A Framework for Simulating Heterogeneous Virtual Processors
32nd Annual Simulation Symposium, System Simulation session
April 12, 1999

Dale Parson, Paul Beatty, John Glossner and Bryan Schlieder
Bell Labs Innovations for Lucent Technologies
dparson@lucent.com
Design Patterns Generate Behavior across Media

The question is whether it is possible to write down rules or patterns for architecture—and software and art—so that ordinary people can follow the rules or patterns and, by the nature of the patterns and using only the abilities of ordinary people, beauty is generated. If this happens, then the rules or patterns are generative, which is a rare quality.

— Richard Gabriel, *Patterns of Software*

LUxWORKS *luxdbg* simulator / debugger applies three design patterns across system layers:

- Build and extend abstract virtual processors.
- Build reflective entities.
- Build a covariant extensible system.
Universal Design Patterns constitute a *lingua franca*

“I can go anywhere in the system and recognize what is going on.”
— a processor designer using LUxWORKS framework.

**PATTERNS**

Key Domain Entity       Reflection       Extension covariance

- Extension medium (meta-data, extension language)  
  - on-the-fly configuration

- Aggregate medium (construction process, modules)  
  - multi-processor system on a chip

- Primitive medium (domain building blocks)  
  - registers, instructions
Layers of the LUxWORKS Framework

Tcl/Tk graphical user interface

widget event callback

Tcl processor monitor / control commands

script callback

Processor monitor / control

Processor model

IO callback

Modeling infrastructure

Tcl/Tk subsystem

Tcl subsystem

0..1

1

1

*

1

*

1

0..1

cosimulate

Vendor simulator interface
Modules of the LUxWORKS Framework

- **BEHAVIORAL API**
  - INST
  - PHASE
  - Tel/Tk
  - LUxDBG
  - Embedded API
  - TVM API
  -DSP1600
  - DSP16K
  - Scheduler
  - DSP1600 TAP Mgr
  - DSP16K TAP Mgr
  - Chain Manager
  - JTAG Driver

- **ALGORITHMIC API**
  - func( input1[], input2[], output[] )
  - DATA
  - COEF
  - Finite Arithmetic effects
  - DATA

- **DATA**

- **COEF**
PATTERN #1: Build and extend abstract virtual processors.

```c
for (tap = 0 ; tap < numtap ; tap++) {
    accum += sample[tap]
        * coefficient[(tap+offset) % numtap];
}
```
PATTERN #1: Build and extend abstract virtual processors.

Every layer is a virtual processor.

voice coders, feature logic, control logic

C/C++ debugging, profiling

DSP1600, DSP16K, SC140, ARM-7, ...

models, layouts, chips

Tcl test scripts, profile scripts, FAE extension scripts

Tcl subsystem

processor monitor / control

processor model

modeling infrastructure

extension language VM

application VM

procedural VM

processor VM

circuit VM
What can one do with an Abstract Virtual Processor?

- Update output and state from input and state
- Run, stop, trigger breakpoint, fire exception, program
- Reset, synchronize, connect to world

A concrete class derived from Circuit implements a circuit.

A concrete class derived from InterfaceProcessor implements an instruction set.

A concrete class derived from DriverProcessor implements a processor chip.
How can one combine Abstract Virtual Processors?

Connect them at construction time.
Abstract Virtual Processors for architectural prototypes

Run, stop, trigger breakpoint, fire exception, program

Update output and state from input and state

Reset, synchronize, connect to world
PATTERN #2: Build reflective entities.

What are your sources, sinks, features, databases?

What are your classes, functions, variables, files?

What is your instruction pointer, instruction memory, byte ordering?

What are your registers, pins, memories?

What are your extensions, tests, sample data frames?

Every layer is reflective.
Reflective processors yield configurable tools.

GUI uses relational and hierarchical mega-widgets. GUI configures itself to processors & programs at run time.
Reflective processors yield configurable bridges to tools.

Behavioral simulator bridge
queries simulator & LUxWORKS
model for pin & clock timing
bindings.

Algorithmic simulator bridge
queries simulator & LUxWORKS
symbol table for function &
parameter bindings.
PATTERN #3: Build a covariant extensible system.

- Breakpoint targets covary with application.
- Breakpoint commands covary with types.
- Breakpoint trigger types covary with processor.
- Signal types & probe types covary.
- Covariance keeps generic classes generic.
- Callback extensions covary with processor type or instance.
Forces driving simulation & debugging architectural evolution

Run-time feature update via Java networked loaders for optimized code

networked application (e.g., Internet telephony)

C++ on controllers, embedded Java

Partial definition by processor core vendors

Multiple heterogeneous processors

Distributed embedded systems need networked debugging & profiling.

Third party cores

extension language VM

application VM

procedural VM

processor VM

circuit VM

D. Parson, Lucent Technologies, 4/12/99

32nd Annual Simulation Symposium
Conclusions

• Virtual Processor pattern gives universal ability to trigger events, reset, execute, halt execution and deliver break events to clients.

• Universal inheritance from Virtual Processor base classes — Circuit and Interface Processor — supports on-the-fly composition of processors into higher-order system simulations.

• Universal reflection allows tools and tool bridges to configure themselves to processors and programs at execution time.

• Processor-oriented covariance allows tool capabilities to vary with process-specific support without perturbing generic tool code.

• Current direction is unification of simulation, embedded hardware execution & real-time operating system task execution.

• Increase in support for networked simulation & debugging.