

# Formal methods meet domain-specific languages

Jean-Paul Bodeveix<sup>1</sup>   **Mamoun Filali**<sup>1</sup>   Julia Lawall<sup>2</sup>  
Gilles Muller<sup>3</sup>

<sup>1</sup>IRIT

<sup>2</sup>DIKU

<sup>3</sup>EMN

Integrated Formal Methods  
Eindhoven

29 November - 2 December, 2005

# Outline

- 1 Aims
- 2 Introduction
  - Bossa
  - B
- 3 Applying B
  - The Bossa methodology in B
  - B machines architecture
  - The B machines
- 4 Conclusion

# Outline

- 1 Aims
- 2 Introduction
  - Bossa
  - B
- 3 Applying B
  - The Bossa methodology in B
  - B machines architecture
  - The B machines
- 4 Conclusion

# Outline

- 1 Aims
- 2 Introduction
  - Bossa
  - B
- 3 Applying B
  - The Bossa methodology in B
  - B machines architecture
  - The B machines
- 4 Conclusion

# Outline

- 1 Aims
- 2 Introduction
  - Bossa
  - B
- 3 Applying B
  - The Bossa methodology in B
  - B machines architecture
  - The B machines
- 4 Conclusion

# Aims

- Bossa framework
- B method
- Reuse the B methodology and B proving framework

# Outline

- 1 Aims
- 2 Introduction
  - Bossa
  - B
- 3 Applying B
  - The Bossa methodology in B
  - B machines architecture
  - The B machines
- 4 Conclusion

## Starting point

Bossa DSL for  
process scheduling

OS: Gilles Muller

---

Languages: Julia Lawall

B formal method  
process scheduling

Jean-Paul Bodeveix

Mamoun Filali

**Short-term goal:** Use the B methodology and the B framework with Bossa

**Long-term goal:** Automate and extend Bossa verification

# Outline

- 1 Aims
- 2 Introduction
  - Bossa
  - B
- 3 Applying B
  - The Bossa methodology in B
  - B machines architecture
  - The B machines
- 4 Conclusion

## B

- Abstract machines.
  - Static part: variables and an invariant.
  - Dynamic part: initialisation and operations.
  - Proof obligations: invariant preservation.
- Development
  - Refinements:  $\rightsquigarrow$  implementation machine  $\rightsquigarrow$  C code.
  - Proof obligations: gluing invariant preservation.
- Proof environment.

# Outline

- 1 Aims
- 2 Introduction
  - Bossa
  - B
- 3 Applying B**
  - The Bossa methodology in B**
  - B machines architecture
  - The B machines
- 4 Conclusion

# The Bossa DSL

- Specification language for schedulers:
  - interactions kernel-scheduler: **events, state classes.**
  - interactions environment-scheduler: **interrupts.**
  - interactions context: **automaton.**
- Implementation language: **handlers.**

# The Bossa methodology in B

**Goal:** Verification of static properties of scheduling policies.

- Verification of state properties, e.g. `Singleton` states.
- Verification of event types pre-post conditions.

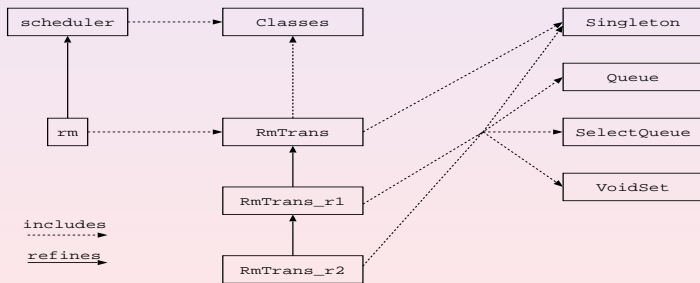
**Method:**

- B machine associated to event types (abstract scheduler)
- Refinements associated to scheduling policies implementations.

# Outline

- 1 Aims
- 2 Introduction
  - Bossa
  - B
- 3 Applying B**
  - The Bossa methodology in B
  - B machines architecture**
  - The B machines
- 4 Conclusion

# B machines architecture



# Outline

- 1 Aims
- 2 Introduction
  - Bossa
  - B
- 3 Applying B**
  - The Bossa methodology in B
  - B machines architecture
  - The B machines**
- 4 Conclusion

# State classes

**MACHINE** Classes

**SETS** Process

**VARIABLES**

Running, Ready, Blocked, Terminated, running

**INVARIANT**

Running  $\subseteq$  Process

& Ready  $\subseteq$  Process

& Blocked  $\subseteq$  Process

& Terminated  $\subseteq$  Process

& running  $\in$  Process

& Running  $\cap$  Ready =  $\emptyset$

& Running  $\cap$  Terminated =  $\emptyset$

& Running  $\cap$  Blocked =  $\emptyset$

& Ready  $\cap$  Terminated =  $\emptyset$

& Ready  $\cap$  Blocked =  $\emptyset$

& Terminated  $\cap$  Blocked =  $\emptyset$

# scheduler

**MACHINE** scheduler

**INCLUDES** Classes

**OPERATIONS**

...

/\*

[tgt in BLOCKED] -> [tgt in TERMINATED]

\*/

Process\_end(tgt)  $\triangleq$

**PRE** tgt : Process & tgt  $\in$  Blocked **THEN**

    CBlockedToTerminated(tgt)

**END**

...

**END**

## B model of a policy

```
states = {  
  RUNNING running : process;  
  READY ready : select queue;  
  READY yield : process;  
  KERNEL_BLOCKED blocked : queue;  
  POLICY_BLOCKED computation_ended : queue;  
  TERMINATED terminated;  
}
```

# RmTrans

**MACHINE** RmTrans

**INCLUDES**

Classes,

```
ru.Singleton(Process),      /* running state */
rd.SelectQueue(Process,period), /* ready state */
yl.Singleton(Process),      /* yield state */
bl.Queue(Process),          /* blocked state */
ce.Queue(Process),          /* computation_ended */
tm.VoidSet(Process)        /* terminated state */
```

# RmTrans

## INVARIANT

```

bl.elems  $\cap$  ce.elems =  $\emptyset$ 
 $\wedge$  rd.elems  $\cap$  yl.elems =  $\emptyset$ 
 $\wedge$  Running = ru.elems
 $\wedge$  Ready = rd.elems  $\cup$  yl.elems
 $\wedge$  Blocked = bl.elems  $\cup$  ce.elems
 $\wedge$  Terminated = tm.elems
  
```

## OPERATIONS

```

RMRunning2Yield  $\triangleq$ 
  PRE ru.elems  $\neq \emptyset \wedge$  yl.elems =  $\emptyset$  THEN
    yl.add(ru.elem) || ru.suppress || CRunning2Rea
  END;
  ...
END
  
```

```
On unblock.preemptive {
  if (e.target in blocked) {
    if ((!empty(running)) && (e.target > running))
      running => ready;
  }
  e.target => ready;
}
```

**REFINEMENT** *rm*

**REFINES** *scheduler*

**INCLUDES** *RmTrans /\* state transition machine \*/*

**VARIABLES**

*missed\_deadlines, timer /\* policy specific vari*

**INVARIANT**

*missed\_deadlines : Process --> NATURAL*

*& timer : Process --> INTEGER*

**INITIALISATION**

*missed\_deadlines := Process \* {0}*

*|| timer := Process \* {0}*

## OPERATIONS

Unblock\_preemptive(tgt)  $\triangleq$

**VAR** isbk **IN**

isbk  $\leftarrow$  RMisBlocked(tgt);

**IF** isbk = TRUE **THEN**

**VAR** hru **IN**

hru  $\leftarrow$  RMhasRunning;

**IF** hru = TRUE  $\wedge$

period(tgt) < period(running) **THEN**

RMRunning2Ready

**END;**

RMBlocked2Ready(tgt)

**END**

**END**

**END**

**END**

# Conclusion

## Past:

- Expression of the Bossa methodology and properties in B.
- Proofs remain tedious.

## Present:

- Covering of other Bossa properties.
- Proof automation: WS1S, model checking, abstraction.

## Future:

- Application level proofs.
- Proof annotated code.