iUML-B
Class Diagrams

Motivation

Provide a more approachable interface for newcomers to Event-B

Provide diagrams to help visualise models

Provide extra modelling features to Event-B
- Sequencing of Events (see State-machines)
- Lifting of Behaviour to a set of instances (O-O)

N.b. not trying to formalise UML

What is iUML-B?

A Graphical front-end for Event-B

- Plug-in for Rodin

Not UML …

- Has its own meta-model (abstract syntax)
- Semantics inherited from translation to Event-B

… but it has some similarities with UML

- Class Diagrams
- State Machine Diagrams

Translator generates Event-B automatically

- Into the same machine (generated = read only)
- Can also write standard Event-B in the same machine + events

What are the benefits?

Visualisation

- Helps understanding
- communication

Faster modelling

- One drawing node = several lines of B
- Extra information inferred from position (containment) of elements
- Experiment with different abstractions

finding useful abstractions is hard

Provides structuring constructs

- Hierarchical state-machines

  Event-B has no *event sequencing* mechanism
- Class Data Entity relationships, Lifted methods and state-machines

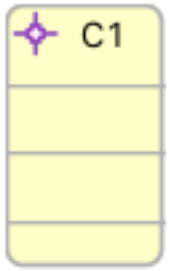
  Event-B has no *lifting* mechanism

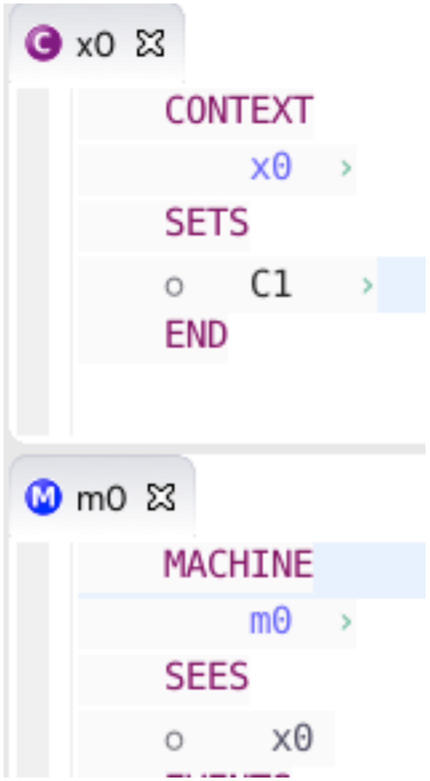
Class Diagrams

Class Diagrams provide a way to model Data as sets (classes) of objects.

Each class links to a Carrier Set, Constant or Variable in the Event-B

Linking to an existing data element allows more flexibility.
A button enables data to be created from the class diagram





x0

```
CONTEXT
     x0
SETS
  o     C1
END
```

m0

```
MACHINE
     m0
SEES
  o     x0
```

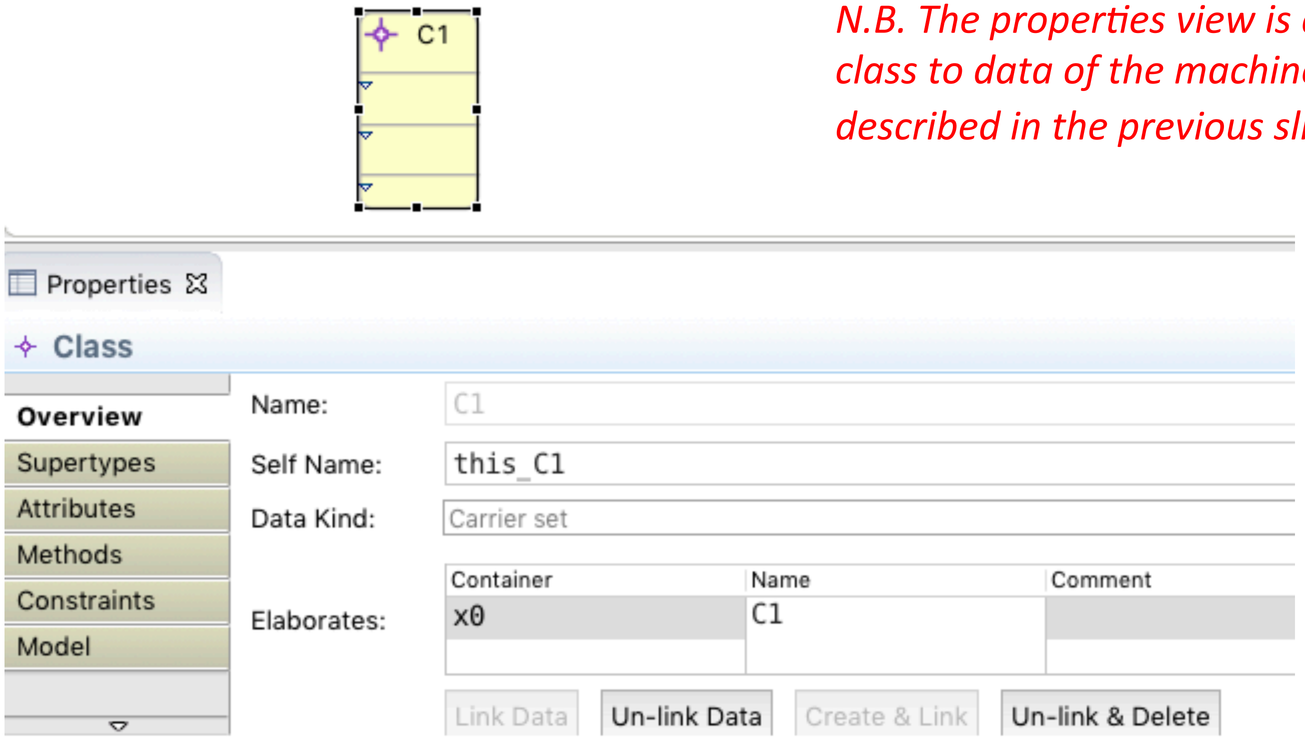
Class C1 is linked to the carrier set C1
in context x0 seen by machine m0
(The class diagram is in m0)

Self Name

Each class has a *self name*. This is an identifier that refers to an example instance of the class.

The default self name is *this_<classname>.* It can be changed in the properties view after selecting the class.

N.B. The properties view is also used to link the class to data of the machine or context as described in the previous slide.



Name: C1

Self Name: this_C1

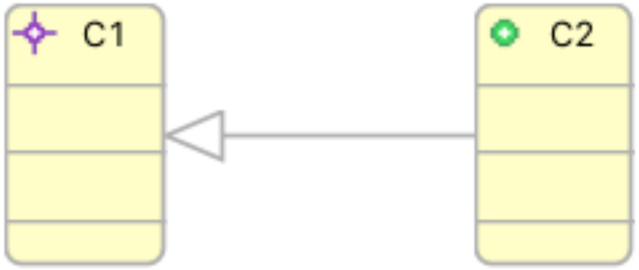
Data Kind: Carrier set

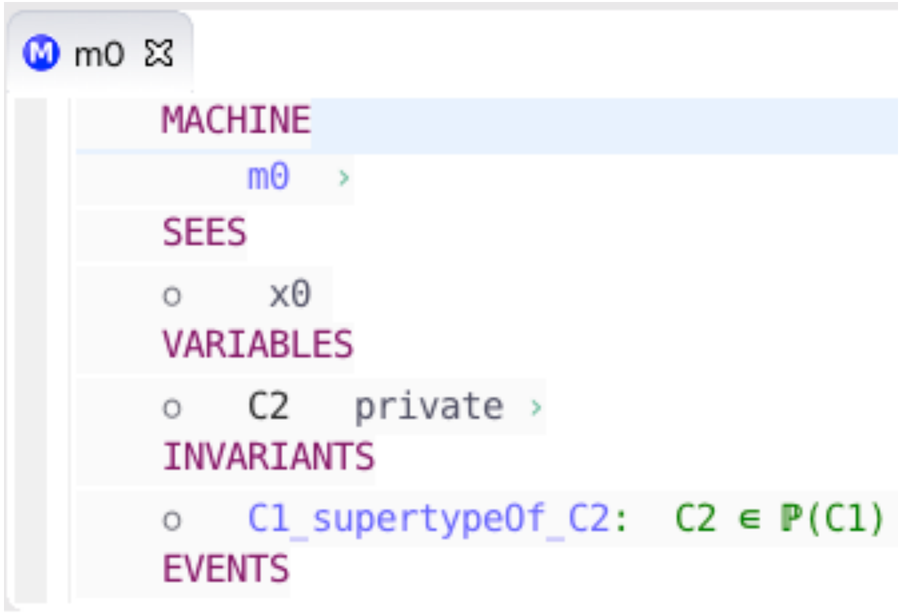
Elaborates:

Container	Name	Comment
x0	C1	

Supertype

Class Supertype relationships generate an axiom or invariant.





```
MACHINE
      m0
SEES
  o     x0
VARIABLES
  o     C2     private
INVARIANTS
  o     C1_supertypeOf_C2:   C2 ∈ ℙ(C1)
EVENTS
```

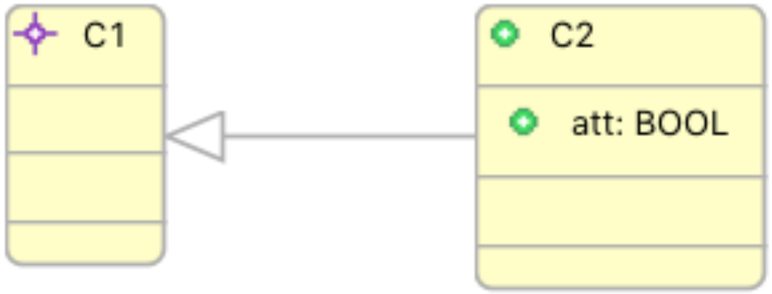
Class C2 is linked to a variable
C2 has C1 as its supertype
This generates an invariant in m0 which types C2

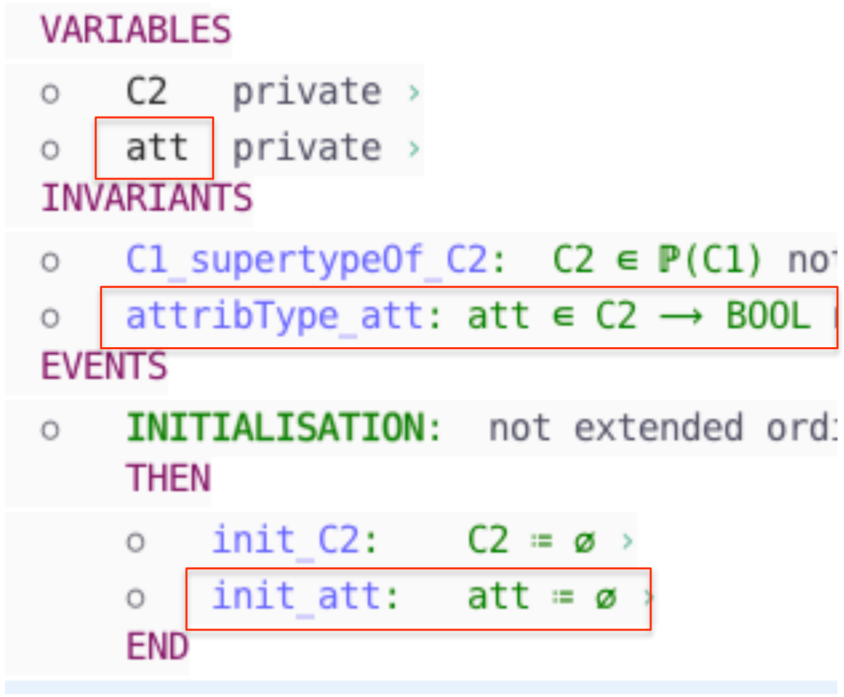
(n.b. subtypes are NOT assumed to be disjoint)

Attribute

A class (i.e. each of its objects) may have attributes

The attribute has a type which may be any valid Event-B (sub)set





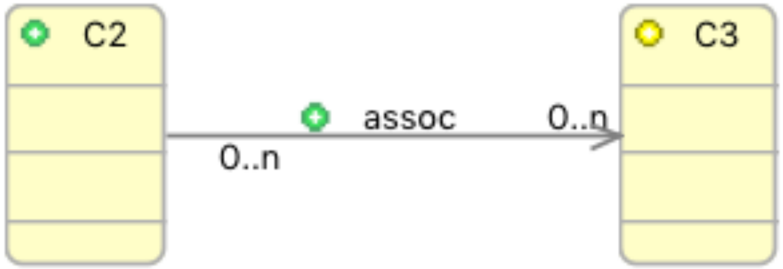
```
VARIABLES
  o   C2    private ›
  o   att   private ›
INVARIANTS
  o   C1_supertypeOf_C2:  C2 ∈ ℙ(C1) no
  o   attribType_att: att ∈ C2 ⟶ BOOL
EVENTS
  o   INITIALISATION:  not extended ord
      THEN
        o   init_C2:     C2 ≔ ∅ ›
        o   init_att:    att ≔ ∅ ›
      END
```

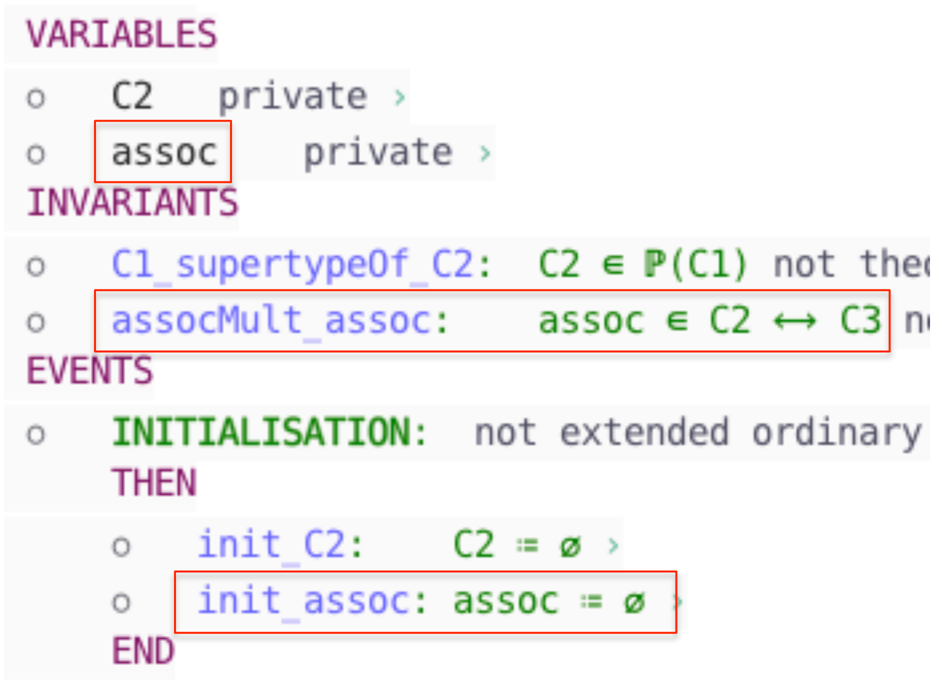
The attribute links to a variable (or constant) which is a relation between class instances and values from the attributes type.

(for attributes we usually make the relation a *total function*)

Association

A class may have associations with other classes





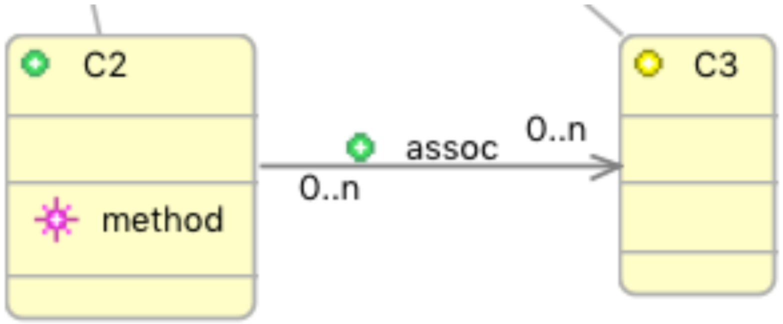
```
VARIABLES
  o   C2    private ›
  o   assoc    private ›
INVARIANTS
  o   C1_supertypeOf_C2:   C2 ∈ ℙ(C1) not thed
  o   assocMult_assoc:     assoc ∈ C2 ↔ C3 n
EVENTS
  o   INITIALISATION:  not extended ordinary
      THEN
        o   init_C2:     C2 ≔ ∅ ›
        o   init_assoc: assoc ≔ ∅ ›
      END
```

The association links to a variable (or constant) which is a relation between class instances and target class instances.

(the cardinalities determine the kind of relation: surjective injective, functional etc.)

Methods

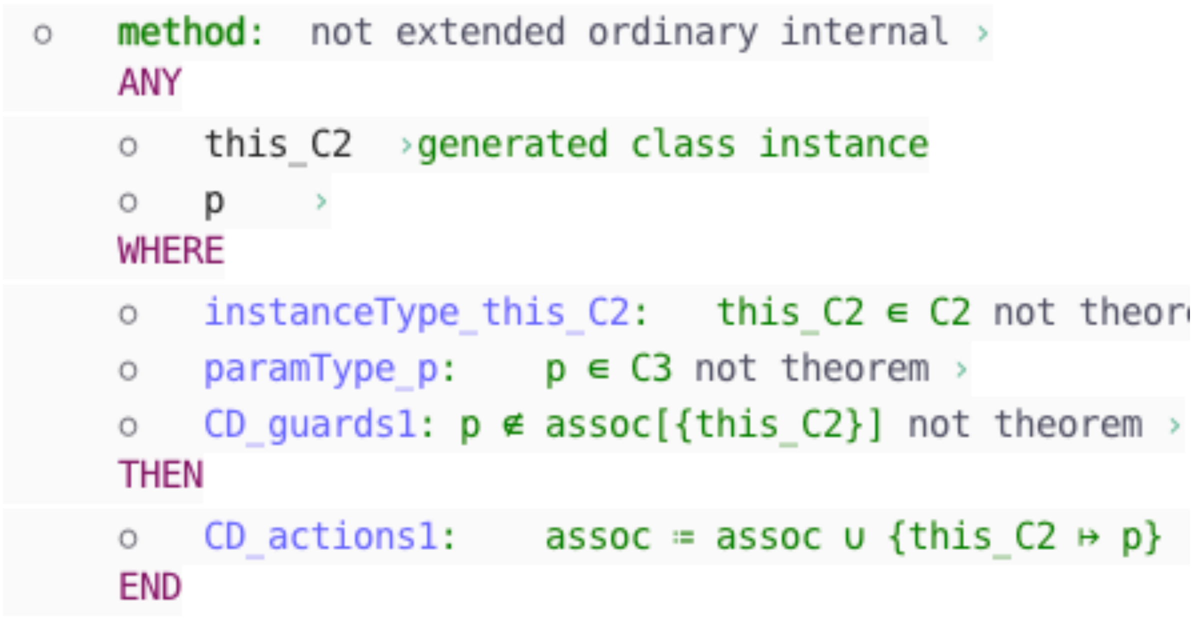
A class may have methods



The method links to an event in the machine.

The event automatically gets a parameter for the class instance.

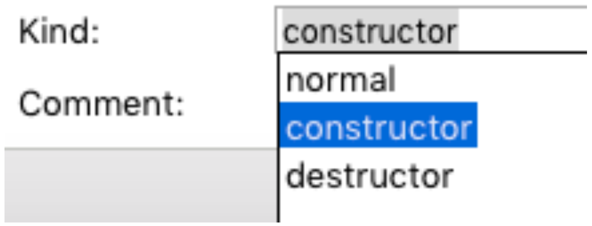
Other parameters, guards and actions can be added to the method (using the properties view)

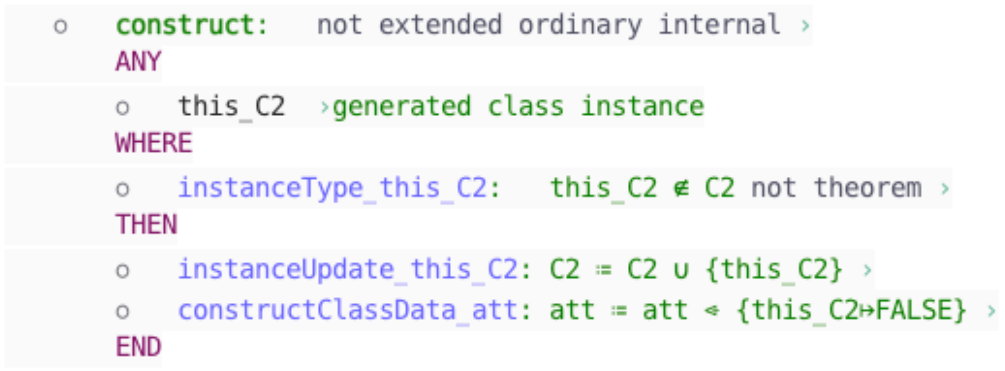


```
o     method:  not extended ordinary internal ›
      ANY
        o    this_C2   ›generated class instance
        o    p      ›
      WHERE
        o    instanceType_this_C2:    this_C2 ∈ C2 not theor
        o    paramType_p:      p ∈ C3 not theorem ›
        o    CD_guards1: p ∉ assoc[{this_C2}] not theorem ›
      THEN
        o    CD_actions1:      assoc ≔ assoc ∪ {this_C2 ↦ p}
      END
```

Constructors and Destructors

A class method can be made a constructor or destructor by setting the kind parameter in the properties view. (The class must link to a variable set of instances)

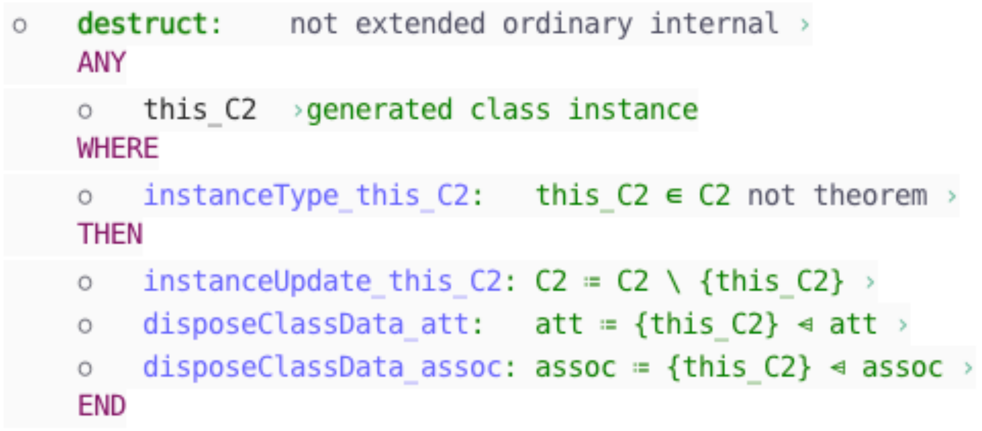




```
o   construct:   not extended ordinary internal ›
ANY
    o   this_C2   ›generated class instance
WHERE
    o   instanceType_this_C2:   this_C2 ∉ C2 not theorem ›
THEN
    o   instanceUpdate_this_C2: C2 ≔ C2 ∪ {this_C2} ›
    o   constructClassData_att: att ≔ att  {this_C2↦FALSE} ›
END
```

For a constructor, an unused instance is added to the class.

Any attributes are initialised for the instance



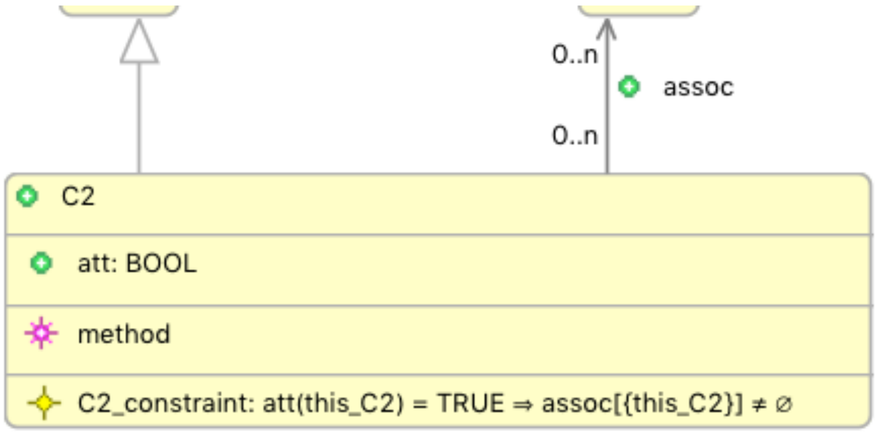
```
o   destruct:   not extended ordinary internal ›
ANY
    o   this_C2   ›generated class instance
WHERE
    o   instanceType_this_C2:   this_C2 ∈ C2 not theorem ›
THEN
    o   instanceUpdate_this_C2: C2 ≔ C2 \ {this_C2} ›
    o   disposeClassData_att:   att ≔ {this_C2} ⩤ att ›
    o   disposeClassData_assoc: assoc ≔ {this_C2} ⩤ assoc ›
END
```

For a destructor, a used instance is removed from the class.

Any attributes are also removed for the instance

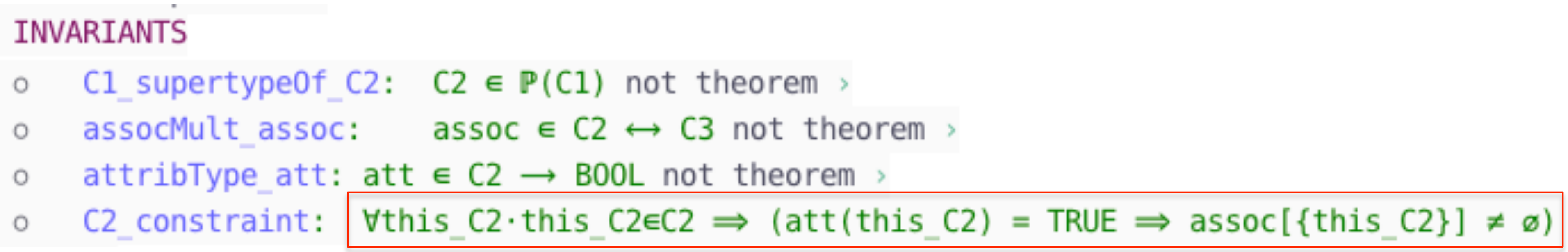
Constraints

A class may contain constraints



The constraint generates an invariant in the machine*

The generation adds an antecedent that universally quantifies the constraint for all class instances. (This quantification is implicit in the class diagram)



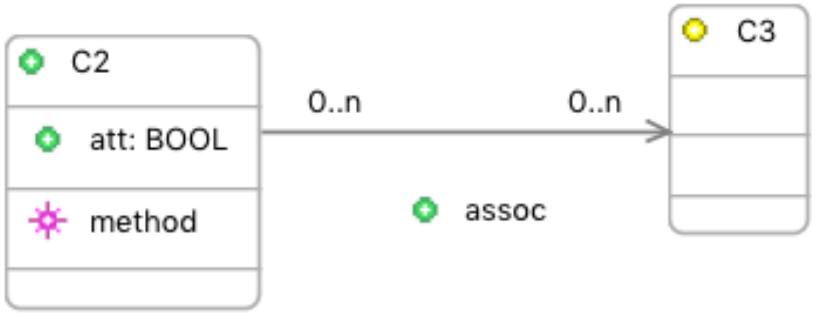
```
INVARIANTS
o   C1_supertypeOf_C2:  C2 ∈ ℙ(C1) not theorem ›
o   assocMult_assoc:    assoc ∈ C2 ↔ C3 not theorem ›
o   attribType_att: att ∈ C2 → BOOL not theorem ›
o   C2_constraint: ∀this_C2·this_C2∈C2 ⇒ (att(this_C2) = TRUE ⇒ assoc[{this_C2}] ≠ ∅)
```

* an axiom if the class diagram is contained in a context

Refinement

When a machine with a class diagram is refined,

a refined class diagram is created in the refined machine.



It contains a *refined class* instead of each original class. They do not generate any invariants but can be used in for new modelling (either adding to their content or as targets in association and supertype).

Attributes and associations are retained as *inherited*. They do not generate any invariants.

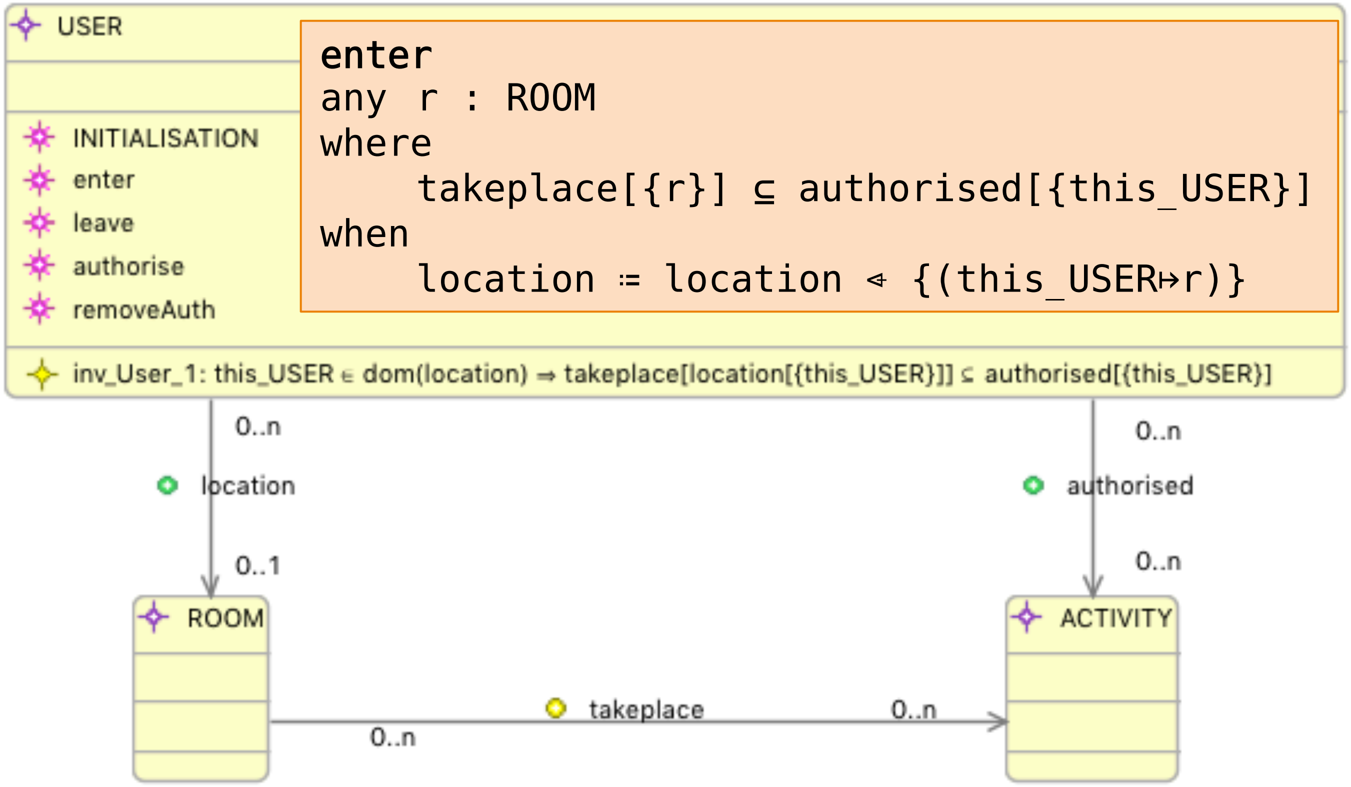
Methods are retained as refined versions linking to the corresponding refined events providing a basis for refinement.

Example – Access Control

1) Activities take place in Rooms

2) Users are authorised to carry out activities

3) Users may enter Rooms only if they are authorised to carry out all the activities that take place in that room



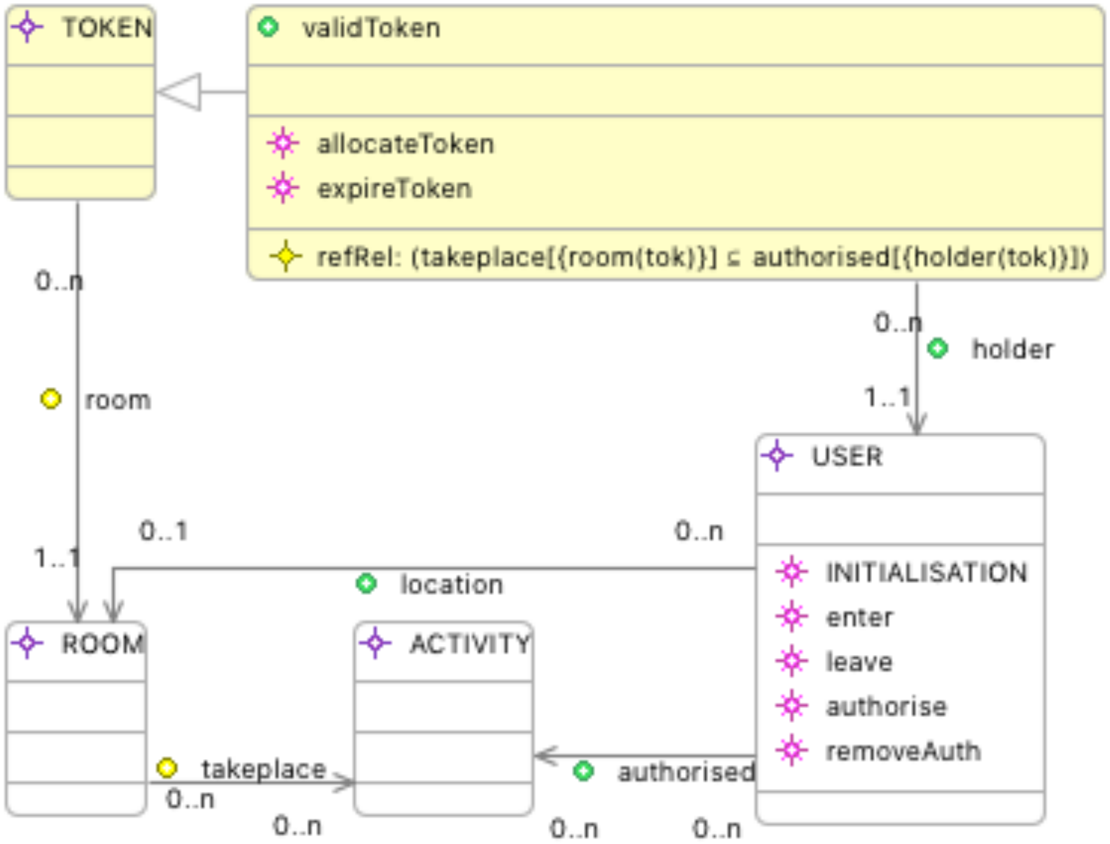
```
enter
any r : ROOM
where
    takeplace[{r}] ⊆ authorised[{this_USER}]
when
    location ≔ location ⩤ {(this_USER↦r)}
```

Example – Access Control Refinement

4) Users are allocated Tokens *(allocateToken is a constructor and sets holder)*

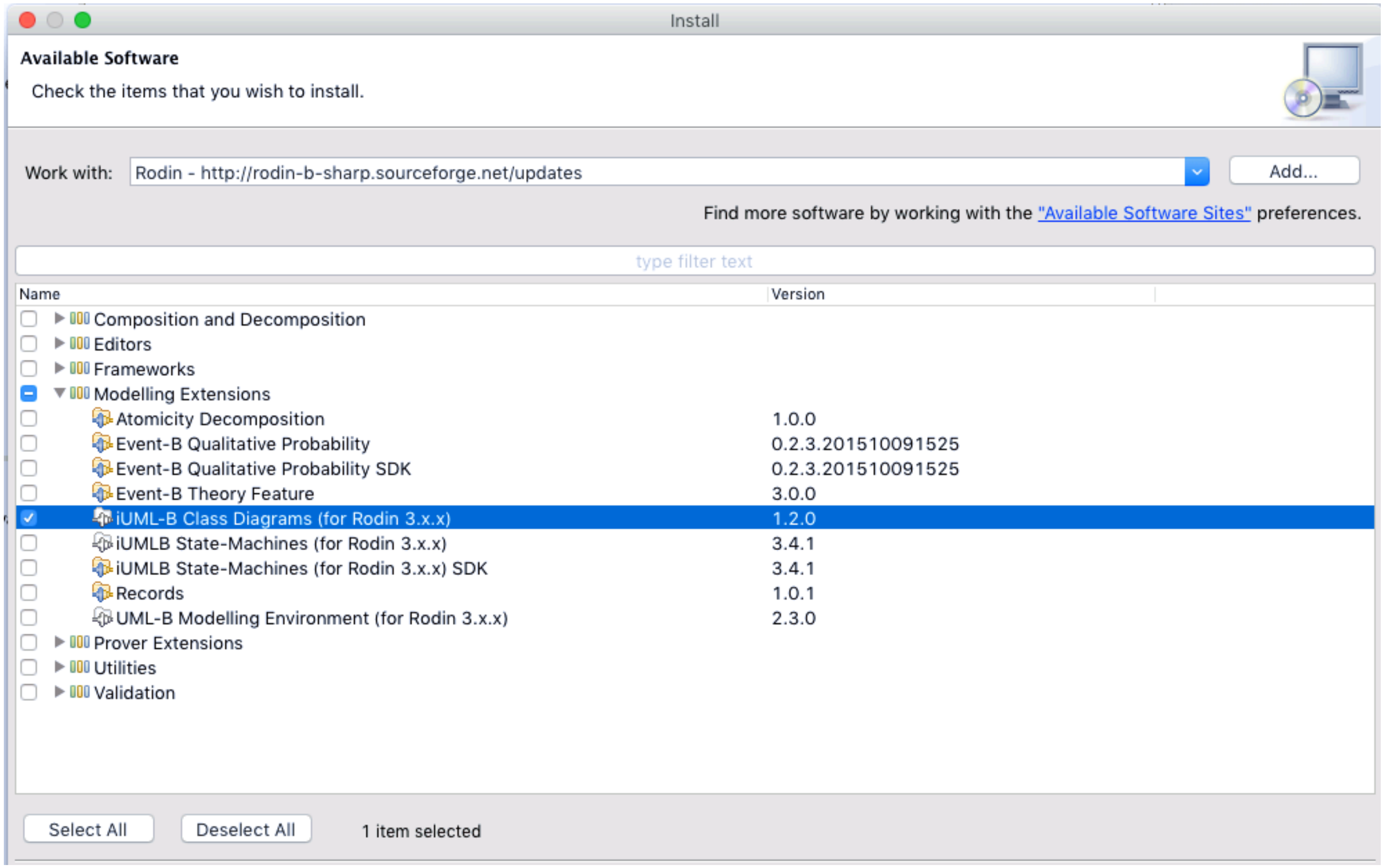
5) Holding a Token enables access to Rooms

6) Tokens expire *(expireToken is a destructor)*



Installation

Help – Install New Software…



Install

Available Software

Check the items that you wish to install.

Work with: Rodin - http://rodin-b-sharp.sourceforge.net/updates

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
☐ ▶ Composition and Decomposition	
☐ ▶ Editors	
☐ ▶ Frameworks	
▬ ▼ Modelling Extensions	
☐ Atomicity Decomposition	1.0.0
☐ Event-B Qualitative Probability	0.2.3.201510091525
☐ Event-B Qualitative Probability SDK	0.2.3.201510091525
☐ Event-B Theory Feature	3.0.0
✓ iUML-B Class Diagrams (for Rodin 3.x.x)	1.2.0
☐ iUMLB State-Machines (for Rodin 3.x.x)	3.4.1
☐ iUMLB State-Machines (for Rodin 3.x.x) SDK	3.4.1
☐ Records	1.0.1
☐ UML-B Modelling Environment (for Rodin 3.x.x)	2.3.0
☐ ▶ Prover Extensions	
☐ ▶ Utilities	
☐ ▶ Validation	

1 item selected

Summary

Class Diagrams for modelling data relationships

Classes – sets (Carrier Sets, Constants Variables)

  self name

Supertypes - subsets

Attributes – relations (usually functions)

Associations - relations

Methods – events

  constructors and destructors

Constraints – invariants or Axioms

Refined Classes