UNIVERSITY OF Southampton

Michael Butler
mjb@ecs.soton.ac.uk
Dana Dghaym
D.Dghaym@ecs.soton.ac.uk
Thai Son Hoang
T.S.Hoang@ecs.soton.ac.uk

COMP1216. Software Modelling and Design (2019-20)

Solution Sheet 6: Modelling with Relations

Issue date: 9 March 2020

This Problem Class relates to the following simple system. You can use **Rodin** to create the model. The intruction on installing *Rodin* can be found in the following link https://secure.ecs.soton.ac.uk/noteswiki/w/COMP1216#Rodin_installation. More information about *Rodin* can be found in the Rodin Handbook at https://www3.hhu.de/stups/handbook/rodin/current/html/index.html.

System Descriptions

In a university degree programme, students are registered on courses. Students must be enrolled on the degree programme to be registered in a course. There is no need to consider multiple degree programmes - just assume we are modelling a single degree programme.

- Specify an Event-B context for this system declaring the types STUDENT and COURSE
- Define Event-B variables and invariants that represent the enrolled students in the degree programme, courses available for the degree programme, and also the courses that students are registered in. Ensure the invariants are sufficiently strong.
- Include an event for enrolling a student on the programme.
- Include an event for adding a course to the programme.
- Include an event for registering a student in a course. Ensure the guards are sufficiently strong.
- Include an event for de-enrolling a student from the degree programme.

- Include an event for removing a course from the degree programme.

Solution:
The static part of the model is specified in the context as follows.

```
/*
 * This context models the static part of the model
 * by declaring the set of students and courses
 *
 * @author: htson
 */
context c0
sets
 STUDENT // The set of students (both registered and unregistered)
 COURSE // The set of courses
end
```

The dynamic part of the model is specfied in the machine as follows.

```
/*
 * This machine specifies the dynamic part of the model.
 * It models the set of enrolled students and their registered course.
 *
 * @author: htson
 */
machine m0
sees c0 // See the context to get access to the carrier sets STUDENT and COURSE.
variables
 students // The set of enrolled students.
 courses // The set of courses available for registration
 register // The registration information for the students.
invariants
 @typeof-students: students ⊆ STUDENT
 @typeof-courses: courses ⊆ COURSE

 /*
  * Only enrolled students can register for courses.
  * Only courses for the programme can be registered.
  * Students can register for zero or more courses.
  * A course can have zero or more registered students.
  *
  * NOTE: The usage of ↔ for many-to-many relations.
  */
 @typeof-register: register ∈ students ↔ courses
events
 /*
  * The initialisation.
  */
 event INITIALISATION
 then
  @init-students: students := ∅ // Initially, there are no enrolled students.
  @init-course: courses := ∅ // Initially, there are no courses for the programme.
  @init-register: register := ∅ // Initially, there are no registration.
 end
```

```
36
37  /*
38   * Enroll a student @s to the programme.
39   */
40  event Enroll
41  any
42   s // The student to enroll.
43  where
44   @grd1: s ∉ students // @s is not yet enrolled.
45  then
46   @act1: students := students ∪ {s} // @s is now enrolled.
47  end
48
49  /*
50   * Add a new course @c to the programme.
51   */
52  event AddCourse
53  any
54   c // The course to be added.
55  where
56   @grd1: c ∉ courses // @c is not yet part of the programme.
57  then
58   @act1: courses := courses ∪ {c} // @c is not part of the programme.
59  end
60
61  /*
62   * Register student @s to a course @c.
63   */
64  event Register
65  any
66   s // The student to register.
67   c // The course that the student registers.
68  where
69   @grd1: s ∈ students // @s is a enrolled student.
70   @grd2: c ∈ courses // @c is an existing course.
71   @grd3: s ↦ c ∉ register // @s is not yet registed for @c.
72  then
73   // NOTE: The usage of set union for extending a relation.
74   @act1: register := register ∪ {s ↦ c} // @s is now registered for @c.
75  end
76
77  /*
78   * De−enroll a student @s from the programme.
79   */
80  event Deenroll
81  any
82   s // The student to de−enroll.
83  where
84   @grd1: s ∈ students // @s is an enrolled student.
85  then
86   @act1: students := students \ {s} // @s is now de−enrolled.
87
88   // NOTE: The usage of domain restriction for removing information from a relation.
89   @act2: register := {s} ⩤ register // Remove all registrations for @s.
90  end
91
```

```
92  /*
93   * Remove an existing course @c from the programme.
94   */
95  event RemoveCourse
96  any
97   c // The course to be removed.
98  where
99   @grd1: c ∈ courses // @c is an existing course.
100 then
101  @act1: courses := courses \ {c} // Remove @c from the set of existing courses.
102
103  // NOTE: The usage of range restriction for removing information from a relation.
104  @act2: register := register ⩥ {c} // Remove all registrations for @c.
105 end
106
107 end
```