Ptolemy II
The automotive challenge problems version 4.1

Johan Eker
Edward Lee

with thanks to Jie Liu,
Paul Griffiths, and Steve Neuendorffer

MoBIES Working group meeting, 27-28 September 2001, Dearborn

Overview

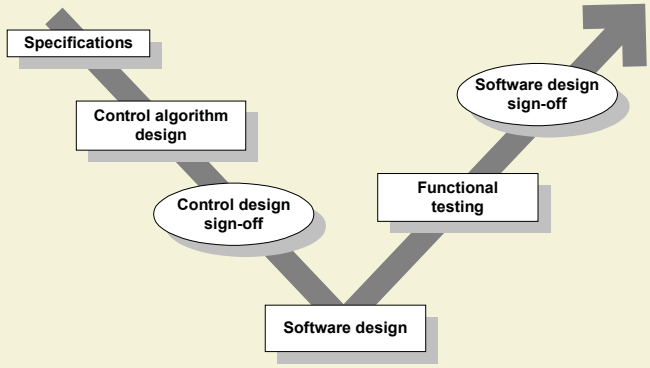
- Yellowstone recap
- Selected challenge problems
  - 1.1 Multiple-view modeling
  - 1.2 Automated composition of subcomponents
  - 3.3 Code generation

Yellowstone recap: Design of embedded control systems

- Different phases, different tools, different people makes it difficult to debug:
  - Control engineer view
    - plant dynamics, stability, phase margins, rise time, etc.
    - assumes: equidistant sampling with no or little latency
  - Embedded system engineer view
    - scheduling, priorities, memory usage, communication setup, etc
    - assumes: fixed controller design
- A good toolset supports close interaction between the different phases/teams
- The only interesting performance metric is the behavior of the controlled system

"Classical" development cycle

- Sign-offs are expensive
- Feedback slow



Closing the "system design/control design" loop

hardware setup
communication
priorities
RTOS tuning

evaluate system performance

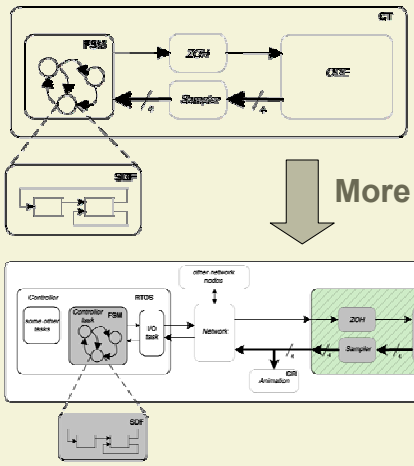
system design

control design

evaluate system performance

controller parameters
delay compensation
reviewing specs

Idealized Model



Assumes equidistant sampling constant latency

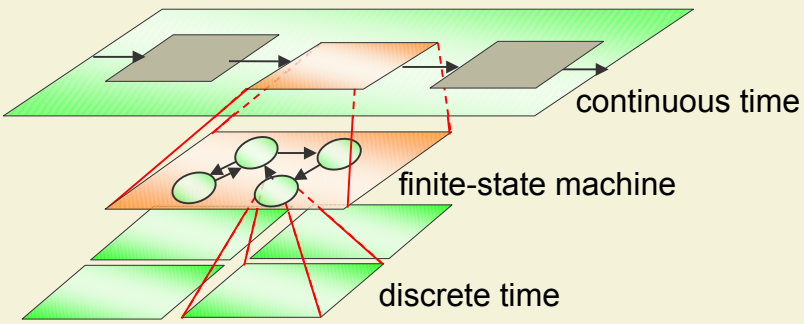
More realistic model

- **Multitasking**
  - jitter, execution time
  - RTOS domain
- **Communication**
  - Transport, routing, medium access

1.1 Multi-view modeling

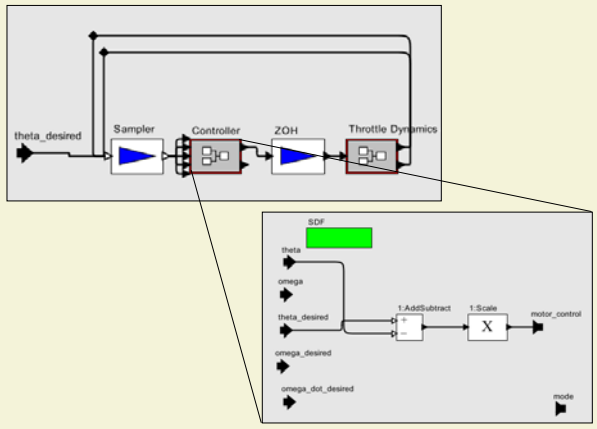
- Different granularity models
  - Level 1: Hybrid automata w/ cont. dynamics
  - Level 2: Discrete controllers and some scheduling info
  - Level 3: Platform specific info
- Component refinement
  - Start with a naïve implementation and make it gradually more complex
- Ptolemy II
  - Component based
  - Hierarchical & heterogeneous
  - Functional behavior & control flow decoupled through the use of directors
  - Composite actors treated like atomic

Multi-view modeling in Ptolemy II

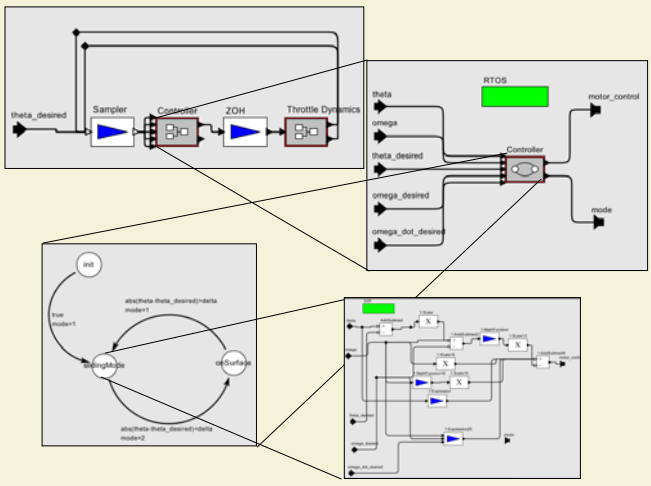


Hierarchical, heterogeneous model

Component refinement in Ptolemy II
Example model 1

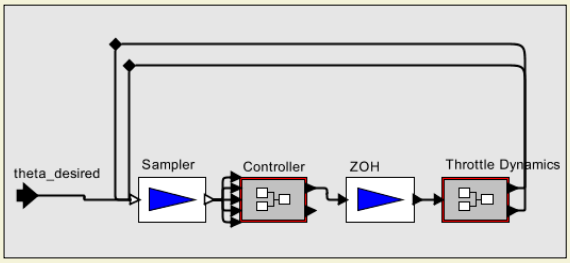


Component refinement in Ptolemy II
Example model 2



Composite actors

- From top level view: the behavioral semantics of the component has not changed!



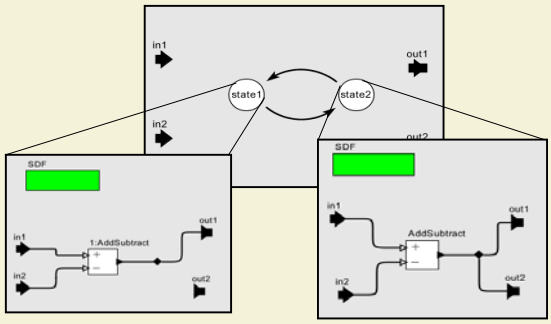
- Aggregation not just syntactical
- Composite actor is opaque

1.2 Automated composition of sub-components

- What is the actual problem?
  - Example: Many states and many signals in a Stateflow + Simulink gets means and whole lot of wiring
- Lack of proper aggregation!
- Ptolemy addresses the problem through hierarchy
- Smarter editor vs. new languages

The ModalModel in Ptolemy II

- Wiring of the state refinements is done automatically,
- All wires are hidden under the hood



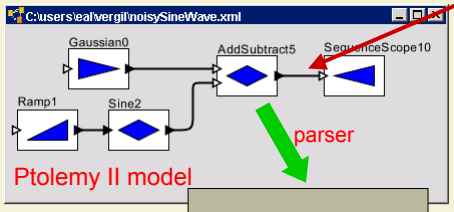
3.3 Code generation

- From Java to Java & Java to C at Maryland
- Actor libraries are built and maintained in Java
  - polymorphic libraries are rich and small
- Collapsing composite actors to atomic actors
  - Director + actors => actor
- Efficiency gotten through code transformations
  - specialization of polymorphic types
  - code substitution using MoC semantics
  - removal of unnecessary code

Outline of our Approach

Jeff Tsay,
Christopher Hylands,
Steve Neuendorffer

Model of Computation semantics defines communication, flow of control



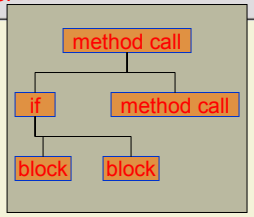
Ptolemy II model

parser

scheduler

Schedule:
- fire Gaussian0
- fire Ramp1
- fire Sine2
- fire AddSubtract5
- fire SequenceScope10

All actors are given in Java, then translated to embedded Java, C, VHDL, etc.



abstract syntax tree

```
…
for (int i = 0; i < plus.getWidth(); i++) {
  if (plus.hasToken(i)) {
    if (sum == null) {
      sum = plus.get(i);
    } else {
      sum = sum.add(plus.get(i));
    }
  }
}
…
```

target code

Conclusions

- Hierarchically heterogeneous modeling matches the applications well.
- Component based technologies and hierarchical heterogeneity gives good support for
  - Multi-view modeling
  - Piecewise refinement
- Tool integration as a more fundamental problem
  - About designing the proper protocol for communication between subsystems