UNIVERSITY OF Southampton

ECS Electronics & Computer Science University of Southampton

Composition, Renaming and Generic Instantiation in Event-B Development

Renato Silva
ras07@ecs.soton.ac.uk

Supervisor: Michael Butler

DSSE Dependable Systems & Software Engineering

17/07/2009

Overview

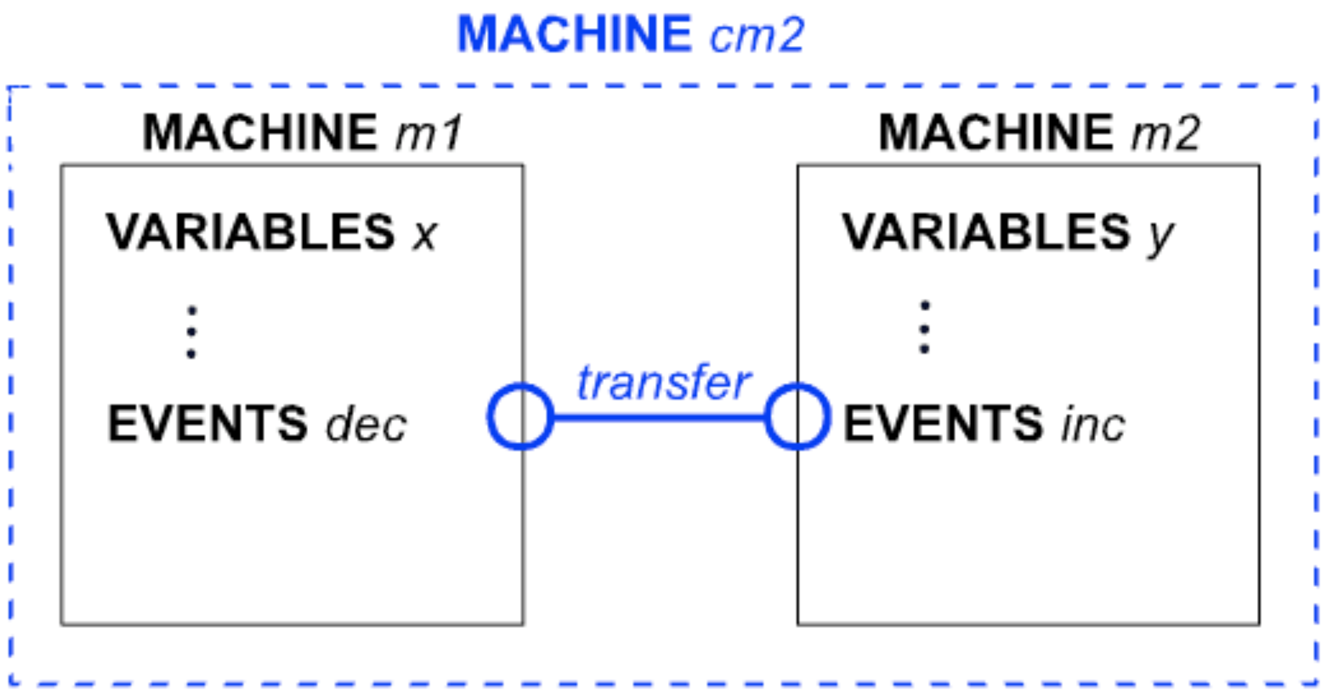
- Simple Composition Model
- Composition
- Renaming/Refactory
- Generic Instantiation
- Conclusions
- Future Work

Machine M1 and Machine M2

MACHINE m1
VARIABLES
  x
INVARIANTS
  $inv1: x \in \mathbb{N}$
EVENTS
Initialisation
  begin
    $act1: x := 100$
  end
Event dec $\widehat{=}$
  any
    i
  where
    $grd2: x > 0$
    $grd1: i \in 1 \,..\, x$
  then
    $act1: x := x - i$
  end
END

MACHINE m2
VARIABLES
  y
INVARIANTS
  $inv1: y \in \mathbb{N}$
EVENTS
Initialisation
  begin
    $act1: y := 0$
  end
Event inc $\widehat{=}$
  any
    i
  where
    $grd1: i \in \mathbb{N}$
  then
    $act1: y := y + i$
  end
END

Simple Composition



Simple Composition

MACHINE cm1

VARIABLES

x

y

INVARIANTS

$inv1 : x \in \mathbb{N}$

$inv2 : y \in \mathbb{N}$

$inv3 : x + y = 100$

EVENTS

Initialisation

begin

$act1 : x := 100$

$act2 : y := 0$

end

Event transfer $\widehat{=}$

any

i

where

$grd1 : x > 0$

$grd2 : i \in 1 \mathrel{..} x$

then

$act1 : x := x - i$

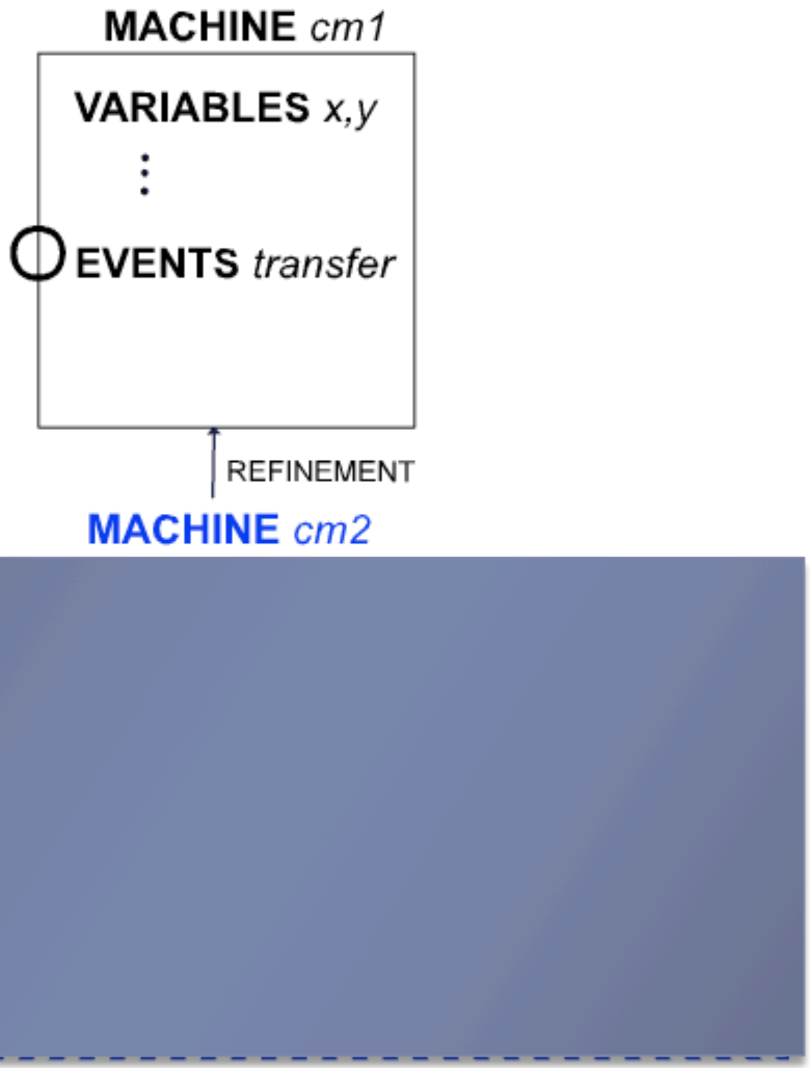
$act2 : y := y + i$

end

END

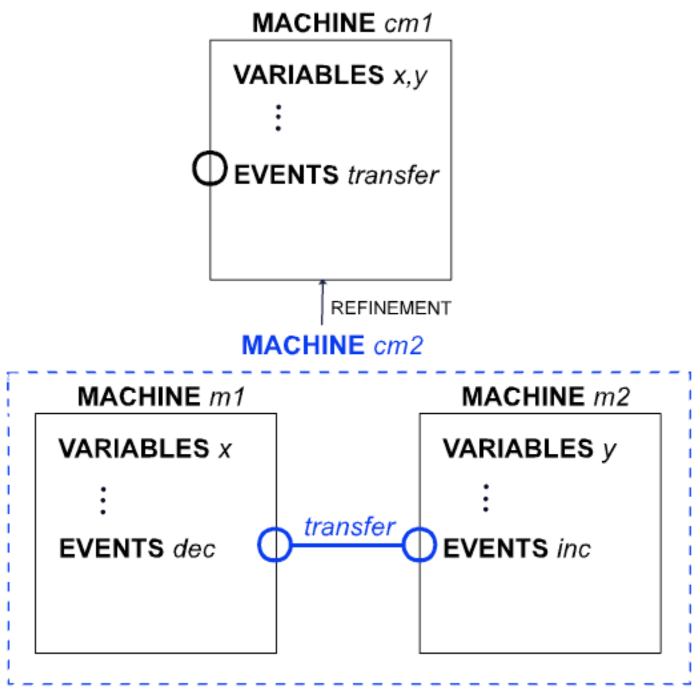
Machine *cm1* – Abstract Machine

Simple Composition



Demonstration

- Demonstration of the Composition plug-in for Rodin Platform using the Simple Composition



Composition

- Allows aggregation of sub-systems and generate a larger system (interaction between sub-systems)
- Reusability of systems that are already created and validated
- Sub-systems may be refined independently

UNIVERSITY OF Southampton

Composition Plug-in

- Shared Event Composition: systems are composed through synchronisation of events
- Properties of each machine are merged (Contexts, Variables,…)
- Conjunction of the invariant predicates
- Composed events:
  - Merged parameters
  - Conjunction of guards
  - Assignment of actions are done in parallel

- $evt3 \mathrel{\widehat{=}} \textbf{ANY } t?, x \textbf{ WHERE } t? \in A \wedge G(t?, x, m) \textbf{ THEN } S(t?, x, m) \textbf{ END}$
- $evt4 \mathrel{\widehat{=}} \textbf{ANY } t!, y \textbf{ WHERE } H(t!, y, n) \textbf{ THEN } T(t!, y, n) \textbf{ END}$
- $evt3 \parallel evt4 \mathrel{\widehat{=}}$
  $\textbf{ANY } t!, x, y \textbf{ WHERE } t! \in A \wedge G(t!, x, m) \wedge H(t!, y, n) \textbf{ THEN } S(t!, x, m) \parallel T(t!, y, n) \textbf{ END}$

Semantics of Composition

- Event-B has the same semantics structure and refinement definitions as Action Systems
- It is possible to make a correspondence between parallel composition in CSP and an event-based view of parallel composition for Action Systems
- A failure-divergence definition (CSP) can be applied to Event-B machines

$S \in Machine \rightarrow FD$ where *FD* is the set of Failure-Divergence for *Machine*

$PAR(P,Q)$ where $P,Q \in FD$ (function that defines the semantics of the process P||Q in CSP)

$S(M \parallel N) = PAR(S(M), S(N))$

- *PAR* is monotonic, so machines M and N can be refined independently

Renaming/Refactory: WHY?

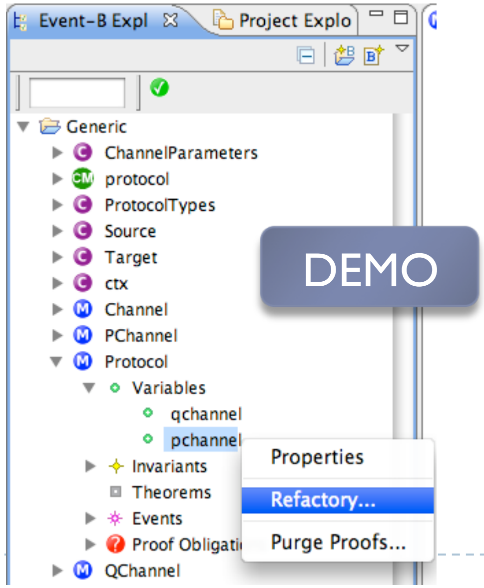
- Shared Event Composition constraint:
  - No shared variables
- If machines to be composed have the same variable name, it is necessary to rename (at least) one of the variables
- Occurrences of variables in other elements need to reflect renaming (invariants, actions, guards,…)
- Occurrences also need to propagate over related files like refinements…
- (Long time) request by Event-B developers in the Rodin Platform

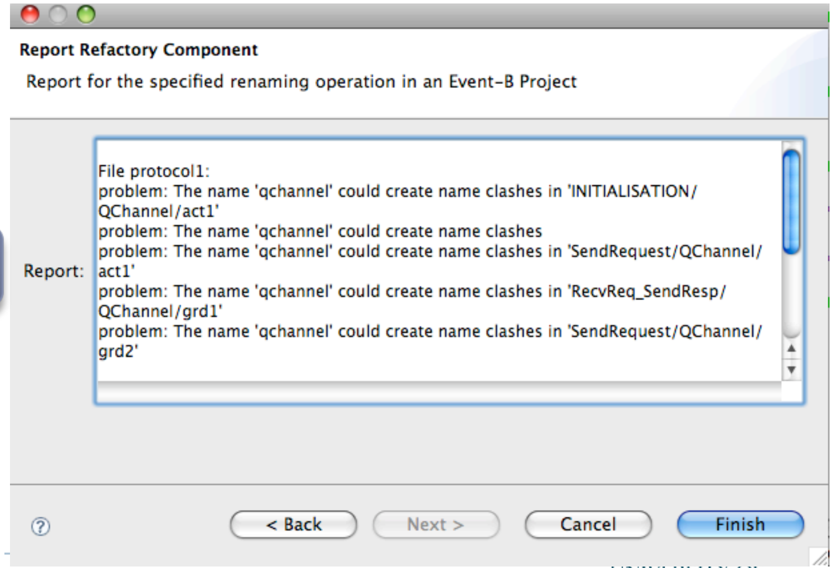
Renaming/Refactory plug-in

- Renaming/Refactory plug-in allows the renaming of variables but not only:
  - Carrier Sets
  - Constants
  - Events
  - Labelled elements (invariants, axioms, guards, etc)
- Uses Rodin Indexer to accelerate search of elements and be more accurate
- Goal: refactory of elements should not affect the behaviour of machines/contexts - no change at the semantic level

Renaming/Refactory plug-in

- How it works?
  - User selects element to be renamed
  - User introduces new element name
  - A list of related files is created
  - Looks for possible clashes and returns a report
  - User decides if he wants to execute renaming





Renaming/Refactory plug-in

- Prototype (not a final version)
- Available to install from Rodin Update Site in version Rodin 1.0

- Limitation
  - Renaming is not applied to proofs obligations (but the intention is to be applied in the future)

Generic Instantiation

- Generic Instantiation reuses components and tries to solve difficulties raised by the construction of large models
- We propose a Generic Instantiation approach for Event-B by instantiating machines.
- Instances inherit properties from the *pattern* and *personalised* it by renaming/replacing those properties to more specific names to the instance.
- This approach uses the Renaming plug-in

Generic Instantiation

- Generation of proof obligations to ensure assumptions (axioms) used in the patterns are satisfied in the instance.
- *Contexts* work as Parameterisation of *instantiated machines*

Instantiated Machine

CONTEXT Ctx
SETS $S_1 \ldots S_m$
CONSTANTS $C_1 \ldots C_n$
AXIOMS $Ax_1 \ldots Ax_p$

MACHINE M
SEES Ctx
VARIABLES $v_1 \ldots v_q$
EVENTS $ev_1 \ldots ev_r$

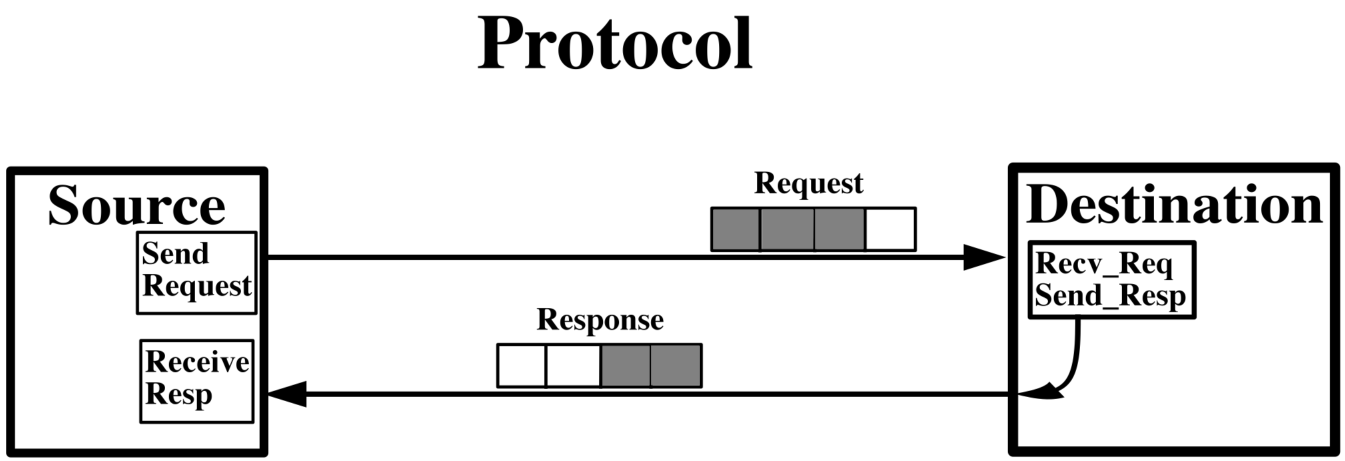
INSTANTIATED MACHINE IM
INSTANTIATES M **VIA** Ctx
SEES D /* context containing the instance properties */
REPLACE /* replace parameters defined in context C */
  SETS $S_1 := DS_1, \ldots, S_m := DS_m$ /* Carrier Sets or Constants */
  CONSTANTS $C_1 := DC_1, \ldots, C_n := DC_n$
RENAME /* rename variables, events and parameters at machine M */
  VARIABLES $v_1 := nv_1, \ldots, v_q := nv_q$
  EVENTS $ev_1 := nev_1, \ldots, ev_r := nev_r$ /* optional */
    $p_1 := np_1, \ldots, p_s := np_s$ /* parameters: optional */
END

Instantiated Machine

- An INSTANTIATED MACHINE that instantiates a generic machine (*pattern*):
  - Has a name.
  - Defines which machine is used as generic.
  - Defines which variables and events are renamed.
  - Elements (sets and constants) from seen contexts are replaced by instance elements.
- Axioms in the pattern are converted into theorems in the instance.
  - The proof obligations associated with theorems assure that the assumptions in the pattern are satisfied in the instance.

Instantiated Machine: example

- Case study: Model a Protocol (*Problem*)



Generic Context and Machine: *Pattern*

CONTEXT ChannelParameters

SETS

$\quad$ ***Message***

CONSTANTS

$\quad$ ***max_size***

AXIOMS

$\quad$ ***axm1*** : $max\_size \in \mathbb{N}$

END

MACHINE Channel

SEES ChannelParameters

VARIABLES

$\quad$ ***channel***

INVARIANTS

$\quad$ ***inv1*** : $channel \subseteq Message$

$\quad$ ***inv2*** : $card(channel) \leq max\_size$

EVENTS

Initialisation

$\quad$ **begin**

$\quad\quad$ ***act1*** : $channel := \varnothing$

$\quad$ **end**

Event *Send* $\widehat{=}$

$\quad$ **any**

$\quad\quad$ ***m***

$\quad$ **where**

$\quad\quad$ ***grd1*** : $m \in Message$

$\quad\quad$ ***grd2*** : $card(channel) < max\_size$

$\quad$ **then**

$\quad\quad$ ***act1*** : $channel := channel \cup \{m\}$

$\quad$ **end**

Event *Receive* $\widehat{=}$

$\quad$ **any**

$\quad\quad$ ***m***

$\quad$ **where**

$\quad\quad$ ***grd1*** : $m \in channel$

$\quad$ **then**

$\quad\quad$ ***skip***

$\quad$ **end**

END

Instantiated Machine

- Context is used as Parameterisation of machines where the instance properties are defined

CONTEXT ProtocolTypes

SETS

Request

Response

CONSTANTS

$qmax\_size$

$pmax\_size$

AXIOMS

$axm1$: $qmax\_size \in \mathbb{N}$

$axm2$: $pmax\_size \in \mathbb{N}$

END

CONTEXT ChannelParameters

SETS

Message

CONSTANTS

max_size

AXIOMS

$axm1$: $max\_size \in \mathbb{N}$

END

Instantiated Machine: QChannel

INSTANTIATED MACHINE QChannel
INSTANTIATES Channel **VIA** ChannelParameter
SEES ProtocolTypes /* context containing the instance properties */
REPLACE /* replace parameters defined in ChannelParameters */
  SETS *Message := Request*
  CONSTANTS *max_size := qmax_size*
RENAME /* rename variables and events at Channel machine */
  VARIABLES *channel := qchannel*
  EVENTS *Send := QSend*
    m := q
   Receive := Receive
    m := q
END

```
machine Channel sees ChannelParameters

variables channel

invariants
  @inv1 channel ⊆ Message
  @inv3 finite(channel)
  @inv2 card(channel) ≤ max_size

events
  event INITIALISATION
    then
      @act1 channel ≔ ∅
  end

  event Send
    any m
    where
      @grd1 m ∈ Message
      @grd2 card(channel) < max_size
    then
      @act1 channel ≔ channel ∪ {m}
  end

  event Receive
    any m
    where
      @grd1 m ∈ channel
  end
end
```

```
machine QChannel sees ProtocolTypes

variables qchannel

invariants
  @inv1 qchannel ⊆ Request
  @inv3 finite(qchannel)
  @inv2 card(qchannel) ≤ qmax_size
  theorem @thm1 qmax_size ∈ ℕ

events
  event INITIALISATION
    then
      @act1 qchannel ≔ ∅
  end

  event QSend
    any q
    where
      @grd1 q ∈ Request
      @grd2 card(qchannel) < qmax_size
    then
      @act1 qchannel ≔ qchannel ∪ {q}
  end

  event Receive
    any q
    where
      @grd1 q ∈ qchannel
  end
end
```

Instantiated Machine + Composition

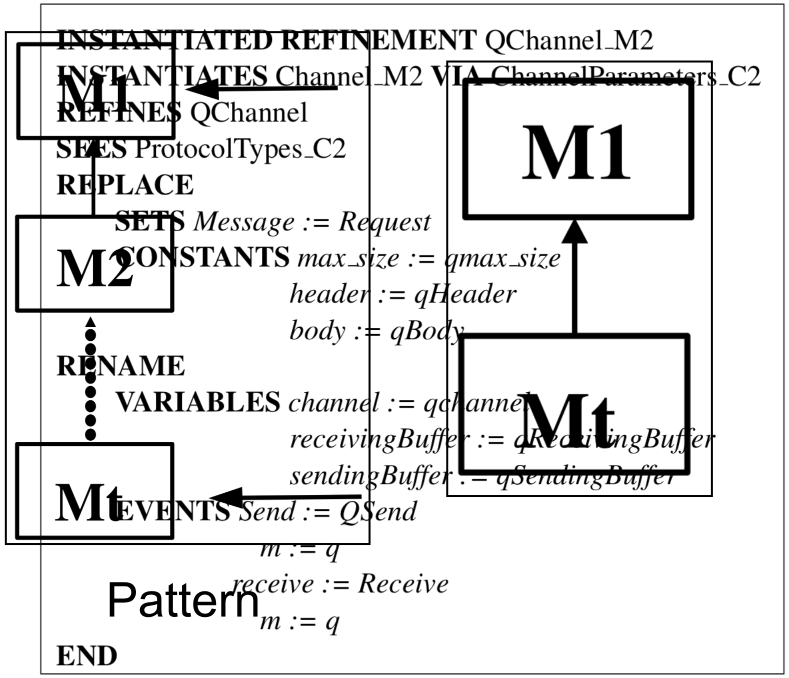
```
machine Protocol sees ProtocolTypes

variables qchannel pchannel

invariants
  @inv1 qchannel ⊆ Request
  @inv2 pchannel ⊆ Response
  @inv3 card(pchannel) ≤ pmax_size
  @inv4 card(qchannel) ≤ qmax_size
  theorem @QChannel/thm1 qmax_size ∈ ℕ
  theorem @PChannel/thm2 pmax_size ∈ ℕ

events
  event INITIALISATION
    then
      @act1 qchannel ≔ ∅
      @act2 pchannel ≔ ∅
  end
```

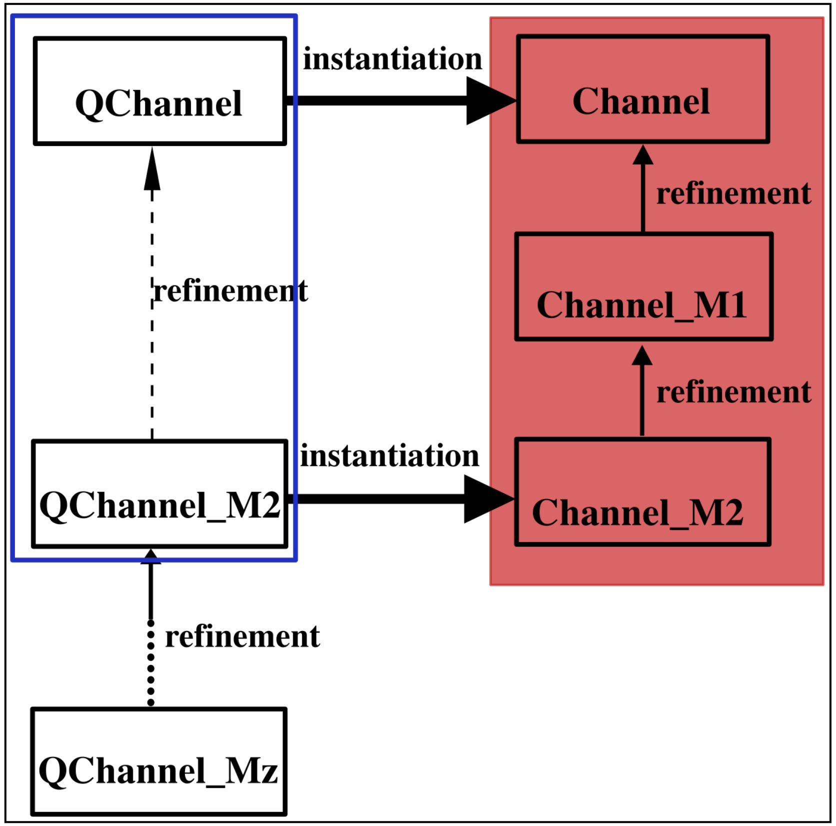
```
event SendRequest
  any q
  where
    @grd1 q ∈ Request
    @grd2 card(qchannel) < qmax_size
  then
    @act1 qchannel ≔ qchannel ∪ {q}
end

event RecvReq_SendResp
  any q p
  where
    @grd1 q ∈ qchannel
    @grd2 p ∈ Response
    @grd3 card(pchannel) < pmax_size
  then
    @act1 pchannel ≔ pchannel ∪ {p}
end

event RecvResp
  any p
  where
    @grd1 p ∈ pchannel
end
```

COMPOSED MACHINE Protocol
INCLUDES
PChannel
QChannel
EVENTS
SendRequest
Combines Events QChannel.QSend
RecvReq_SendResp
Combines Events QChannel.Receive || PChannel.Send
RecvResp
Combines Events combines PChannel.Receive
END

Instantiation of a chain of refinements

- Expand notion of reuse to a chain of refinements
- Creation of *Instantiated Refinements*





Instantiated Machine/Refinement: Instantiating Theorems and Invariants

- Invariants define model properties in machines
- Theorems work as assertions
  - If theorem proof obligation is discharged, the same should happen in the instance: no re-proving
- Ideally:
  - add to the instance the assumptions and assertions given by the theorems and invariants without the hassle of re-proving them.
- Possible solution: proved-theorem, similar to a theorem but without a proof obligation associated

UNIVERSITY OF Southampton

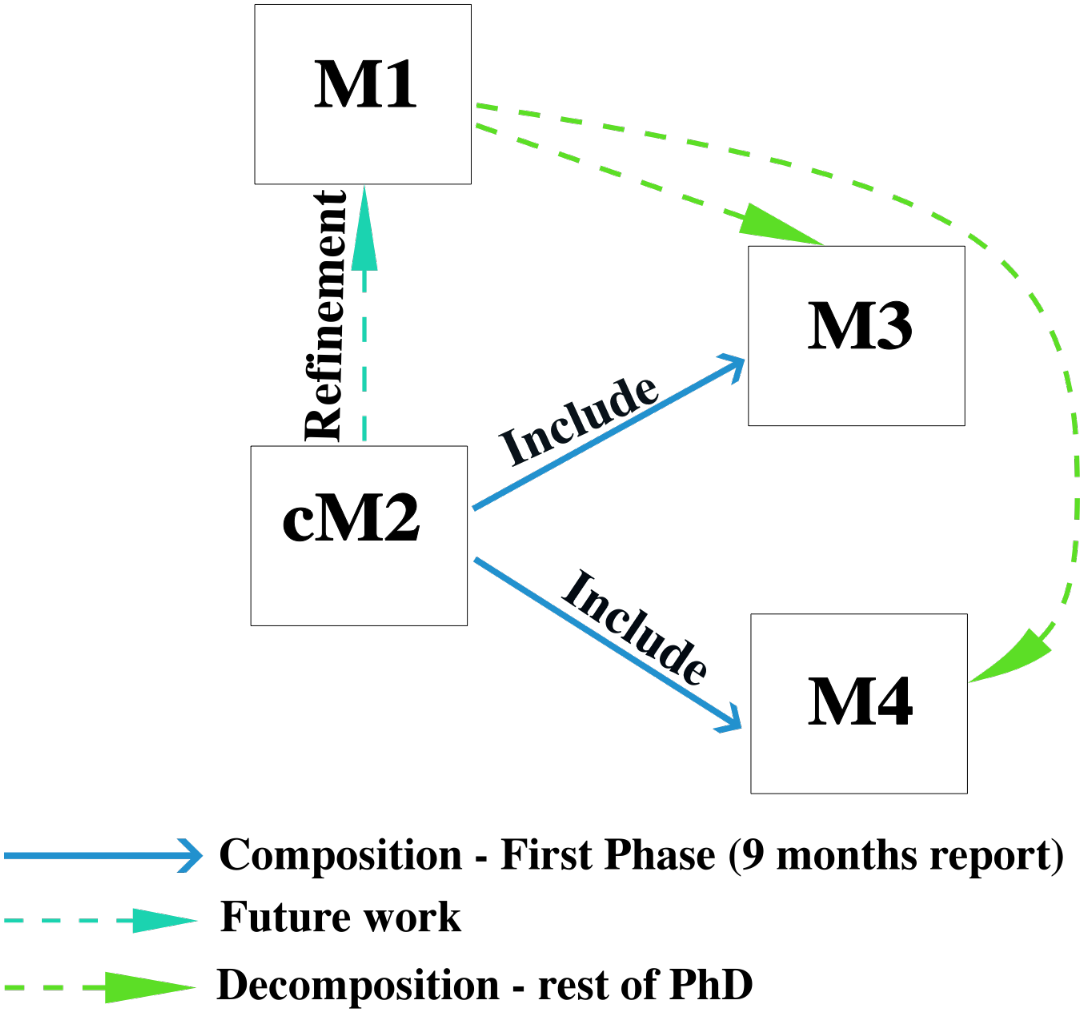
Conclusions

- Composition plug-in: Ability to apply shared event parallel composition of machines using Event-B
- Rodin works as a modelling tool support
- Renaming plug-in: renaming of elements in Event-B models using Rodin
- Generic Instantiation: proposal for instantiation of machines

Future Work

- Validation of output machine in the composed machine file while using the Composition plug-in
- Development in the Rodin platform of the generic instantiation – Instantiated Machines
- Instantiated Contexts??
- Study of **Decomposition** (can be considered the inverse operation of Composition) using Event-B and the Rodin platform.
  - **Shared Event:** event-based viewpoint
  - Application of case-studies

Future Work


M1
M3
cM2
M4
Refinement
Include
Include
Composition - First Phase (9 months report)
Future work
Decomposition - rest of PhD

The End

QUESTIONS???

THANK YOU

FOR

YOUR ATTENTION