Lecture 6 Part 1.

REFINEMENT IN EVENT-B

Event-B (reminder)

- Event-B models are organised in terms of two basic constructs: *contexts* and *machines*:
  - contexts specify the static part of a model;
  - machines specify the dynamic part.
- The role of the contexts is to isolate the parameters of a formal model and their properties, which are assumed to hold for all instances.
- A machine encapsulates a transition system with the state specified by a set of variables and transitions modelled by a set of guarded events.

Model development with Event-B

- Event-B allows models to be developed gradually via mechanisms such as ***context extension*** and ***machine refinement***.
- These techniques enable users to develop target systems from their abstract specifications, and subsequently introduce more implementation details.
- More importantly, properties that are proved at the abstract level are maintained through refinement, and hence are also guaranteed to be satisfied by later refinements.
- As a result, correctness proofs of systems are broken down and distributed amongst different levels of abstraction, which is easier to manage.

A course management system: Requirements

- A club has some fixed *members*; amongst them are *instructors* and *participants*.
- A member can be both an instructor and a participant.

REQ1	**Instructors are members of the club.**

REQ2	**Participants are members of the club.**

A course management system (cont.)

- There are predefined *courses* that can be offered by a club.
- Each course is associated with exactly one fixed instructor.

REQ3	**There are predefined courses.**

REQ4	**Each course is assigned to one fixed instructor.**

A course management system (cont.)

A course is either *opened* or *closed* and is managed by the system.

REQ5	**A course is either *opened* or *closed*.**

REQ6	**The system allows a closed course to be opened.**

REQ7	**The system allows an opened course to be closed.**

A course management system (cont.)

The number of opened courses is limited.

REQ8	**The number of opened courses cannot exceed a given limit.**

Only when a course is opened, can participants *register* for the course. An important constraint for registration is that an instructor cannot attend his own courses.

REQ9	**Participants can only register for an open course.**

REQ10	**Instructors cannot attend their own courses.**

A course management system: development with Event-B

- Next, we will develop a formal model based on the above requirements document:
  - we will refer to the above requirements in order to justify how they are formalised in the Event-B model.
- In the initial model, we focus on opening and closing of courses by the system.
- We start our modelling with defining a context **Courses_c0.**

A course management system: Context

CONTEXT Courses_c0
SETS *COURSES* // a carrier set *COURCES* denoting the set of courses that can be offered by the club (REQ3)
CONSTANTS m // REQ8: the maximum number of courses that the club can open
AXIOMS

axm0_1: **finite**(*COURSES*)

axm0_2: $m \in \mathbb{N}$

axm0_3: $m > 0$

axm0_4: **card**(*COURSES*) $\geq m$ // number of all possible courses is no less than m

END

A course management system: Machine

- We develop machine **Courses_m0** of the initial model, focusing on courses opening and closing.
  - This machine sees context **Courses_c0** developed before.
- We model the set of opened courses by a variable *courses*

```
MACHINE Courses_m0
SEES Courses_c0
VARIABLES  courses   // The machine state is represented by the variable, courses,
                               denoting the open courses
 INVARIANTS
          inv0_1: courses ⊆ COURCES  // open courses is a subset of all available courses
          inv0_2: card(courses) ≤ m
EVENTS
INITIALISATION ≙
          then
             act1: courses:= ∅  // Initially, all courses are closed
                                 (so, the set of opened courses is set to the empty set);
         end
…
```

A course management system: Machine

- We model the opening and closing of courses using two events **OPENCOURSE** and **CLOSECOURSE** as follows:

OPENCOURSE $\triangleq$ // **REQ6:** The system allows a closed course to be opened.
any *crs*
where
grd1: **card**(*courses*) $<$ *m* // the current number of opened courses has not yet reached the limit
grd2: $crs \notin courses$ // a course *crs* is not opened yet
then
act1: $courses := courses \cup \{crs\}$ // add *crs* course to the set *courses*
end

CLOSECOURSE $\triangleq$ // **REQ7:** The system allows an opened course to be closed.
any *crs*
where
grd1: $crs \in courses$ // the course crs has been opened before
then
act1: $courses := courses \setminus \{crs\}$ // remove *crs* from the set *courses*
end

Context Extension

- Context extension is a mechanism for introducing more static details into an Event-B development.
  - A context can *extend* one or more contexts.
- When describing a context D as extending another context C, we call C and D the abstract and concrete context, respectively.
- By extending C, D “inherits” all the abstract elements of C, i.e., carrier sets, constants, axioms and theorems.

Course Management System:

We extend context **Courses_c0** by the context **Members_c1**

```
CONTEXT Members_c1 EXTENDS Courses_c0
SETS  MEMBERS            // a carrier set MEMBERS represents the set of club members

CONSTANTS PARTICIPANTS  // constant PARTICIPANTS denotes the set of participants
          INSTRUCTORS   // constant INSTRUCTORS denotes the set of instructors
          courseInstructor // constant models a relationship between courses and instructors
AXIOMS
    axm1_1: finite(MEMBERS)
    axm1_2: PARTICIPANTS ⊆ MEMBERS // participants must be members of the club
    axm1_3: INSTRUCTORS ⊆ MEMBERS  // instructors must be members of the club
    axm1_4: courseInstructor ∈ COURSES → INSTRUCTORS // a total function from COURSES to
                                         INSTRUCTORS (thus we formalise REQ4)
END
```

Machine Refinement

- *Machine refinement* is a mechanism for introducing details about the dynamic properties of a model
  - When speaking about machine N refining another machine M, we refer to M as the *abstract* machine and to N as the *concrete* machine.
- Two kinds of refinement: ***superposition refinement*** and ***data refinement***
  - In superposition refinement, the abstract variables of M are retained in the concrete machine N, with possibly some additional concrete variables.
  - In data refinement, the abstract variables *v* are replaced by concrete variables *w* and, subsequently, the connections between M and N are represented by the relationship between *v* and *w*.
  - More often, Event- B refinement is a mixture of both superposition and data refinement: some of the abstract variables are retained, while others are replaced by new concrete variables.

Superposition Refinement

- In superposition refinement, variables *v* of the abstract machine M are kept in the refinement, i.e. as part of the state of N.
- N can have some additional variables *w*.
- The concrete invariants ***J*(*v*,*w*)** specify the relationship between the old and new variables.
- Each abstract event e is refined by a concrete event f
- Assume that the abstract event e and the concrete event f are as follows:

e = any *x* where *G*(*x*, *v*) then *Q*(*x*, *v*) end

f = any *x* where *H*(*x*,*v*,*w*) then *R*(*x*,*v*,*w*) end

- **f refines *e*** if the guard of f is stronger than that of e (*guard strengthening*), concrete invariants *J* are maintained by f, and abstract action *Q* simulates the concrete action *R* (*simulation*).

Superposition Refinement

- In the course of refinement, *new events* are often introduced into a model.
- Lets go back to our *Course Management System*…

Refinement of a machine **Courses_m0**

MACHINE Courses_m0
SEES Courses_c0
VARIABLES *courses*
INVARIANTS
inv0_1: $courses \subseteq COURCES$
inv0_2: $\mathbf{card}(courses) \leq m$
EVENTS
INITIALISATION ≙ …
OPENCOURSE ≙ …
CLOSECOURSE ≙ …

we had before ….

Courses_m0 machine is refined by a machine **Members_m1**

MACHINE Members_m1
REFINES Courses_m0
SEES Members_c1
VARIABLES *courses participants*
INVARIANTS
inv1_1: $participants \in courses \leftrightarrow PARTICIPANTS$
inv1_2: $\forall c.\ c \in courses \Rightarrow courseInstructor(c) \notin participants[\{c\}]$
EVENTS
INITIALISATION ≙
then
…
act2: $participants := \emptyset$ *// The variable is initialised to the empty set.*
end

- New variable ***participants*** representing information about course participants (modelled as a relation between the sets of open courses ***courses*** *and the set PARTICIPANTS*)
- Invariant ***inv1_2:*** $\forall c.\ c \in courses \Rightarrow courseInstructor(c) \notin participants[\{c\}]$ states that *"for every opened course c, the instructor of this course is not amongst its participants"* ***(REQ10)***

Modelling machine **Members_m1**

The original abstract event **OPENCOURSE** stays unchanged in this refinement, while an additional assignment is added to **CLOSECOURSE** to update ***participants*** by removing the information about a closing course *crs* from it.

OPENCOURSE refines **OPENCOURSE** $\triangleq$ // no changes in this event

  any *crs*

  where

    grd1: **card**(*courses*) $<$ *m*

    grd2: *crs* $\notin$ *courses*

  then

    act1: *courses* := *courses* $\cup$ {*crs*}

  end

CLOSECOURSE refines **CLOSECOURSE** $\triangleq$ // we add in to the event an additional action

  any *crs*

  where

    grd1: *crs* $\in$ *courses*

  then

    act1: *courses* := *courses* $\setminus$ {*crs*}

    act2: *participants* := {*crs*} $\lhd$ *participants* // *removing all the relationships between this course and its participants.*

end

Machine **Members_m1**

- A new event **REGISTER** is added. It models the registration of a participant *p* for an opened course *c*.
- The guard of the event ensures that *p* is not the instructor of the course (***grd1_3***) and is not yet registered for the course (***grd1_4***).
- The action of the event updates ***participants*** accordingly by adding the mapping $\boldsymbol{c \mapsto p}$ to it.

```
REGISTER ≙   //  the registration of a participant p for an opened course c
        any  p  c
        where
                grd1_1: c ∈ courses
                grd1_2: p ∈ PARTICIPANTS
                grd1_3: p≠CourseInstructor(c)      // p is not the instructor of the course
                grd1_4: c ↦p ∉ participants        // p is not yet registered for the course
        then
                act1: participants  := participants  ∪ {c ↦p}  // adding all the relationships between this
                                                                  course and its participants.
        end
```

Proving consistency of **Members_m1**

- Now we discuss some of the important proof obligations for **Members_m1:**
  - **invariant preservation**
- **CLOSECOURSE/inv1_2/INV –** this obligation is to ensure that ***inv1_2*** is maintained by **CLOSECOURSE** event.

$$\forall c.\ c \in courses \Longrightarrow courseInstructor(c) \notin participants[\{c\}]$$

$$\vdash$$

$$\forall c.\ c \in courses \setminus \{crs\} \Longrightarrow courseInstructor(c) \notin (\{crs\} \mathbin{⩤} participants)[\{c\}]$$

The obligation is trivial, because, given that $c \neq crs$, $(\{crs\} \mathbin{⩤} participants)[\{c\}]$ is the same as $participants[\{c\}]$.

Proving consistency of **Members_m1** (cont.)

- **REGISTER/inv1_1/INV –** this obligation is to ensure that ***inv1_2*** is maintained by the new **REGISTER** event.

…

participants ∈ *cources* ↔ *PARTICIPANTS*

c ∈ *courses*

p ∈ *PARTICIPANTS*

⊢

participants ∪ *{c* ↦*p}* ∈ *cources* ↔ *PARTICIPANTS*

Verifying that the **REGISTER**, *participants* := *participants* ∪ *{c* ↦*p}* , establishes the invariant *participants* ∈ *cources* ↔ *PARTICIPANTS.*

We have following reasoning:

participants ∈ *cources* ↔ *PARTICIPANTS* holds trivially. *{c* ↦*p}* ∈ *cources* ↔ *PARTICIPANTS* also holds, since *c* ∈ *courses* and *p* ∈ *PARTICIPANTS.* Then we conclude that *participants* ∪ *{c* ↦*p}* ∈ *cources* ↔ *PARTICIPANTS*

Data Refinement

- In data refinement, abstract variables *v* are removed and replaced by concrete variables *w*.
- The states of abstract machine M are related to the states of concrete machine N by ***gluing invariants*** *J*(*v*, *w*).
- In Event-B, the gluing invariants *J* are declared as invariants of N and also contain the *local* concrete invariants, i.e., those constraining only concrete variables *w*.
- Coming back to the *Course Management System*…

Data refinement of **Members_m1** machine

- We perform a data refinement by replacing abstract variables ***courses*** and ***participants*** by a new concrete variable ***attendants***:

inv2_1: $attendants \in COURSES \nrightarrow \mathbb{P}(PATICIPANTS)$

- is a *partial function* from *COURSES* to some set of participants.

The following invariants at as gluing invariants, linking abstract variables ***courses*** and ***participants*** with concrete variable ***attendants***

inv2_2: $courses = \mathbf{dom}(attendants)$

inv2_3: $\forall c.\ c \in courses \Longrightarrow participants[\{c\}] = attendants(c)$ // for every opened course *c*, the set of participants attending that course represented abstractly as *participants*[{*c*}] is the same as *attendants*(*c*).

Members_m2 machine

MACHINE **Members_m2**
REFINES **Members_m1**
SEES **Members_c1**
VARIABLES *attendants*
INVARIANTS

inv2_1: $attendants \in COURSES \nrightarrow \mathbb{P}(PATICIPANTS)$

inv2_2: $courses = \mathbf{dom}(attendants)$

inv2_3: $\forall c.\ c \in courses \Longrightarrow participants[\{c\}] = attendants(c)$

EVENTS

…

Refinement of **OPENCOURSE** event

MACHINE **Members_m1** **REFINES** **Courses_m0**
....
OPENCOURSE refines **OPENCOURSE** $\triangleq$
  any *crs*
  where
    grd1: **card**(*courses*) $<$ *m*
    grd2: *crs* $\notin$ *courses*
  then
    act1: *courses* := *courses* $\cup$ {*crs*}
  end

we had before ….

MACHINE **Members_m2** **REFINES** **Members_m1**
…
OPENCOURSE_new refines **OPENCOURSE** $\triangleq$
  any *crs*
  where
    grd2_1: *crs* $\notin$ *dom(attendants)*
    grd2_2: **card**(*attendants*) $\neq$ *m*
  then
    act1: *attendants(crc)* := $\emptyset$
  end

now

- The concrete guards ensure that *crs* is a closed course and the number of opened courses (**card(***attendants***)**) has not reached the limit *m*.
- The action of **OPENCOURSE_new** sets the initial participants for the newly opened course *crs* to be the empty set.

Refinement of **CLOSECOURSE** event

- Abstract event **CLOSECOURSE** is refined by concrete event **CLOSECOURSE_new**, where one course ***crs*** is closed at a time. The guard and action of concrete event **CLOSECOURSE_new** are as expected:

MACHINE Members_m1 **REFINES** Courses_m0
....
CLOSECOURSE refines CLOSECOURSE ≙
    any *crs*
    where
        grd1: $crs \in courses$
    then
        act1: $courses := courses \setminus \{crs\}$
        act2: $participants := \{crs\} ⩤ participants$
    end

we had before ....

MACHINE Members_m2 **REFINES** Members_m1
....
CLOSECOURSE_new refines CLOSECOURSE ≙
    any *crs*
    where
        grd1: $crs \in dom(attendants)$
    then
        act1: $attendants := \{crs\} ⩤ attendants$
    end

now

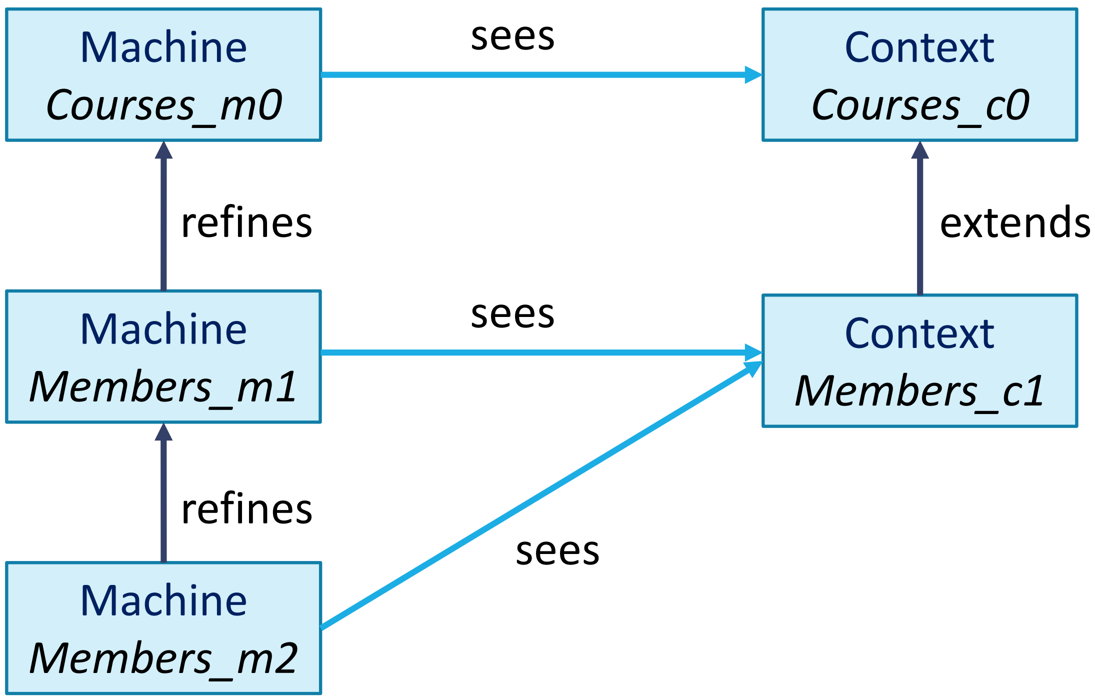
Refinement of **REGISTER** event

MACHINE Members_m1 **REFINES** Courses_m0
…
REGISTER ≙
    any *p c*
    where
        grd1_1: $c \in courses$
        grd1_2: $p \in PARTICIPANTS$
        grd1_3: $p \neq CourseInstructor(c)$
        grd1_4: $c \mapsto p \notin participants$
    then
        act1: $participants := participants \cup \{c \mapsto p\}$
end

MACHINE Members_m2 **REFINES** Members_m1
…
REGISTER_new refines **REGISTER** ≙
    any *p c*
    where
        grd2_1: $c \in dom(attendants)$
        grd2_2: $p \in PARTICIPANTS$
        grd2_3: $p \neq CourseInstructor(c)$
        grd2_4: $p \notin attendants(c)$
    then
        act1: $attendants(c) := attendants(c) \cup \{p\}$
end

Summary of the development

The hierarchy of the development:



Requirements tracing:

REQ id	Models
REQ1	*Members_c1*
REQ2	*Members_c1*
REQ3	*Courses_c0*
REQ4	*Members_c1*
REQ5	*Courses_m0*
REQ6	*Courses_m0*
REQ7	*Courses_m0*
REQ8	*Courses_m0*
REQ9	*Members_m1*
REQ10	*Members_m1*

One more example of refinement

Switching on and off air conditioner after checking temperature

Introducing roles and permissions of users

```
MACHINE
  ExamplLec6
SEES
  ExL6Cont
VARIABLES
  temperature
  aircond
INVARIANTS
  inv1   :   temperature ∈ ℕ
  inv2   :   aircond ∈ AIRCONDSTATES
EVENTS
  INITIALISATION   ≙
  STATUS
   ordinary
  BEGIN
   act1   :   temperature ≔ 23
   act2   :   aircond ≔ OFF
  END

  TEMPSENSING   ≙
  STATUS
   ordinary
  BEGIN
   act1   :   temperature :∈ ℕ
  END

  AIRCONDOFFON   ≙
  STATUS
   ordinary
  WHEN
   grd1   :   temperature > 24
  THEN
   act1   :   aircond ≔ ON
  END

  AIRCONDONOFF   ≙
  STATUS
   ordinary
  WHEN
   grd1   :   temperature < 22
  THEN
   act1   :   aircond ≔ OFF
  END

END
```

```
MACHINE
  ExamplLec60Ref1
REFINES
  ExamplLec6
SEES
  ExL6Cont0R1
VARIABLES
  temperature
  aircond
  flag
INVARIANTS
  inv1  :  flag ∈ PHASE
  inv2  :  (flag = sen ∧ temperature > 24) ⇒ aircond = ON
  inv3  :  (flag = sen ∧ temperature < 22) ⇒ aircond = OFF
EVENTS
  INITIALISATION  ≙
  STATUS
   ordinary
  BEGIN
   act1  :  temperature ≔ 23
   act2  :  aircond ≔ OFF
   act3  :  flag ≔ sen
  END
```

```
TEMPSENSING  ≙
STATUS
 ordinary
REFINES
 TEMPSENSING
WHEN
 grd1  :  flag = sen
THEN
 act1  :  temperature :∈ N
 act2  :  flag ≔ cont
END
```

```
AIRCONDOFFON  ≙
STATUS
 ordinary
REFINES
 AIRCONDOFFON
WHEN
 grd1  :  temperature > 24
 grd2  :  flag = cont
THEN
 act1  :  aircond ≔ ON
 act2  :  flag ≔ sen
END
```

```
AIRCONDONOFF  ≙
 extended
STATUS
 ordinary
REFINES
 AIRCONDONOFF
WHEN
 grd1  :  temperature < 22
 grd2  :  flag = cont
THEN
 act1  :  aircond ≔ OFF
 act2  :  flag ≔ sen
END

END
```

```
  ExL6Cont0R2C2
EXTENDS
  ExL6Cont0R1
SETS
  USERSSET
  ROLES
CONSTANTS
  SUPERVISOR
  ORDINARY
AXIOMS
  axm1   :   USERSSET ≠ ∅
  axm2   :   partition(ROLES, {SUPERVISOR}, {ORDINARY})
END
```

ExampLec60Ref2

```
MACHINE
  ExamplLec60Ref20
REFINES
  ExamplLec60Ref1
SEES
  ExL6Cont0R2C2
VARIABLES
  temperature
  aircond
  flag
  users
  permissions
  request
  act_user
INVARIANTS
  inv1  :  users ⊆USERSSET
  inv2  :  permissions ∈ USERSSET → ROLES
  inv3  :  request ∈ BOOL
  inv4  :  act_user ∈ USERSSET
```

```
AIRCONDOFFON  ≙
  extended
STATUS
  ordinary
REFINES
  AIRCONDOFFON
WHEN
  grd1  :  temperature > 24
  grd2  :  flag = cont
  grd3  :  request=TRUE ∧ permissions(act_user)=SUPERVISOR
THEN
  act1  :  aircond ≔ ON
  act2  :  flag ≔ sen
  act3  :  request ≔ FALSE
END
```

```
AIRCONDONOFF  ≙
  extended
STATUS
  ordinary
REFINES
  AIRCONDONOFF
WHEN
  grd1  :  temperature < 22
  grd2  :  flag = cont
  grd3  :  request=TRUE ∧ permissions(act_user)=ORDINARY
THEN
  act1  :  aircond ≔ OFF
  act2  :  flag ≔ sen
  act3  :  request ≔ FALSE
END
```

ExampLec60Ref2

```
ADDUSER   ≙
STATUS
 ordinary
ANY
 usr
 rl
WHERE
 grd1   :   usr ∈ USERSSET
 grd3   :   rl ∈ ROLES
 grd4   :   usr ∉ dom(permissions)
THEN
 act1   :   users ≔ users ∪ {usr}
 act2   :   permissions ≔ permissions ∪ {usr ↦ r
END

SENDREQUEST   ≙
STATUS
 ordinary
ANY
 usr
WHERE
 grd1   :   usr ∈ users
THEN
 act1   :   request ≔ TRUE
 act2   :   act_user ≔ usr
END

END
```