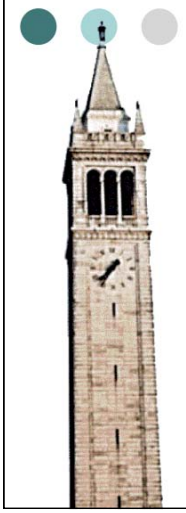


Introduction to Embedded Systems

Edward A. Lee & Sanjit Seshia
UC Berkeley
EECS 124
Spring 2008

Lecture X: More Concurrency Models

There are many ways to view computation and its integration with physical dynamics.

Concerns:

- timeliness
- modularity
- composability
- reliability
- robustness
- concurrency
- resource usage (energy, time, …)

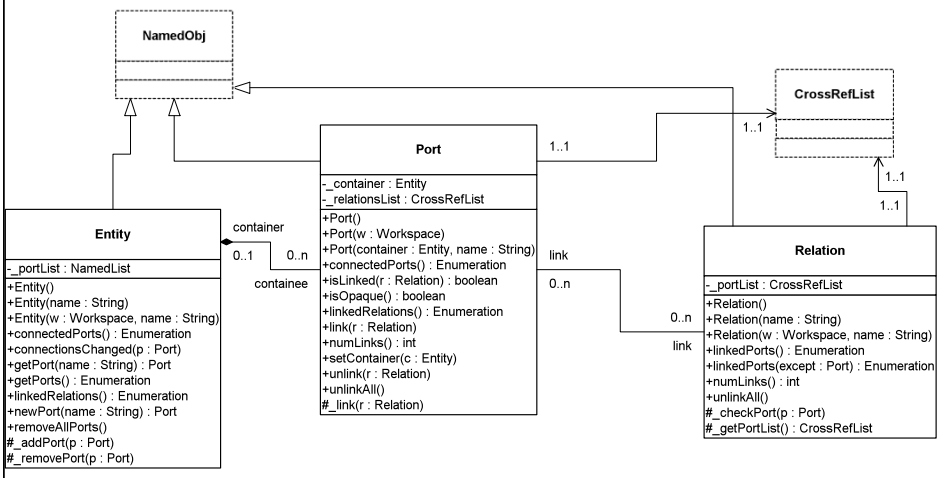
Model-based design

models are abstractions of systems:

- structural (OO design)
- ontological (type systems)
- imperative logic ("procedural epistemology")
- functional logic
- actor-oriented ("dynamical systems")

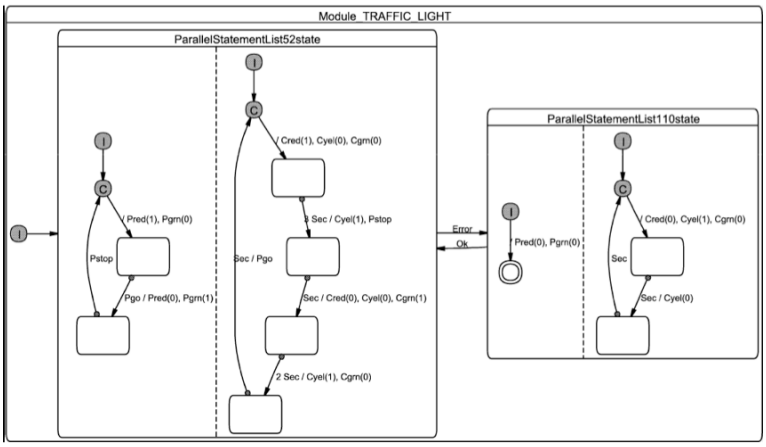
All of these have their place…

Example: UML static structure diagrams



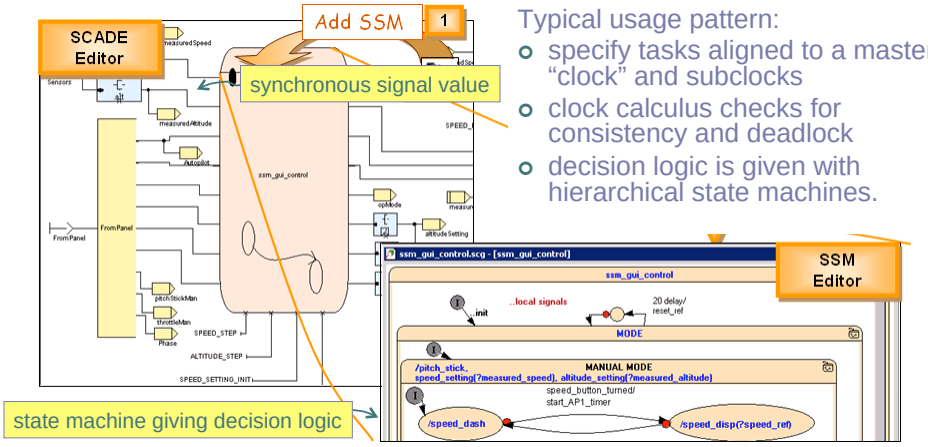
Used to design object-oriented structure. Can be a *model* or a *meta model* (a model of a modeling technique). The above example is a meta model.

Example: UML Statecharts



Statecharts combine finite state machines with concurrent composition. They are due to David Harel [1987]. The above example models a simple traffic light system, and is due to Reinhard von Hanxleden [2007].

Example: Synchronous Languages

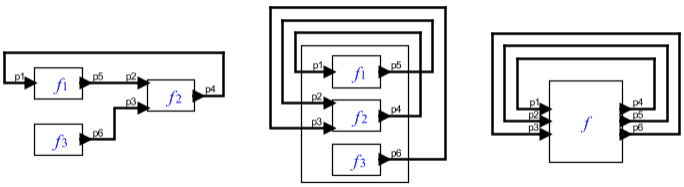


Typical usage pattern:

- specify tasks aligned to a master "clock" and subclocks
- clock calculus checks for consistency and deadlock
- decision logic is given with hierarchical state machines.

Lustre/SCADE, from http://www.esterel-technologies.com/

Synchronous/Reactive (SR) models



A signal is a function of form $x: \mathbb{N} \rightarrow R \cup \{\varepsilon\}$ where the domain represents "ticks" of a "clock" and ε is "absent." An actor with n inputs and m outputs is

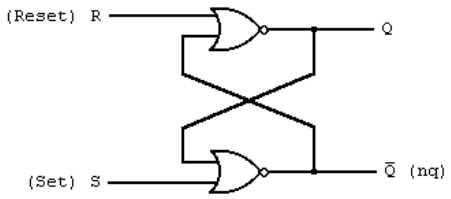
$$S: (\mathbb{N} \rightarrow R \cup \{\varepsilon\})^n \rightarrow (\mathbb{N} \rightarrow R \cup \{\varepsilon\})^m$$

At tick n of the clock, the actor realizes a function

$$f_n: (R \cup \{\varepsilon\})^n \rightarrow (R \cup \{\varepsilon\})^m$$

At tick n, the system above has $m = n = 3$ and signal values $\mathbf{x}(n)$ satisfying $\mathbf{x}(n) = f_n(\mathbf{x}(n))$, a "fixed point."

Example: Hardware Description Languages



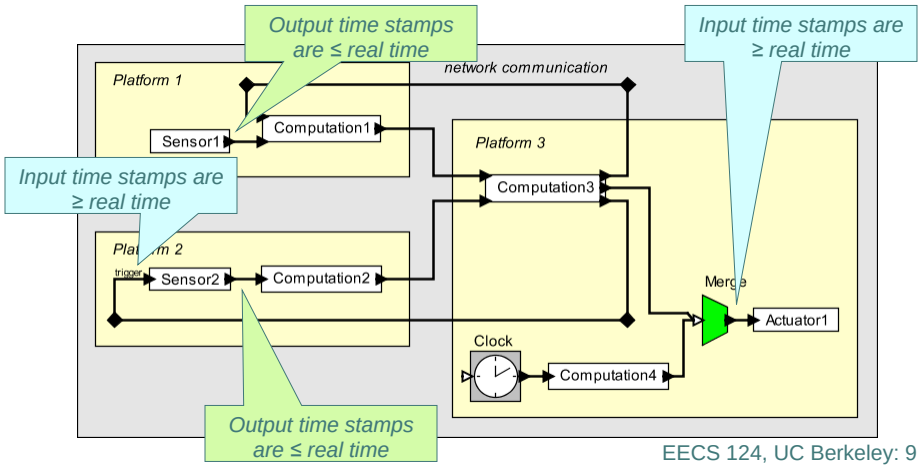
SR Latch

e.g. VHDL:

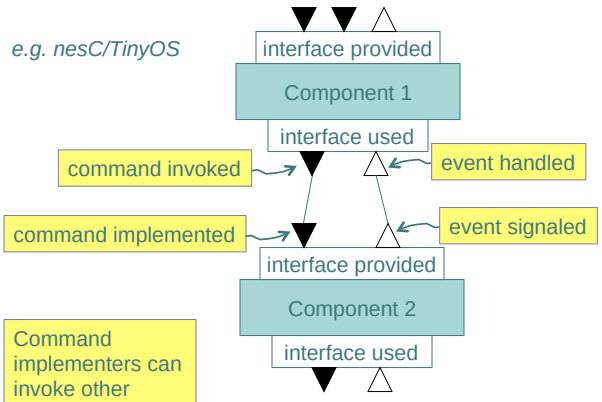
```
entity latch is
  port (s,r : in bit;
        q,nq : out bit);
end latch;

architecture dataflow of latch is
begin
  q<=r nor nq;
  nq<=s nor q;
end dataflow;
```

Example: PTIDES: Programming Temporally Integrated Distributed Embedded Systems

Distributed execution under DE semantics, with "model time" and "real time" bound at sensors and actuators.



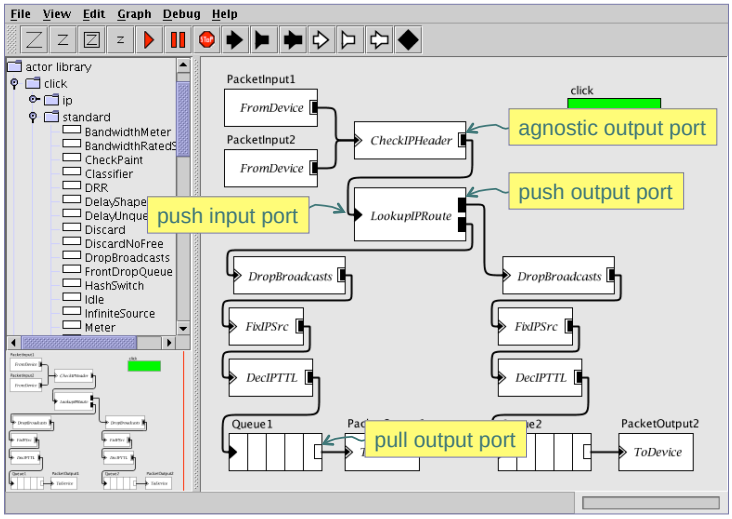
Example: Sensor Network Languages



Typical usage pattern:

- hardware interrupt signals an event.
- event handler posts a task.
- tasks are executed when machine is idle.
- tasks execute atomically w.r.t. one another.
- tasks can invoke commands and signal events.
- hardware interrupts can interrupt tasks.
- exactly one mutex, implemented by disabling interrupts.

Example: Network Languages

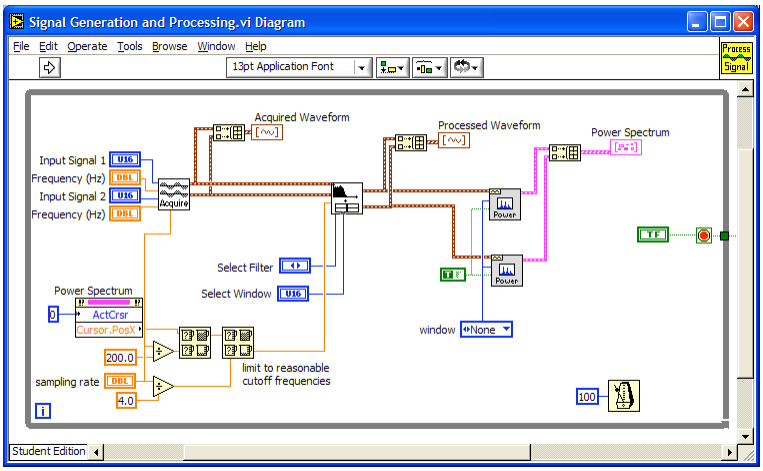


Typical usage:

- queues have push input, pull output.
- schedulers have pull input, push output.
- thin wrappers for hardware have push output or pull input only.

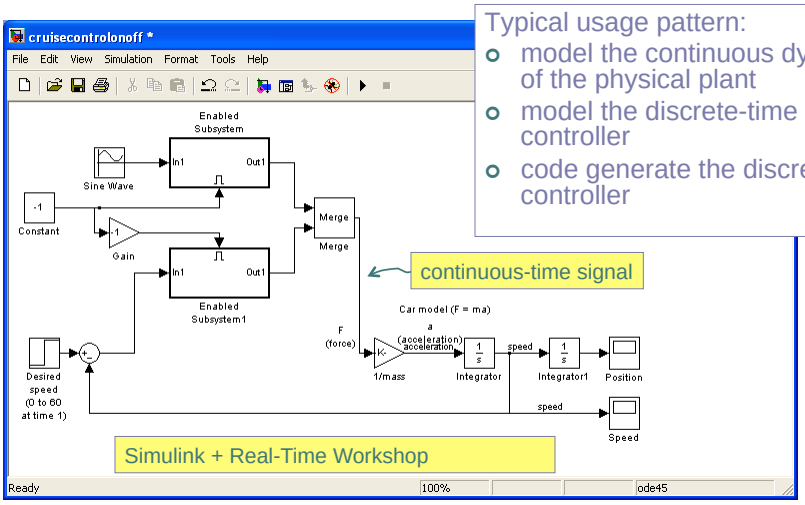
Click (Kohler) with a visual syntax in Mescal (Keutzer)

Example: Dataflow Languages



e.g. LabVIEW, Structured dataflow model of computation

Example: Continuous-Time Languages



Typical usage pattern:

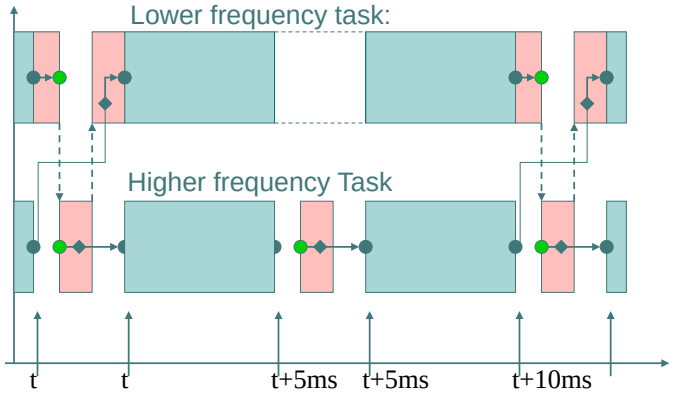
- model the continuous dynamics of the physical plant
- model the discrete-time controller
- code generate the discrete-time controller

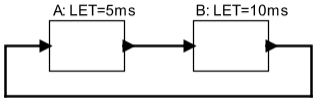
continuous-time signal

Simulink + Real-Time Workshop

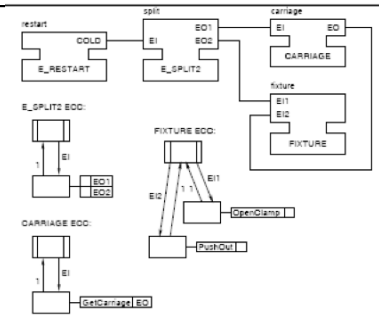
Example: Time-Triggered Models

In time-triggered models (e.g. Giotto, TDL, Simulink/RTW), each actor has a logical execution time (LET). Its actual execution time always appears to have taken the time of the LET.





Some efforts get confused: IEC 61499



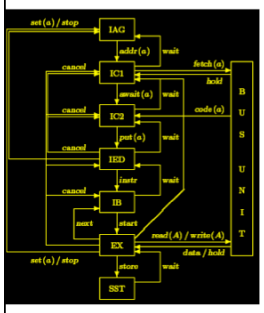
International Electrotechnical Commission (IEC) 61499 is a standard established in 2005 for distributed control systems software engineering for factory automation.

The standard is (apparently) inspired by formal composition of state machines, and is intended to facilitate formal verification.

Regrettably, the standard essentially fails to give a concurrency model, resulting in radically different behaviors of the same source code from on runtime environments from different vendors, and (worse) highly nondeterministic behaviors on runtimes from any given vendor.

See: Ĉengić, G., Ljungkrantz, O. and Åkesson, K., Formal Modeling of Function Block Applications Running in IEC 61499 Execution Runtime. in *11th IEEE International Conference on Emerging Technologies and Factory Automation*, (Prague, Czech Republic 2006).

These higher abstractions rely on WCET, which is increasingly problematic



Example of what it takes to do WCET analysis:

Ferdinand et al. determine the WCET of astonishingly simple avionics code from Airbus running on a Motorola ColdFire 5307, a pipelined CPU with a unified code and data cache. Despite the software consisting of a fixed set of non-interacting tasks containing only simple control structures, their solution required detailed modeling of the seven-stage pipeline and its precise interaction with the cache, generating a large integer linear programming problem. The technique successfully computes WCET, but only with many caveats that are increasingly rare in software.

Fundamentally, the ISA of the processor has failed to provide an adequate abstraction.

C. Ferdinand et al., “Reliable and precise WCET determination for a real-life processor.” EMSOFT 2001.

A Key Problem

Electronics technology delivers highly and precise timing…

… and the overlaying software abstractions discard it.

"Correct" execution of a C program has nothing to do with how long it takes to do anything.

Instead of a Program Specifying…

$$f : \{0,1\}^* \rightarrow \{0,1\}^*$$

… a (partial) function from bit sequences to bit sequences …

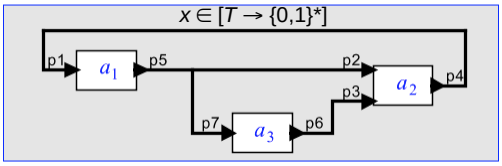
… A Program Should Specify

$$f : \underbrace{[T \to \{0,1\}^*]^P}_{\text{“signal”}} \to \underbrace{[T \to \{0,1\}^*]^P}_{\text{“signal”}}$$

“actor”

…where T is a (partially) ordered set representing time, precedence ordering, causality, synchronization, etc.

This is the Basis for Actor-Oriented Models



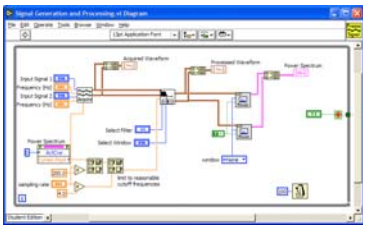
Cascade connections
Parallel connections
Feedback connections

If actors are functions on signals, then the nontrivial part of this is feedback.

Many of the Models of Computation we have seen fit this view:

- *Time-Triggered*
- *Discrete Events*
- *Dataflow*
- *Rendezvous*
- *Synchronous/Reactive*
- *Continuous Time*
- *Mixtures of the above*
- *…*

Examples of Actor-Oriented "Languages"



- CORBA event service (distributed push-pull)
- LabVIEW (dataflow, National Instruments)
- Modelica (continuous-time, Linkoping)
- OPNET (discrete events, Opnet Technologies)
- Occam (rendezvous)
- ROOM and UML-2 (dataflow, Rational, IBM)
- SCADE and synchronous languages (synchronous/reactive)
- SDL (process networks)
- Simulink (Continuous-time, The MathWorks)
- SPW (synchronous dataflow, Cadence, CoWare)
- VHDL, Verilog (discrete events, Cadence, Synopsys, ...)
- …

Many of these are domain specific.

Many of these have visual syntaxes.

The semantics of these differ considerably, but all can be modeled as

$$f : [T \rightarrow \{0,1\}^*]^P \rightarrow [T \rightarrow \{0,1\}^*]^P$$

with appropriate choices of the set T.

The notion of time is not so easy to nail down

- Is "current time" knowable to all players in a system?
- Can cause and effect take zero time?
- Does simultaneity imply nondeterminism?
- What is concurrency?
- Does concurrency imply nondeterminism?