UNIVERSITY OF Southampton
School of Electronics and Computer Science

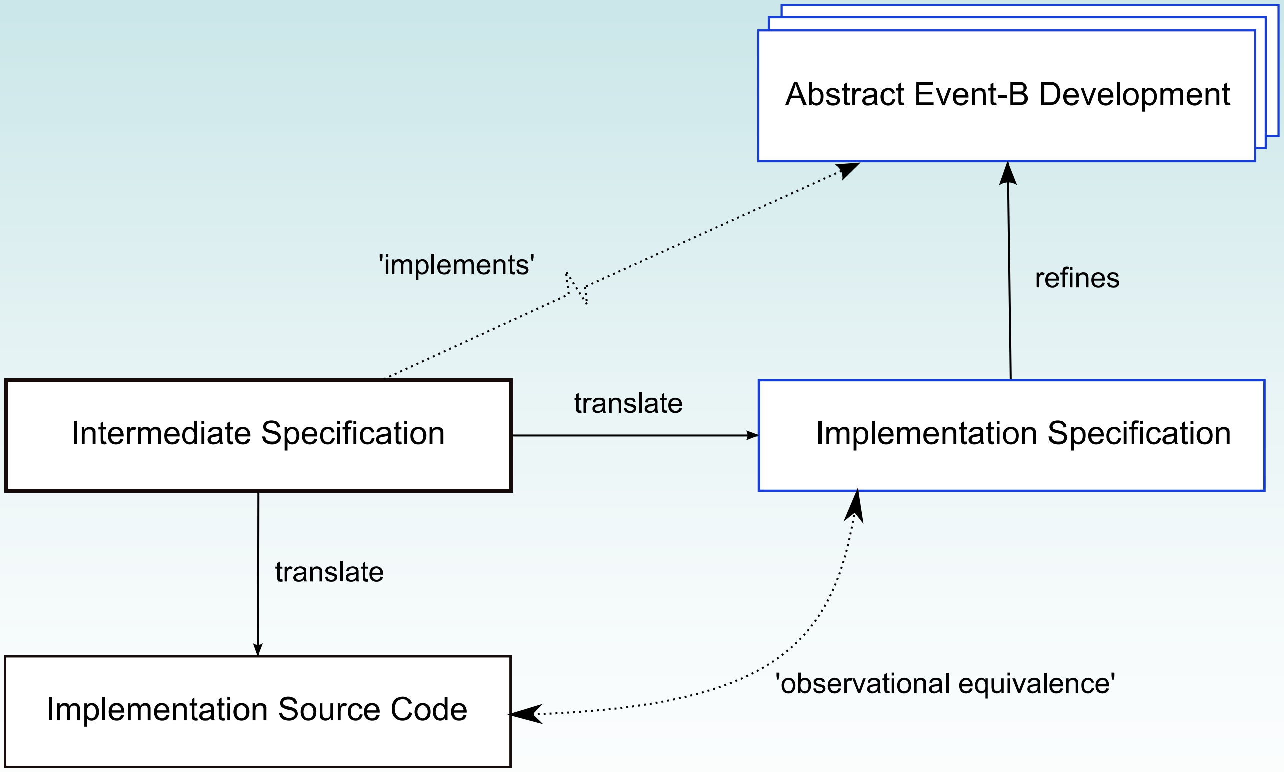
Rodin User + Developer Workshop

Tasking Event-B for Code Generation

Andy Edmunds ae2@ecs.soton.ac.uk
and
Michael Butler mjb@ecs.soton.ac.uk

Previous work

OCB – Linking Event-B and Object-Oriented Implementations


Abstract Event-B Development
'implements'
refines
Intermediate Specification
translate
Implementation Specification
translate
Implementation Source Code
'observational equivalence'

Previous work

The Intermediate Specification (OCB),

- was Object Oriented in style; java-like.
- mapped to a Java implementation.
- had a large semantic gap between the Event-B model and OCB.
- gave rise to difficult refinements, due to the abstraction large gap.

Tasking Event-B

Tasking Event-B is an extension of Event-B,

- with a smaller semantic gap (between Event-B and Implementation specification) than in previous work.
- with smaller refinement steps which should make proofs easier.
- with translators that map to Ada (and in the future, C).

Tasking Event-B

Targeting implementations with,

- Multi-tasking capability
- Tasking
  - for shared memory systems.
  - i.e. task/lightweight process/thread.
  - using interleaving atomic executions.
- Sharing data between tasks using ‘protected objects’,
  - using atomic procedure calls,
  - with blocking behaviour.

Tasking Event-B

Has Loop, Branch, Sequence, and Synchronisation Constructs.

Protected Object's updates Modelled by Shared Event Composition

Events can map to,

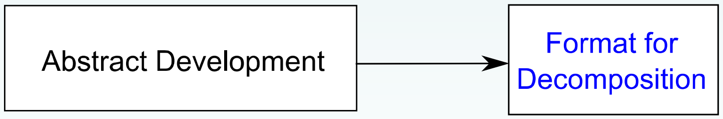
- part of a loop /branch implementation.
- a subroutine definition.
- part of a subroutine call (parameters).

Before Decomposition

1. Specify the abstract development.
2. Prepare for decomposition. For each event,

    - identify and specify parameters (using event guards),

    - substitute expressions by parameters, in event actions, where applicable.



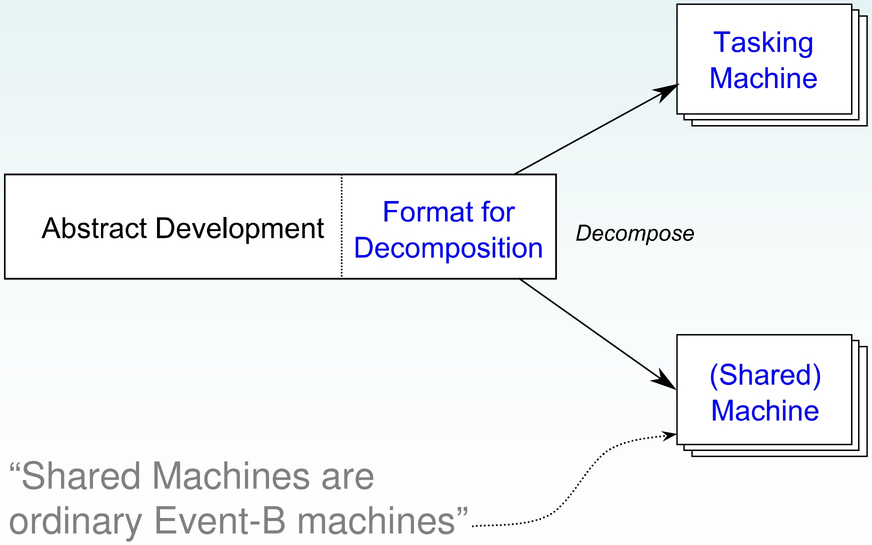
Abstract Development

Format for Decomposition

Decompose

3. Allocate variables to machines during shared event decomposition (typically to multiple Tasking/ Shared Machines)

4. Complete the decomposition.



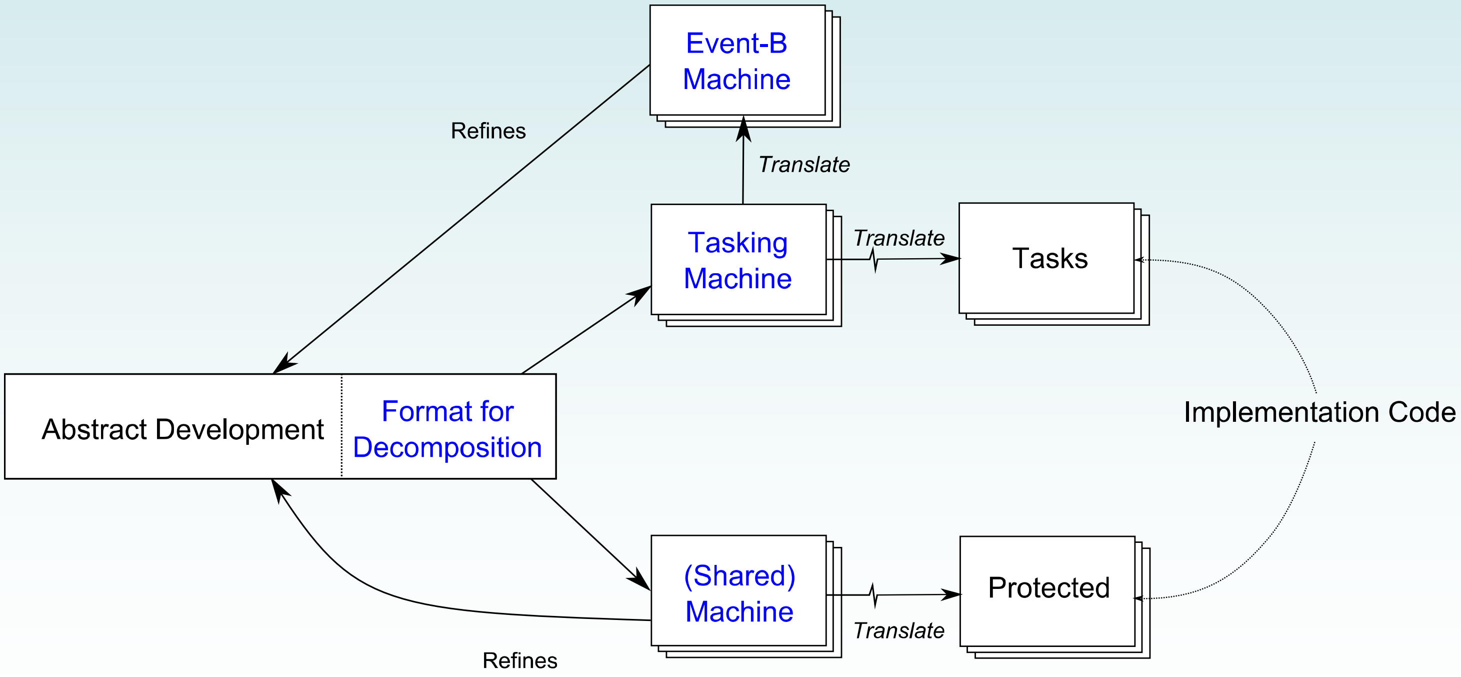
“Shared Machines are ordinary Event-B machines”

Translation

5. Copy, or reference, decomposed machines for use in the tasking model.

6. Add Tasking Constructs to create Tasking and Shared Machines.
   e.g. synch, loop, branch, sequence, priority, etc.

7. Automatic Translation to Code and Event-B



Tasking Event-B Notation V1

```
TaskBody ::=
  TaskBody ; TaskBody
  | if EventSynch endif
    [ elseif EventSynch endelseif ] …
    [ else EventSynch endelse ]
  | do EventSynch [ finally EventSynch ] od
  | EventSynch
```

More details @
http://wiki.event-b.org/images/TranslationV20100722.pdf

tasktype ::= Periodic(p) | One Shot | Repeating | Triggered

priority(n)

Event Synchronisation

EventSynch ::= LocalEvent RemoteEvent

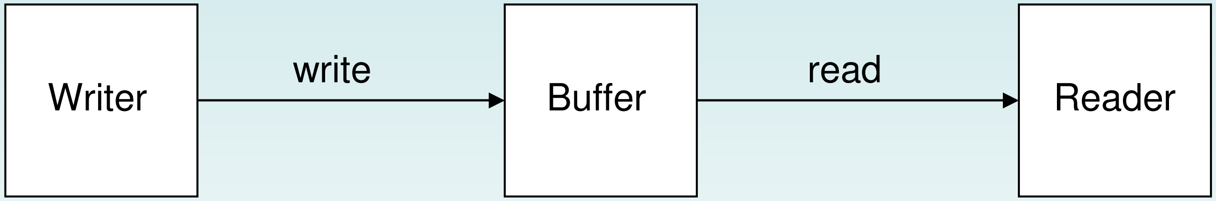
- Tasking Local/Remote Events are annotated Event-B Events

- Local/remote is relative to a particular task.
    - A local event belongs to a tasking machine, and only updates the task's state.
    - A remote event belongs to a shared machine, and only updates a shared machine's state.

- Specifies 'synchronisation' of a local and remote events
    - decomposition semantics; guards are conjoined.
    - with parallel updates.

One-Place Buffer Example

“write a single NAT value to buffer”



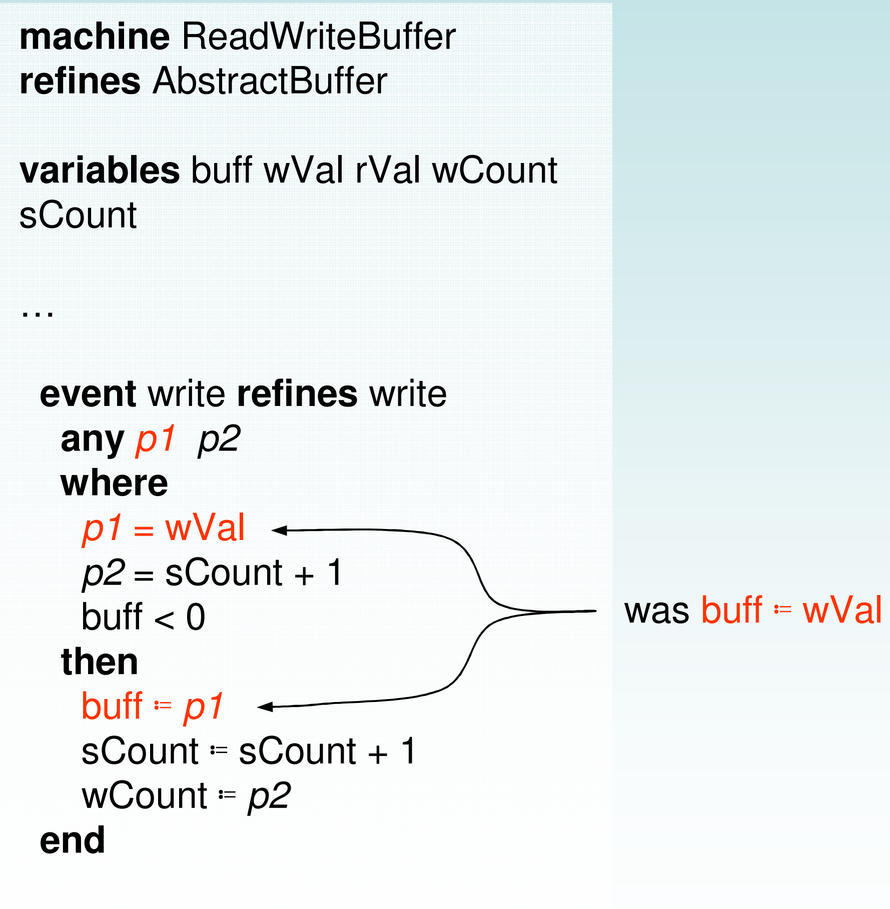
“read the value from the buffer”

Abstract Machine

```
machine AbstractBuffer

variables buff wVal rVal wCount sCount

…

  event write
    where
      buff < 0
    then
      buff ≔ wVal
      sCount ≔ sCount + 1
      wCount ≔ sCount + 1
  end
```

“buff is initially -1”

Parameterised for Decomposition (i)



```
machine ReadWriteBuffer
refines AbstractBuffer

variables buff wVal rVal wCount
sCount

…

  event write refines write
    any p1  p2
    where
      p1 = wVal
      p2 = sCount + 1
      buff < 0
    then
      buff ≔ p1
      sCount ≔ sCount + 1
      wCount ≔ p2
  end
```

was buff ≔ wVal

“The parameter wVal”

Parameterised for Decomposition (ii)

```
machine ReadWriteBuffer
refines AbstractBuffer

variables buff wVal rVal wCount
  sCount


…

  event write refines write
    any p1  p2
    where
      p1 = wVal
      p2 = sCount + 1
      buff < 0
    then
      buff ≔ p1
      sCount ≔ sCount + 1
      wCount ≔ p2
  end
```

was wCount ≔ sCount + 1

“The parameter: sCount + 1”

Decomposed Machines

```
machine Writer

variables wVal wCount

…

  event write
    any p1_out  p2_in
    where
      p2_in ∈ ℤ
      p1_out ∈ ℤ
      p1_out = wVal
    then
      wCount ≔ p2_in
  end
```

```
machine Shared

variables buff sCount

…

  event write
    any p1_in p2_out
    where
      p2_out ∈ ℤ
      p1_in ∈ ℤ
      p2_out = sCount + 1
      buff < 0
    then
      buff ≔ p1_in
      sCount ≔ sCount + 1
  end
```

- Parameter renaming is for clarity only,
  - but parameters will be ‘paired’ in order of declaration for translation.

Adding the Tasking Constructs

```
tasking machine Writer
priority 5
tasktype triggered
variables wVal wCount

…

body
  w1: ◁ write || Shared.write ▷ ;
  w2: …

  event sync write
    any
      out p1_out
      in p2_in
    where
      p2_in ∈ ℤ
      p1_out ∈ ℤ
      p1_out = wVal
    then
      wCount ≔ p2_in
  end
```

```
machine Shared

variables buff sCount

  …

  event write
    any
      in p1_in
      out p2_out
    where
      p2_out ∈ ℤ
      p1_in ∈ ℤ
      p2_out = sCount + 1
      buff < 0
    then
      buff ≔ p1_in
      sCount ≔ sCount + 1
  end
```

The Resulting Event-B model

```
machine Writer refines Writer
sees autoGenCTX_Writer

variables
  wVal wCount wCount2 Writer_pc

Invariants
  …
  Writer_pc∈ Writer_pc_Set

events
  event write refines write
    any p1_out  p2_in
    where
      p2_in ∈ ℤ
      p1_out ∈ ℤ
      p1_out = wVal
      Writer_pc = w1
    then
      wCount ≔ p1_out
      Writer_pc ≔ w2
  end
```

```
machine Shared

variables buff sCount

invariants
  … // various typing

  event write
    any p1_in p2_out
    where
      p2_out ∈ ℤ
      p1_in ∈ ℤ
      p2_out = sCount + 1
      buff < 0
    then
      buff ≔ p1_in
      sCount ≔ sCount + 1
  end
```

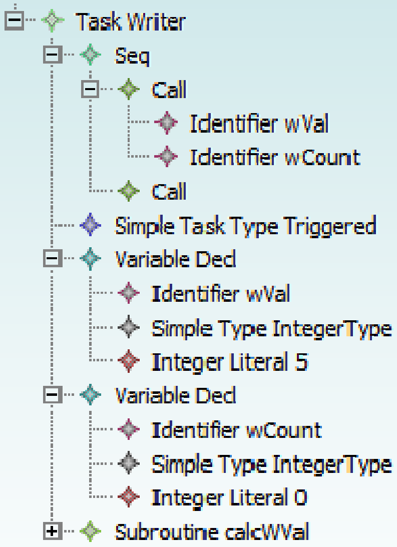
“Using Program Counters”

The Resulting Event-B model

```
machine Writer refines Writer
sees autoGenCTX_Writer

variables
  wVal wCount wCount2  write

Invariants
  …
 write ∈ BOOL

events
  event write refines write
    any p1_out  p2_in
    where
      p2_in ∈ ℤ
      p1_out ∈ ℤ
      p1_out = wVal
      write = TRUE
    then
      wCount ≔ p1_out
      write ≔ FALSE
  end
```

The Writer, Common Language Model

- Common Language Model
  - for further translation to AdaEMF etc.



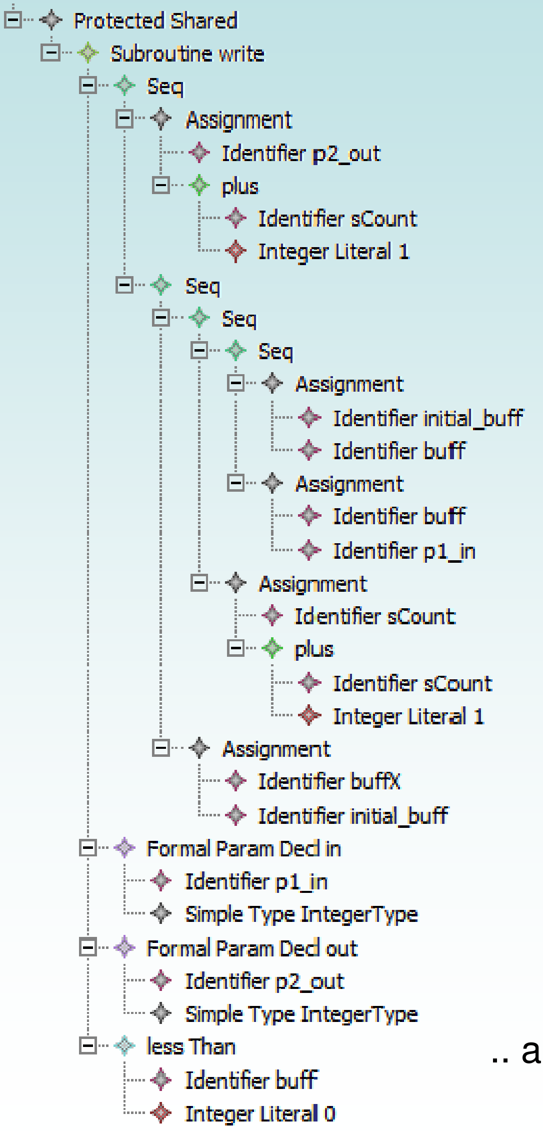
```
Task Writer
  Seq
    Call
      Identifier wVal
      Identifier wCount
    Call
  Simple Task Type Triggered
  Variable Decl
    Identifier wVal
    Simple Type IntegerType
    Integer Literal 5
  Variable Decl
    Identifier wCount
    Simple Type IntegerType
    Integer Literal 0
  Subroutine calcWVal
```

```
Task Writer
   taskType triggered
Declarations
   s: Shared
   wVal: Integer := 5
   wCount: Integer := 0

   Subroutine calcWVal(){
     wVal := wVal * 2
   }

Task Body
   s.write(wVal, wCount);
   …
```

.. or as pretty print

"CLM only needs to be machine readable"

The Shared, Common Language Model



```
Protected Shared
  Subroutine write
    Seq
      Assignment
        Identifier p2_out
        plus
          Identifier sCount
          Integer Literal 1
      Seq
        Seq
          Seq
            Assignment
              Identifier initial_buff
              Identifier buff
            Assignment
              Identifier buff
              Identifier p1_in
          Assignment
            Identifier sCount
            plus
              Identifier sCount
              Integer Literal 1
        Assignment
          Identifier buffX
          Identifier initial_buff
    Formal Param Decl in
      Identifier p1_in
      Simple Type IntegerType
    Formal Param Decl out
      Identifier p2_out
      Simple Type IntegerType
    less Than
      Identifier buff
      Integer Literal 0
```

.. as pretty print

```
Protected Shared
Declarations
   buff: Integer := -1
   sCount: Integer := 0
   buffX: Integer := -1

   Subroutine write(p1_in: in Integer,
                        p2_out: out Integer)
     when buff < 0 {
       p2_out := sCount + 1;
       initial_buff := buff;
       buff := p1_in;
       …
    }
   …
```

"Conditional waiting
in implementations"

TODO

- Common Language Metamodel V1 to AdaEMF translation,
  - for use with AdaEMF to AdaText Source Translator,
    - from Alexei in Newcastle.
- Testing and Evaluation of Common Language Metamodel V1 and tools.
- Version 2 of the Intermediate Language ??