UNIVERSITY OF Southampton
School of Electronics and Computer Science

Towards Modular Development in Event-B

Thai Son Hoang[1] Hironobu Kuruma[2] Michael Butler[1]

[1]ECS, University of Southampton, U.K.

[2]Research and Development Group, Hitachi Ltd., Japan

RODIN Workshop 2016

Linz, Austria

23rd May 2015

Background

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Event-B top-down development

- Refinement
- Decomposition

Pros and Cons

✓ Details can be introduced gradually into the formal model

✗ Large models with monolithic structures

Existing Composition Approaches

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Modularisation (Iliasov et. al., 2010)
- Feature composition (Poppleton, 2008)
- Parallel composition (Silva and Butler, 2009)
- Patterns (Hoang el. al., 2009)
- Generic instantiation (Silva and Butler, 2009)

The presented proposal borrows many ideas from these approaches.

The Aims

- Reuse models through composition.
  - Including refinement-chain of the sub-models.
- Integrated smoothly with the Event-B development process.
  - Accomodate changes seamlessly.

Machine Inclusion

Concepts

- Machine A includes machine B

  machine A
  includes B

  - A inherits B's variables
  - A inherits B's invariants
  - B's variables can only be modified via event synchronisation

- Multiple instances of B can be included via prefixing.

  machine A
  includes *p*_B

  - Variables, events are renamed accordingly.

Machine Inclusion

Illustration

machine B

variables y
invariants
  $J(y)$

events
 event f
  any u where
   $G_B(y, u)$
  then
   $y :| BAP_B(y, u, y')$
  end

Machine Inclusion

Illustration

machine B

variables y
invariants
$J(y)$

events
event f
any u where
$G_B(y, u)$
then
$y :\mid BAP_B(y, u, y')$
end

machine A

variables x

machine (flatten_)A

variables x

Machine Inclusion

Illustration

machine B

variables y
invariants
  $J(y)$

events
 event f
  any u where
   $G_B(y, u)$
  then
   $y :| BAP_B(y, u, y')$
  end

machine A
includes p_B
variables x

machine (flatten_)A

variables x , p_y

Machine Inclusion

Illustration

UNIVERSITY OF Southampton
School of Electronics and Computer Science

machine B

variables y
invariants
$J(y)$

events
event f
any u where
$G_B(y, u)$
then
$y :| BAP_B(y, u, y')$
end

machine A
includes p_B
variables x
invariants
$I(x, p\_y)$

machine (flatten_)A

variables x , p_y
invariants
$I(x, p\_y)$
$J(y)$

Machine Inclusion

Illustration

UNIVERSITY OF Southampton
School of Electronics and Computer Science

machine B

variables y
invariants
$J(y)$

events
event f
any u where
$G_B(y, u)$
then
$y :| BAP_B(y, u, y')$
end

machine A
includes p_B
variables x
invariants
$I(x, p\_y)$

events
event e
synchronises p_f
any t where
$G_A(x, t)$
$H_{AB}(x, p\_y, t, p\_u)$

then
$x :| BAP_A(x, t, x')$
end

machine (flatten_)A

variables x , p_y
invariants
$I(x, p\_y)$
$J(y)$

Machine Inclusion

Illustration

machine B

variables y
invariants
$J(y)$

events
event f
any u where
$G_B(y, u)$
then
$y :| BAP_B(y, u, y')$
end

machine A
includes p_B
variables x
invariants
$I(x, p\_y)$

events
event e
synchronises p_f
any t where
$G_A(x, t)$
$H_{AB}(x, p\_y, t, p\_u)$

then
$x :| BAP_A(x, t, x')$
end

machine (flatten_)A

variables x , p_y
invariants
$I(x, p\_y)$
$J(y)$

events
event e

any t, p_u where
$G_A(x, t)$
$H_{AB}(x, p\_y, t, p\_u)$
$G_B(p\_y, p\_u)$
then
$x :| BAP_A(x, t, x')$
$y :| BAP_B(p\_y, p\_u, p\_y')$
end

Refinement-Chain Inclusion

Concepts

University of Southampton
School of Electronics and Computer Science

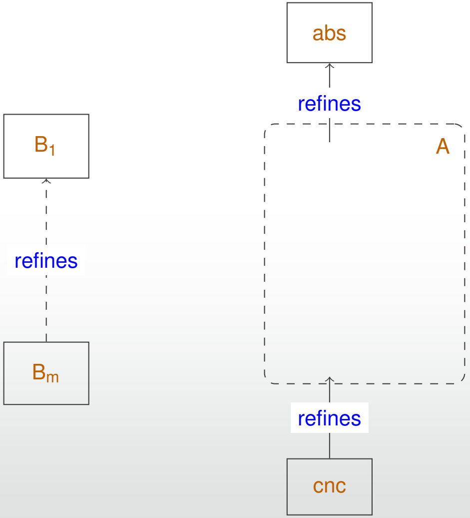
- Machine A includes refinement-chain $B_1 \longrightarrow B_m$.

  machine A
  includes $B_1 \longrightarrow B_m$

- A has double interfaces
  - To its abstract machine, A includes B_1
  - To its refinement, A includes B_m

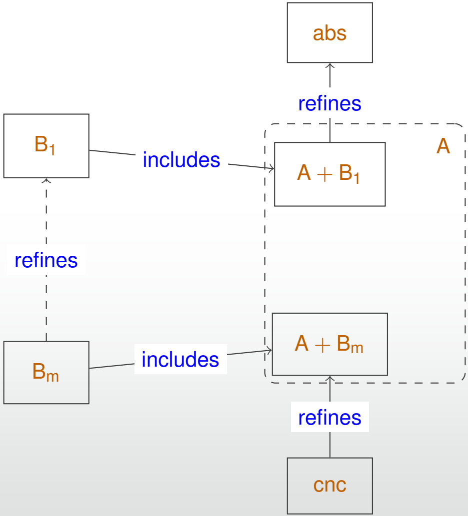
Refinement-Chain Inclusion

Illustration



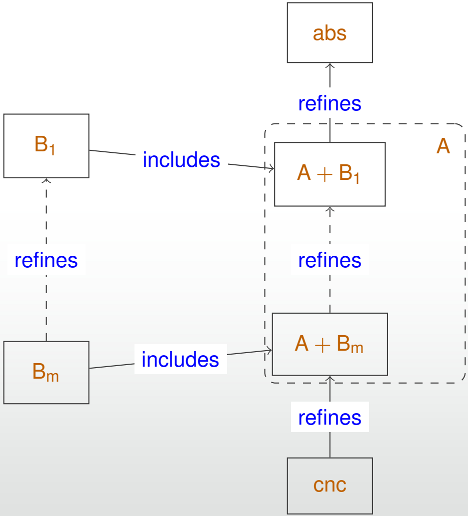
Refinement-Chain Inclusion

Illustration



Refinement-Chain Inclusion

Illustration



Refinement-Chain Inclusion

The Two "Interfaces"


```
machine A + B1

events
 event e
  any t, u1 where
   G_A(x, t)
   H_AB1(x, y1, t, u1)
   G_B1(y1, u1)
  then
   x :| BAP_A(x, t, x')
   y1 :| BAP_B1(y1, u1, y')
  end
```

```
machine A + Bm
refines A + B1
events
 event e
  any t, um where
   G_A(x, t)
   H_ABm(x, ym, t, um)
   G_Bm(ym, um)
  then
   x :| BAP_A(x, t, x')
   ym :| BAP_Bm(ym, um, y')
  end
```

The refinement is "almost" correct-by-construction.

Refinement-Chain Inclusion

The Two "Interfaces"

machine A + B_1

events
 event e
  any t, u_1 where
   $G_A(x, t)$

  then
   $x :| BAP_A(x, t, x')$

  end

machine A + B_m
refines A + B_1
events
 event e
  any t, u_m where
   $G_A(x, t)$

  then
   $x :| BAP_A(x, t, x')$

  end

The refinement is "almost" correct-by-construction.

- Guard-strenthening for G_A is trivial
- Action-strengthing for BAP_A is trivial

Refinement-Chain Inclusion

The Two "Interfaces"

machine A + B_1

events
 event e
  any t, u_1 where

  $G_{B1}(y_1, u_1)$
 then

  $y_1 :| BAP_{B1}(y_1, u_1, y')$
 end

machine A + B_m
refines A + B_1
events
 event e
  any t, u_m where

  $G_{Bm}(y_m, u_m)$
 then

  $y_m :| BAP_{Bm}(y_m, u_m, y')$
 end

The refinement is "almost" correct-by-construction.

- Guard-strengthening for G_{B1} is guaranteed by B_m refines B_1
- Action-strengthening for BAP_{B1} is guaranteed by B_m refines B_1

Refinement-Chain Inclusion

The Two "Interfaces"

machine A + B_1	machine A + B_m
	refines A + B_1
events	events
event e	event e
any t, u_1 where	any t, u_m where
$H_{AB1}(x, y_1, t, u_1)$	$H_{ABm}(x, y_m, t, u_m)$
then	then
end	end

The refinement is "almost" correct-by-construction.

- Guard-strengthening for H_{AB1} needs to be proved manually.

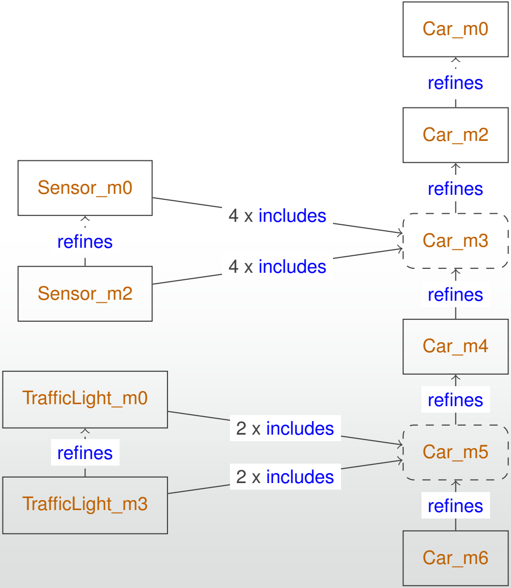
Some Evaluations

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- We have applied the idea to several examples.
- Resulting in hierarchical development.
- Modelling effort is reduced.
- Proving effort is reduced.
- Easier to understand specification.

Example. Car on a Bridge

Development hierarchy



Machine Car_m6

machine Car_m6
includes
 ML_out_Sensor_m2
 ML_in_Sensor_m2
 IL_out_Sensor_m2
 IL_in_Sensor_m2

 ML_TrafficLight_m3
 IL_TrafficLight_m3

Event ML_in

UNIVERSITY OF Southampton
School of Electronics and Computer Science

```
(flatten_)ML_in
 when
  ML_in_SNSR = TRUE
  ML_in_Snsr_01 = FALSE
  ML_in_ctrl_Snsr_01 = FALSE
  ML_in_DEP ≠ IL_out_DEP
 then
  ML_in_SNSR := FALSE
  ML_in_DEP := ML_in_DEP + 1
  ML_in_Snsr_10 := TRUE
 end
```

```
event ML_in
 synchronises ML_in_SNSR_off
 when
  ML_in_DEP ≠ IL_out_DEP
 then
  SKIP
 end
```

Summary

UNIVERSITY OF Southampton
School of Electronics and Computer Science

- Inclusion of refinement-chain
- Syntactical
- Tool support is needed.
- Reference machines/contexts from another project
  (see the next talk).