The Internet of Important Things

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

Keynote

Software Engineering and Formal Methods (SEFM)
September 11, 2015.
York, England, UK
The TerraSwarm Research Center 2013-2018

What it is:
The TerraSwarm Research Center is addressing the huge potential (and associated risks) of pervasive integration of smart, networked sensors and actuators into our connected world.

The Goal
To lead the world in development of the platforms, methodologies, and tools that enable invention of creative, secure, and sound applications using networked sensors and actuators.

Lee, Berkeley
TerraSwarm Sites

U. of Washington
- Learning as a service
- Learning on streams

UC Berkeley
- Composable architecture
- Design, verification, synthesis
- Networking
- Distributed services
- Control synthesis

Caltech
- Control synthesis

UC San Diego
- Smart grids
- Smart buildings

Michigan
- Localization
- Smart dust
- Energy scavenging
- Security and privacy

CMU
- Mobile vehicles
- Privacy
- Control synthesis

Penn
- Localization

Illinois
- Resource management

UT Dallas
- Context awareness

Lee, Berkeley
Buzzword du jour: The Internet of Things

Using Internet technology to interact with physical devices (“things”).

http://www.gartner.com/technology/research/hype-cycles/
... but the idea has been around for a while...
Our Focus: Cyber-Physical Systems (CPS)
The Internet of *Important* Things (IoIT)

Using Internet technology to interact with physical devices (“things”).

We are interested in systems where safety and reliability loom large.

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.
It’s not just information technology anymore:
• Cyber + Physical
• Computation + Dynamics
• Security + Safety

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

Innovation:
Cyber-physical systems are fundamentally different from computational systems and from physical systems. They require new engineering models that embrace temporal dynamics and algorithmic computation.
A RESTful service [Fielding & Taylor 2002] is accessed using a design pattern common on the web that we call *Asynchronous Atomic Callbacks* (AAC).

In the Web, AAC is widely used. It is central to many popular internet programming frameworks such as Node.js & Vert.x, and to CPS frameworks such as TinyOS.

Lee, Berkeley
// Import a module providing network services
var http = require("http");

// Construct a URL encoding a request
var url = "http://foo.com/deviceID/...";

// Issue the request and provide a callback
http.get(url, function(response) {
    // ... handle the response ...
});

The callback function will be called atomically some time later when the server response arrives.
Another Common Design Pattern: Actors

Streaming requests:

Sequence of requests for a service (a *stream*) triggers a sequence of responses.
Streaming requests:

But the responses may not come back in the same order as the requests!
This is a rudimentary timing problem.
The order and timing of events matters a lot when interacting with physical processes.

The system at the right orchestrates hundreds of microcontrollers to deposit ink on paper flying through the printer at 100 kmh with micron precision.

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.
Actor-oriented wrappers for networked devices and services. Some examples:
E.g. time-stamped events processed in time-stamp order (a discrete-event (DE) model of computation (MoC)).

E.g. asynchronous atomic callbacks (AAC).

Lee, Berkeley
• Semantics of timed systems
  – Fixed point theorems, contractions, causality, and constructiveness on generalized ultrametric semilattices [Matsikoudis & Lee, 2013/14/15].

• Interface theories
  – Type systems, behavioral interfaces, and concurrent composition [Lohstroh & Lee, 2015, Lee & Xiong 2003].
Behavioral Interfaces to Codify Contracts

- Encode the possible sequences of interactions using *interface automata* [de Alfaro & Henzinger, 2001].
- Use automated tools to compose the automata, checking for incompatibilities.

*Example automaton for an accessor*

Checking Compatibility of Contracts

**DE Director**

**Accessor**

**JavaScript**

Horizontal Contract

Vertical Contract


Lee, Berkeley
Accessor Architecture Today
Version 0.1a

Accessor Interface
implements

Accessor Interface + JavaScript

Accessor Interface

requires

Module API Definition
CommonJS + Text
implements

Accessor Host
Nashorn + Ptolemy II

Accessor Host
Nashorn + Java

Accessor Host
Node.js

Accessor Host
Browser

Module Implementation
JavaScript + Java (Nashorn)

Module Implementation
JavaScript (Node.js)

Module Implementation
JavaScript
Using Accessors

Today, the TerraSwarm project is demonstrating the use of accessors to compose services to orchestrate a fleet of robots.

In St. Louis, MO, at DARPA’s *Wait, What?* Exposition.

*Lee, Berkeley*
Because the interfaces are formal objects, virtual prototypes and simulations behave the same as a (correct) implementation.
Because the interfaces are formal objects, virtual prototypes and simulations behave the same as a (correct) implementation.

Scarab robot simulation has the same interface as the robot itself.
Focus on the horizontal contract

horizontal contract governs actor interactions

- Actor → Accessor → Actor
- Swarmlet

vertical contract governs the interaction between the accessor and the service or thing

- Request → Service Implementation → Response
- Swarm service or thing

runs on an accessor host

Time-stamped events processed in time-stamp order (a discrete-event (DE) model of computation (MoC)).

E.g. asynchronous atomic callbacks (AAC).

Lee, Berkeley
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.
If the thing is black box with a RESTful interface, then we time stamp the response... but if we can design the thing, we can do much better!

Networks of causal actors are deterministic under the DE MoC.
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...
Ptides: First step: Time stamps bind to real time at sensors and actuators

- Actors wrap sensors
- Time stamp value is time of measurement
- Time stamp value is a deadline
- Actors wrap actuators

Lee, Berkeley
Ptides: Second step: Time-stamped messages.

Actors specify computation

Messages carry time stamps that define their interleaving
Ptides: Third step: Network clock synchronization

GPS, NTP, IEEE 1588, TSN, time-triggered busses, ... they all work. We just need to bound the clock synchronization error.

Assume bounded clock error $e$

Clock synchronization gives global meaning to time stamps

Messages are processed in time-stamp order
Global latencies between sensors and actuators become controllable, which enables analysis of system dynamics.

Feedback through the physical world

Model includes manipulations of time stamps, which control latencies between sensors and actors

Actuators may be designed to interpret input time stamps as the time at which to take action.
Safe-to-process analysis guarantees that events are processed in time-stamp order, given some assumptions.

Assume bounded sensor delay $s$

Assume bounded network delay $d$

Assume bounded clock error $e$

Technical: Need to have deadlines on network interfaces, to guarantee time-stamp order irrespective of execution times of actors.

Application specification of latency $d_2$

An earliest event with time stamp $t$ here can be safely merged when real time exceeds $t + s + d + e - d_2$
So Many Assumptions?

Solomon Wolf Golomb: On keeping the model distinct from the thing being modeled:

You will never strike oil by drilling through the map!

All of the assumptions are achievable with today’s technology, and in fact are requirements anyway for hard-real-time systems. The Ptides model makes the assumptions explicit.

Violations of the assumptions are detectable as out-of-order events and can be treated as faults.
A “fault” is a violation of assumptions in the model.

As with any model, the physical world may not conform to its rules. Violations should be treated as faults.

If an event arrives here with an earlier time stamp…

… after an event here with a later time stamp has been processed, then one or more assumptions was violated.

Lee, Berkeley
Google independently developed a very similar technique and applied it to distributed databases.
Distributed database with redundant storage and query handling across data centers.

Update to a record comes in. Time stamp $t_1$.

Query for the same record comes in. Time stamp $t_2$. 
Update to a record comes in. Time stamp $t_1$. 

Query for the same record comes in. Time stamp $t_2$. 

If $t_2 < t_1$, the query response should be the pre-update value. Otherwise, it should be the post-update value.
Update to a record comes in. Time stamp $t_1$.

Communication latency bound $b$.

Query for the same record comes in. Time stamp $t_2$.

Synchronize clocks with error bound $e$.

When the local clock time exceeds $t_2 + e + d$, issue the current record value as a response.
If after sending a response, we receive a record update with time stamp \( t_1 < t_2 \) declare a fault. Spanner handles this with a transaction schema.
See

- Chapter 8: Discrete-Event Models
- Chapter 10: Modeling Timed Systems

Free download at:
http://ptolemy.org/systems
IoIT applications operate in an intrinsically nondeterministic world.

*Does it really make sense to insist on deterministic models?*
Deterministic Models of Nondeterministic Systems

Physical System

Model

Synchronous digital logic

Lee, Berkeley
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>31</th>
<th>27</th>
<th>26</th>
<th>22</th>
<th>21</th>
<th>17</th>
<th>16</th>
<th>7</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rd</td>
<td>rs1</td>
<td>rs2</td>
<td>funct10</td>
<td>opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dest</td>
<td>src1</td>
<td>src2</td>
<td>ADD/SUB/SLT/SLTU</td>
<td>OP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dest</td>
<td>src1</td>
<td>src2</td>
<td>AND/OR/XOR</td>
<td>OP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dest</td>
<td>src1</td>
<td>src2</td>
<td>SLL/SRL/SRA</td>
<td>OP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dest</td>
<td>src1</td>
<td>src2</td>
<td>ADDW/SUBW</td>
<td>OP-32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dest</td>
<td>src1</td>
<td>src2</td>
<td>SLLW/SRLW/SRAW</td>
<td>OP-32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


**Instruction Set Architectures (ISAs)**

Lee, Berkeley
Deterministic Models of Nondeterministic Systems

Physical System

Model

```java
/** Reset the output receivers, which are the inside receivers of
 * the output ports of the container.
 * @exception IllegalArgumentException If getting the receivers fails.
 */
private void _resetOutputReceivers() throws IllegalArgumentException {
    List<IOPort> outputs = ((Actor) getContainer()).outputPortList();
    for (IOPort output : outputs) {
        if (_debugging) {
            _debug("Resetting inside receivers of output port: " + output.getName());
        }
        Receiver[][] receivers = output.getInsideReceivers();
        if (receivers != null) {
            for (int i = 0; i < receivers.length; i++) {
                if (receivers[i] != null) {
                    for (int j = 0; j < receivers[i].length; j++) {
                        if (receivers[i][j] instanceof FSMReceiver) {
                            receivers[i][j].reset();
                        }
                    }
                }
            }
        }
    }
}
```

Single-threaded imperative programs
Deterministic Models of Nondeterministic Systems

Physical System

Model

Differential Equations

\[ \dot{x}(t) = \dot{x}(0) + \frac{1}{M} \int_{0}^{t} F(\tau) d\tau \]
A Major Problem for CPS: Combinations of these Models are Nondeterministic

```java
/** Reset the output receivers, which are the inside receivers of
 * the output ports of the container.
 * @exception IllegalArgumentException If getting the receivers fails.
 */
private void _resetOutputReceivers() throws IllegalArgumentException {
    List<IOPort> outputs = ((Actor) getContainer()).outputPortList();
    for (IOPort output : outputs) {
        if (_debugging) {
            _debug("Resetting inside receivers of output port: " + output.getName());
        }
        Receiver[][] receivers = output.getInsideReceivers();
        if (receivers != null) {
            for (int i = 0; i < receivers.length; i++) {
                if (receivers[i] != null) {
                    for (int j = 0; j < receivers[i].length; j++) {
                        if (receivers[i][j] instanceof FSMReceiver) {
                            receivers[i][j].reset();
                        }
                    }
                }
            }
        }
    }
}
```

\[
\dot{x}(t) = \dot{x}(0) + \frac{1}{M} \int_{0}^{t} F(\tau) \, d\tau
\]
Correct execution of a program in C, C#, Java, Haskell, OCaml, Esterel, etc. has nothing to do with how long it takes to do anything. Nearly all our computation and networking abstractions are built on this premise.

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

Programmers have no map!
One Last Comment... Model Fidelity
... or ... How often will faults occur?

- In *science*, a good model matches well the behavior of the physical world.
- In *engineering*, a good physical implementation matches well the behavior 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 be able to construct a faithful physical realization.*
A Model
A Physical Realization
To a *scientist*, the model is flawed.

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

I’m an engineer...
PTIDES offers better models with less flawed physical realizations.
Determinism?

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!
IoIT and CPS demand new ways of modeling. They require models that embrace some notion(s) of *time*, and models with *strong formal properties*.

*Raffaello Sanzio da Urbino – The Athens School*