CMPE 117: Embedded Software
Instructor: Luca de Alfaro

MWF 3:30-4:40, Earth & Marine

Lecture 1
© Luca de Alfaro, 2002

General Class Information

- **Office hours**: MW 5-6 (after class)
- **Exams**: the course will have two midterms and one final.
- **Homeworks**: weekly (see next).
- **Projects**: none, except by special request.
- **Class web page**: www.soe.ucsc.edu/classes/cmpe117/Spring02/
- **Prerequisite**: CMPS 111, Operating Systems.
Textbooks

- Other reading (papers, tutorials) will be indicated and provided on the class web page.

Homework

- **Homework:** weekly, handed out on Friday, due next Friday, collected in class.
- **Late homework policy:** late homeworks are not accepted. However...
- **Homework grading:** when deciding the class grade or evaluation for a student, I will not take into account his/her homework with the lowest grade. In practice, this means that if you do the other homeworks well, you can avoid doing one of the homeworks without any grade/evaluation consequence.
Programming Assignments & Lego

- A few of the homeworks will consist in programming assignments. The languages used (Esterel, perhaps Giotto) have a syntax that resembles C. Location/platform: CATS cluster, Sun machines.

- Still in development: use of LegOS and Lego Mindstorm robots. The obstacle seems to be student access to linux running PCs; we are trying to see if we can do it on Windows. Comments?

Open Possibilities

- Class discussions on special topics: students (in pairs?) could present for discussion topics of special interest, and lead the ensuing discussion.

  ➔Comments??

- Guest lectures: we will have some guest lectures: one from Scott Brandt on soft real time systems and scheduling, another (perhaps) on synchronous languages for embedded systems, and maybe more.
Organization...

- **Send me an email** (luca@soe) with your name: I will use to make an email list for announcements.
- I am generally not dropping anyone, so drop yourself if you need.
- Attendance is open to **anyone**.

Why embedded software?
It's Relevant

$4 billion development cost
>50% system integration and validation
Largest private industrial project (1995)

It's Everywhere

>99% of CPUs are used in embedded systems
It's Challenging

Electronics: >30% of cost, >90% of innovation

Powertrain control:
> 100 embedded software components

It's Difficult

Mars, December 3, 1999
Lander lost due to embedded software design error
$ 184 million development cost
It's Risky

Ariane 5, French Guyana, June 4, 1996
$800 million embedded software failure

What's difficult about it?

We need to integrate the controllers
The real world behaves concurrently.
Presently, concurrency is hard

Resource competition:
- CPU Time
- Resources: I/O, ...
- Locks
- Power
- ...

Each controller consists of concurrent activities

Task 1

Task 2

Task 3

Read temp
Read accel.

Compute fuel

Control injection

Control injection

Control injection

Control injection

CPU

MEM

I/O
Presently, concurrency is hard

Programming Languages Make Concurrency Difficult

**Easy:**
- Functional composition:
  \[\text{do\_this}(\text{foo});\]
- Sequential composition:
  \[\text{do\_this}(\text{foo});\]
  \[\text{do\_that}(\text{blah});\]

**Hard:**
- Concurrency
  \[\text{do\_this} \parallel \text{do\_that}\]
- Timed communication
  The value of the temperature should be communicated every 10ms.
Programming Languages Make Concurrency Difficult

Our vocabulary:
- Schedule alarms (no periodic alarms).
- Priorities
- Read the clock
- Send a message to a remote socket

What we want to say:
- Execute this control algorithm every 10ms
- Broadcast the new values with a delay of at most 200ms

There is a semantic gap!

Software/hardware have been optimized for:
- Best average case performance
- Speed independence
- Resource sharing
- Sequentiality
Caches, pipelines, interrupt driven, long context switches, complex memory management

For embedded software we need:
- Repeatable timing
- Resource accounting
- Concurrency

Paradigm shift!
Feature Interaction

- Engine
  - DOHC I-4
  - DOHC V-6
- Transmission
  - 2-wheel drive
  - 4-wheel drive
- Brakes etc.
  - shocks
  - shocks, ABS
- Access
  - Door locks
  - Doors, antitheft
Feature Interaction

- Customization, sharing of platforms, and feature diversification is crucial to profit.
- How can we add features without breaking a system that works?
- How can we certify a combination of features based on the certification of the components?

The Result: Productivity is LOW

- The component based approach does not work well.
- The majority of real time embedded software projects are not completed (source: EMSOFT 2001 panel discussion).
- Productivity gap:
  - Expectation: \(\approx 3\) months
  - Reality: > 1 year
In current programming languages: concurrency is hard!

In the design of real time embedded systems: concurrency is easy!

Designers are adopting alternative languages where concurrency, as well as simple and precise notions of time, are available (e.g. Matlab/Simulink).

[G. Berry, EMSOFT 01]

A vision for the future

• Components and compositionality
• Concurrency
• Machine independent model of real time computation
• Component interfaces
Example of real-time programmer's model

**Giotto** [Henzinger, Horowitz, Kirsch; UC Berkeley]:

- **Computation**
  - read rpm and gyros
  - compute control surface outputs

- **Communication**
  - 20ms

[UCB: Helicopter flight control software, Software Enabled Control Project]

---

**A vision for embedded software**

Start with high-level requirements, domain-specific knowledge, etc.
Implement the components in terms of high-level, concurrent programmer's models, and languages.

Decompose the design in components of manageable size, and specify the interfaces. Check that the interfaces match.
A vision for embedded software

Compile the high level model onto a physical architecture

Course Outline

- Basic organization of real time operating systems.
- Real time scheduling.
- Examples of real time operating systems.
- Models for real time controllers and systems.
- Distribution: time distribution, computation distribution.
- Real time communication protocols: CAN, TTP.
- The time triggered vs. event triggered approaches.
Course Outline (cont.)

• Synchronous languages for embedded systems: Lustre, Esterel, Giotto.
• Fault detection, fault tolerance.