Summary of Event-B Modeling Notation

Jean-Raymond Abrial
(edited by Thai Son Hoang)

Department of Computer Science
Swiss Federal Institute of Technology Zürich (ETH Zürich)

Bucharest DEPLOY 2-day Course, 14th-16th July, 2010
Purpose of this Presentation

- Showing the structure of the Event-B modeling notation
- Machines, contexts, and events
- Presenting a small example
- Event-B is not a programming language (even very abstract)

- Event-B is a notation used for developing mathematical models of discrete transition systems

- Event-B is to be used together with the Rodin Platform
Such models, once finished, can be used to eventually construct:

- sequential programs,
- distributed programs,
- concurrent programs,
- electronic circuits,
- large systems involving a possibly fragile environment,
- etc.

The underlined statement is an important case.

In this presentation, we shall construct a small sequential program.
A model is made of several components

A component is either a machine or a context:

- **Machine**
  - variables
  - invariants
  - events
  - variant

- **Context**
  - sets
  - constants
  - axioms

- Machines and contexts have names

- Such names must be distinct in a given model
Machines and Contexts (cont’d)

- **Contexts** contain the static structure of a discrete system (constants and axioms)

- **Machines** contain the dynamic structure of a discrete system (variables, invariants, and events)

- Machines see contexts

- Contexts can be extended

- Machines can be refined
Relationship Between Machines and Contexts

Machine \xrightarrow{\text{sees}} \text{Context}

refines

extends

Machine \xrightarrow{\text{sees}} \text{Context}

refines

extends
Visibility Rules  
(can be Skipped at First Reading)

- A machine **can see several contexts** (or no context at all).
- A context may **extend several contexts** (or no context at all).
- A machine **implicitly sees** all contexts extended by a seen context.
- A machine only sees a context either **explicitly** or **implicitly**.
- A machine only **refines at most one other machine**.
- **No cycle** in the “refines” or “extends” relationships.
Example (can be Skipped at First Reading)

- $M_0$ sees $C_{01}$ and $C_{02}$ explicitly.
- $M_1$ sees $C_1$ explicitly.
- $M_1$ sees $C_{01}$ and $C_{02}$ implicitly.
Context Structure

```
context
   < context_identifier >
extends *
   < context_identifier >
   ...
sets *
   < set_identifier >
   ...
constants *
   < constant_identifier >
   ...
axioms *
   < label >: < predicate >
   ...
end
```

- Sections with “∗” might be empty
- All keyword sections are predefined in the Rodin Platform
- All labels are generated automatically by the Rodin Platform (but can be modified)
Explaining Context Sections

- "sets" lists various carrier sets, which define pairwise disjoint types.
- The only property we can assume about a set is that it is not empty.
- "constants" lists the different constants introduced in the context.
- "axioms" defines the main properties of the constants.
- Axioms can be marked as "theorems" denotes derived properties (to be proved) from previously declared the axioms.
A set $D$ is defined in context $\text{ctx}_0$

Moreover, three constants, $n$, $f$, and $v$, are defined in this context:

- $n$ is a natural number ($\text{axm1}$)
- $f$ is a total function from the interval $1..n$ to the set $D$ ($\text{axm2}$)
- $v$ is supposed to belong to the range of $f$ ($\text{axm3}$)

A theorem is proposed: $n$ is a positive number ($\text{thm1}$)
Pictorial Representation of the Context

v is somewhere
Each machine has exactly one **initialisation** event

- All keyword sections are predefined in the Rodin Platform
- All labels are generated automatically by the Rodin Platform (but can be modified)
“variables” lists the state variables of the machine

“invariants” states the properties of the variables

Invariants are defined in terms the seen sets and constants

Invariants can be marked as “theorems” which are derivable from previously declared invariants and seen axioms

“events” defines the dynamics of the transition system (slide 17)

“variant” is explained later (slide 29)
Machine (and Context) Example

Machine **m_0a** sees the previously defined context **ctx_0**

- A variable **i** is defined
- **i** is a member of the interval 1 .. n (**inv1**)  

**events:** next slide
Notice that keyword “where” becomes “when” in the Rodin Platform Pretty Print when there is no “any”.

Notice that keyword “then” becomes “begin” in the Rodin Platform Pretty Print when there are no “any” and no “where/when”.

Again, all keyword sections are predefined in the Rodin Platform.

All labels are generated automatically by the Rodin Platform (but can be modified).
Explaining Event Sections

- An event is a **state transition** in a discrete **dynamic system**.

- **refines** contains the **name(s)** of the refined event(s) (if any)

- Can be skipped at first reading:
  - Several refined events are possible in case of a **merging refining event** concentrating **more than one refined event**
  - **Merged events** must have the same actions
Explaining Event Sections (cont’d)

- "status" is either:
  - ordinary,
  - convergent: it has to decrease the variant (slide 29),
  - anticipated: to be convergent later in a refinement.

- "any" contains the parameters of the event (might be empty)

- "where" (or "when") contains the various guards of the event

- A guard is a necessary condition for an event to be enabled

- Guards can be marked as "theorems" which are derivable from invariants, seen axioms and previously declared guards.

- "actions" see next slide
An action describes the ways one or several *state variables* are modified by the *occurrence* of an event.

An action might be either *deterministic* or *non-deterministic*. 
Deterministic Action (Example)

- Here is the form of some deterministic actions on variables $x$, $y$ and $z$:

  \[
  x := x + y \\
  y := y - x - z
  \]

- Notice that $x$ and $y$ should be distinct.

- Actions are supposed to be “performed” in parallel.

- Variables $x$ and $y$ are assigned to $x + y$ and $y - x - z$ respectively.

- Variable $z$ is used but not modified by these actions.
$x, y : | x' > x \land y' < x'$

- On the LHS of operator $: |$, we have two distinct variables.
- On the RHS, we have a, so-called, before-after predicate.
- The RHS contains occurrences of $x$ and $y$ (before values) and primed occurrences $x'$ and $y'$ (after values).
- As a result (in this example):
  - $x$ is assigned a value greater than its previous value.
  - $y$ is assigned a value smaller than that, $x'$, assigned to $x$. 
Second Form of Non-deterministic Action (Example)

\[ x \in \{x + 1, y - 2, z + 3\} \]

Here \( x \) is assigned any value from the set \( \{x + 1, y - 2, z + 3\} \)
The Most General Form of an Action

- The **second form** of non-deterministic action is **equivalent** to the following **first form**:
  \[ x : | x' \in \{x + 1, y - 2, z + 3\} \]

- Likewise, a **deterministic** action has an **equivalent non-deterministic form**:
  \[ x, y : | x' = x + y \land y' = y - x - z \]

- The **non-det. first form** can thus **always be assumed** (by the tools)
This machine is the model specification of a searching program.

**Machine m_0a**

- **Initialisation**
  ```
  status \triangleq ordinary
  begin
    act1 : i := 1
  end
  ```

- **Search**
  ```
  status \triangleq ordinary
  any
  k
  where
  grd1 : k ∈ 1 .. n
  grd2 : f(k) = v
  then
    act1 : i := k
  end
  ```

- Event **search** assigns to \( i \)
- Any value \( k \) such that \( f(k) = v \),
- Provided \( k \) is in interval \( 1 .. n \)
Pictorial Representation of the State after “search”
Another Machine \( m_{0b} \)

\[
\begin{align*}
\text{machine} & \quad m_{0b} \\
\text{sees} & \quad \text{ctx}_0 \\
\text{variables} & \quad i \\
\text{invariants} & \quad \text{inv1} : \quad i \in 1 \ldots n \\
\text{events} & \quad \ldots \\
\text{end} & \quad \\
\end{align*}
\]

\[
\begin{align*}
\text{initialisation} & \quad \cong \\
\text{status} & \quad \text{ordinary} \\
\text{begin} & \quad \text{act1} : \quad i := 1 \\
\text{end} & \quad \\
\end{align*}
\]

\[
\begin{align*}
\text{search} & \quad \cong \\
\text{status} & \quad \text{ordinary} \\
\text{begin} & \quad \text{act1} : \quad i :| i' \in 1 \ldots n \land f(i') = v \\
\text{end} & \quad \\
\end{align*}
\]

- The \textbf{only difference} between \( m_{0a} \) and \( m_{0b} \) is in event \textbf{search}.
- \( i \) is assigned \textbf{non-deterministically} a values \( i' \) such that \( i' \in 1 \ldots n \) and \( f(i') = v \).
- Notice that event \textbf{search} has \textbf{no guard}. 
“with” contains the witnesses of a refining event.

A witness has to be provided in a refining event

- for each disappearing parameter of the refined event (see m_1a)
- after value of each disappearing variable.

The witness for parameter \( a \) is defined as follows:

\[ a : P(a) \]

where \( P(a) \) is a predicate involving \( a \)

The witness for after value of variable \( b \) is defined as follows:

\[ b' : P(b') \]

where \( P(b') \) is a predicate involving \( b' \)

For a deterministic witness \( P(x) \) is \( x = E \) (with \( E \) free of \( x \))
The variant of a machine is either a natural number expression or a finite set expression.

It has to be present in any machine with convergent events.

A numeric variant must be decreased by all convergent events.

A set variant must be made strictly included in its previous value by all convergent events.
Refinement Machine m_1a Refining Machine m_0a

**machine**
- m_1a
- refines m_0a
- sees ctx_0
- variables i, j
- invariants
  - inv1: j \in 0 .. n - 1
  - inv2: v \notin f[1 .. j]
  - thm1: v \in f[j + 1 .. n]
- variant n – j
- events...
- end

**initialisation** \( \equiv \)
- status ordinary
  - begin
    - act1: i := 1
    - act2: j := 0
  - end

**search** \( \equiv \)
- status ordinary
  - refines search
    - when
      - grd1: f(j + 1) = v
      - with
        - k: j + 1 = k
    - then
      - act1: i := j + 1
    - end

**progress** \( \equiv \)
- status convergent
  - when
    - grd1: f(j + 1) \neq v
  - then
    - act1: j := j + 1
  - end

- A new variable \( j \) is introduced
- Notice invariant inv2 and theorem thm1
- Notice the with section in event search
- A new convergent event progress is introduced
- Notice the numeric variant \( n – j \)
Pictorial Representation of the State

f

\( j + 1 \)

\( v \text{ is somewhere} \)

\( v \text{ not found} \)

\( j \)

\( 1 \)
Refinement Machine $m_{1b}$ Refining Machine $m_{0b}$

```
machine m_1b
refines m_0b
sees ctx_0
variables i j
invariants
  inv1 : j \in 0 .. n - 1
  inv2 : v \notin f[i .. j]
  thm1 : v \in f[j + 1 .. n]
variant j .. n
events ...
end
```

initialisation \( \cong \)
status ordinary
begin
  act1 : i := 1
  act2 : j := 0
end

```
search \( \cong \)
status ordinary
refines search
when
  grd1 : f(j + 1) = v
then
  act1 : i := j + 1
end
```

```
progress \( \cong \)
status convergent
when
  grd1 : f(j + 1) \neq v
then
  act1 : j := j + 1
end
```

- The `with` section in event `search` is not needed
- Notice the finite set `variant j .. n`
- These are the only differences with refining machine $m_{1a}$
A sequential program can be constructed from m_1a (or m_1b)

This is done by applying a number of event merging rules (NOT DEFINED HERE)

The application of these rules yields the following program:

\[
\begin{align*}
  i, j &:= 1, 0; & \text{initialisation} \\
  \text{while } & f(j + 1) \neq v \text{ do } \begin{align*}
    j &:= j + 1 & \text{progress} \\
  \end{align*} \\
  \text{end; } \\
  i &:= j + 1 & \text{search}
\end{align*}
\]
Exercise

Modify refinement **m_1a** (or **m_1b**), in order to obtain the following final program from the same specification **m_0a** (or **m_0b**):

\[
\begin{align*}
i, j &:= 1, n + 1 ; \\
&\text{initialisation} \\
\textbf{while} &\quad f(j - 1) \neq \nu \quad \textbf{do} \\
&\quad j := j - 1 \\
&\quad \text{progress} \\
&\textbf{end} ; \\
&\quad \text{search} \\
i &:= j - 1
\end{align*}
\]