

# Concurrency, Parallelism and Distribution

## Concurrency Module

Jorge Castro, Joaquim Gabarro  
{castro, gabarro}@cs.upc.edu

Computer Science  
Universitat Politècnica de Catalunya

Introduction

Shared memory

Message Passing

Course organization

# Introduction

# Lecturers



Figure: Joaquim Gabarro



Jorge Castro

# Concurrency

**Purpose:** Concurrency Foundations, mainly through examples.

Concurrency is about interacting programs (or processes).

- ▶ **Modelling concurrency:** LTS (Labeled Transition Systems)

There are **two main** paradigms:

- ▶ **Shared Memory:** **Java**. It is object oriented and imperative. Class Thread
- ▶ **Message Passing:** **Erlang**. It is functional. Each process has a mailbox

# Paradigms

In concurrent programs, process interact between the.

- ▶ **Shared Memory**: Process interact accessing **shared objects**. Intuitively process interact reading and writing to common variables (shared variables). Different processes can access the same memory position. There is no other form of interaction.
- ▶ **Message Passing**: Process interact **sending (and receiving) messages**. There are no shared variables between the processes. Like in your email, any process has a MailBox.

# Basic material

- ▶ **Models**

Jeff Magee and Jeff Cramer

Concurrency, State Models & Java Programs

John Wiley & Sons, 2006.

- ▶ **Java**

Part of the slides are strongly based in:

<https://docs.oracle.com/javase/tutorial/>

- ▶ **Erlang**

Joe Armstrong

Programming Erlang, Software for a Concurrent World

Pragmatic Bookshelf, 2007 (first edition).

# Shared memory



# Modelling Processes

- ▶ A **process** is the **execution of a sequential program**.
- ▶ As a process executes, it **transforms** its **states** by executing statements.
- ▶ Each statement consists of a sequence of one or more atomic **actions**.

You have to download from

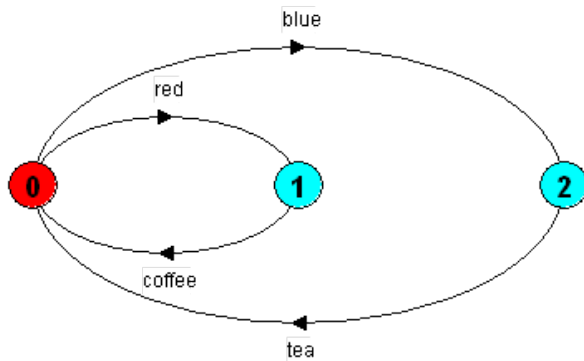
<http://www.doc.ic.ac.uk/~jnm/book/> and install the **Labelled Transition System Analyzer, LTS**

# FSP

- ▶ We introduce a **simple algebraic notation** called **FSP** (for Finite State Process)
- ▶ Every FSP description has a corresponding Labelled Transition System.

## Example: Drinking machine

```
DRINKS = (red->coffee->DRINKS  
          |blue->tea->DRINKS  
          ) .
```



## Example: Drinking machine, traces

A process may have **many possible traces**

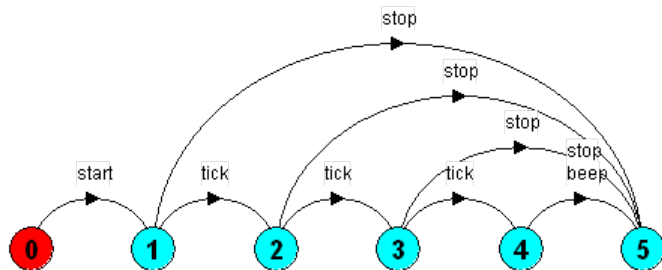
red->coffee-> blue->tea-> blue->tea->...

blue->tea-> red->coffee-> blue->tea->...

Look at the *Animator* part of the LST.

## Example: Countdown Timer

A countdown timer which beeps after N ticks, or can be stopped.



```
COUNTDOWN (N=3) = (start->COUNTDOWN[N]),  
COUNTDOWN[i:0..N] =  
  (when (i>0) tick->COUNTDOWN[i-1]  
   | when (i==0) beep->STOP  
   | stop->STOP) .
```

# Java: Classes and Objects

A class is a **blueprint or prototype** from which objects are created.

- ▶ A class has **atributes** (usually private) and **methods** (usually public).
- ▶ A class has at least a **constructor**. A constructor has the name of the class.
- ▶ The **toString** methods returns a string. Usually prints an object of the class.

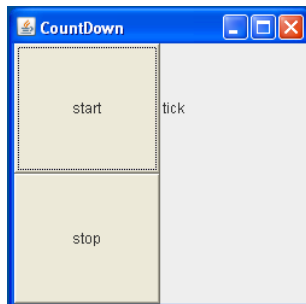
# Concurrency: Threads

- ▶ A **Thread** class manages a **single sequential** thread of control.
- ▶ Threads may be **created** and **deleted** dynamically.
- ▶ An application that creates an instance of Thread must **provide the code** that will run in that thread.

`http://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html`

# LTS to Java: Countdown

Actions become methods (more or less)



There are two `Buttons` to start and stop the process. Action `tick` and `beep` are displayed as a `Label`.



# Modelling Concurrency

- ▶ How should we **model process** execution speed?
  - ▶ Arbitrary speed.  
We **abstract** away time.
- ▶ How do we **model concurrency**?
  - ▶ Arbitrary relative order of actions from different processes.  
**Interleaving** but preservation of each process order.
- ▶ What is the **result**?
  - ▶ Provides a general model **independent** of scheduling.  
**Asynchronous** model of execution.

## Example: Parallel composition; Action interleaving

$(P||Q)$  represents the concurrent execution of  $P$  and  $Q$ . The operator  $||$  is the parallel composition operator.

```
ITCH = (scratch->STOP).  
CONVERSE = (think->talk->STOP).  
||CONVERSE_ITCH = (ITCH || CONVERSE).
```

Possible traces as a result of action interleaving.

```
think->talk->scratch  
think->scratch->talk  
scratch->think->talk
```

# Message Passing

# Reasons why you should learn Erlang

- ▶ You want to write programs that **run faster when you run them on a multicore computer**.
- ▶ You want to write **fault-tolerant** applications that can be modified without taking them out of service.
- ▶ You've heard about **functional programming** and you're wondering whether the techniques really work.
- ▶ You don't **want to wear your fingers out** by typing lots of lines of code.

# Variables

All variable names **must start with an uppercase letter**.

1> X = 123456789.

123456789

2> X.

123456789

3> X\*X\*X\*X.

232305722798259244150093798251441

Erlang variables can only be bound —to values— **once**.

# Pattern matching

1> Rectangle = {rectangle, 10, 5}.

{rectangle, 10, 5}.

2> Circle = {circle, 2.4}.

{circle, 2.40000}

3> {rectangle, Width, Ht} = Rectangle.

{rectangle, 10, 5}

4> Width.

10

5> Ht.

5

6> {circle, R} = Circle.

{circle, 2.40000}

7> R.

2.40000

# The Concurrency Primitives

- ▶ Creation of a process: `Pid = spawn(Fun)`
- ▶ Sending a Message: `Pid ! Message`
- ▶ Receiving a message: `receive ... end`

## area as process: area\_server0.erl

Download area\_server0.erl

```
-module(area_server0).
```

```
-export([loop/0]).
```

```
loop() ->
```

```
    receive
```

```
        {rectangle, Width, Ht} ->
```

```
            io:format("Area of rectangle is ~p~n", [Width * Ht]),
```

```
            loop();
```

```
        {circle, R} ->
```

```
            io:format("Area of circle is ~p~n", [3.14159 * R * R]),
```

```
            loop();
```

```
        Other ->
```

```
            io:format("I don't know the area of a ~p is ~n",[Other]),
```

```
            loop()
```

```
    end.
```



# Evaluating loop/0 in the shell

```
1> Pid = spawn(fun area_server0:loop/0).
```

```
<0.36.0>
```

```
2> Pid ! {rectangle, 6, 10}.
```

```
Area of rectangle is 60
```

```
{rectangle,6,10}
```

```
3> Pid ! {circle, 23}.
```

```
Area of circle is 1661.90
```

```
{circle,23}
```

```
4> Pid ! {triangle,2,4,5}.
```

```
I don't know the area of a {triangle,2,4,5} is
```

```
{triangle,2,4,5}
```

# Course organization

# Prerequisites

Some basic knowledge on:

- ▶ Object oriented programming: classes and objects.
- ▶ Finite state machines.
- ▶ Recursive programming.

**Question:** If I do not have some of the prerequisites could I take this course?

**Answer:** Yes, but you will need to work more. It will be a way to know something on those topics!

Any case we are there to **help you!**

# Scheduling: Theory

## Five lectures

- ▶ Shared Memory:
  - ▶ **Models and Problems**: Modelling concurrency: FSP and LTS. Interference, lack of coordination and deadlock.
  - ▶ **Problems and Solutions**: Monitors. Wait and notify. Partial solution to deadlock.
  - ▶ **Correctness**: Lamport approach: safety and progress.
- ▶ Message Passing:
  - ▶ **Concurrency on Erlang**: Spawning processes. Remote procedure call. Servers.
  - ▶ **Parallelism and Dynamic Coding**: Parallel quicksort. Dynamic code updating.

# Scheduler: Lab

- ▶ Shared Memory:
  - ▶ LTS, LTS Analyser and FSP
  - ▶ Models and Java Programs
  - ▶ Safety and Progress
- ▶ Message Passing:
  - ▶ Basics on Erlang
  - ▶ Example of Process Interaction Architecture

# Do not forget

If any problem or comment or..., do not wait, send and email!  
gabarro@cs.upc.edu, castro@cs.upc.edu

THANKS!