Embedded Intelligent Systems

Edward A. Lee
Robert S. Pepper Distinguished Professor
UC Berkeley

Keynote: ARTEMIS Sprint Event

April 13, 2016
Vienna, Austria
ARTEMIS: 2006 Annual Conference, Graz, Austria, May 22-24, 2006
Cyber-Physical Systems
Full of Contradictory Requirements

It’s not just information technology anymore:

- Cyber + Physical
- Computation + Dynamics
- Security + Safety

Contradictions:

- Adaptability vs. Repeatability
- High connectivity vs. Security and Privacy
- High performance vs. Low Energy
- Asynchrony vs. Coordination/Cooperation
- Scalability vs. Reliability and Predictability
- Laws and Regulations vs. Technical Possibilities
- Economies of scale (cloud) vs. Locality (fog)
- Open vs. Proprietary
- Algorithms vs. Dynamics

Innovation:

Cyber-physical systems require new engineering methods and models to address these contradictions.

Lee, Berkeley
Industrial automation example from 2008: Bosch-Rexroth printing press.

The term “IoT” includes the technical solution “Internet technology” in the problem statement “connect things”.

The term CPS does not.

Lee, Berkeley

This Bosch Rexroth printing press is a cyber-physical factory using Ethernet and TCP/IP with high-precision clock synchronization (IEEE 1588) on an isolated LAN.
Focus on Models
Models vs. Reality

\[ x(t) = x(0) + \int_0^t v(\tau) d\tau \]
\[ v(t) = v(0) + \frac{1}{m} \int_0^t F(\tau) d\tau. \]

The model

The target (the thing being modeled).

In this example, the *modeling framework* is calculus and Newton’s laws.

*Fidelity* is how well the model and its target match.

Lee, Berkeley
Engineers often confuse the model with its target

You will never strike oil by drilling through the map!

But this does not in any way diminish the value of a map!

Solomon Wolf Golomb
Determinacy

Some of the most valuable models are \textit{deterministic}.

A model is \textit{deterministic} if, given the initial state and the inputs, the model defines exactly one behavior.

Deterministic models have proven extremely valuable in the past.
“We may regard the present state of the universe as the effect of its past and the cause of its future. An intellect which at a certain moment would know all forces that set nature in motion, and all positions of all items of which nature is composed, if this intellect were also vast enough to submit these data to analysis, it would embrace in a single formula the movements of the greatest bodies of the universe and those of the tiniest atom; for such an intellect nothing would be uncertain and the future just like the past would be present before its eyes.”

— Pierre Simon Laplace, A Philosophical Essay on Probabilities

Pierre-Simon Laplace (1749–1827). Posthumous portrait by Joan-Baptiste Paulin Guérin, 1838

Lee, Berkeley
But hasn’t Laplace’s demon been debunked?

Deterministic model

\[ x(t) = x(0) + \int_0^t v(\tau) d\tau \]
\[ v(t) = v(0) + \frac{1}{m} \int_0^t F(\tau) d\tau. \]

Laplace reflected a period of optimism in science that tended to equate the model with the truth.

He was drilling through a map!
A Model Need not be *True* to be *Useful*

“Essentially, all models are wrong, but some are useful.”

What kinds of models should we use?

Let’s look at the most successful kinds of models from the cyber and the physical worlds.
Software is a Model

Lee, Berkeley

Physical System

Model

Single-threaded imperative programs are deterministic models
Consider single-threaded imperative programs

This program defines exactly one behavior, given the input x.

Note that the modeling framework (the C language, in this case) defines “behavior” and “input.”

The target of the model is nondeterministic (electrons and holes sloshing around in silicon).

Lee, Berkeley
Software relies on another deterministic model that abstracts the hardware.

**Physical System**

**Model**

---

**Integer Register-Register Operations**

RISC-V defines several arithmetic R-type operations. All operations read the $rs1$ and $rs2$ registers as source operands and write the result into register $rd$. The `funct` field selects the type of operation.

<table>
<thead>
<tr>
<th>Field</th>
<th>rd</th>
<th>rs1</th>
<th>rs2</th>
<th>funct10</th>
<th>7 6</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>src1</td>
<td>dest</td>
<td>src1</td>
<td>src2</td>
<td>ADD/SUB/SLT/SLTU</td>
<td>OP</td>
<td></td>
</tr>
<tr>
<td>src1</td>
<td>dest</td>
<td>src1</td>
<td>src2</td>
<td>AND/OR/XOR</td>
<td>OP</td>
<td></td>
</tr>
<tr>
<td>src1</td>
<td>dest</td>
<td>src1</td>
<td>src2</td>
<td>SLL/SRL/SRA</td>
<td>OP</td>
<td></td>
</tr>
<tr>
<td>src1</td>
<td>dest</td>
<td>src1</td>
<td>src2</td>
<td>ADDW/SUBW</td>
<td>OP-32</td>
<td></td>
</tr>
<tr>
<td>src1</td>
<td>dest</td>
<td>src1</td>
<td>src2</td>
<td>SLLW/SRLW/SRAW</td>
<td>OP-32</td>
<td></td>
</tr>
</tbody>
</table>

---


---

Lee, Berkeley
... which relies on yet another deterministic model

Physical System

Model

Synchronous digital logic is a deterministic model

Lee, Berkeley
Deterministic Models for the Physical Side of CPS

Physical System

Model

\[ \dot{x}(t) = \dot{x}(0) + \frac{1}{M} \int_{0}^{t} F(\tau) d\tau \]

**Differential Equations are deterministic models**

Lee, Berkeley
A Major Problem for CPS: Combinations of Deterministic Models are Nondeterministic

```c
1 void initTimer(void) {
2     SysTickPeriodSet(SysCtlClockGet() / 1000);
3     SysTickEnable();
4     SysTickIntEnable();
5 }
6 volatile uint timer_count = 0;
7 void ISR(void) {
8     if(timer_count != 0) {
9         timer_count--;
10    }
11 }
12 int main(void) {
13     SysTickIntRegister(&ISR);
14     // other init
15     timer_count = 2000;
16     initTimer();
17     while(timer_count != 0) {
18         ... code to run for 2 seconds
19     }
20     ... // other code
21 }
```

\[ \dot{x}(t) = \dot{x}(0) + \frac{1}{M} \int_{0}^{t} F(\tau) d\tau \]
Correct execution of a program in all widely used programming languages, and correct delivery of a network message in all general-purpose networks has nothing to do with how long it takes to do anything.

Programmers have to step outside the programming abstractions to specify timing behavior.

Embedded software designers have no map!

Lee, Berkeley
Despite using TCP/IP on Ethernet, this network achieves highly reliable bounded latency.

TSN (time-sensitive networks) technology is starting to become pervasive…

This Bosch Rexroth printing press is a cyber-physical factory using Ethernet and TCP/IP with high-precision clock synchronization (IEEE 1588) on an isolated LAN.

Lee, Berkeley
CPS applications operate in an intrinsically nondeterministic world.

Does it really make sense to insist on deterministic models?
In *science*, the value of a *model* lies in how well its behavior matches that of the physical system.

In *engineering*, the value of the *physical system* lies in how well its behavior matches that of the model.

*In engineering, model fidelity is a two-way street!*

*For a model to be useful, it is necessary (but not sufficient) to be able to construct a faithful physical realization.*
A Model
A Physical Realization

Lee, Berkeley
Model Fidelity

• To a *scientist*, the model is flawed.

• To an *engineer*, the realization is flawed.

I’m an engineer...
For CPS, we need to Change the Question

The question is *not* whether deterministic models can describe the behavior of cyber-physical systems (with high fidelity).

The question is whether we can build cyber-physical systems whose behavior matches that of a deterministic model (with high probability).
Deterministic models do not eliminate the need for robust, fault-tolerant designs.

In fact, they *enable* such designs, because they make it much clearer what it means to have a fault!
Suppose that events flowing in the network are time stamped.

Then violations of the assumptions on network latency bounds are detectable, for example.

This Bosch Rexroth printing press is a cyber-physical factory using Ethernet and TCP/IP with high-precision clock synchronization (IEEE 1588) on an isolated LAN.
We have to fix the models!

But how?

If I had more time, I would describe:

- PTIDES: deterministic distributed real-time software
- PRET machines: CPUs with deterministic timing

Invite me back...
Existence Proofs

Deterministic models with highly faithful implementations exist for distributed real-time systems.

- **PTIDES**: distributed real-time software
  - Deterministic timing across networks

- **PRET** machines
  - Deterministic timing at the processor level
Roots of the Idea

Using Time Instead of Timeout for Fault-Tolerant Distributed Systems

LESLIE LAMPORT
SRI International

A general method is described for implementing a distributed system with any desired degree of fault-tolerance. Instead of relying upon explicit timeouts, processes execute a simple clock-driven algorithm. Reliable clock synchronization and a solution to the Byzantine Generals Problem are assumed.


General Terms: Design, Reliability

Additional Key Words and Phrases: Clocks, transaction commit, timestamps, interactive consistency, Byzantine Generals Problem

ACM Transactions on Programming Languages and Systems, 1984.

Lee, Berkeley
Abstract: Discrete-event (DE) models are formal system specifications that have analyzable deterministic behaviors. Using a global, consistent notion of time, DE components communicate via time-stamped events. DE models have primarily been used in performance modeling and simulation, where time stamps are a modeling property bearing no relationship to real time during execution of the model. In this paper, we extend DE models with the capability of relating certain events to physical time...
Google independently developed a very similar technique and applied it to distributed databases.

Spanner: Google’s Globally-Distributed Database


Google, Inc.

Abstract

Spanner is Google’s scalable, multi-version, globally-distributed, and synchronously-replicated database. It is the first system to distribute data at global scale and support externally-consistent distributed transactions. This paper describes how Spanner is structured, its feature set, the rationale underlying various design decisions, and a novel time API that exposes clock uncertainty. This API and its implementation are critical to supporting external consistency and a variety of powerful features: non-blocking reads in the past, lock-free read-only transactions, and atomic schema changes, across all of Spanner.

Proceedings of OSDI 2012
Time-stamped events that are processed in time-stamp order.

This MoC is widely used in simulation and HDLs.

Given time-stamped inputs, it is a deterministic concurrent MoC.

A few texts that use the DE MoC
Ptides: Time stamps bind to real time at sensors and actuators

- Time stamp value is a deadline
- Time stamp value is time of measurement
- Actors wrap sensors
- Actors wrap actuators
- Messages are processed in time-stamp order

Lee, Berkeley
Assume bounds on:

- *clock synchronization error*
- *network latency*

then *events are processed in time-stamp order* at every component. If in addition we assume

- *bounds on execution time*

then events are delivered to actuators on time.

See [http://chess.eecs.berkeley.edu/ptides](http://chess.eecs.berkeley.edu/ptides)
All of the assumptions are achievable with today’s technology, and are requirements anyway for hard-real-time systems. A Ptides model makes the requirements explicit.

Violations of the requirements are detectable as out-of-order events and can be treated as faults.

Lee, Berkeley
A fault manifests as out-of-order events.

Occurrence of a fault implies one or more of the assumptions was violated.

If an event arrives here with an earlier time stamp…

… after an event here with a later time stamp has been processed, then fault!
High-precision clock synchronization will become ubiquitous. Networks will become able to “guarantee” bounded latencies.

The time is right.

*But what about the execution time of software?*
Existence Proofs

Deterministic models with highly faithful implementations exist for distributed real-time systems.

- **PTIDES**: distributed real-time software
  - Deterministic timing across networks
- **PRET machines**
  - Deterministic timing at the processor level
The hardware out of which we build computers is capable of delivering “correct” computations and precise timing…

The synchronous digital logic abstraction removes the messiness of transistors.

… but the overlaying software abstractions discard the timing precision.

```c
// Perform the convolution.
for (int i=0; i<10; i++) {
    x[i] = a[i]*b[j-i];
    // Notify listeners.
    notify(x[i]);
}
```
PRE Machines – Giving Software the Capabilities their Hardware Already Has.

- **PREcision-Timed processors = PRET**
- **Predictable, REpeatable Timing = PRET**
- **Performance with REpeatable Timing = PRET**

*http://chess.eecs.berkeley.edu/pret*

```java
// Perform the convolution.
for (int i=0; i<10; i++) {
    x[i] = a[i]*b[j-i];
    // Notify listeners.
    notify(x[i]);
}
```

Lee, Berkeley
Major Challenges
and existence proofs that they can be met

• Pipelines
  – fine-grain multithreading
• Memory hierarchy
  – memory controllers with controllable latency
• I/O
  – threaded interrupts, with bounded effects on timing
PRET Memory Systems:
- DRAM controller
  - [Reineke et al., CODES+ISSS 2011]
- Scratchpad management
  - [Kim et al., RTAS, 2014]
- Mixed criticality DRAM controller
  - [Kim et al., RTAS 2015]

PRET Principle:
- The case for PRET
  - [Edwards & Lee, DAC 2007]
- PRET ISA extensions
  - [Edwards at al., ICCD 2009]
- Temporal isolation
  - [Bui et al., DAC, 2011]
- Design challenges
  - [Broman et al., ESLsyn, 2013]
- Cyber-physical systems
  - [Lee., Sensors, 2015]
Three Generations of PRET Machines at Berkeley

• PRET1, Sparc-based (simulation only)
  – [Lickly et al., CASES, 2008]

• PTARM, ARM-based (FPGA implementation)
  – [Liu et al., ICCD, 2012]

• FlexPRET, RISC-V-based (FPGA + simulation)
Our Second Generation PRET

PTArm, a soft core on a Xilinx Virtex 5 FPGA (2012)

Interleaved pipeline with one set of registers per thread

SRAM scratchpad shared among threads

DRAM main memory, separate banks per thread

I/O devices

Our Third-Generation PRET: Open-Source FlexPRET (Zimmer 2014/15)

- 32-bit, 5-stage thread interleaved pipeline, RISC-V ISA
  - **Hard real-time HW threads:** scheduled at constant rate for isolation and repeatability.
  - **Soft real-time HW threads:** share all available cycles for efficiency.
- Deployed on Xilinx FPGA (area comparable to Microblaze)

---

Lee, Berkeley
FlexPRET

Hard-Real-Time (HRT) Threads Interleaved with Soft-Real-Time (SRT) Threads

HRT threads have deterministic timing. SRT threads share available cycles.

SRAM scratchpad shared among threads.

DRAM main memory provides deterministic latency for HRT threads. Conventional behavior for the rest.

Michael Zimmer
Interrupt-Driven I/O is notorious for disrupting timing.

Interrupts have *no effect* on HRT threads, and *bounded effect* on SRT threads!

*Michael Zimmer*
No!

The PRET project has shown that you do not need to sacrifice performance to get control over timing.

Lee, Berkeley

FlexPRET Shows:

- Not only is there no performance cost for appropriate workloads, but there is also no performance cost for inappropriate workloads!

- Pipelining, memory hierarchy, and interrupt-driven I/O can all be done without losing timing determinacy!
Deterministic models with highly faithful implementations exist for distributed real-time systems.

- PTIDES: distributed real-time software
  - Deterministic timing across networks
- PRET machines
  - Deterministic timing at the processor level

We have run out of excuses for sloppy engineering!