1 – Introduction

Alexandre David

adavid@cs.aau.dk

Credits for the slides:
Claus Brabrand
Jeff Magee & Jeff Kramer

Course

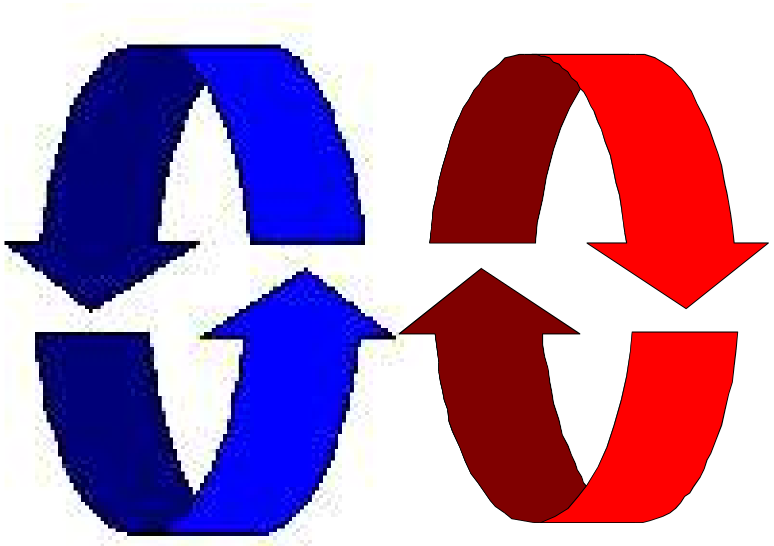
- Teachers:
  - Alexandre David *adavid@cs.aau.dk*
  - Emmanuel Fleury *fleury@cs.aau.dk*
- Page: *http://www.cs.aau.dk/~adavid/teaching/MTP-05/*
- Lectures:
  - tuesdays/fridays 8h-12h
  - lecture + exercises
  - follow the Concurrency book + additional materials

Materials

- *Concurrency – State Models and Java Programs*, by Jeff Magee and Jeff Kramer.
- Other useful books, see on the web site [3] [4] [5] in particular.
- Other materials:
  - slides
  - photocopies
  - recommended readings on the web

Concurrency

State Models and Java Programs



Jeff Magee and Jeff Kramer

adapted by Claus Brabrand

modified by Alexandre David

Why this Course?

- Story: Between 1985 and 1987, a computer controlled *therapy radiation machine*, the Therac-25, caused 6 known accidents with massive overdoses causing serious injuries and deaths. The fault came from race conditions between concurrent activities in the control program.
- Lesson: If you are going to design Therac-26, then do it right.

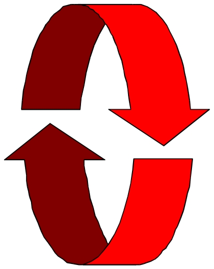
Is it Useful?

- Concurrent programming is used in a wide range of applications, most are either:
  - life critical
  - money critical
  - important for quality of life
- This course is about the principles and practices of concurrent programming.
- It is useful even if you don't design Therac-26.

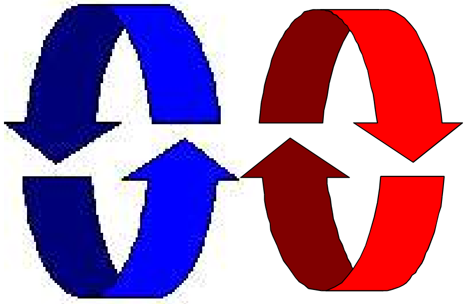
Concurrent Programs

- Example: activities involved in building a house include bricklaying, carpentry, plumbing, electrical installation, painting… Some activities may occur at the same time and have precedence constraints (no painting before bricklaying).
- It is similar for computer programs: execution of a program (or subprogram) is termed as a *process*.
- Concurrent programs are often interleaved.

What is a Concurrent Program?



- *Sequential* program: one *process*, one single *thread* of control.
  - sequential computations only



- *Concurrent* program: one or more *processes*, one or more *threads* of control *per process*.
  - multiple computations in parallel
  - control of several activities at the same time

Advantages of Concurrent Programming

- Performance gain from multiprocessing hardware
  - parallelism
  - future of computing (multi-core CPU)
- Increased application throughput
  - I/O calls block only their threads
- Increased application responsiveness
  - high priority threads for user requests
  - reactive systems

Advantages and Drawbacks!

- More appropriate program structure
  - concurrency reflected in programs
- But it is more difficult to reason about concurrent activities than sequential activities:
  - shared resources
  - mutual exclusion
  - preemption
  - precedence constraints
  - how to write and debug!!!
  - etc…

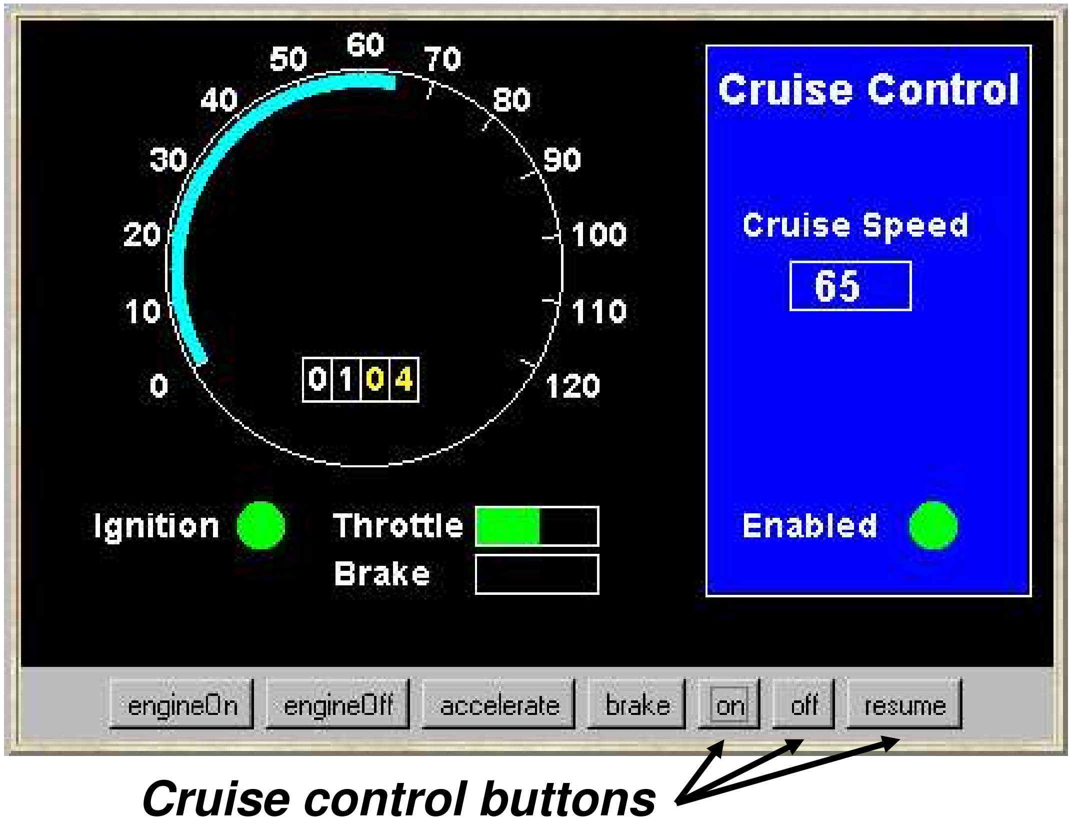
Be Careful!

- Therac-25: concurrent programming error with race conditions – caused deaths.
- Mars Rover: problems with interaction between concurrent tasks (deadlock caused by a priority inversion of tasks holding shared resources) that caused periodic software resets – not nice when it is on Mars!
- We need to be rigorous.

Cruise Control Example

- Requirements: controlled by 3 buttons (*resume*, *on*, *off*) with simple rules for the behaviour.
- How to design such a program?
- How to ensure the programs meets its specifications?
- How to define the specifications?
- How to define unsafe behaviours?

Java Applet



Cruise control buttons

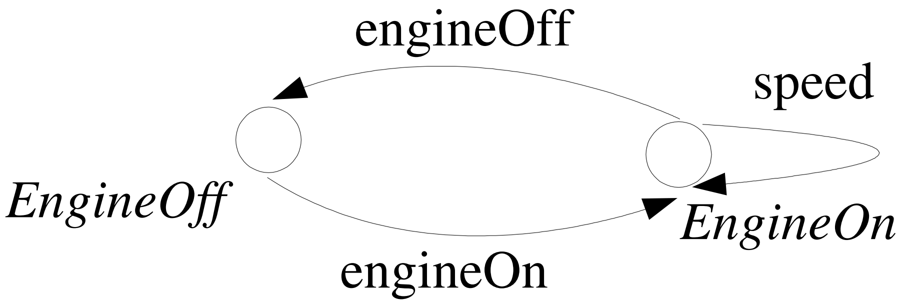
- ***Is the system safe?***
- ***Would testing be sufficient to discover all errors?***

Cruise Controller cont.

- What you would do:
  - use your own experience and design it as best as you can.
  - test it with a simulator of some kind, use a number of scenarios or *test cases*.
- Testing is difficult: how much testing do we need? Coverage problems.
- Note: concurrent events may occur in any order, difficult to (re-)produce right/wrong sequences.

Let's Make a Model!

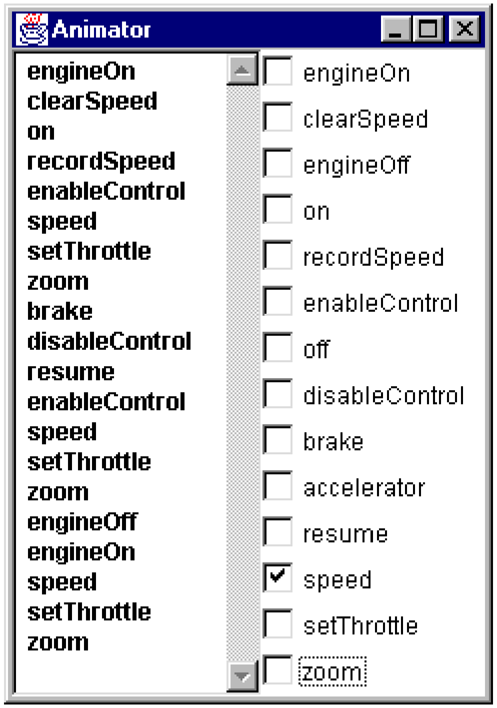
- A model is a simplified representation of the real world that *focuses on certain aspects* to analyze properties. For us: *concurrency*.
- Based on Labelled Transition Systems (LTS) .



EngineOff = engineOn->*EngineOn*
EngineOn = engineOff->*EngineOff*
| speed->*EngineOn*

LTSA

- LTSA in Java provided on the CD of the book.
- Animation of models to visualize behaviours.
- Mechanical verification of safety properties.



```
engineOn
clearSpeed
on
recordSpeed
enableControl
speed
setThrottle
zoom
brake
disableControl
resume
enableControl
speed
setThrottle
zoom
engineOff
engineOn
speed
setThrottle
zoom
```

☐ engineOn
☐ clearSpeed
☐ engineOff
☐ on
☐ recordSpeed
☐ enableControl
☐ off
☐ disableControl
☐ brake
☐ accelerator
☐ resume
☑ speed
☐ setThrottle
☐ zoom

Engineers use models to gain confidence in the adequacy and validity of a proposed design

State Machines

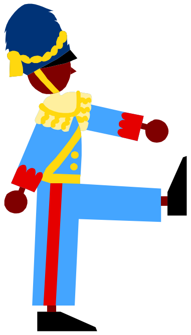
- States: indicate in which states the system is in, e.g., engine switched *on* or *off*.
- Transitions between states: when given *events* occur or *actions* are taken, the system changes state.
- The point is to *analyse the behaviour* of the system *before* it is implemented.
- Analysis done by a *model-checker*. When prooblems are found, it generates the sequence of actions that lead to the problem.

Practice

- Java used for the examples:
  - widely available, accepted, and portable
  - provides good concurrency abstractions
- Later in the course, C:
  - common on all operating systems

”Toy problems”:

crystallize concurrency programming issues and problems!



Course Objectives

This course is intended to provide a ***sound understanding of*** *the* ***concepts****,* ***models*** *and* ***practice*** *involved in designing concurrent software.*

- Concepts: thorough understanding of concurrency problems and solution techniques.
- Models: provide insight into concurrent behaviour and aid reasoning about particular designs.
- Practice: programming practice and experience.

Course Outline

- ***Processes and Threads***
- ***Concurrent Execution***
- ***Shared Objects & Interference***
- ***Monitors & Condition Synchronization***
- ***Deadlock***
- ***Safety and Liveness Properties***
- ***Model-based Design***

Concepts

Models

Practice

- *Dynamic systems*
- *Message Passing*
- *Concurrent Software Architectures*
- *Timed Systems*

Summary

- Concepts:
  Model based approach for the design and construction of concurrent programs.
- Models:
  Finite State models to represent concurrent behaviours.
- Practice:
  Java and C for constructing concurrent programs.