

# Toward Robust Oses for Appliances: a New Approach Based on Domain-Specific Languages

---

Gilles Muller, Charles Consel,  
Renaud Marlet, Luciano Barreto, Fabrice  
Mérillon, Laurent Réveillère

*Compose Group*

*<http://www.irisa.fr/compose>*

Irisa/LaBRI, Rennes/Bordeaux (France)

# Issues in Developing Appliances

---

- ◆ High pace of innovation
  - ⇒ reduction of development time
    - improve code re-use
    - improve expertise re-use
  
- ◆ Upgrades and bug fixes difficult
  - ⇒ high level of confidence in the software
    - improve robustness and safety
    - statically verify critical properties

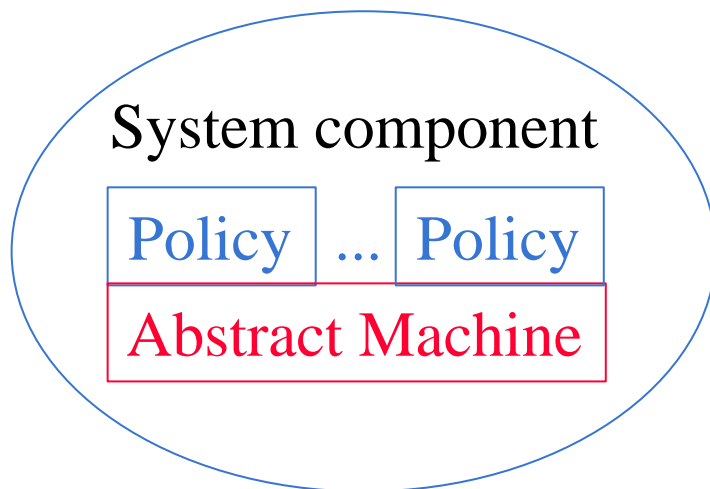
# Domain Specific Language: Characterization

---

- ◆ Restricted language that offers domain-specific abstractions
  - properties embedded in the language
  - concise programming
- ◆ Interface to a library/Abstract Machine
  - glue language
  - force a correct usage of the library
  - integration of an expertise in the language

# Our Approach

---



- ◆ Family of components
- ◆ Enforce a two-stage design:
  - algorithms/policy
  - common mechanisms
- ◆ DSL
  - syntax
  - domain properties
  - language restrictions

# DSL and OSEs: Interests and Advantages

---

- ◆ Improved expertise re-use
  - expertise repository in the AM (mechanisms)
  - separate What/How
- ◆ Improved code re-use
  - well-identified common mechanisms
  - enforced re-use of the AM
- ◆ Improved safety and robustness
  - property verification (predictable)

# DSL and OS: our first experiments

---

- ◆ GAL (generation of X11 drivers)
  - conciseness 10/C
  - properties on registers/clocks
- ◆ Devil (specification of hardware devices)
  - consistency of specifications
  - run-time checks
- ◆ PLAN-P (active networks)
  - safety and security properties
  - JIT compilation/kernel execution

# DSL and OS: some of the issues

---

## Increasing number of languages

- ◆ Babel Tower syndrome
- ◆ Efficient (JIT) Compilers
- ◆ Methodology/Tools for designing DSLs

# Babel Tower Syndrome

---

May be not a hard issue

- ◆ Reuse existing well-known syntaxes
  - restricted language constructions
  - PLAN-P: ML, C
  
- ◆ Syntax close to domain abstractions
  - domain expertise of the programmer
  - GAL, Devil: device documentation

# Efficient (JIT) Compilers

---

- ◆ Optimizing compilers
  - complex to develop
  - specific to processors
  - complex to maintain and evolve
- ◆ Interpreters
  - simple to develop
  - slow execution speed
  - ➔ Dilemma performance/maintainability

# Generating Compilers from Interpreters using Specialization

---

## Tempo, a Program Specializer for C

- ◆ Evaluate away the interpretation
  - “glue code” eliminated
- ◆ CT/RT specialization
  - C interpreter -> off-line compiler
  - C interpreter -> JIT compiler
- ◆ DSL programs as efficient as C equivalents
  - PLAN-P, GAL

# Methodology/Tools

---

- ◆ First methodology: SPRINT [PLILP-98]
  - steps toward defining a DSL
  
- ◆ Still
  - domain analysis is the “hard part”
  - there is a need for additional tools
    - » interpreter design
    - » graphical interfaces

# Conclusion

---

DSLs are a realistic approach to OS design and development

- ◆ Several DSLs successfully developed
- ◆ Coming soon: a DSL for scheduling
- ◆ Future work:
  - gather more experience in domain analysis
  - tools for designing/developing DSLs

# Questions

---

Devil, PLAN-P, GAL et Tempo are publicly available

<http://www.irisa.fr/compose/>