Component-Based Design of Embedded Control Systems

Edward A. Lee & Jie Liu
UC Berkeley

with thanks to the entire Berkeley and Boeing SEC teams

SEC PI Meeting
Annapolis, May 8-9, 2001
Precise Mode Change Problem

How do you get the processes to a quiescent state to take a mode change?
Components and their Relationships

An abstract syntax - clustered graphs - is well suited to a wide variety of component-based modeling strategies, ranging from state machines to process networks.
Actor View of Producer/Consumer Components

Basic Transport:

Producer/consumer styles:
- continuous-time
- dataflow
- discrete events
- synchronous
- time-driven
- publish/subscribe
- …
A Laboratory for Exploring Models of Computation

Ptolemy II – Java based, network integrated

- A realization of a model of computation is called a “domain.” Multiple domains can be mixed hierarchically in the same model.
Basic Object Model for Executable Components

- **ComponentEntity**
  - +getDirector() : Director
  - +getExecutiveDirector() : Director
  - +getManager() : Manager
  - +inputPortList() : List
  - +newReceiver() : Receiver
  - +outputPortList() : List

- **CompositeEntity**
  - 0..n

- **CompositeActor**
  - 0..1

- **Executable**
  - «Interface»
  - Executable
    - +fire()
    - +initialize()
    - +postfire() : boolean
    - +prefire() : boolean
    - +preinitialize()
    - +stopFire()
    - +terminate()
    - +wrapup()

- **Actor**
  - «Interface»
  - Actor
    - +fire()
    - +initialize()
    - +prefire() : boolean
    - +preinitialize()
    - +postfire() : boolean
    - +stopFire()
    - +terminate()
    - +wrapup()

- **Director**

- **AtomicActor**
Abstract Semantics – How Components Interact

**flow of control**
- Initialization
- Execution
- Finalization

**communication**
- Structure of signals
- Send/receive protocols
Abstract Semantics – How Components Interact

**flow of control**
- **Initialization**
- **Execution**
- **Finalization**

**communication**
- Structure of signals
- Send/receive protocols

- **preinitialize()**
  - declare static information, like type constraints, scheduling properties, temporal properties, structural elaboration

- **initialize()**
  - initialize variables
Abstract Semantics – How Components Interact

flow of control
- Initialization
- Execution
- Finalization

communication
- Structure of signals
- Send/receive protocols

iterate()
Abstract Semantics – How Components Interact

flow of control
- Initialization
- **Execution**
- Finalization

communication
- Structure of signals
- Send/receive protocols

- **iterate()**
- prefire()
- fire()
- postfire()
- stopFire()
The Key Action Methods

- **Prefire()**
  - obtain required resources
  - may read inputs
  - may start computations
  - returns a boolean indicating readiness

- **Fire()**
  - produces results

- **Postfire()**
  - commits state updates (transactional)

- **StopFire()**
  - request premature termination

All of these are atomic (non-preemptible)
This Abstract Semantics has Worked For

- Continuous-time models
- Finite state machines
- Dataflow
- Discrete-event systems
- Synchronous/reactive systems
- Time-driven models (Giotto)
- …

Hybrid systems

Can we make it work for priority-driven multitasking (RTOS style)?
Benefits

- Composable semantics
  - arbitrarily deep hierarchies
  - heterogeneous hierarchies

- Precise mode switching
  - nest FSMs with anything else

Hierarchical, heterogeneous, system-level model
Objective:
- understand and improve OCP semantics
- support priority-driven preemptive scheduling
- use atomic execution, to get composability
- solve the precise mode change problem

Solution:
- Atomic execution when possible
- Façade to long-running processes when not
Atomic Façade to Long-Running Computations

- Each component defines the interaction between the atomic façade and the long-running process.

- There are several useful patterns:
  - allow task to complete
  - enforce declared timing
  - “anytime” computation
  - transactional
RTOS Domain Implementation

- priority
- executionTime

RT-Q

OS-Q

(clock, 2.0) (clock, 1.0)

(actor, output time)

(T3, p3, t3) (T1, p2, t2) (T1, p1, t1)

(task, priority, remaining processing time)
Example: two simple tasks

nonpreemptive

preemptive
Inter-domain example: shared-resource controllers

plant1
plant2

computer
controller1
controller2
Background process example: Data acquisition and processing

atomic processes

background processes
What a Modal Control System Might Look Like

- Initial state: $d\Phi$, $\Phi$
- Transition: $Th < region1 \&\& Th > -region1$ mode = 1
- Transition: $Th > region2 || Th < -region2$ mode = 0
- Transition: $d\Phi < maxSpeed \&\& d\Phi > -maxSpeed$ mode = 2; stabilizeController.$\Phi$ value = $\Phi$

RTOS model

SEC, Annapolis, 20
Systematic, principled, real-time, heterogeneous, hierarchical composition of:

- Processes and/or threads
- Finite automata (mode controllers)
- Other models of computation
  - Continuous-time models
  - Dataflow models
  - ...

The key is the abstract semantics of Ptolemy II, which defines hierarchical heterogeneous composition of models of computation.