Guide for starting on Event-B

1 Installation

Event-B method is supported by Rodin platform. The installation of Rodin is described on the website: http://www.event-b.org/

- 

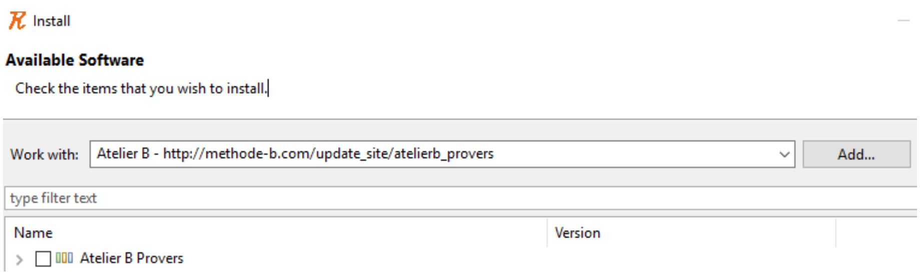
Click on “Install Rodin”
- Instructions for Rodin and plug ins installation are given then:

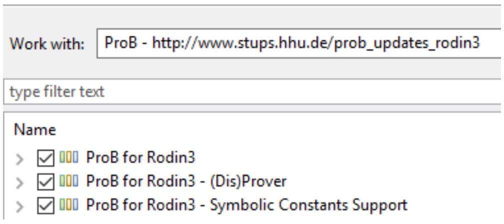
Rodin Platform and Plug-in Installation

Name	Installation
Rodin platform	• Requires Java 1.6 • Download the Core: Rodin Platform file for your platform. To install, just unpack the archive anywhere on your hard-disk and launch the "rodin" executable in it. • Start Rodin • Information on the latest release.
Plug-ins	• Plug-ins are installed from within Rodin by selecting Help/Install New Software. Then select the appropriate update site from the list of download sites. • Details on plug-ins. • Install the Atelier B Provers plugin from the Atelier B Provers Update site to take full advantage of Rodin proof capabilities • Install the ProB plugin from the ProB Update site for powerful model checking and animation
User manual and Tutorial	• Rodin Handbook

WARNING

- Use the version Rodin 3.4
- Atelier B - http://methode-b.com/update_site/atelierb_provers
- ProB - http://www.stups.hhu.de/prob_updates_rodin3

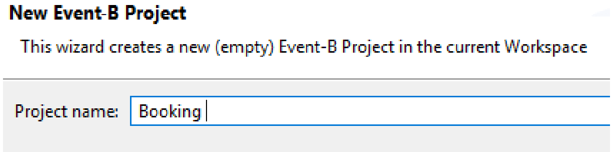




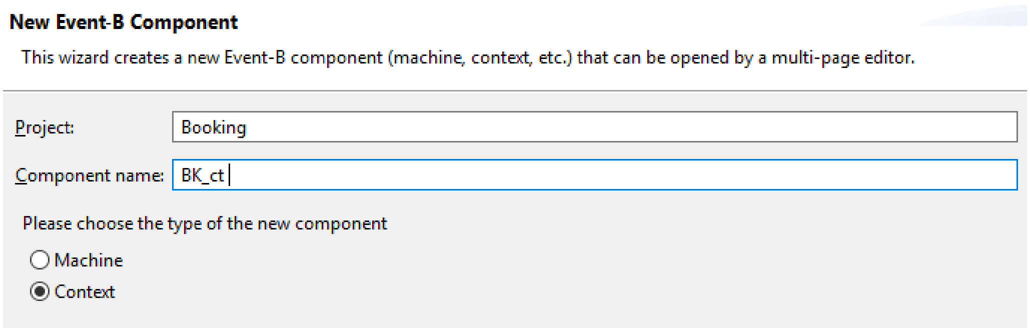
2 First steps with Event-B

After installing Rodin, a "Welcome" page is displayed. First, close this page in order to display the Event-B explorer window on the left-hand side.

- To create a new Event-B project, select into Event-B Explorer the icon so as to open a wizard. Otherwise, open the main menu bar and choose "File ⇒ New ⇒ Event-B Project". As a result, a text box is popped up so as to enter the first project name, "Booking" for instance.



- In the explorer, select the new project folder, then click on the icon to add Event-B component (you can also right-click on the folder and select "New ⇒ Event-B Component"). To create the first context, enter its name and choose the context type.



To open the component with Event-B editor, right-click on the context and select "Open With ⇒ Event-B Context Editor".

Similarly, create a new machine and open it.

- At the bottom of the editor, select "Edit" tab to add useful data for your model: constants, axioms, variables… You can also use the wizards under the menu bar:

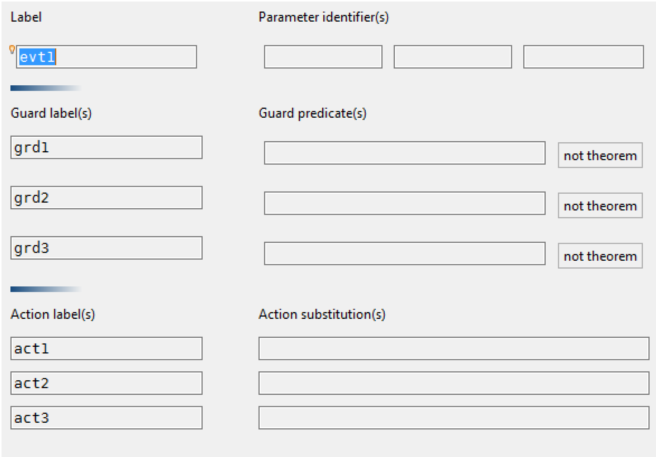
new Carrier Set (to give the name of an abstract set)

Enumerated Set (to give name of the set and a list of elements)

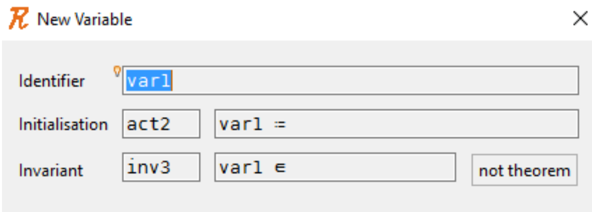
new Constant (to give constant name and its defined axiom)

Axiom (to define axioms)

Event



Variable



Invariant (to define invariants)

3 Exercise #1

Model the Booking system.

In the Context component:

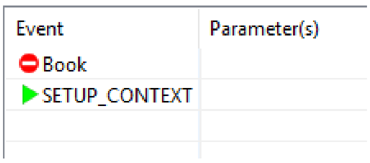
- Constant: max_seat
- Axioms: max_seat is a natural number; its fixed value is 350

In the Machine component:

- Variable: free_seat
- Invariants: free_seat is a natural number; it is lower or equal to max_seat
- Events:
    - free_seat is initialized to max_seat
    - Book event: parameter n, which is a natural number; its action is to decrease free_seat by n

Check the PO rules by clicking on the “Proving” perspective, at the top right .

To simulate your model with ProB, right-click on the machine, then select “Start Animation / Model Checking”. In the explorer, double-click on the green icon to run it step by step.



Can we implement a DeadLock Freedom rule? Explain and justify.

4 Example: abstract & refined models

Let consider an abstract model of “primary/secondary” redundancy mechanism.

- Weak property: primary component active and failed first, then secondary component could fail
- Invariant inv3: must not get secondary component failed when primary component ok
- Invariant inv4: must not lose both components

```
MACHINE
   CoSSAP_0
VARIABLES
   s_C1
   s_C2
```

```
INVARIANTS
   inv1   :   s_C1 ∈ {0,1,2}
   inv2   :   s_C2 ∈ {0,1,2}
   inv3   :   ¬(s_C1 = 0 ∧ s_C2 ≠ 0)
   inv4   :   ¬(s_C1 = 2 ∧ s_C2 = 2)
```

```
INITIALISATION   ≙
STATUS
  ordinary
BEGIN
  act1   :   s_C1 ≔ 0
  act2   :   s_C2 ≔ 0
END
```

```
failed_C1   ≙
STATUS
  ordinary
WHEN
  grd1   :   s_C2 = 0
THEN
  act1   :   s_C1 :∈{1,2}
END
```

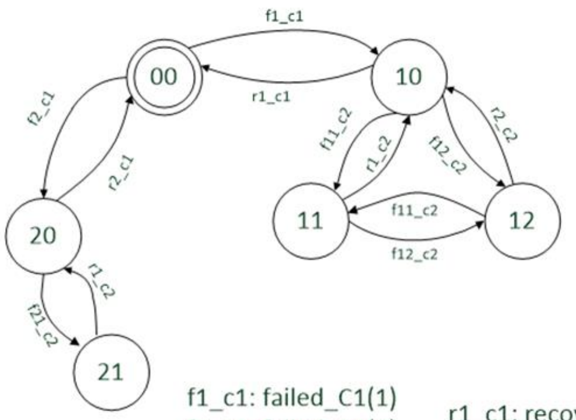
```
failed_C2_1   ≙
STATUS
  ordinary
WHEN
  grd1   :   s_C1 = 1
THEN
  act1   :   s_C2 :∈ {1,2}
END
```

```
failed_C2_2   ≙
STATUS
  ordinary
WHEN
  grd1   :   s_C1 = 2
THEN
  act1   :   s_C2 ≔ 1
END
```

```
recovered_C1   ≙
STATUS
  ordinary
WHEN
  grd1   :   s_C1 ≠ 0
  grd2   :   s_C2 = 0
THEN
  act1   :   s_C1 ≔ 0
END
```

```
recovered_C2   ≙
STATUS
  ordinary
WHEN
  grd1   :   s_C2 ≠ 0
THEN
  act1   :   s_C2 ≔ 0
END
```

- Graph model: a trace gives the finite sequence of states from initial state



f1_c1: failed_C1(1)
f2_c1: failed_C1(2)
f11_c2: failed_C2_1(1)
f21_c2: failed_C2_2(1)
f12_c2: failed_C2_1(2)

r1_c1: recovered_C1(1)
r2_c1: recovered_C1(2)
r1_c2: recovered_C2(1)
r2_c2: recovered_C2(2)

- State-transition model: set notation

L = {(0, 0)}

f1_c1 = {(0, 0) → (1, 0)}
f2_c1 = {(0, 0) → (2, 0)}
f11_c2 = {(1, 0) → (1,1), (1, 2) → (1,1)}
f12_c2 = {(1, 0) → (1,2), (1, 1) → (1,2)}
f21_c2 = {(2, 0) → (2,1)}

r1_c1 = {(1, 0) → (0, 0)}
r2_c1 = {(2, 0) → (0, 0)}
r1_c2 = {(1, 1) → (1,0), (2, 1) → (2,0)}
r2_c2 = {(1, 2) → (1,0)}

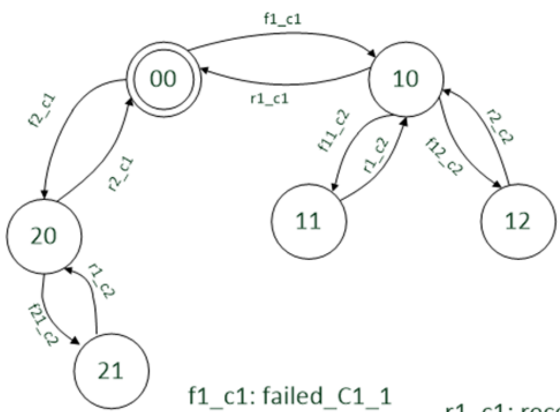
- Set of all abstract events

ae = {(0, 0) → (1, 0), (0, 0) → (2, 0), (1, 0) → (1,1), (1, 0) → (1,2),
(2, 0) → (2,1), (1, 2) → (1,1), (1, 1) → (1,2),
(1, 0) → (0, 0), (1, 1) → (1,0), (1, 2) → (1,0),
(2, 0) → (0, 0), (2, 1) → (2,0)}

- Refined model of "primary/secondary" redundancy mechanism

  - Strong property: less than 2 consecutive failures without recovery

State-transition graph model ?



f1_c1: failed_C1_1
f2_c1: failed_C1_2
f11_c2: failed_C2_1
f21_c2: failed_C2_2
f12_c2: failed_C2_12

r1_c1: recovered_C1(1)
r2_c1: recovered_C1(2)
r1_c2: recovered_C2(1)
r2_c2: recovered_C2(2)

- Strong property: less than 2 consecutive failures without recovery

```
failed_C1_1
STATUS
 ordinary
REFINES
 failed_C1
WHEN
 grd1   :   s_C2 = 0
THEN
 act1   :   s_C1 ≔ 1
END
```

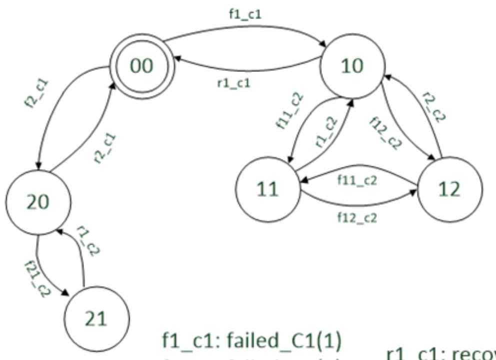
```
failed_C1_2
STATUS
 ordinary
REFINES
 failed_C1
WHEN
 grd1   :   s_C2 = 0
THEN
 act1   :   s_C1 ≔ 2
END
```

```
failed_C2_1
STATUS
 ordinary
REFINES
 failed_C2_1
WHEN
 grd1   :   s_C1 = 1
 grd2   :   s_C2 = 0
THEN
 act1   :   s_C2 ≔ 1
END
```

```
failed_C2_12
STATUS
 ordinary
REFINES
 failed_C2_1
WHEN
 grd1   :   s_C1 = 1
 grd2   :   s_C2 = 0
THEN
 act1   :   s_C2 ≔ 2
END
```

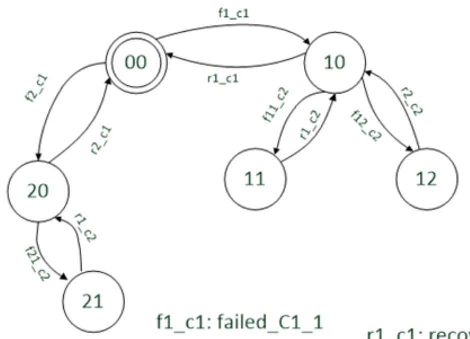
```
failed_C2_2
STATUS
 ordinary
REFINES
 failed_C2_2
WHEN
 grd1   :   s_C1 = 2
THEN
 act1   :   s_C2 ≔ 1
END
```

- Any trace of the refined model is included in the abstract model
- Refinement holds the abstract behavior



f1_c1: failed_C1(1)
f2_c1: failed_C1(2)
f11_c2: failed_C2_1(1)
f21_c2: failed_C2_2(1)
f12_c2: failed_C2_1(2)

r1_c1: recovered_C1(1)
r2_c1: recovered_C1(2)
r1_c2: recovered_C2(1)
r2_c2: recovered_C2(2)



f1_c1: failed_C1_1
f2_c1: failed_C1_2
f11_c2: failed_C2_1
f21_c2: failed_C2_2
f12_c2: failed_C2_12

r1_c1: recovered_C1(1)
r2_c1: recovered_C1(2)
r1_c2: recovered_C2(1)
r2_c2: recovered_C2(2)

- State-transition model: set notation

$L' = \{(0, 0)\}$

$f1\_c1 = \{(0, 0) \rightarrow (1, 0)\}$
$f2\_c1 = \{(0, 0) \rightarrow (2, 0)\}$
$f11\_c2 = \{(1, 0) \rightarrow (1,1)\}$
$f12\_c2 = \{(1, 0) \rightarrow (1,2)\}$
$f21\_c2 = \{(2, 0) \rightarrow (2,1)\}$

$r1\_c1 = \{(1, 0) \rightarrow (0, 0)\}$
$r2\_c1 = \{(2, 0) \rightarrow (0, 0)\}$
$r1\_c2 = \{(1, 1) \rightarrow (1,0), (2, 1) \rightarrow (2,0)\}$
$r2\_c2 = \{(1, 2) \rightarrow (1,0)\}$

- Set of all refined events

$$re = \{(0, 0) \rightarrow (1, 0), (0, 0) \rightarrow (2, 0), (1, 0) \rightarrow (1,1), (1, 0) \rightarrow (1,2), (2, 0) \rightarrow (2,1), (1, 0) \rightarrow (0, 0), (1, 1) \rightarrow (1,0), (1, 2) \rightarrow (1,0), (2, 0) \rightarrow (0, 0), (2, 1) \rightarrow (2,0)\}$$

Comparison

$L = \{(0, 0)\}$

$ae = \{(0, 0) \rightarrow (1, 0), (0, 0) \rightarrow (2, 0), (1, 0) \rightarrow (1,1), (1, 0) \rightarrow (1,2),$
$(2, 0) \rightarrow (2,1), (1, 2) \rightarrow (1,1), (1, 1) \rightarrow (1,2),$
$(1, 0) \rightarrow (0, 0), (1, 1) \rightarrow (1,0), (1, 2) \rightarrow (1,0),$
$(2, 0) \rightarrow (0, 0), (2, 1) \rightarrow (2,0)\}$

$L' = \{(0, 0)\}$

$re = \{(0, 0) \rightarrow (1, 0), (0, 0) \rightarrow (2, 0), (1, 0) \rightarrow (1,1), (1, 0) \rightarrow (1,2),$
$(2, 0) \rightarrow (2,1),$
$(1, 0) \rightarrow (0, 0), (1, 1) \rightarrow (1,0), (1, 2) \rightarrow (1,0),$
$(2, 0) \rightarrow (0, 0), (2, 1) \rightarrow (2,0)\}$

Sufficient conditions for refinement →

$L' \subseteq L$
$L \neq \emptyset$
$re \subseteq ae$
$dom(ae) \subseteq dom(re)$