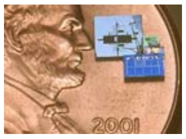
Introduction to Embedded Systems

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

Lecture 1: Cyber Physical Systems

2001 National Research Council Report *Embedded Everywhere*

"Information technology (IT) is on the verge of another revolution. Driven by the increasing capabilities and ever declining costs of computing and communications devices, IT is being embedded into a growing range of physical devices linked together through networks and will become ever more pervasive as the component technologies become smaller, faster, and cheaper... These networked systems of embedded computers ... have the potential to change radically the way people interact with their environment by linking together a range of devices and sensors that will allow information to be collected, shared, and processed in unprecedented ways. ... The use of [these embedded computers] throughout society could well dwarf previous milestones in the information revolution."



What are Embedded Systems?

Computational systems
- but not first-and-foremost a computer

Integral with physical processes
- sensors, actuators, physical dynamics

Reactive
- at the speed of the environment

Heterogeneous
- hardware/software/networks, mixed architectures

Networked
- concurrent, distributed, dynamic

Example: Automotive electronics today

Up to 80 computers (*electronic control units,* ECUs) in a premium car today:

- engine control, transmission, anti-lock brakes, electronic suspension, parking assistance, climate control, audio system, “body electronics” (seat belt, etc.), display and instrument panel, etc.
- linked together by CAN bus (today), FlexRay (tomorrow) with up to 2km of wiring.
- growing fraction of development costs, manufacturing costs, and fuel consumption.

Traditionally, embedded systems was an industrial (not academic) problem, principally about resource limitations.

- small memory
- small data word sizes
- relatively slow clocks

When these are the key problems, emphasize efficiency:
- write software at a low level (in assembly code or C)
- avoid operating systems with a rich suite of services
- develop specialized computer architectures:
  - programmable DSPs
  - network processors
- develop specialized networks
  - Can, FlexRay, TTP/C, MOST, etc.

This is how embedded SW has been designed for 30 years

But embedded systems do have more fundamental differences from general-purpose computation:

time matters
- "as fast as possible" is not good enough

concurrency is intrinsic
- it's not an illusion (as in time sharing), and
- it's not (necessarily) about exploiting parallelism

processor requirements can be specialized
- predictable, repeatable timing
- support for common operations (e.g. FIR filters)
- need for specialized data types (fixed point, bit vectors)

programs need to run (essentially) forever
- memory usage has to be bounded
- rebooting is not acceptable

Prevailing software engineering methods fall short

programming languages have no temporal semantics

- "correct" execution of C has nothing to do with how long it takes.

emphasis on expressiveness over analyzability

- programming languages are expected to be Turing complete
- bounded memory usage is undecidable
- termination is undecidable (we usually want non-termination)

emphasis on programmer convenience

- automated memory management (garbage collection)

modularity methods focus on static structure

- object-oriented design and type systems

performance optimization is not enough

- behavior has to be repeatable and predictable

MY ACCOMPLISHMENTS THIS MONTH INCLUDE COMPLYING WITH OUR ISO 9000, SARBANES-OXLEY AND SEI-5 POLICIES.
AND IF YOU MAKE A NEW POLICY, I WILL COMPLY WITH IT SO FAST IT WILL MAKE YOUR HEAD SPIN!
IS IT MY IMAGINATION OR IS PRETENDING TO WORK GETTING EASIER?
www.dilbert.com
scottadams@aol.com
© 2005 Scott Adams, Inc./Dist. by UFS, Inc.
8-5-05

© Scott Adams, Inc./Dist. by UFS, Inc.

One possibility is to augment standard software engineering with "non-functional properties"

Time
Security
Fault tolerance
Power consumption

("quality of service")

But the formulation of the question is very telling:

How is it that *when* a braking system applies the brakes is any less a *function* of the braking system than *how much* braking it applies?

What about "real time"?

Make it faster!

What if you need "absolutely positively on time"?

Today, most embedded software engineers write code, build your system, and test for timing. Model-based design seeks to specify dynamic behavior (including timing) and "compile" implementations that meet the behavior.

Real-time systems should not be about "quality of service" but rather about "correctness of service."

Traditionally, "***faster* is *better***."

This is like saying that for a roller coaster, "**stronger is better**."

We have to change the mindset to "**not fast enough is *wrong***!"

Real-Time Multitasking?

Prioritize and Pray!

All too often, real-time operating systems (RTOSs) are used in a rather ad hoc way. Without any particular principles, engineers tweak priorities until the prototype works under test.

The resulting system is *brittle*, meaning the small changes in the operating conditions (or in the design of the system) can cause big changes in behavior. For example, replacing the processor with a faster one can cause real-time failures.

An engineer's responsibility

- Korean Air 747 in Guam, 200 deaths (1997)
- 30,000 deaths and 600,000 injuries from medical devices (1985-2005)
  - perhaps 8% due to software?

source: D. Jackson, M. Thomas, L. I. Millett, and the Committee on Certifiably Dependable Software Systems, "Software for Dependable Systems: Sufficient Evidence?," National Academies Press, May 9 2007.

Beyond embedded systems: Cyber-Physical Systems (CPS)

CPS: Orchestrating networked computational resources with physical systems.

Some CPS applications:

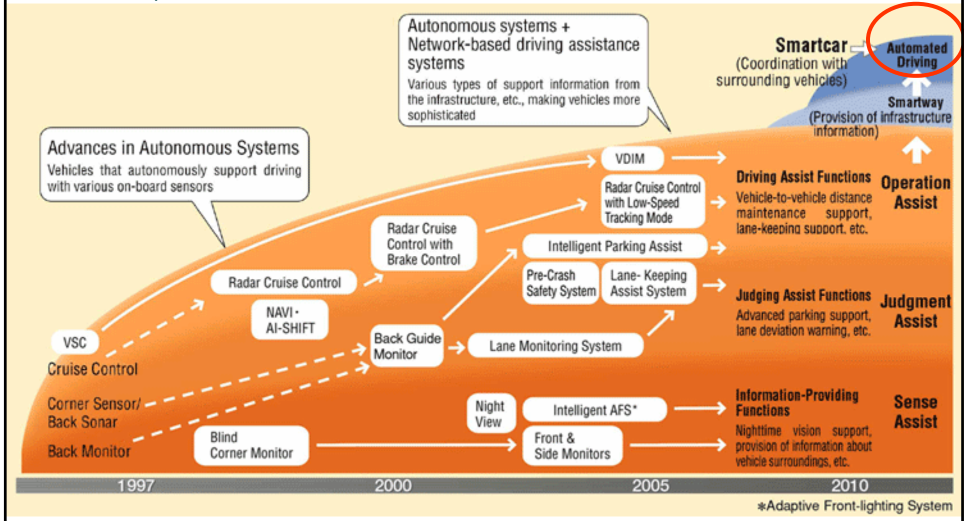
Dec. 11, 2006: Dancers in Berkeley dancing in real time with dancers in Urbana-Champagne

telepresence
distributed physical games
traffic control and safety
financial networks
medical devices and systems
assisted living
advanced automotive systems,
energy conservation
environmental control
aviation systems
critical infrastructure (power, water)
distributed robotics
military systems
smart structures
biosystems (morphogenesis,…)

Potential impact
social networking and games
safe/efficient transportation
fair financial networks
integrated medical systems
distributed micro power generation
military dominance
economic dominance
disaster recovery
energy efficient buildings
alternative energy
pervasive adaptive communications
distributed service delivery
…

Example: Toyota autonomous vehicle technology roadmap

Source: Toyota Web site



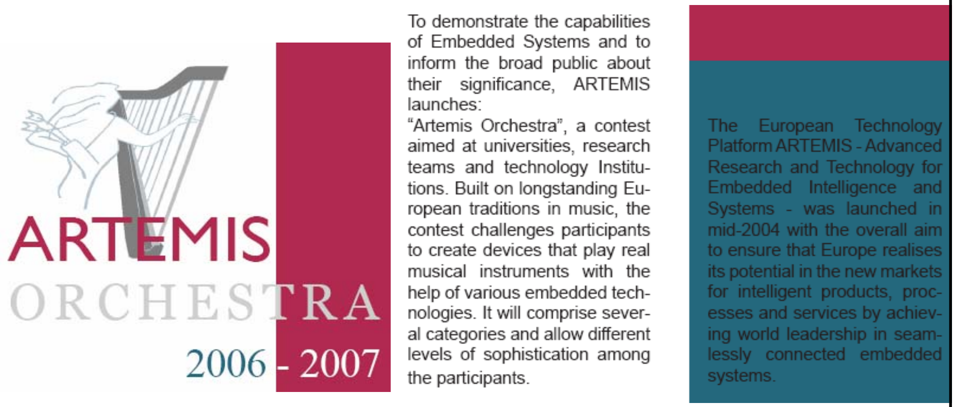
Autonomous systems + Network-based driving assistance systems
Various types of support information from the infrastructure, etc., making vehicles more sophisticated
Smartcar
(Coordination with surrounding vehicles)
Automated Driving
Smartway
(Provision of infrastructure information)
Advances in Autonomous Systems
Vehicles that autonomously support driving with various on-board sensors
VDIM
Radar Cruise Control with Low-Speed Tracking Mode
Driving Assist Functions
Vehicle-to-vehicle distance maintenance support, lane-keeping support, etc.
Operation Assist
Radar Cruise Control with Brake Control
Intelligent Parking Assist
Radar Cruise Control
Pre-Crash Safety System
Lane- Keeping Assist System
Judging Assist Functions
Advanced parking support, lane deviation warning, etc.
Judgment Assist
NAVI・AI-SHIFT
VSC
Cruise Control
Back Guide Monitor
Lane Monitoring System
Corner Sensor/ Back Sonar
Night View
Intelligent AFS*
Information-Providing Functions
Nighttime vision support, provision of information about vehicle surroundings, etc.
Sense Assist
Back Monitor
Blind Corner Monitor
Front & Side Monitors
1997
2000
2005
2010
*Adaptive Front-lighting System

EECS 124, UC Berkeley: 15

DARPA Grand Challenge

EECS 124, UC Berkeley: 16

Current European “grand challenge”


ARTEMIS
ORCHESTRA
2006 - 2007

To demonstrate the capabilities of Embedded Systems and to inform the broad public about their significance, ARTEMIS launches:
“Artemis Orchestra”, a contest aimed at universities, research teams and technology Institutions. Built on longstanding European traditions in music, the contest challenges participants to create devices that play real musical instruments with the help of various embedded technologies. It will comprise several categories and allow different levels of sophistication among the participants.

The European Technology Platform ARTEMIS - Advanced Research and Technology for Embedded Intelligence and Systems - was launched in mid-2004 with the overall aim to ensure that Europe realises its potential in the new markets for intelligent products, processes and services by achieving world leadership in seamlessly connected embedded systems.

Where CPS Differs from the traditional embedded systems problem:

The traditional embedded systems problem:

Embedded software is software on small computers. The technical problem is one of optimization (coping with limited resources).

The CPS problem:

Computation and networking integrated with physical processes. The technical problem is managing time and concurrency in networked computational systems.

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…

A premise in computing is the universality of the notion of “computability.” But is it really universal?

Correct execution of a program in C, C#, Java, Haskell, etc. has nothing to do with how long it takes to do anything. All our computation and networking abstractions are built on this premise.

Timing of programs is not repeatable, except at very coarse granularity.

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

A Story

In “fly by wire” aircraft, certification of the software is extremely expensive. Regrettably, it is not the software that is certified but the entire system. If a manufacturer expects to produce a plane for 50 years, it needs a 50-year stockpile of fly-by-wire components that are all made from the same mask set on the same production line. Even a slight change or “improvement” might affect timing and require the software to be re-certified.