

On Designing a Target-Independent DSL for Safe OS Process Scheduling Components

Julia Lawall, DIKU, University of Copenhagen

Anne-Françoise Le Meur, Jacquard Group, LIFL, Lille

Gilles Muller, OBASCO Group, Ecole des Mines de Nantes

Overview

- ▶ Introduction to Domain-Specific Languages (DSLs).
- ▶ Our proposal for DSL design.
- ▶ Instantiation in the Bossa DSL for process scheduling.
- ▶ Conclusions.

Domain-Specific Languages (DSLs)

DSL: A language dedicated to a particular domain.

- ▶ Captures a family of programs.
- ▶ Provides high-level domain-specific abstractions that
 - ▶ Simplify programming.
 - ▶ Enable verifications, optimizations.

Useful when:

- ▶ Programming within the program family is often needed.
- ▶ Programming within the program family requires highly specialized knowledge.

Examples: lex, yacc, SQL, languages for graphics, Web programming, etc.

Our target domain: process scheduling

Process scheduling: How an OS selects a process for the CPU.

- ▶ Many scheduling policies (round-robin, rate monotonic, etc.).
- ▶ Policies form a program family.
- ▶ No policy is perfect for all applications.

Implementing a scheduler requires:

- ▶ Understanding the scheduling policy.
- ▶ Understanding the target OS.
 - ▶ Any error can crash the machine.

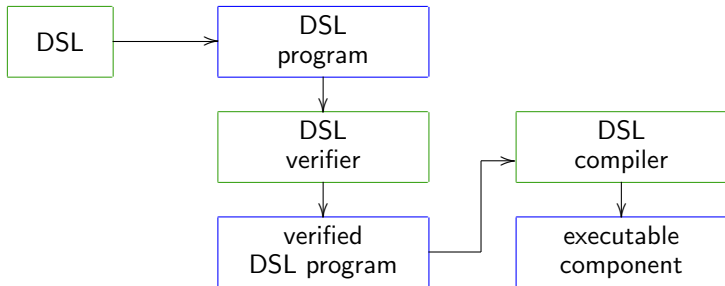
⇒ An ideal DSL target ...

- ▶ Bossa [Muller, Lawall, et al., EW2002, ASE2003, PEPM2004]

Creating a DSL

A domain expert uses domain expertise to [Consel, Marlet, PLILP'98]:

- ▶ Select language abstractions.
- ▶ Develop a language syntax.
- ▶ Implement language support (**verifier**, **compiler**, etc.)



Problem: Multiple kinds of expertise may be needed.

Expertise needed to create a DSL for process scheduling

Expertise in scheduling policies:

- ▶ Liveness, bounded response time, etc.
- ▶ What kinds of operations are needed to provide these properties?

Expertise in operating systems:

- ▶ How does existing scheduling code work?
- ▶ What existing scheduling code should be replaced?
- ▶ What invariants must scheduling code maintain?

Problem: Expertises required at different times.

Our proposal

Divide the role of the domain expert:

- ▶ Scheduling expert: Expert in the program family.
 - ▶ Identifies relevant language constructs.
- ▶ OS expert: Expert in each specific execution environment.
 - ▶ Identifies relevant OS properties.

Our proposal

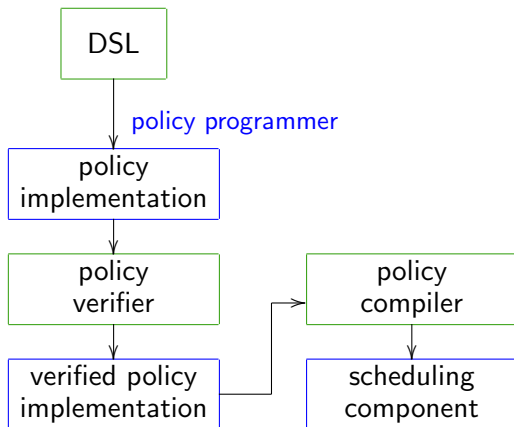
Divide the role of the domain expert:

- ▶ Scheduling expert: Expert in the program family.
 - ▶ Identifies relevant language constructs.
- ▶ OS expert: Expert in each specific execution environment.
 - ▶ Identifies relevant OS properties.

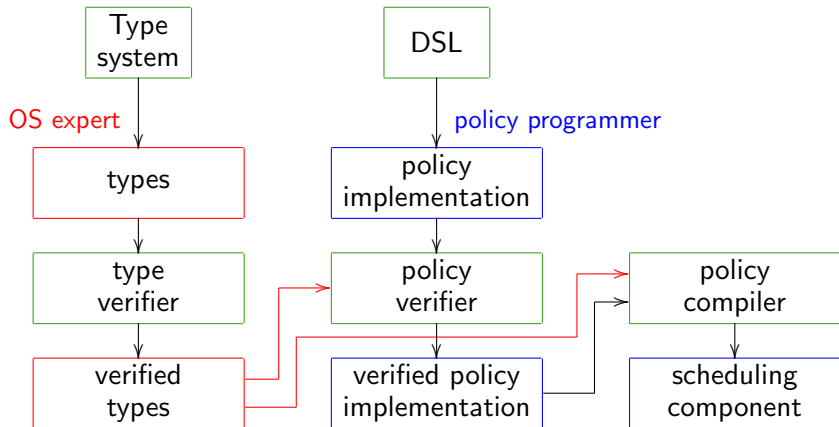
Introduce a type system:

- ▶ Developed by the scheduling expert based on an analysis of the range of relevant properties.
- ▶ Used by the OS expert to describe OS properties.
- ▶ Types used in verifying and compiling DSL programs.

Instantiation in the Bossa DSL



Instantiation in the Bossa DSL



Issues

- ▶ Can relevant properties be expressed in a concise and understandable way?
- ▶ Can type information be used to detect errors?
- ▶ Can type information improve the result of compilation?

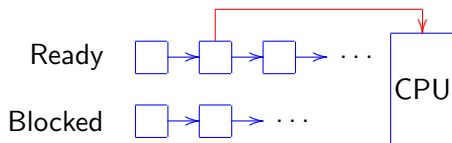
The Bossa DSL, in more detail

- ▶ The scheduling domain.
- ▶ Contribution of the scheduling expert
- ▶ Contribution of the OS expert
- ▶ Tying things together: the verification process.

The scheduling domain

Goal of process scheduling:

- ▶ Elect a new process.
- ▶ Only ready processes are eligible.



A scheduler must:

- ▶ Elect an eligible process.
- ▶ Adjust process states in response to kernel events.

Contribution of the scheduling expert

Language infrastructure (OS independent)

- ▶ Syntax
 - ▶ main elements: process states and event handlers
- ▶ Type system
- ▶ Verifier
- ▶ Compiler

Process states

```
states = {  
    RUNNING    running    : process;  
    READY      ready      : select queue;  
    READY      expired    : queue;  
    BLOCKED    blocked    : queue;  
    TERMINATED terminated;  
}
```

States: running, ready, etc.

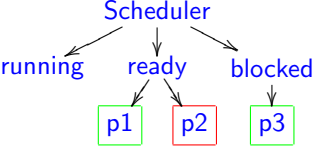
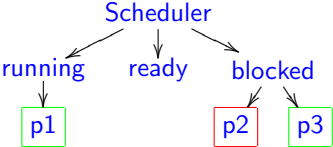
State classes: Describe state semantics:

- ▶ RUNNING: the state of the running process
- ▶ READY: states containing eligible processes
- ▶ BLOCKED: states containing blocked (ineligible) processes
- ▶ TERMINATED: a dummy state for terminating processes

Event handlers

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

unblock p2



Contribution of the OS expert (Linux 2.4)

Events:

bossa.schedule, block.*, unblock.preemptive.*, unblock.nonpreemptive.*, ...

Interrupt events: unblock.preemptive.*, unblock.nonpreemptive.*, ...

Event sequences: block.* \xrightarrow{u} bossa.schedule, ...

Type rules:

- ▶ unblock.preemptive.*:
 - ▶ [tgt in BLOCKED] -> [tgt in READY]
 - ▶ [p in RUNNING, tgt in BLOCKED] -> [p, tgt] in READY
 - ▶ [tgt in RUNNING] -> []
 - ▶ [tgt in READY] -> []
- ▶ 11 events, 60 rules for Linux 2.4.

Tying things together

Verifier and compiler:

- ▶ Implemented by the scheduling expert.
- ▶ Configured with information provided by the OS expert.

Verifier:

- ▶ Checks that all handlers are present.
- ▶ Checks that handlers implement allowed transitions.

Compiler:

- ▶ Generates C code.
- ▶ Uses information collected by the verifier.

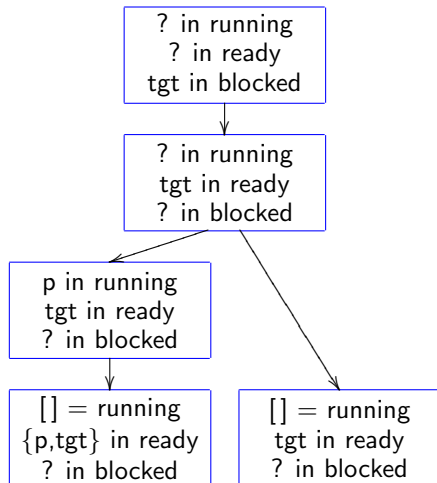
Verification example

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

Verification with respect to:
[tgt in BLOCKED] -> ...

Matches:
[p in RUNNING, tgt in BLOCKED] ->
[{p, tgt} in READY]

[tgt in BLOCKED] -> [tgt in READY]



Conclusions

Multiple kinds of expertise required to implement a DSL.

- ▶ May not all be available at the same time.

For scheduling, we propose:

- ▶ A scheduling expert.
- ▶ An OS expert.
- ▶ A type system to connect them.

DSL can be constructed so that the contribution of the OS expert can be usefully exploited.

Availability

- ▶ Implementation in Linux 2.4, with and without high-resolution timers.
- ▶ Example policies and applications.
- ▶ Teaching lab, based on Knoppix.
- ▶ MPlayer demo.

<http://www.emn.fr/x-info/bossa/>