Building Unreliable Systems out of Reliable Components: 
*The Real Time Story*

Edward A. Lee  
Professor, Chair of EE, and Associate Chair of EECS  
CHESS: Center for Hybrid and Embedded Software Systems  
UC Berkeley

Shannon Lecture  
IEEE-Computer Society, Silicon Valley Chapter  
Stanford University, November 17, 2005.
Electronics Technology Delivers Timeliness

... and the overlaying abstractions discard it.
Computation in the 20th Century

\[ f : \{0,1\}^* \rightarrow \{0,1\}^* \]
Computation in the 20th Century

- Time is irrelevant
- All actions are ordered
- Nontermination is a defect
- Concurrency is an illusion

Alan Turing
Exploiting the 20th Century Abstraction

- Programming languages
- Debuggers
- Virtual memory
- Caches
- Dynamic dispatch
- Speculative execution
- Power management (voltage scaling)
- Memory management (garbage collection)
- Just-in-time (JIT) compilation
- Multitasking (threads and processes)
- Networking (TCP)
- Theory (complexity)
What about timeliness?

Moore’s law has saved us!
In Core Software Abstractions: Real-Time is Not

- Time is not in the semantics of programs.
  - Have to step outside the semantics to specify timing.

- Timing is a consequence of implementation not a property of design.
  - Measured on the bench
  - For a particular realization

- Resulting systems are brittle.
  - Small changes have big consequences
  - Ports to new platforms require redesign
The Myth of WCET

Worst-Case Execution Time

- True WCET can be thousands of times bigger than actual execution time.

- In many implementations, true WCET is not a useful number.

- Dubious WCET is what is actually used.

- Correctness of even safety-critical systems depends on WCET being correct.
What is Done in Practice

- Real-time systems are boxes, not software services.
- Critical real-time systems use idiosyncratic, non-mainstream processors (like DSPs).
- Designs are bench tested, then encased.
The question: What would have to change to achieve *absolutely, positively on time* (APOT)?

The answer: *nearly everything.*
What to do?

- Put time into programming languages
  - *Promising start:* Simulink, Giotto, Discrete-event models
- Rethink the OS/programming language split
  - *Promising start:* TinyOS/nesC
- Rethink the hardware/software split
  - *Promising start:* FPGAs with programmable cores
- Memory hierarchy with predictability
  - *Promising start:* Scratchpad memories vs. caches
- Memory management with predictability
  - *Promising start:* Bounded pause time garbage collection
- Predictable, controllable deep pipelines
  - *Promising start:* Pipeline interleaving + stream-oriented languages
- Predictable, controllable, understandable concurrency
  - *Promising start:* Synchronous languages, SCADE
- Networks with timing
  - *Promising start:* Time triggered architectures, time synchronization
- Computational dynamical systems theory
  - *Promising start:* Hybrid systems, schedulability analysis
Recall: Computation in the 20th Century

\[ f: \{0,1\}^* \rightarrow \{0,1\}^* \]
Computation in the 21\textsuperscript{st} Century

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

…where \( T \) is a set representing time, precedence ordering, causality, synchronization, etc.
A Consequence: Component Abstractions Need to Change

Object-oriented:

- class name
- data
- methods

What flows through an object is sequential control
Stuff happens to objects

Actor oriented:

- actor name
- data (state)
- parameters
- ports

What flows through an object is streams of data
 Actors make things happen

Input data    Output data
The First (?) Actor-Oriented Platform

*The On-Line Graphical Specification of Computer Procedures*

W. R. Sutherland, Ph.D. Thesis, MIT, 1966

Bert Sutherland used the first acknowledged object-oriented framework (Sketchpad, created by his brother, Ivan Sutherland) to create the first actor-oriented programming framework.

Partially constructed actor-oriented model with a class definition (top) and instance (below).
Your Speaker in 1966
Modern Examples of Actor-Oriented Platforms

- Simulink (The MathWorks)
- LabVIEW (National Instruments)
- Modelica (Linkoping)
- OPNET (Opnet Technologies)
- Giotto and xGiotto (UC Berkeley)
- Polis & Metropolis (UC Berkeley)
- Gabriel, Ptolemy, and Ptolemy II (UC Berkeley)
- OCP, open control platform (Boeing)
- GME, actor-oriented meta-modeling (Vanderbilt)
- SPW, signal processing worksystem (Cadence)
- System studio (Synopsys)
- ROOM, real-time object-oriented modeling (Rational)
- Easy5 (Boeing)
- Port-based objects (U of Maryland)
- I/O automata (MIT)
- VHDL, Verilog, SystemC (Various)
- …
Ptolemy II: Our Laboratory for Actor-Oriented Models of Computation

Concurrency management supporting dynamic model structure.

Extensible, behaviorally-polymorphic component library.

Director from an extensible library defines component interaction semantics or "model of computation."

Type system for transported data

Visual editor supporting an abstract syntax
Models of Computation Implemented in Ptolemy II

- CI – Push/pull component interaction
- Click – Push/pull with method invocation
- CSP – concurrent threads with rendezvous
- CT – continuous-time modeling
- DE – discrete-event systems
- DDE – distributed discrete events
- DDF – Dynamic dataflow
- DPN – distributed process networks
- DT – discrete time (cycle driven)
- FSM – finite state machines
- Giotto – synchronous periodic
- GR – 2-D and 3-D graphics
- PN – process networks
- SDF – synchronous dataflow
- SR – synchronous/reactive
- TM – timed multitasking

Most of these are actor oriented.
A New Foundation for Computation

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

...where \( T \) is a set representing time, precedence ordering, causality, synchronization, etc.
A Start on a 21st Century Theory of Computation: The Tagged Signal Model

[Lee & Sangiovanni-Vincentelli, 1998]

- A set of values $V$ and a set of tags $T$
- An event is $e \in T \times V$
- A signal $s$ is a set of events. I.e. $s \subseteq T \times V$
- A functional signal is a (partial) function $s : T \rightarrow V$
- The set of all signals $S = 2^{T \times V}$

Related models:
- Interaction Categories [Abramsky, 1995]
- Interaction Semantics [Talcott, 1996]
- Abstract Behavioral Types [Arbab, 2005]
Actors, Ports, and Behaviors

An actor has a set of ports $P$

\[ P_A = \{ p_1, p_2, p_3, p_4 \} \]

A behavior is a function $\sigma: P_A \rightarrow S$

An actor is a set of behaviors $A \subset [P_A \rightarrow S] = S^{P_A}$
Actor Composition

Composition is simple intersection
(of sets of functions)

\[ A = A_1 \wedge A_2 \quad P = P_1 \cup P_2 \]

\[ A = A_1 \wedge A_2 = \{ \sigma \mid \sigma \downarrow_{P_1} \in A_1 \text{ and } \sigma \downarrow_{P_2} \in A_2 \} \subseteq [P \rightarrow S] \]
Connectors are trivial actors.

\[ P_1 = \{p_1, p_2\} \quad P_2 = \{p_3, p_4\} \]

\[ P_c = \{p_2, p_3\} \]

\[ c \subseteq [P_c \rightarrow S], \quad \forall \sigma \in c, \quad \forall p_1, p_2 \in P_c, \quad \sigma(p_1) = \sigma(p_2) \]

\[ A = A_1 \land A_2 \land c \]
Tagged Signal Model Gives a Fixed-Point Semantics to Arbitrary Composition

(a)

(b)

(c)
Tagged Signal Model can be used on a Wide Variety of Concurrent and Timed Models of Computation

- CSP – concurrent threads with rendezvous
- CT – continuous-time modeling
- DE – discrete-event systems
- DDF – Dynamic dataflow
- DT – discrete time
- Giotto – synchronous periodic
- PN – process networks
- SDF – synchronous dataflow
- SR – synchronous/reactive
Application of this Theory of Computation: Discrete-Event Systems

- CI – Push/pull component interaction
- Click – Push/pull with method invocation
- CSP – concurrent threads with rendezvous
- CT – continuous-time modeling
- **DE – discrete-event systems**
- DDE – distributed discrete events
- DDF – Dynamic dataflow
- DPN – distributed process networks
- DT – discrete time (cycle driven)
- FSM – finite state machines
- Giotto – synchronous periodic
- GR – 2-D and 3-D graphics
- PN – process networks
- SDF – synchronous dataflow
- SR – synchronous/reactive
- TM – timed multitasking

Lee, Berkeley 27
Discrete Events (DE): A Timed Concurrent Model of Computation

DE Director implements timed semantics using an event queue

Reactive actors

Event source

Signal

Time line
By default, an actor produces events with the same time as the input event. But in this example, we expect (and need) for the BooleanSwitch to “see” the output of the Bernoulli in the same “firing” where it sees the event from the PoissonClock. Events with identical time stamps are also ordered, and reactions to such events follow data precedence order.
Semantics Clears Up Subtleties: Feedback

Data precedence analysis has to take into account the non-strictness of this actor (that an output can be produced despite the lack of an input).
Semantics Clears Up Subtleties: Zeno Systems

DE systems may have an infinite number of events in a finite amount of time. Carefully constructed semantics gives these systems meaning.

This model illustrates a Zeno condition, where an infinite number of events occur before time 2.0, and hence the Clock actor is unable to ever produce its output at time 2.0.
Example of Current Research Challenges

Use distributed discrete-event systems as a timed model of computation for embedded software in unreliable, sporadically connected networks, such as wireless sensor networks.

The most interesting possibilities are based on distributed consensus algorithms (as in Croquet, Reed, Lamport).

Research challenges include:
- Defining the semantics
- Combining the semantics heterogeneously with others. E.g.:
  - Signal processing for channel modeling
  - TinyOS for node functionality
- Creating efficient runtime environments
- Building the design environment
Current Projects in the Ptolemy Group

- Abstract semantics (Cataldo, Liu, Matsikoudis, Zheng)
  - Behavioral polymorphism
  - Actor semantics (prefire, fire, postfire)
  - Compositional directors
  - Time semantics
  - Causality interfaces

- Distributed computing (Feng, Zhao)
  - Robust distributed consensus
  - Data coherence (distributed caches)
  - Time synchronization

- Real-time software (Bandyopadhyay, Cheong, Zhou)
  - Time-based models vs. dataflow models
  - Deterministic, understandable multitasking
  - Memory hierarchy with scratchpad memory
  - Code generation

- Hybrid systems (Cataldo, Zheng)
  - Operational semantics
  - Stochastic hybrid systems
  - Aspect-oriented multi-view modeling
  - Code generation
Conclusion

*The time is right to create the 21-st century theory of (embedded) computing.*