Formal Modelling for Ada Implementations: Tasking Event-B

A. Edmunds A.Rezazadeh M. Butler I. Maamria

Department of Electronics and Computer Science
University of Southampton

Ada Europe 2012

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

Motivation

- Automatic Code Generation from Event-B To Ada,
  - for Multi-Tasking Embedded Systems.
  - Modelling of Controllers / Protected, Shared Data and Environment.
  - with a stream-lined approach.
- Extensibility: add new Types, and their Implementations.
- Latest Work:
  - Gone from from 'demonstrator' tool to an integrated tool.
  - Improved static checking.
  - Perform code generation from Event-B State-machines.

Motivation

- Automatic Code Generation from Event-B To Ada,
    - for Multi-Tasking Embedded Systems.
    - Modelling of Controllers / Protected, Shared Data and Environment.
    - with a stream-lined approach.
- Extensibility: add new Types, and their Implementations.
- Latest Work:
    - Gone from from 'demonstrator' tool to an integrated tool
    - Improved static checking.
    - Perform code generation from Event-B State-machines.

Motivation

- Automatic Code Generation from Event-B To Ada,
  - for Multi-Tasking Embedded Systems.
  - Modelling of Controllers / Protected, Shared Data and Environment.
  - with a stream-lined approach.
- Extensibility: add new Types, and their Implementations.
- Latest Work:
  - Gone from from 'demonstrator' tool to an integrated tool.
  - Improved static checking.
  - Perform code generation from Event-B State-machines.

Resources

- From the EU funded RODIN, and DEPLOY projects:
  - http://www.event-b.org/
  - http://wiki.event-b.org/index.php/Main_Page
- Continuing with the Advance project:
  - 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, Operators.
  - Add new Translation, Re-write Rules etc.

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

- 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

- 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 ∈ ℤ
    axm4   :   Min ∈ ℤ
END
```

Event-B - Macines, 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
```

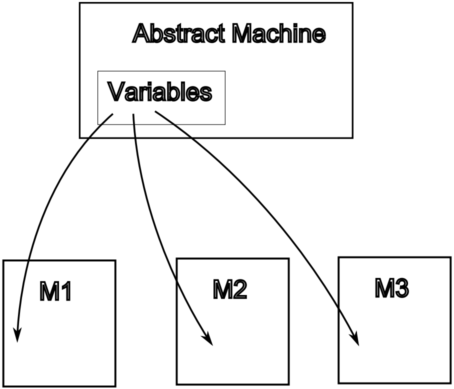
- The **ANY** construct admits parameters:
  - Parameters are typed in the Guard;
  - but may not be assigned to.

Outline

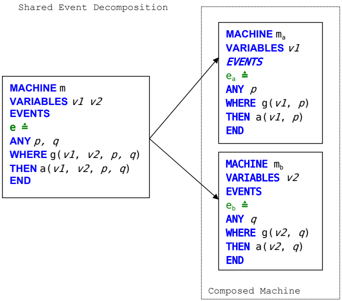
1. Event-B
   - Background
   - Overview of Event-B
   - Composition / Decomposition
2. Implementation-Level Modelling
   - Tasking Event-B
   - The User Interface: Machine and Event Annotations
3. Adding New Types, and Translation Rules
   - Translation Rules for Ada
   - Example of Adding a New Type

Decomposition

Distribute Variables Between Machines



Automatic Decomposition



Shared Event Decomposition

MACHINE m
VARIABLES *v1 v2*
EVENTS
e ≙
ANY *p, q*
WHERE g(*v1, v2, p, q*)
THEN a(*v1, v2, p, q*)
END

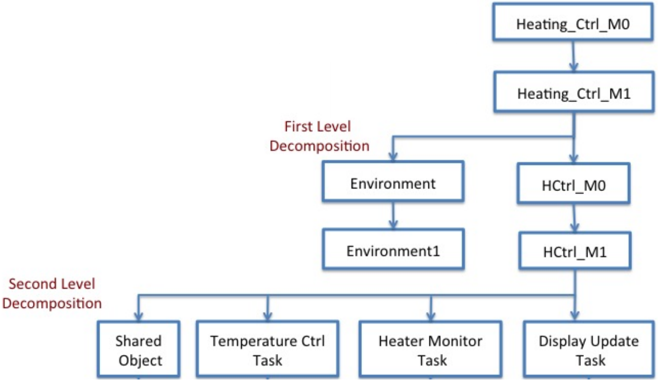
MACHINE m_a
VARIABLES *v1*
EVENTS
e_a ≙
ANY *p*
WHERE g(*v1, p*)
THEN a(*v1, p*)
END

MACHINE m_b
VARIABLES *v2*
EVENTS
e_b ≙
ANY *q*
WHERE g(*v2, q*)
THEN a(*v2, q*)
END

Composed Machine

- 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
  - 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

- 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

- 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

- 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
    - 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.

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.

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

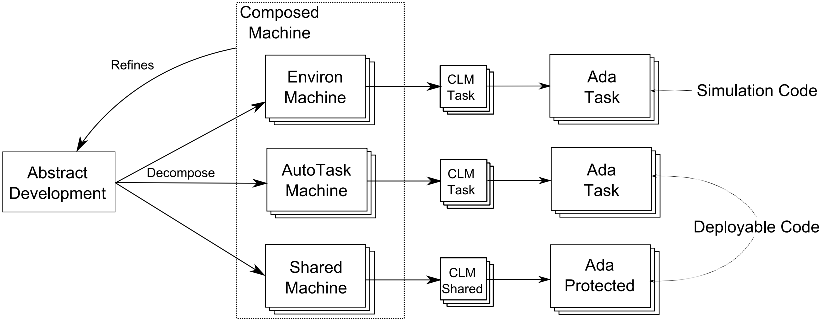
- Mapping of events
  - depends on use in task body.
  - Some event guards and actions are ‘in-lined’.
  - Some events map to ’subroutines’, and are *called*.
  - Guards
    - map to entry barriers,
    - or, looping/branching statements.
  - The code generator takes care of this.
- Synchronizations: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



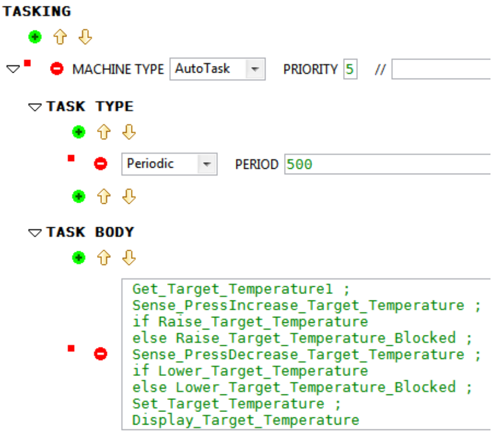
Outline

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 AutoTask PRIORITY 5 //

TASK TYPE

Periodic PERIOD 500

TASK BODY

```
Get_Target_Temperaturel ;
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
```

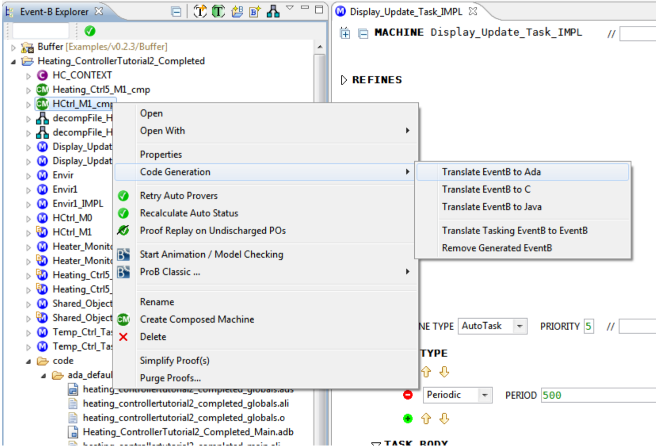
UI - Events

- Synchronized Events
- Parameter Directions.
- Typing.

```
Get_Target_Temperature1   ≙
    COMBINES EVENT
 Shared Object IMPL.Get Target Temperature1 ||
 Display_Update_Task_IMPL.Get_Target_Temperature1
REFINES
 Get_Target_Temperature1
```

```
Get_Target_Temperature1   ≙
REFINES
    Get_Target_Temperature1
ANY
    in tm
WHERE
    grd1   :   tm ∈ ℤ   TYPING
THEN
    act1   :   cttm1 ≔ tm
END
```

Generating Code



```
MACHINE Display_Update_Task_IMPL //
```

NE TYPE AutoTask PRIORITY 5 //

Periodic PERIOD 500

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 ∈ ℤ, b ∈ ℤ, c ∈ Q, d ∈ Q
Translator Rules
     ...
     trns2:   a − b ↦ a - b
     trns9:   c = d ↦  c = d
     trns19:  a ≠ b ↦  a /= b
     trns21:   a mod b    ↦  a mod b
     trns22:   ¬$c    ↦   not($c)
     trns23:   $c ∨ $d  ↦   ($c) or ($d)
     trns24:   $c ∧ $d  ↦   ($c) and ($d)
     trns25:   $c ⇒ $d  ↦ not($c) or ($d)
Type Rules
     typeTrns1:   ℤ     ↦ Integer
     typeTrns2:   BOOL  ↦ 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 : $\mathrm{array}(s : \mathbb{P}(T))$
direct definition
$\mathrm{array}(s : \mathbb{P}(T)) \mathrel{\hat=} \{\, n, f \cdot n \in \mathbb{N} \wedge f \in 0 \cdot\cdot (n-1) \rightarrow s \mid f \,\}$

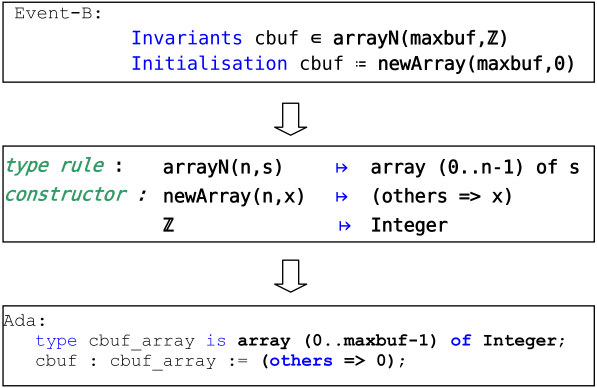
•arrayN : $\mathrm{arrayN}(n : \mathbb{Z}, s : \mathbb{P}(T))$
well-definedness condition $n \in \mathbb{N} \wedge \mathrm{finite}(s)$
direct definition
$\mathrm{arrayN}(n : \mathbb{Z}, s : \mathbb{P}(T)) \mathrel{\hat=} \{\, a \mid a \in \mathrm{array}(s) \wedge \mathrm{card}(s) = n \,\}$

Theory: Translation Rules for Arrays

```
•update      :   update(a : ℤ↔T, i : ℤ, x : T)
 …
•lookup   :   lookup(a : ℤ↔T, i : ℤ)
 …
•newArray   :   newArray(n : ℤ, x : T)
 …

TRANSLATOR Ada
Metavariables s ∈ ℙ(T), n ∈ ℤ, a ∈ ℤ↔T, i ∈ ℤ, x ∈ T
Translator Rules
     trns1    :   lookup(a,i) ↦ a(i)
     trns2    :   a ≔ update(a,i,x) ↦ a(i) := x
     trns3    :   newArray(n,x) ↦ (others => x)
Type Rules
     typeTrns1    :   arrayN(n,s) ↦ array (0..n-1) 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 counterparts,
  - with the Common Language Meta-model; an abstraction of Ada types.
- AutoTask machine, Environ machine or Shared machine.
  - Task body to specify flow of control;
  - with 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 Ada implementation constructs.
    - Decomposition is also the mechanism for breaking up complex systems to make modelling and proof more tractable.
- Data type and operator extensibility.
- Target Language extensible.
- Future work:
    - The Advance project is ongoing.
    - Mindstorms Group Projects.