



Learning Material: Code Generation with Tasking Event-B

A. Edmunds

Electronics and Computer Science, University of Southampton

March 26, 2013

Outline

- 1 Event-B
 - Background
 - Overview of Event-B
 - Composition / Decomposition
- 2 Implementation-Level Modelling
 - Tasking Event-B
 - The User Interface: Machine and Event Annotations
- 3 Adding New Types, and Translation Rules
 - Translation Rules for Ada
 - Example of Adding a New Type

Outline

- 1 **Event-B**
 - Background
 - Overview of Event-B
 - Composition / Decomposition
- 2 **Implementation-Level Modelling**
 - Tasking Event-B
 - The User Interface: Machine and Event Annotations
- 3 **Adding New Types, and Translation Rules**
 - Translation Rules for Ada
 - Example of Adding a New Type

What We Have...

- Automatic Code Generation from Event-B To Ada, C, Java
 - FMI C or OpenMP C.
 - Ada for Multi-Tasking Embedded Systems.
 - Java for concurrent shared memory systems.
 - Modelling of Controllers / Protected, Shared Data and Environment.
- Extensible Mathematical Language Translations:
 - add new Types, and their Implementations.

What We Have...

- Automatic Code Generation from Event-B To Ada, C, Java
 - FMI C or OpenMP C.
 - Ada for Multi-Tasking Embedded Systems.
 - Java for concurrent shared memory systems.
 - Modelling of Controllers / Protected, Shared Data and Environment.
- Extensible Mathematical Language Translations:
 - add new Types, and their Implementations.

Resources

- From the EU funded RODIN, DEPLOY, and ADVANCE projects:
 - <http://www.event-b.org/>
 - http://wiki.event-b.org/index.php/Main_Page
 - <http://www.advance-ict.eu/>
 - *... a unified tool-based framework for automated formal verification and simulation-based validation of cyber-physical systems.*
- Rodin Tools - A new not-for-profit company.

Outline

- 1 **Event-B**
 - Background
 - **Overview of Event-B**
 - Composition / Decomposition
- 2 Implementation-Level Modelling
 - Tasking Event-B
 - The User Interface: Machine and Event Annotations
- 3 Adding New Types, and Translation Rules
 - Translation Rules for Ada
 - Example of Adding a New Type

Event-B

- **Based on Set-Theory + Predicate Logic + Arithmetic,**
 - Tool Support, with Automatic and Interactive proof.
 - Refinement, for incremental development.
- Context Component.
 - Specify Sets, Constants, and Axioms.
- Machine Component.
 - Specify Variables, Invariants, and Events.
- Theory Component
 - Add new types, Operations, and Invariants.
 - Add new translation rules, and proof rules.

Event-B

- Based on Set-Theory + Predicate Logic + Arithmetic,
 - Tool Support, with Automatic and Interactive proof.
 - Refinement, for incremental development.
- Context Component.
 - Specify Sets, Constants, and Axioms.
- Machine Component.
 - Specify Variables, Invariants, and Events.
- Theory Component

Event-B

- Based on Set-Theory + Predicate Logic + Arithmetic,
 - Tool Support, with Automatic and Interactive proof.
 - Refinement, for incremental development.
- Context Component.
 - Specify Sets, Constants, and Axioms.
- Machine Component.
 - Specify Variables, Invariants, and Events.
- Theory Component

Event-B

- Based on Set-Theory + Predicate Logic + Arithmetic,
 - Tool Support, with Automatic and Interactive proof.
 - Refinement, for incremental development.
- Context Component.
 - Specify Sets, Constants, and Axioms.
- Machine Component.
 - Specify Variables, Invariants, and Events.
- Theory Component
 - Add new Types, Operators.
 - Add new Translation, Re-write Rules etc.

Event-B - Context

... from the Heater Controller Example.

CONTEXT

HC_CONTEXT

CONSTANTS

Max

Min

AXIOMS

axm1 : Max = 45

axm2 : Min = 5

axm3 : Max $\in \mathbb{Z}$

axm4 : Min $\in \mathbb{Z}$

END

Event-B - Machines, Variables etc.

MACHINE

HCtrl_M0

SEES

HC_CONTEXT

VARIABLES

hsc // *heat source commanded*

nha // *no heat alarm*

cttm2 // *commanded target temp*

...

INVARIANTS

typing_nha : nha ∈ BOOL

typing_hsc : hsc ∈ BOOL

typing_ota : cttm2 ∈ ℤ

...

EVENTS

INITIALISATION ≐

BEGIN

act3: hsc = FALSE

act4: nha = FALSE

act5: cttm2 :∈ ℤ

...

END

Event-B - Events

```
TurnON_Heat_Source ≐  
REFINES  
TurnON_Heat_Source  
WHEN  
    // average temp less  
grd1: avt < cttm2 // than commanded  
    // value  
THEN  
act1: hsc = TRUE // Turn heat source on  
END
```

- Based on guarded command: $g \rightarrow a$
 - In Event-B, the guard g is an Event-B predicate;
 - the action a is an Event-B expression.

Event-B - Event Parameters

```
Sense_Temperatures ≐  
ANY t1 t2  
WHERE grd1: t1 ∈ ℤ  
      grd2: t2 ∈ ℤ  
THEN act1: stm1 := t1  
      act2: stm2 := t2  
END
```

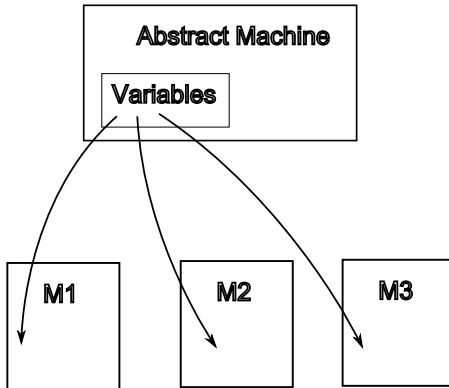
- The **ANY** construct admits parameters:
 - Parameters are typed in the Guard;
 - but may not be assigned to.

Outline

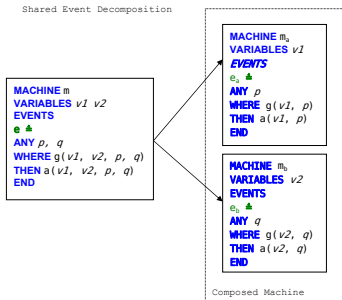
- 1 **Event-B**
 - Background
 - Overview of Event-B
 - **Composition / Decomposition**
- 2 Implementation-Level Modelling
 - Tasking Event-B
 - The User Interface: Machine and Event Annotations
- 3 Adding New Types, and Translation Rules
 - Translation Rules for Ada
 - Example of Adding a New Type

Decomposition

Distribute Variables Between Machines

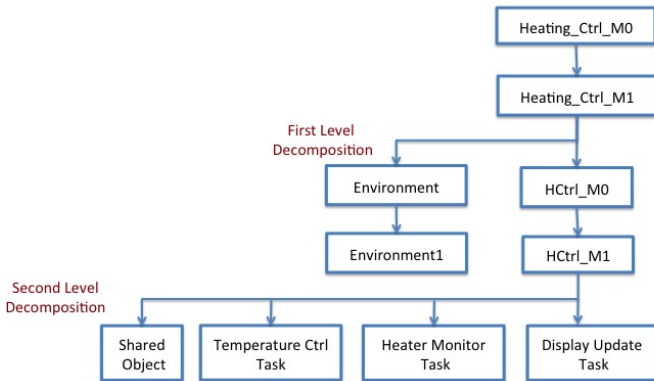


Automatic Decomposition



- Events are Refactored.
- Synchronization $e_a \parallel e_b$ models an atomic subroutine call.
- The Composed Machine is a Refinement.

The Heater Controller Development



Outline

- 1 Event-B
 - Background
 - Overview of Event-B
 - Composition / Decomposition
- 2 Implementation-Level Modelling
 - **Tasking Event-B**
 - The User Interface: Machine and Event Annotations
- 3 Adding New Types, and Translation Rules
 - Translation Rules for Ada
 - Example of Adding a New Type

Implementation Level Modelling

- Using 'Annotated' Event-B models - Tasking Event-B.
 - Specify a task's priority, and type (periodicity etc.) Formal modelling of time is in its early stages.
 - A Machine's Task-Body - formally describes the flow of execution,
 - is the basis for refinement of the Abstract Development.

Implementation Level Modelling

- Using 'Annotated' Event-B models - Tasking Event-B.
- Specify a task's priority, and type (periodicity etc.) Formal modelling of time is in its early stages.
- A Machine's Task-Body - formally describes the flow of execution,
- is the basis for refinement of the Abstract Development.

Implementation Level Modelling

- Using 'Annotated' Event-B models - Tasking Event-B.
- Specify a task's priority, and type (periodicity etc.) Formal modelling of time is in its early stages.
- A Machine's Task-Body - formally describes the flow of execution,
- is the basis for refinement of the Abstract Development.

Implementation Level Modelling

- Using 'Annotated' Event-B models - Tasking Event-B.
- Specify a task's priority, and type (periodicity etc.) Formal modelling of time is in its early stages.
- A Machine's Task-Body - formally describes the flow of execution,
- is the basis for refinement of the Abstract Development.

Correspondence with Ada

- **AutoTask Machines**
 - map to Controller Task Implementations;
 - anonymous tasks declared in main.

● Environ Machines

● Environment Tasks

● Shared Machines

Correspondence with Ada

- AutoTask Machines
 - map to Controller Task Implementations;
 - anonymous tasks declared in main.
- Environ Machines
 - map to Environment Tasks.
- Environment Tasks
- Shared Machines

Correspondence with Ada

- AutoTask Machines
 - map to Controller Task Implementations;
 - anonymous tasks declared in main.
- Environ Machines
 - map to Environment Tasks.
- Environment Tasks
 - simulate the environment,
 - or, provide an interface to the environment.
 - (to be explored in the Advance project)
- Shared Machines

Correspondence with Ada

- AutoTask Machines
 - map to Controller Task Implementations;
 - anonymous tasks declared in main.
- Environ Machines
 - map to Environment Tasks.
- Environment Tasks
 - simulate the environment,
 - or, provide an interface to the environment.
 - (to be explored in the Advance project)
- Shared Machines
 - map to Protected Objects in Ada.

Correspondence with Ada

- Mapping of events

- depends on use in task body.
- Some event guards and actions are 'in-lined'.
- Some events map to 'subroutines', and are *called*.
- Guards

→ can map to entry bodies.

→ can map to subprograms.

- The code generator takes care of this.

- Synchronizations:

→ Tasking & Shared Machine – protected subprograms.

→ Tasking & Shared Machine – new types.

Correspondence with Ada

- Mapping of events
 - depends on use in task body.
 - Some event guards and actions are 'in-lined'.
 - Some events map to 'subroutines', and are *called*.
 - Guards
 - Imperative entry bodies
 - The code generator takes care of this.
- Synchronizations:
 - Tasking & Shared Memory – protected subprograms
 - Tasking & Shared Memory – mutexes

Correspondence with Ada

- Mapping of events
 - depends on use in task body.
 - Some event guards and actions are 'in-lined'.
 - Some events map to 'subroutines', and are *called*.
 - Guards
 - The code generator takes care of this.
- Synchronizations:
 - Tasking & Shared Memory – protected subprograms
 - Tasking & Shared Memory – protected objects

Correspondence with Ada

- Mapping of events
 - depends on use in task body.
 - Some event guards and actions are 'in-lined'.
 - Some events map to 'subroutines', and are *called*.
 - Guards
 - The code generator takes care of this.
- Synchronizations:

Correspondence with Ada

- Mapping of events
 - depends on use in task body.
 - Some event guards and actions are 'in-lined'.
 - Some events map to 'subroutines', and are *called*.
 - Guards
 - map to entry barriers,
 - or, looping/branching statements.
 - The code generator takes care of this.
- Synchronizations:

Correspondence with Ada

- Mapping of events
 - depends on use in task body.
 - Some event guards and actions are 'in-lined'.
 - Some events map to 'subroutines', and are *called*.
 - Guards
 - map to entry barriers,
 - or, looping/branching statements.
 - The code generator takes care of this.
- Synchronizations:

Correspondence with Ada

- Mapping of events
 - depends on use in task body.
 - Some event guards and actions are 'in-lined'.
 - Some events map to 'subroutines', and are *called*.
 - Guards
 - map to entry barriers,
 - or, looping/branching statements.
 - The code generator takes care of this.
- Synchronizations:
 - Tasking & Shared Machine = protected subprogram/entry .
 - Tasking & Environ Machine = rendezvous.

Correspondence with Ada

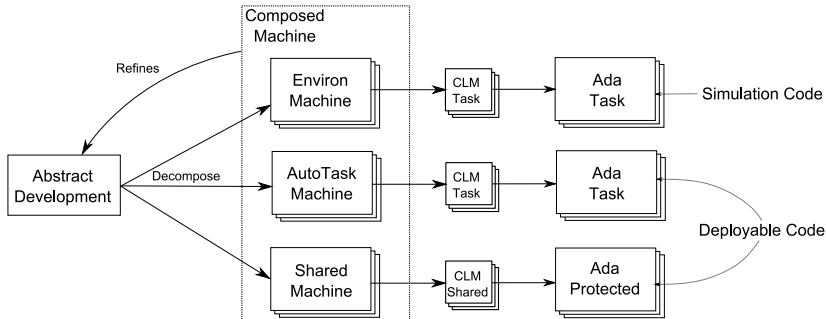
- Mapping of events
 - depends on use in task body.
 - Some event guards and actions are 'in-lined'.
 - Some events map to 'subroutines', and are *called*.
 - Guards
 - map to entry barriers,
 - or, looping/branching statements.
 - The code generator takes care of this.
- Synchronizations:
 - Tasking & Shared Machine = protected subprogram/entry .
 - Tasking & Environ Machine = rendezvous.

Correspondence with Ada

- Mapping of events
 - depends on use in task body.
 - Some event guards and actions are 'in-lined'.
 - Some events map to 'subroutines', and are *called*.
 - Guards
 - map to entry barriers,
 - or, looping/branching statements.
 - The code generator takes care of this.
- Synchronizations:
 - Tasking & Shared Machine = protected subprogram/entry .
 - Tasking & Environ Machine = rendezvous.

The Common Language Model

The Common Language Meta-model is independent of the implementation; an abstraction based on Ada.



Outline

- 1 Event-B
 - Background
 - Overview of Event-B
 - Composition / Decomposition
- 2 Implementation-Level Modelling
 - Tasking Event-B
 - The User Interface: Machine and Event Annotations
- 3 Adding New Types, and Translation Rules
 - Translation Rules for Ada
 - Example of Adding a New Type

UI - Specifying a Task Body

Integrated with

- Machine Editor.

TASKING



MACHINE TYPE PRIORITY //

▼ TASK TYPE



PERIOD



▼ TASK BODY



```

Get_Target_Temperature1 ;
Sense_PressIncrease_Target_Temperature ;
if Raise_Target_Temperature
else Raise_Target_Temperature_Blocked ;
Sense_PressDecrease_Target_Temperature ;
if Lower_Target_Temperature
else Lower_Target_Temperature_Blocked ;
Set_Target_Temperature ;
Display_Target_Temperature

```


UI - Events

- Synchronized Events
- Parameter Directions.
- Typing.

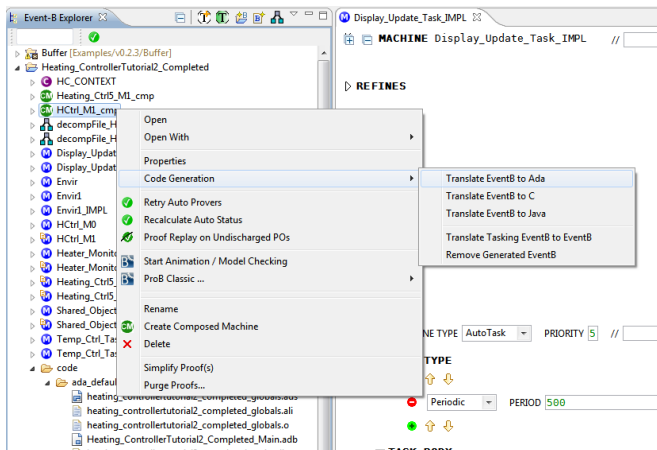
```

Get_Target_Temperature1  $\hat{=}$ 
  COMBINES EVENT
  Shared Object IMPL.Get Target Temperature1 ||
  Display_Update_Task_IMPL.Get_Target_Temperature1
REFINES
  Get_Target_Temperature1
  
```

```

Get_Target_Temperature1  $\hat{=}$ 
REFINES
  Get_Target_Temperature1
ANY
  in tm
WHERE
  grd1 : tm  $\in$   $\mathbb{Z}$  TYPING
THEN
  act1 : cttm1 = tm
END
  
```

Generating Code



Outline

- 1 Event-B
 - Background
 - Overview of Event-B
 - Composition / Decomposition
- 2 Implementation-Level Modelling
 - Tasking Event-B
 - The User Interface: Machine and Event Annotations
- 3 Adding New Types, and Translation Rules
 - Translation Rules for Ada
 - Example of Adding a New Type

Using Mathematical Extensions

THEORY AdaRules

TRANSLATOR Ada

Metavariables ▪ $a \in \mathbb{Z}$, $b \in \mathbb{Z}$, $c \in \mathbb{Q}$, $d \in \mathbb{Q}$

Translator Rules

...

trns2: $a - b \mapsto a - b$

trns9: $c = d \mapsto c = d$

trns19: $a \neq b \mapsto a \neq b$

trns21: $a \bmod b \mapsto a \bmod b$

trns22: $\neg \$c \mapsto \text{not}(\$c)$

trns23: $\$c \vee \$d \mapsto (\$c) \text{ or } (\$d)$

trns24: $\$c \wedge \$d \mapsto (\$c) \text{ and } (\$d)$

trns25: $\$c \Rightarrow \$d \mapsto \text{not}(\$c) \text{ or } (\$d)$

Type Rules

typeTrns1: $\mathbb{Z} \mapsto \text{Integer}$

typeTrns2: $\text{BOOL} \mapsto \text{boolean}$

Outline

- 1 Event-B
 - Background
 - Overview of Event-B
 - Composition / Decomposition
- 2 Implementation-Level Modelling
 - Tasking Event-B
 - The User Interface: Machine and Event Annotations
- 3 Adding New Types, and Translation Rules
 - Translation Rules for Ada
 - Example of Adding a New Type

Adding Arrays

THEORY Array

TYPE PARAMETERS T

OPERATORS

• **array** : $\text{array}(s : \mathbb{P}(T))$

direct definition

$$\text{array}(s : \mathbb{P}(T)) \triangleq \{ n, f \cdot n \in \mathbb{N} \wedge f \in 0 \cdot \dots (n-1) \rightarrow s \mid f \}$$

• **arrayN** : $\text{arrayN}(n : \mathbb{Z}, s : \mathbb{P}(T))$

well-definedness condition $n \in \mathbb{N} \wedge \text{finite}(s)$

direct definition

$$\text{arrayN}(n : \mathbb{Z}, s : \mathbb{P}(T)) \triangleq \{ a \mid a \in \text{array}(s) \wedge \text{card}(s) = n \}$$

Theory: Translation Rules for Arrays

- **update** : $\text{update}(a : \mathbb{Z} \leftrightarrow T, i : \mathbb{Z}, x : T)$
- ...
- **lookup** : $\text{lookup}(a : \mathbb{Z} \leftrightarrow T, i : \mathbb{Z})$
- ...
- **newArray** : $\text{newArray}(n : \mathbb{Z}, x : T)$
- ...

TRANSLATOR Ada

Metavariables $s \in \mathbb{P}(T)$, $n \in \mathbb{Z}$, $a \in \mathbb{Z} \leftrightarrow T$, $i \in \mathbb{Z}$, $x \in T$

Translator Rules

- trns1** : $\text{lookup}(a, i) \mapsto a(i)$
- trns2** : $a = \text{update}(a, i, x) \mapsto a(i) := x$
- trns3** : $\text{newArray}(n, x) \mapsto (\text{others} \Rightarrow x)$

Type Rules

- typeTrns1** : $\text{arrayN}(n, s) \mapsto \text{array } (0..n-1) \text{ of } s$

Theory: Applying the Rules for Arrays

Event-B:

```
Invariants cbuf ∈ arrayN(maxbuf,Z)
Initialisation cbuf := newArray(maxbuf,0)
```



```
type rule : arrayN(n,s)   ↪ array (0..n-1) of s
constructor : newArray(n,x) ↪ (others => x)
                Z           ↪ Integer
```



Ada:

```
type cbuf_array is array (0..maxbuf-1) of Integer;
cbuf : cbuf_array := (others => 0);
```


Wrapping Up

- Tasking Event-B guides code generation.
- Event-B modelling artefacts correspond to:
 - Ada Tasks - Protected Objects;
 - Java threads - monitors.
 - C which uses PThread library.
- The Common Language Meta-model is an abstraction of commonly used programming constructs.
- Tasking Event-B has: AutoTask , Environ and Shared machines
 - AutoTask Machines have a Task-body to specify flow of control.
 - The Tasking Language has sequence, branch and loop constructs.

Wrapping Up

- We make use of the tool-driven decomposition approach, to structure the development.
 - This allows us to partition the system in a modular fashion, reflecting implementation constructs.
 - Decomposition is also the mechanism for breaking up complex systems to make modelling and proof more tractable.
- We have data-type and operator extensibility.
- Target Language specification is extensible.