NWPT2015

Towards Component-based Reuse for Event-B

Andy Edmunds

Åbo Akademi, Turku, Finland

aedmunds@abo.fi

Event-B

- A formal methodology + language.
  - Uses abstraction and non-determinism.
- **Rodin** is the tool.
- The mathematical underpinning,
  - is based on set theory and predicate logic.
  - can provide a precise description of a *system*.
  - uses stepwise development (refinement).
  - can be partly “hidden” by graphical notations.

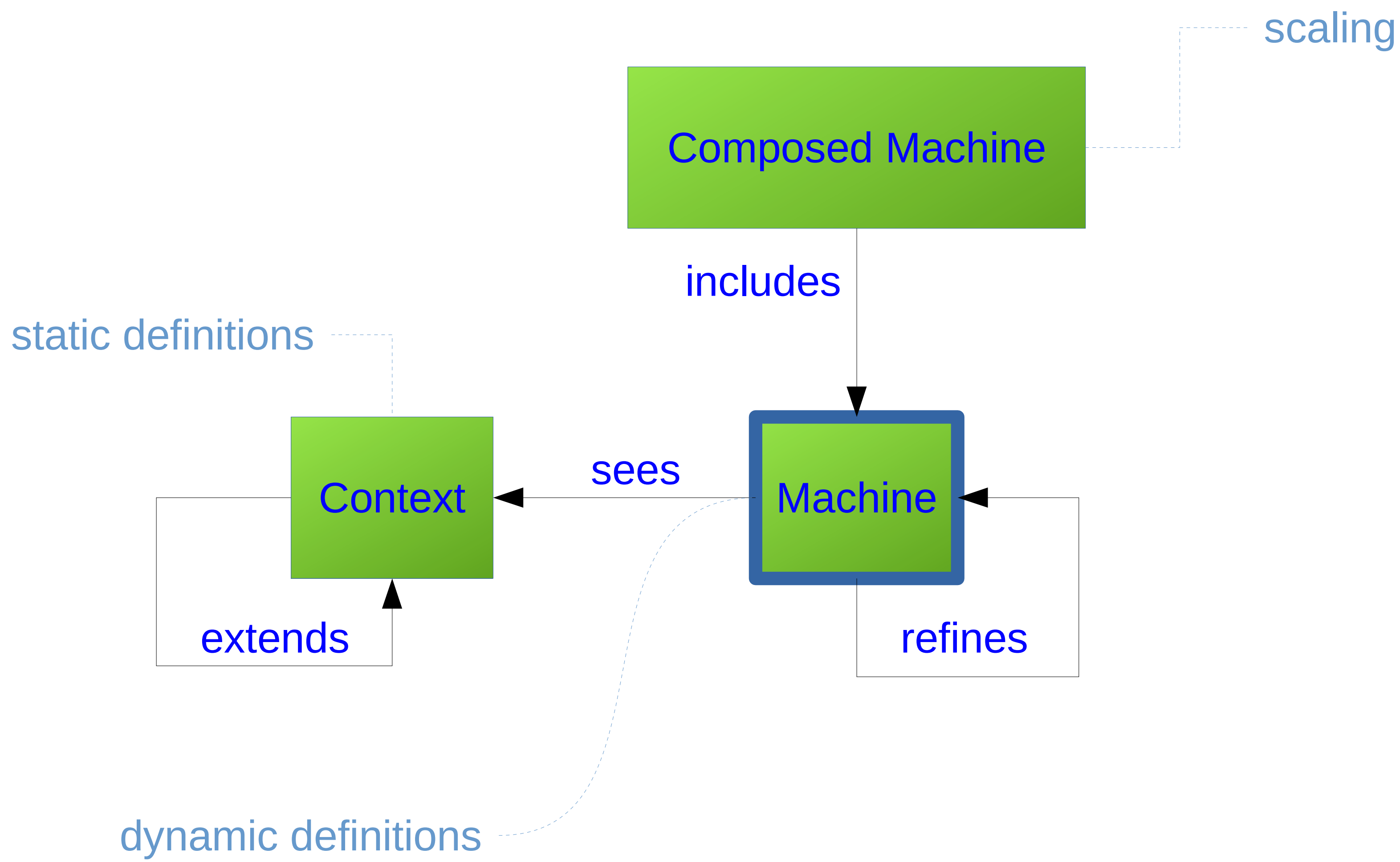
Event-B is for?

- State-based systems modelling,
  - aimed at High Integrity Systems.
  - We specify important 'invariant' properties.
  - Show that state updates don't violate these properties.
  - Show that these properties hold as development progresses.
  - Uses proof and/or model checking.
- ADVICeS Project - More Agility for Event-B!
  - Looking at the engineering process.

Event-B Elements

- Contexts
  - Describing static parts of the system.
  - Have Sets, Constants and Axioms.
- **Machines**
  - Describe the dynamic parts.
  - Have Variables, Invariants and Events.
- **Events**
  - Have parametrised, guarded, atomic state updates.
- **Composed-machines**
  - for structuring and scalability.
- Refinement
  - gradual introduction of detail.

Event-B Artefacts



Why Components?

- Build on Composed-machine features.
- To improve bottom-up scalability.
- To improve 'agility'
  - through reuse of Event-B machines,
    - by defining component interfaces.
      - describing communication flow across component boundaries.
      - adding additional proofs obligations.
    - by adding a component instance diagram.
      - extending iUML-B.
      - adding new Event-B 'generators'.
- Facilitate a searchable library (of components).

Event-B – Events (i)

$e \triangleq$ **ANY** p **WHERE** $G(p, s, c, v)$ **THEN** $A(p, s, c, v)$ **END**

- Event
  - Name e; Parameters p; Guards G; Actions A
- Context
  - Sets s; Constants c
- Machine
  - Variables v

Event-B – Events (ii)

$e \triangleq$ **ANY** p **WHERE** $G(p, s, c, v)$ **THEN** $A(p, s, c, v)$ **END**

- Parameters p
  - models parameters and local variables.
- Guards G
  - blocking predicate.
- Actions A
  - deterministic assignments :=
  - non-deterministic assignments

An Event-B Machine

```
machine M0
sees
 C0
variables
pointPos
invariants
@inv1 "pointPos ∈ pointState"
events
 event INITIALISATION  ordinary
 then @act1 "pointPos :∈ pointState"
 end
 event movePoint
 when @grd0_1 "pointPos = lastKnown"
 then @act0_1 "pointPos ≔ updated"
 end

 event reset ordinary
  when @grd0_1 "pointPos = updated"
  then @act0_1 "pointPos ≔ lastKnown"
 end

end
```

specify properties

atomic, guarded
state updates

Annotating Event Parameters

$$
\begin{array}{ll}
e \triangleq & \textbf{ANY}\ p?\ p!\ x \\
& \textbf{WHERE}\ G(p, x, v) \\
& \textbf{THEN}\ A(p, x, v) \\
& \textbf{END}
\end{array}
$$

- '?' and '!' are just in/out *mode* specifiers in the parameter declaration,
  - not part of the name.
- Input parameters $p?$
- Output parameters $p!$
- Local variables x
- All Parameters $p = p? \cup p!$

“Ignoring Sets and Constants”

Composition Semantics

Composed Machine

MACHINE a

VARIABLES v_a

$e_a \triangleq$ **ANY** $p?_a$, $p!_a$, x_a
WHEN G_a (p_a , x_a , v_a)
THEN A_a (p_a , x_a , v_a)
END

MACHINE b

VARIABLES v_b

$e_b \triangleq$ **ANY** $p?_b$, $p!_b$, x_b
WHEN G_b (p_b , x_b , v_b)
THEN A_b (p_b , x_b , v_b)
END

$\approx$

MACHINE a || b

VARIABLES v_a, v_b

$e_a \parallel e_b \triangleq$ **ANY** p, x_a , x_b
WHEN G_a (p, x_a , v_a) $\wedge$ G_b (p, x_b , v_b)
THEN A_a (p, x_a , v_a) || A_b (p, x_x , v_b)
END

'Reduced' parameter set

“Ignoring Sets and Constants”

Parameter matching

In a single machine, parameter set $p = p? \cup p!$

Parameters q are typed: $q? \in p? \wedge q! \in p!$

In a composition, parameters are typed:

$$q?_a \in p?_a \wedge q!_b \in p!_b \wedge q?_b \in p?_b \wedge q!_a \in p!_a$$

Matching input/output parameters 'reduce',

$(q = q!_a \,||\, q?_b)$ and $(q = q!_b \,||\, q?_a)$

so that, in the composition, p consists of reduced parameters q,

$$q \in p$$

Communicating Events

(A Concrete Example)

Composed Machine

MACHINE a

VARIABLES A $\in T$

$e_a \triangleq$ **ANY prm?**

WHEN prm $\in T$

THEN A := prm

END

MACHINE b

VARIABLES B $\in T$

$e_b \triangleq$ **ANY prm!**

WHEN prm = B ^ prm $\in T$

THEN SKIP

END

$\approx$

Combined event

MACHINE a || b

VARIABLES A $\in T$, **B** $\in T$

$e_a \parallel e_b \triangleq$ **ANY prm**

WHEN prm = B ^ prm $\in T$

THEN A := prm

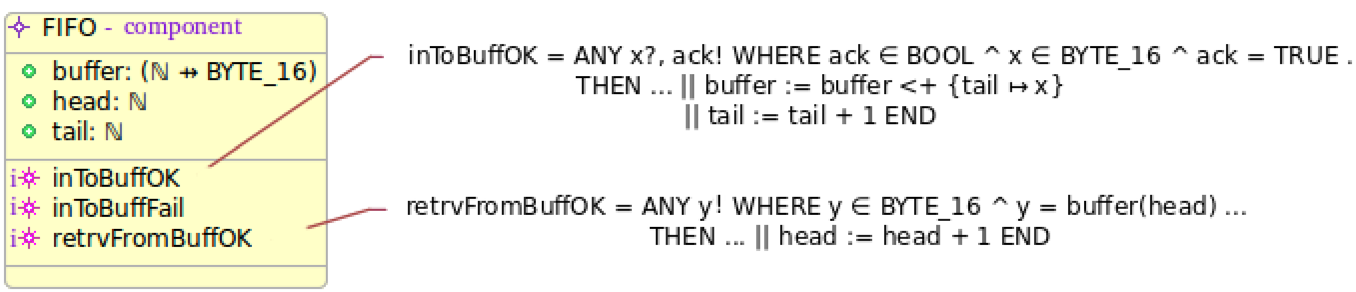
END

So,

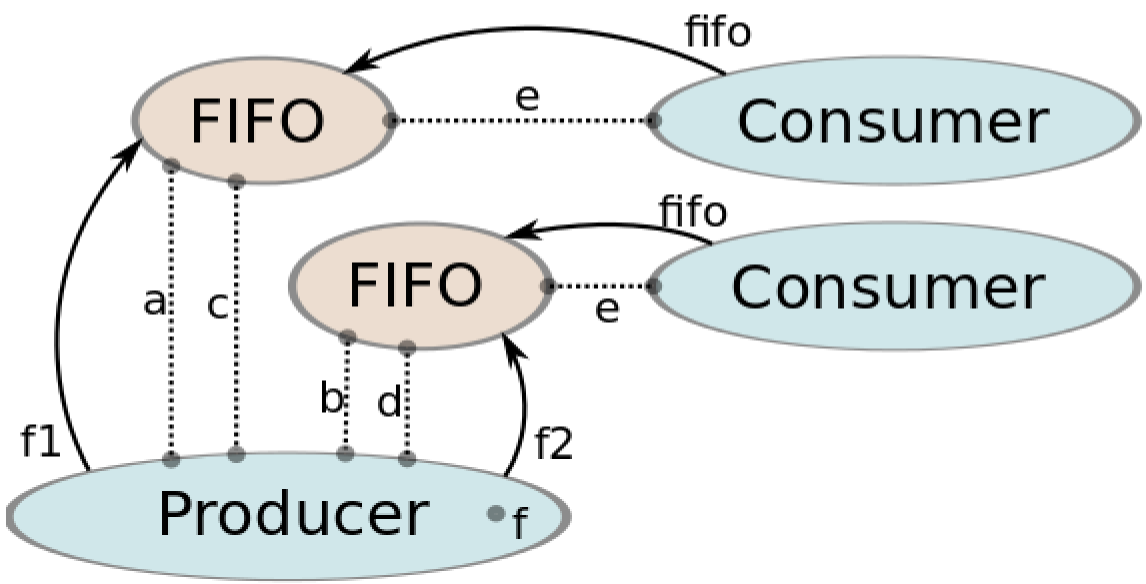
A := **B**

Interface Description (iUML-B)

- Adapted **iUML-B** Class Diagram
  - Identifies a component.
  - Identifies interface events.
  - *(Identifies parameter direction)*.
- FIFO buffer example ...



Component Instance Diagram



Combined Events:
a) <Machine>.<Event> ||<Machine>.<Event>
b) <Machine>.<Event>
....

Machine Invariants

- Invariants – state the required system properties.
- Invariant I of machine a ranges over a machine's sets, constants and variables,

$$I_a(s_a , c_a , v_a)$$

- But it cannot refer to those of another machine.
- A *Composition Invariant* is required.

Composition Invariants

- The Composition Invariant CI,
  - is part of the composed-machine.
  - specifies properties between *internal* elements of *included* machines.
  - ranges over all variables v in a composition.
  - ranges over all included sets and constants, s and c.

- The Composed-Machine Invariant CMI,
  - is formed from the composed-machine CM's composition invariant CI,
  - … and invariants $MI_0..MI_m$ of machines $M_0..M_m$.

$$CMI(CM, M_0 .. M_m) = CI(s, c, v) \wedge MI_0 (s_0 , c_0 , v_0) \wedge .. \wedge MI_m (s_m , c_m , v_m)$$

Combined Event Guard

- We need to add guards to the Combined Event 'Clause'
  - to satisfy the Composition Invariant.
  - remember, combined events reside in the composed machine.
  - The resulting combined event follows,

$e_a \parallel e_b \triangleq$

ANY p, x_a, x_b

WHERE $G_{CI}(v) \wedge G_a(p, x_a, v_a) \wedge G_b(p, x_b, v_b)$

THEN $A_a(p, x_a, v_a) \parallel A_b(p, x_b, v_b)$

END

“Ignoring Sets and Constants”

The New Proof Obligation

- We want to show that the invariant still holds for $e_j \| e_k$

$$
\begin{aligned}
INVe_j \| e_k : \quad & CI(v) \wedge I_j(v_j) \wedge I_k(v_k) \\
& \wedge G_j(p_j, v_j) \wedge G_k(p_k, v_k) \wedge \mathbf{G_{CI}(v)} \\
& \wedge A_j(p_j, v_j, v'_j) \wedge A_k(p_k, v_k, v'_k) \\
& \vdash \\
& i_j(v'_j) \wedge i_k(v'_k) \wedge CI(v')
\end{aligned}
$$

Feasibility of I/O ($\mathbf{i}$)

- The parameter pair's input/output ranges must be compatible, w.r.t
  - type
  - range
- Given an event e and input parameter $q?$, the function typeOfIn returns the type T of $q?$

$$\text{typeOfIn}(e\ ,\ q?) = T$$

- So for a concrete event evt, with prm? and prm $\in \mathbb{N}$

$$\text{typeOfIn}(\text{evt, prm?}) = \mathbb{N}$$

Feasibility of I/O (ii)

- The typeOfOut function is similar.
- We have a new Feasibility Proof Obligation,
  - we call it $FIS_{preStyle}$

$$FIS_{preStyle}\,(e_j(p?_j\ , p!_j\ ), e_k(p?_k\ , p!_k\ ))$$

$$=$$

$$\forall q!, q? \cdot (q! \in p! \wedge q? \in p?)$$

$$\Rightarrow (typeOfOut(e_j\ , q!) \subseteq typeOfIn(e_k\ , q?)$$

Where parameters q are matched by name.

Closing Remarks

- Future Work:
  - **Interface event "calls"** (in Tech. Report).
  - Tool Support.
  - Library, linked data, search and retrieve.

- Acknowledgements:
  - Marina Waldén - *Åbo Akademi University, Turku, Finland.*
  - Colin Snook - *University of Southampton, UK.*