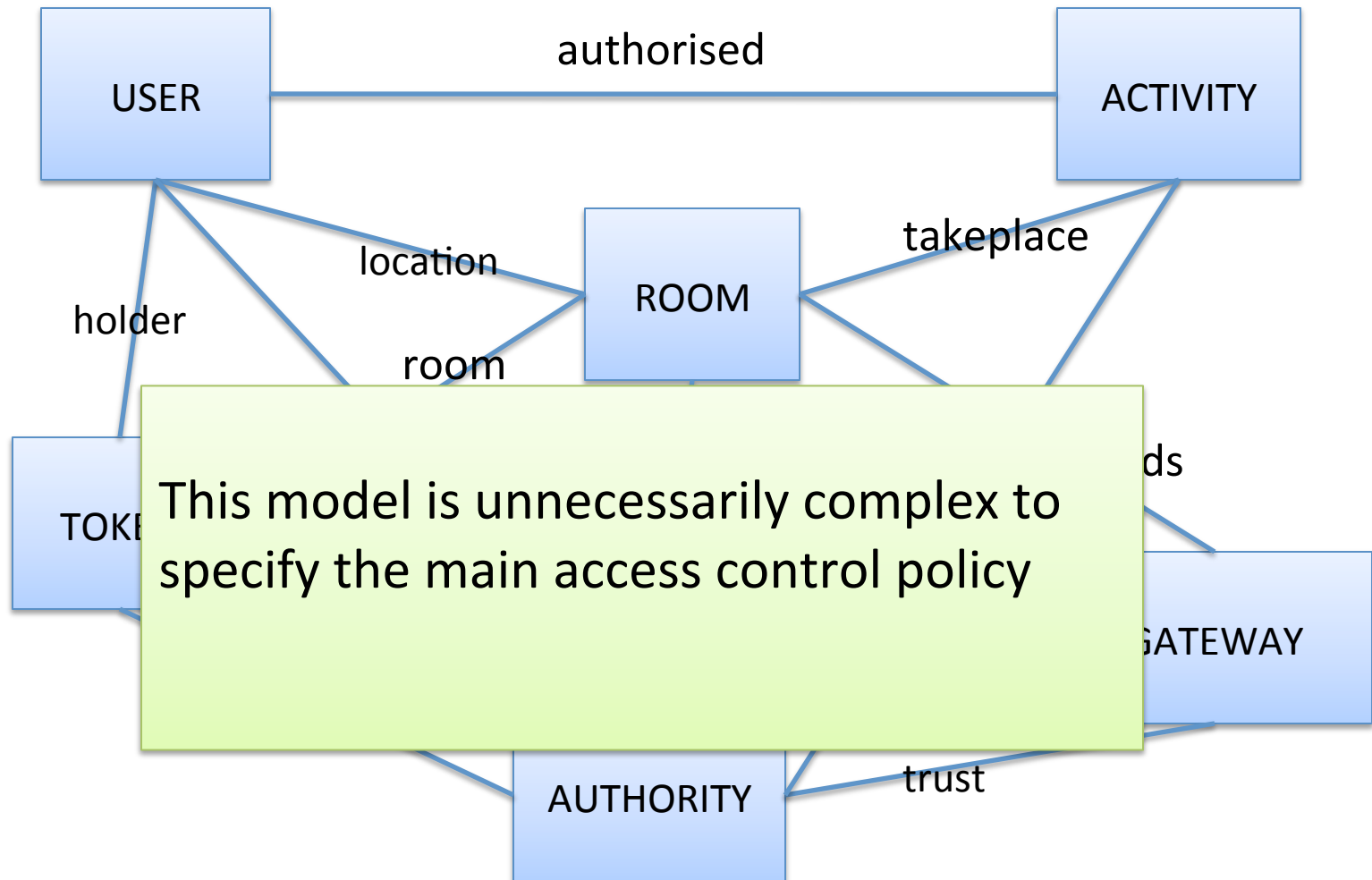
Access control requirements

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



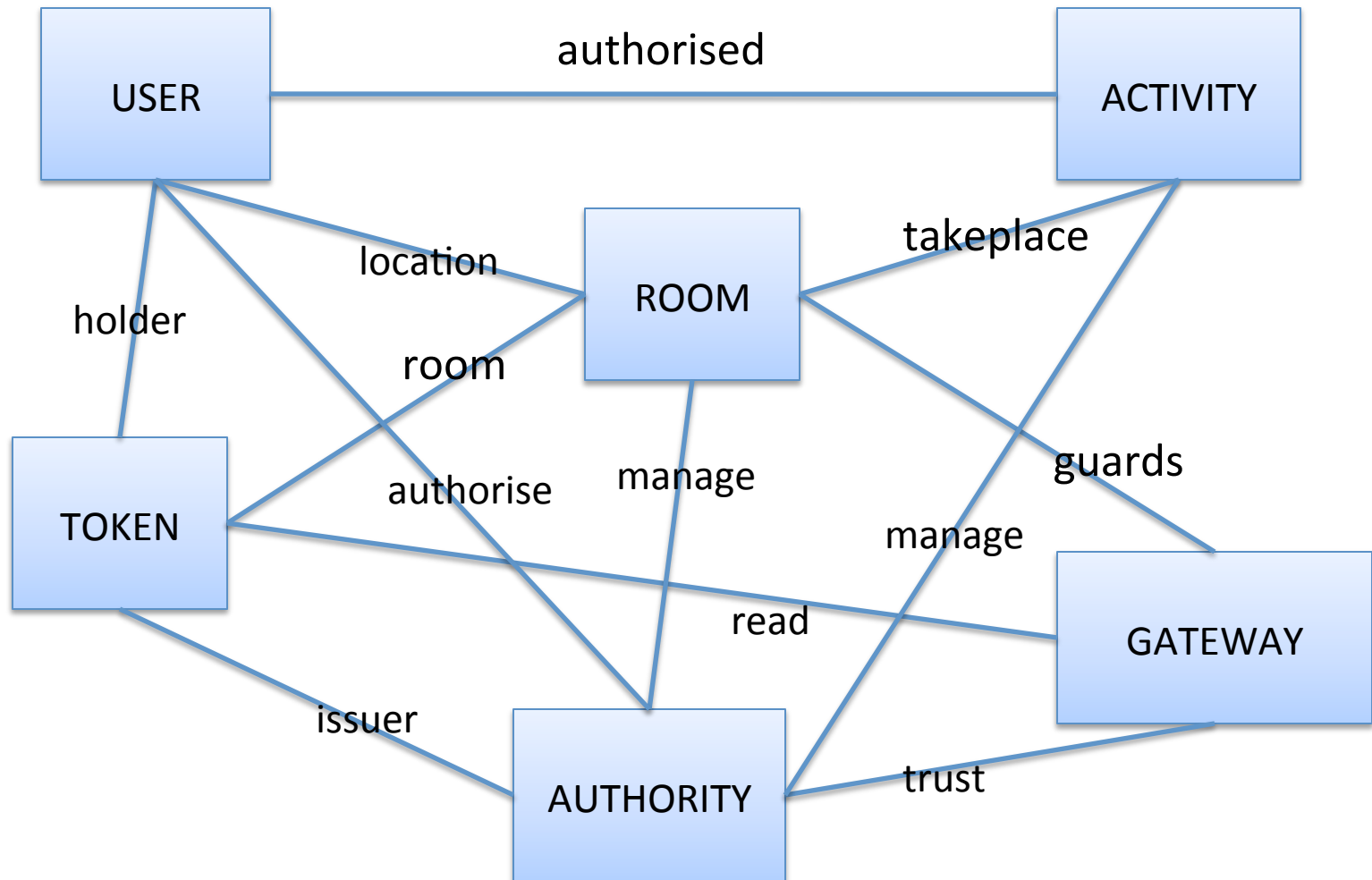
Entities and relationships



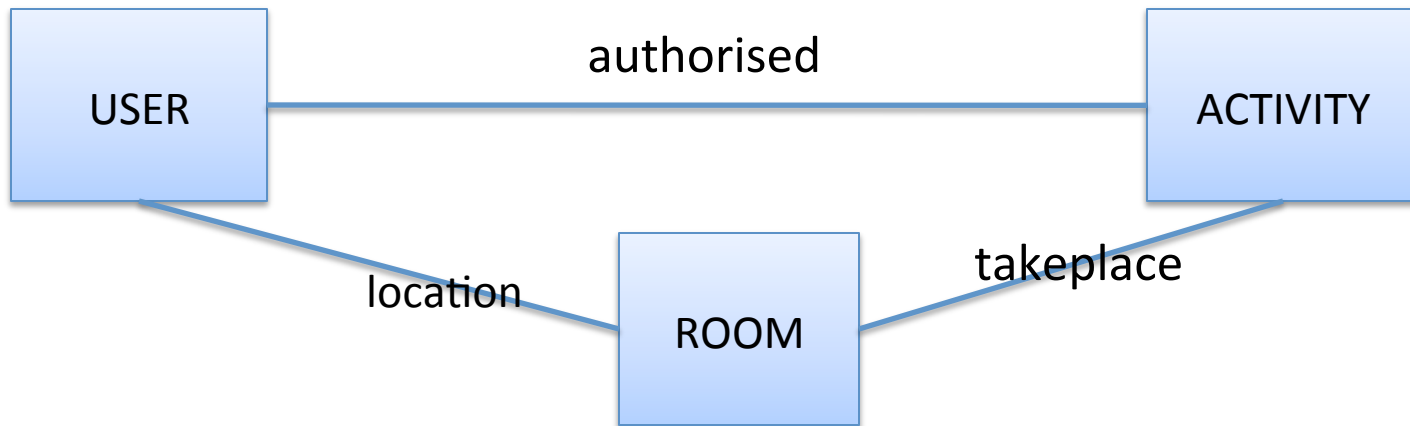
Extracting the essence

- **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

authorised \in USER \leftrightarrow ACTIVITY // relation

takeplace \in ROOM \leftrightarrow ACTIVITY // relation

location \in USER \rightarrow 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	a1

authorised

ROOM	ACTIVITY
r1	a1
r1	a2
r2	a1

takeplace

USER	ROOM
u1	r1
u2	r2
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 like "authorised", "location", and "takeplace". The "authorised" variable is highlighted in red, indicating a violation of an invariant. The "Value" column shows the current values for these variables, such as $\{(User1 \rightarrow Activity1), (User2 \rightarrow Activity2)\}$ for "authorised".

Name	Value
M1	
authorised	$\{(User1 \rightarrow Activity1), (User2 \rightarrow Activity2)\}$
location	$\{(User1 \rightarrow Room2)\}$
takeplace	$\{(Room1 \rightarrow Activity1), (Room1 \rightarrow Activity2), (Room2 \rightarrow Activity1), (Room2 \rightarrow Activity2)\}$

On the right side, the "History" panel shows a list of operations performed during the execution, including `RemAuth(Activity2, User1)`, `Enter(Room2, User1)`, `AddAuth(Activity2, User2)`, `AddAuth(Activity2, User1)`, `AddAuth(Activity1, User1)`, `$initialise_machine({}, {}, {z})`, `$setup_constants()`, and `(root)`.

At the bottom of the interface, a red banner displays the message "invariant violated!", indicating that the model checker has found a counter-example to the invariant being checked.

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)

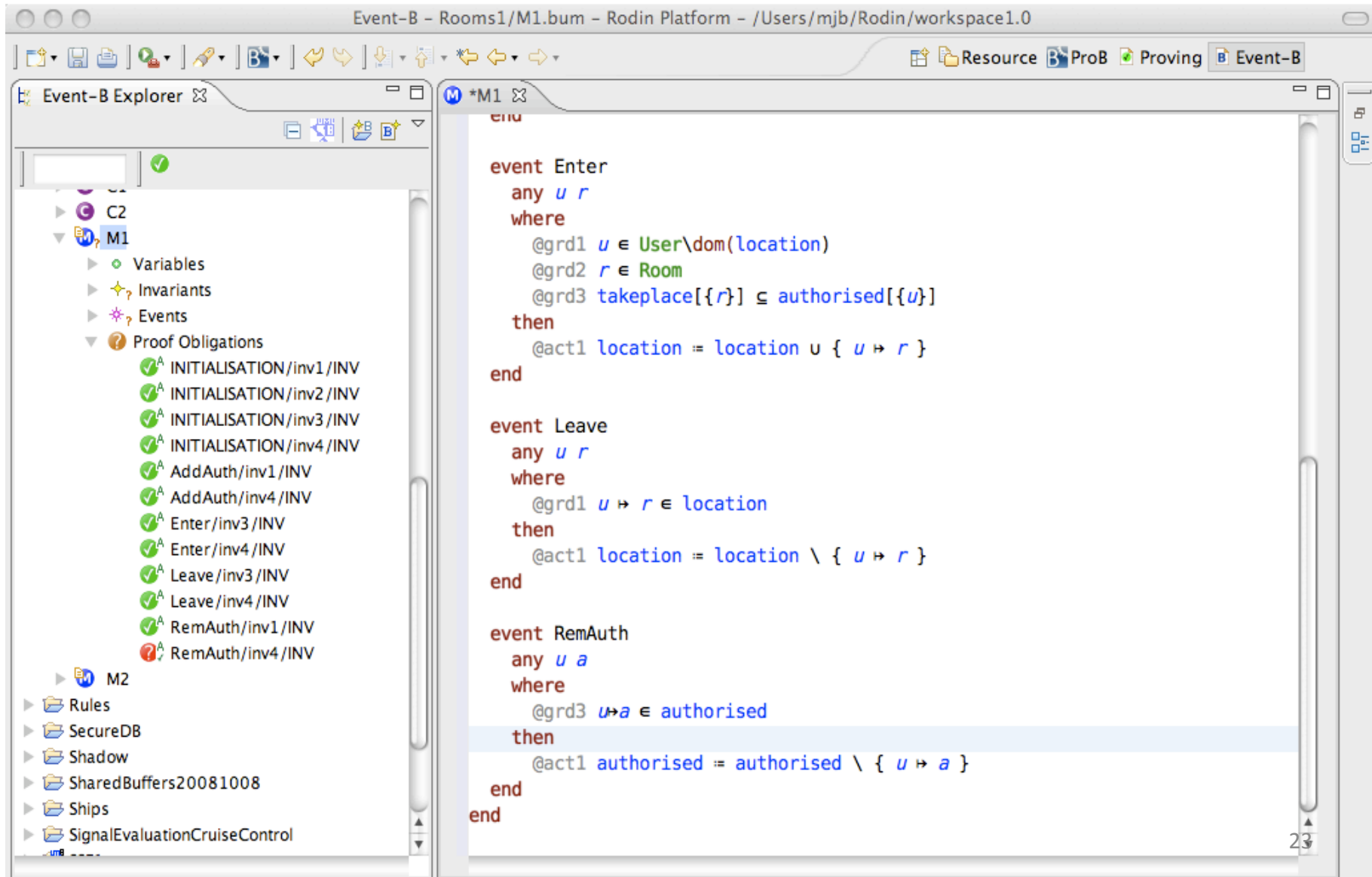
AddAuth(Activity1,User1)

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

\$setup_constants()

(root)

Failing proof



Strengthen guard of *RemAuth*

The screenshot displays the Rodin Platform interface, specifically the Event-B Explorer and the Event-B code editor. The title bar indicates the project is 'Event-B - Rooms1/M1.bum' in the 'Rodin Platform' workspace.

Event-B Explorer (Left Panel):

- Events:** A list of events including INITIALISATION, AddAuth, Enter, Leave, and RemAuth.
- Proof Obligations:** A list of proof obligations for each event, all marked with a green checkmark, indicating they have been successfully proven. The obligations for RemAuth are highlighted.
- Rules:** A list of rules including SecureDB, Shadow, SharedBuffers20081008, Ships, SignalEvaluationCruiseControl, and SSF1.

Event-B Code Editor (Right Panel):

The code editor shows the Event-B code for the 'RemAuth' event. The code is as follows:

```
@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

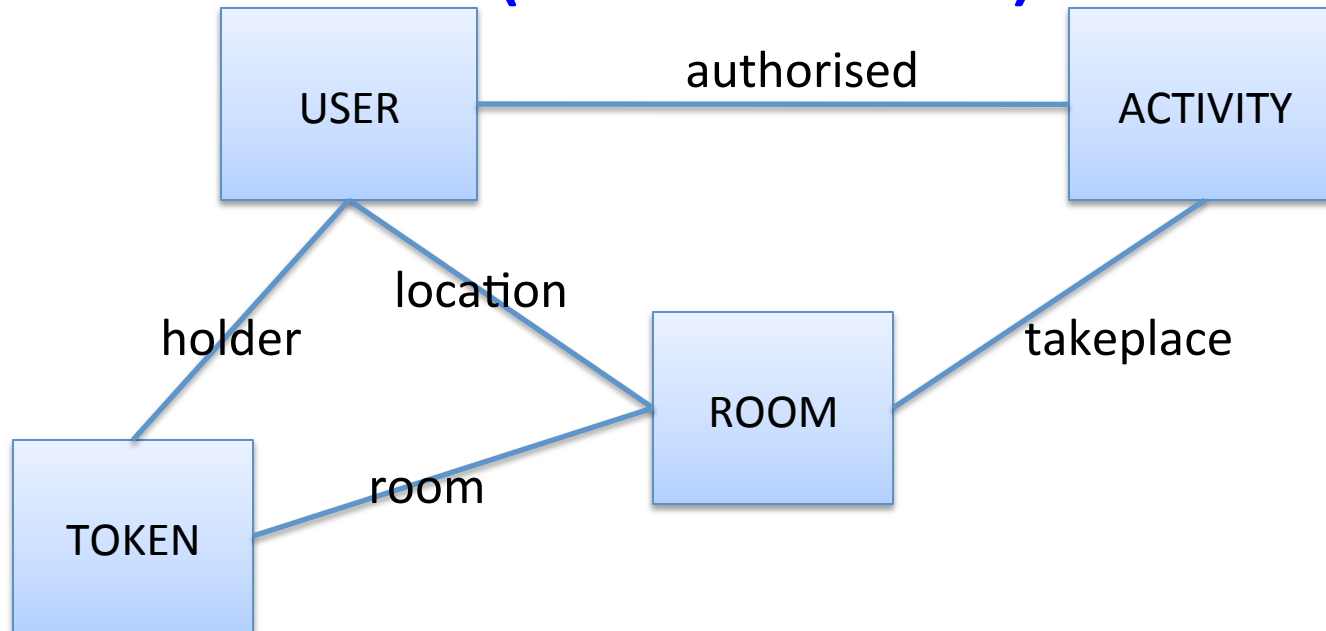
event RemAuth
  any  $u \ a$ 
  where
    @grd3  $u \mapsto a \in \text{authorised}$ 
    @grd4  $\forall rr. u \mapsto rr \in \text{location} \Rightarrow rr \mapsto a \notin \text{takeplace}$ 
  then
    @act1  $\text{authorised} := \text{authorised} \setminus \{u \mapsto a\}$ 
  end
end
```

The bottom status bar shows the 'RODIN Keyboard', 'Progress', 'Symbols', 'Rodin Problems', and 'Statistics' tabs.

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 a toolbar at the top containing icons for undo, redo, and other actions. Below the toolbar, there are three hypotheses listed, each with an unchecked checkbox and a blue underlined 'ct' label:

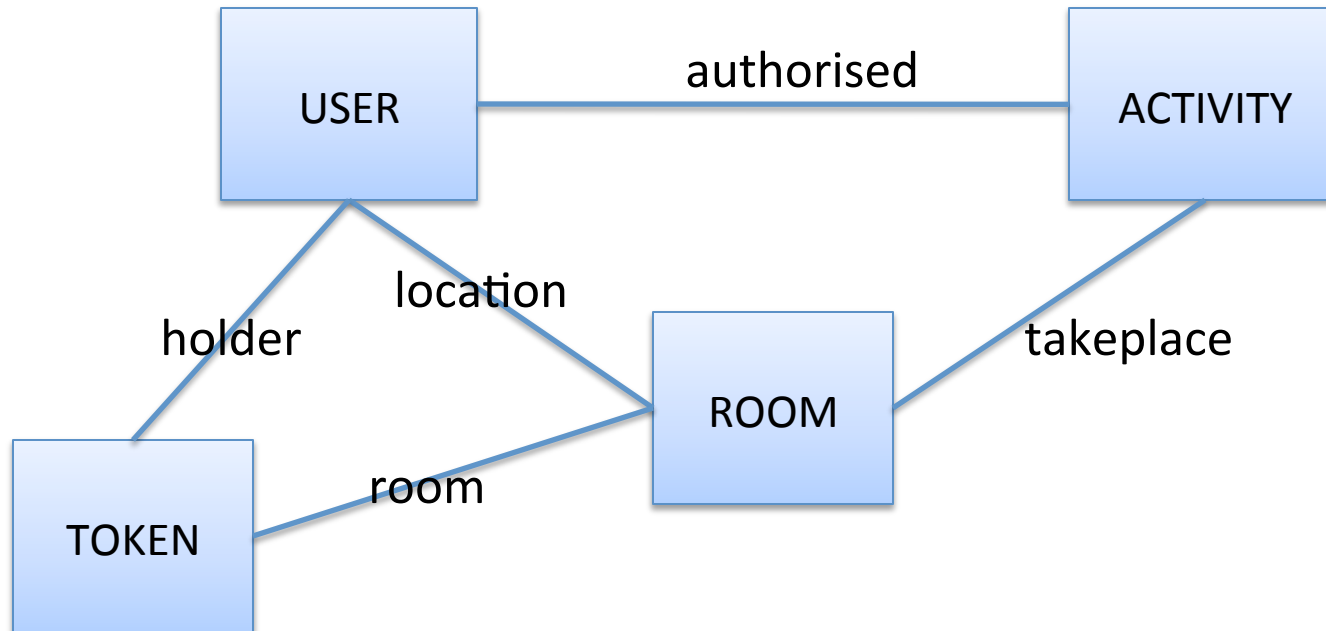
- ☐ ct `t ∈ validToks`
- ☐ ct `r = room(t)`
- ☐ ct `u = holder(t)`

Below the hypotheses is a section labeled "Selected Hypotheses" which is currently empty. At the bottom, there is a "Goal" section with a green checkmark icon and a close button. The goal statement is:

ct `takeplace[{room(t)}] ⊆ authorised[{holder(t)}]`

At the very bottom right, there is a navigation bar with buttons labeled N1, N, P1, P, Z, n, U, v, ~, T, L, o, and g.

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

Proving - Rooms1/M2.bps - Rodin Platform - /Users/mjb/Rodin/workspace1.0

Resource ProB Proving Event-B

Proof T Proof I

G ML

M1 M2 M2

Enter/grd3/GRD

- ☐ **ct** ueUser \ dom(location)
- ☐ **ct** tetok
- ☐ **ct** r=rtok(t)
- ☐ **ct** u=utok(t)

State

Goal

ct takeplace[{r}]⊆authorised[{u}]

Proof Cont Statistics Rodin Prob

nPP R p0 dc ah ae

Rules
SecureDB
Shadow
ShadowBuffer20081008

Progr Event Search

M1
M2
Variables
Invariants
Events
Proof Obligations

- 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

29

But get new failing PO

Proving - Rooms1/M2.bps - Rodin Platform - /Users/mjb/Rodin/workspace1.0

Resource ProB Proving Event-B

Proof Tree Proof Information

RemAuth/inv2/INV

- Event in M1
 - RemAuth:
 - ANY u, a WHERE
 - $\text{grd3: } u \mapsto a \in \text{authorised}$
 - $\text{grd4: } \forall rr. u \mapsto rr \in \text{location} \Rightarrow rr \mapsto a \neq \text{takeplace}$
 - THEN
 - $\text{act1: } \text{authorised} = \text{authorised} \setminus \{ u \mapsto a \}$
 - END
- Event in M2
 - RemAuth:
 - REFINES
 - RemAuth
 - ANY u, a WHERE
 - $\text{grd3: } u \mapsto a \in \text{authorised}$
 - $\text{grd5: } u \in \text{dom}(\text{location}) \Rightarrow \text{location}(u) \mapsto a \neq \text{takeplace}$
 - $\text{grd6: } \tau$
 - THEN
 - $\text{act1: } \text{authorised} = \text{authorised} \setminus \{ u \mapsto a \}$
 - END
- Invariant in M2
 - $\text{inv2: } \forall t. t \in \text{tok} \Rightarrow \text{takeplace}[\{\text{rtok}(t)\}] \subseteq \text{authorised}[\{\text{utok}(t)\}]$

State

Goal

takepla

Proof Obligations

- 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**
- RemAuth/grd4/GRD

Rules

SecureDB

Shadow

ShadowBuffer

30

Strengthen guard of refined *RemAuth*

The screenshot displays the Rodin Platform interface for proving a refinement. The main window shows the refinement of the `RemAuth` event. The refinement is defined as follows:

```
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 to be proved is:

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

The right-hand side of the interface shows a list of proof obligations, all of which are marked as proven (green checkmark):

- inv2 / WD
- INITIALISATI
- AddAuth / inv
- CreateToke
- CreateToke
- CreateToke
- Enter / grd3 /
- Enter / grd4 /
- Enter / inv2 / l
- Enter / grd3 /
- RescindTok
- RemAuth / gr
- RemAuth / gr
- RemAuth / in
- RemAuth / gr

The bottom of the interface shows the Proof Control, Statistics, and Rodin Problems tabs, along with a toolbar and a status bar.

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

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

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**

