Lecture 3

Part 1: Modelling: where to start?

Part 2: Spin tutorial

Natalia Sidorova

Distributed systems

- A distributed system consists of a collection of distinct processes which are specially separated, and which communicate with one another by exchanging messages. . .
- A system is distributed if the message transmission delay is not negligible compared the time between events in a single process
- A distributed system is the one which prevents you from working because of the failure of a machine that you had never heard of.

Leslie Lamport

Models in system development

- Requirements Model: captures functional requirements from user perspective
- Analysis Model: maintainable with logical structure; implementation-independent
- Design Model: impose implementation constraints on analysis model
- Implementation Model: system code written from the design model
- Test Model: documentation and test results

Where are we now?

- We have specified our requirements and want to start with modelling the system.
- Still, we don’t have a global view on the system.
- Traditional thinking maintains requirements describe the "what" is required, whereas subsequent development steps translate from the "what" to the "how".
- To get a better understanding of the system, we consider use cases.
- We DO NOT CONSIDER the use cases of UML here.

Use cases

- The level of use cases is in between the level of requirements and the level of model.
- Use cases constitute the complete course of events initiated by the environment, define interaction between the environment and the system.
- Use cases describe specific scenarios for the system, illustrating one or more key characteristics of its functionality and processes.
- You should describe use cases so that your client can understand and validate them!

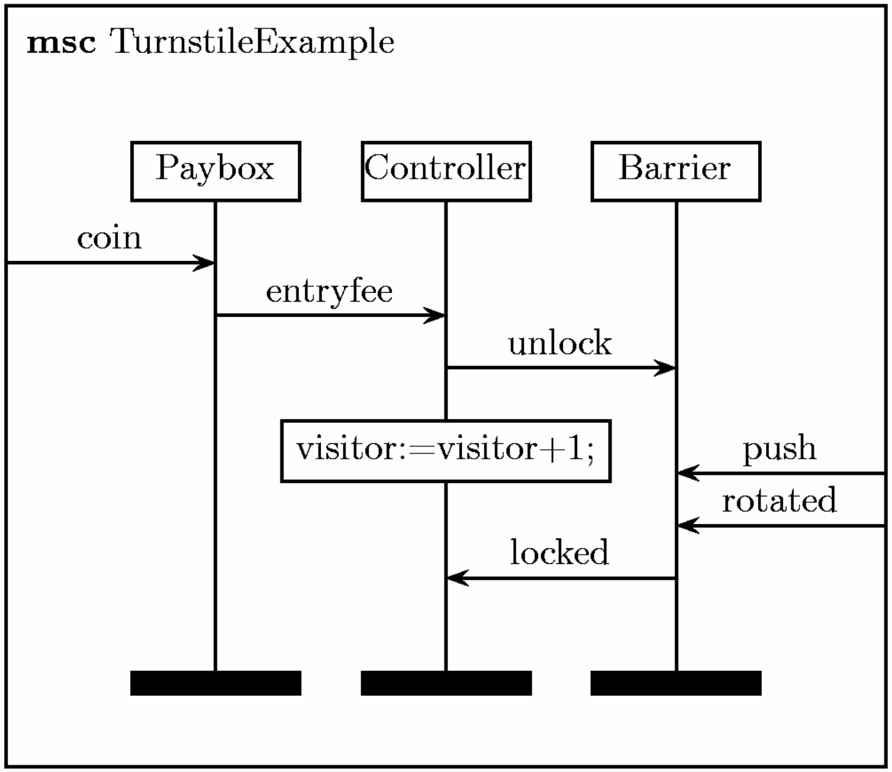
Writing use cases

- Consider some situation,
- Identify main tasks,
- Identify parties participating in the use case, (up to this point you may model it as UML use case)
- Draw allowed and explicitly forbidden scenarios (event sequences) for each use case as a Sequence Diagram of UML or as a Message Sequence Chart (MSC),
- Formulate complex use cases at an abstract level first and then refine them.
- Create Workflow nets for the use cases. Each Workflow net should combine the allowed scenarios for the use case and disallow the forbidden ones.

MSC: Message Sequence Charts

. . . are precedence graphs with locality information.

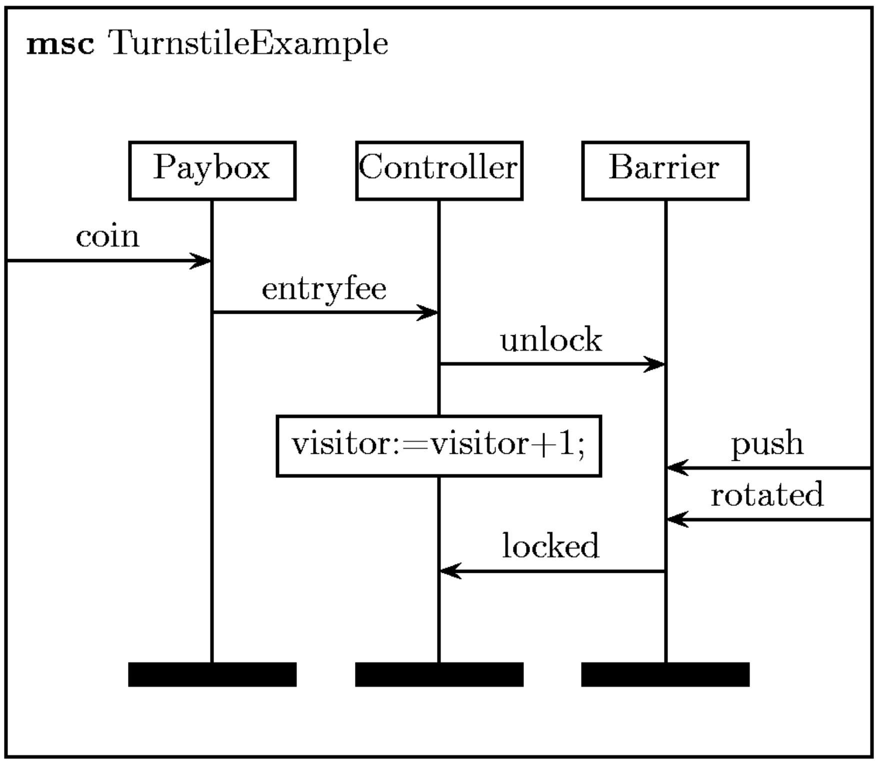
Each vertical line represents a process (or the environment), the arrows represent signals/messages, the blocks represent (internal) process activities.



MSC: Message Sequence Charts

. . . define sets of traces (with locality information).

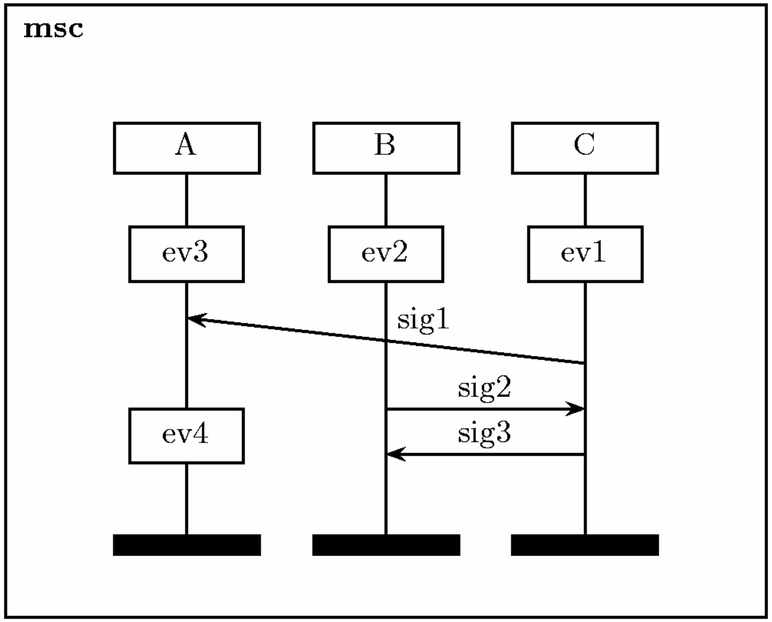
An MSC represents (almost always) more than one trace.




```
coin/ @Environment
/coin @Paybox
entryfee/ @Paybox
/entryfee @Controller
unlock/ @Controller
/unlock @Barrier
visitor:= @Controller
push/ @Environment
/push @Environment
rotated/ @Environment
/rotated @Barrier
locked/ @Barrier
/locked @Controller
```

MSC: Message Sequence Charts

Can you distill some traces?



Is this a trace?
ev1@C
sig1/@C
ev3@A
ev2@B
sig2/@B
/sig2@C
sig3/@C
/sig1@A
/sig3@B
ev4@A

And how many traces are there?

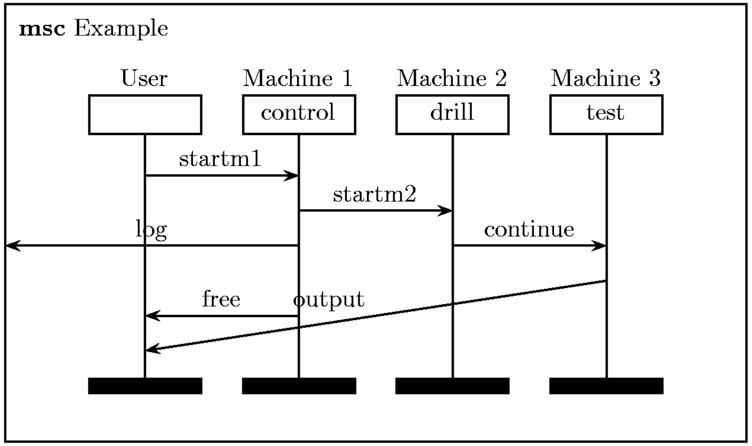
Definition: Basic MSC

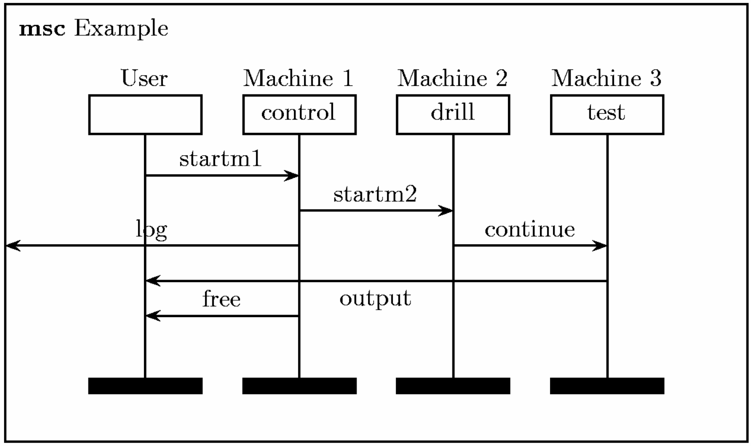
A (basic) MSC M is a tuple $(P, E, L, c, <)$ with

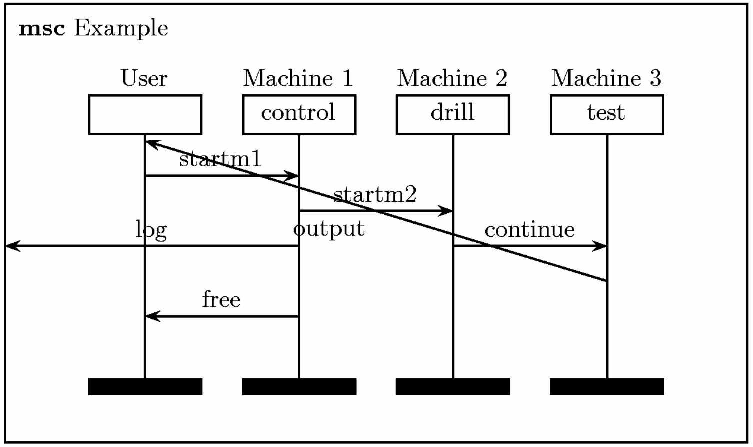
- a set P of process labels (labelling the instance axis),
- a finite set E events $E = S \cup R \cup A$, consisting of
  - send events S (buh/)
  - receive events R (/buh)
  - action events A (task executions etc)
- a labeling function $L : E \to P$ (events on axis),
- a bijection $c : S \to R$ (for send-receive edges)
- precedence relation $< \subseteq E \times E$
  - Sending of a message occurs before its receipt
  - Events on the same instance are totally ordered

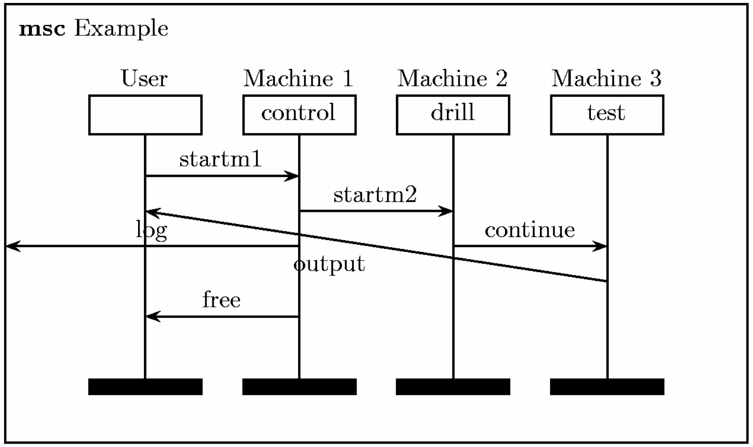
Must be well-formed: no cycles in precedence graph

MSC: Message Sequence Charts



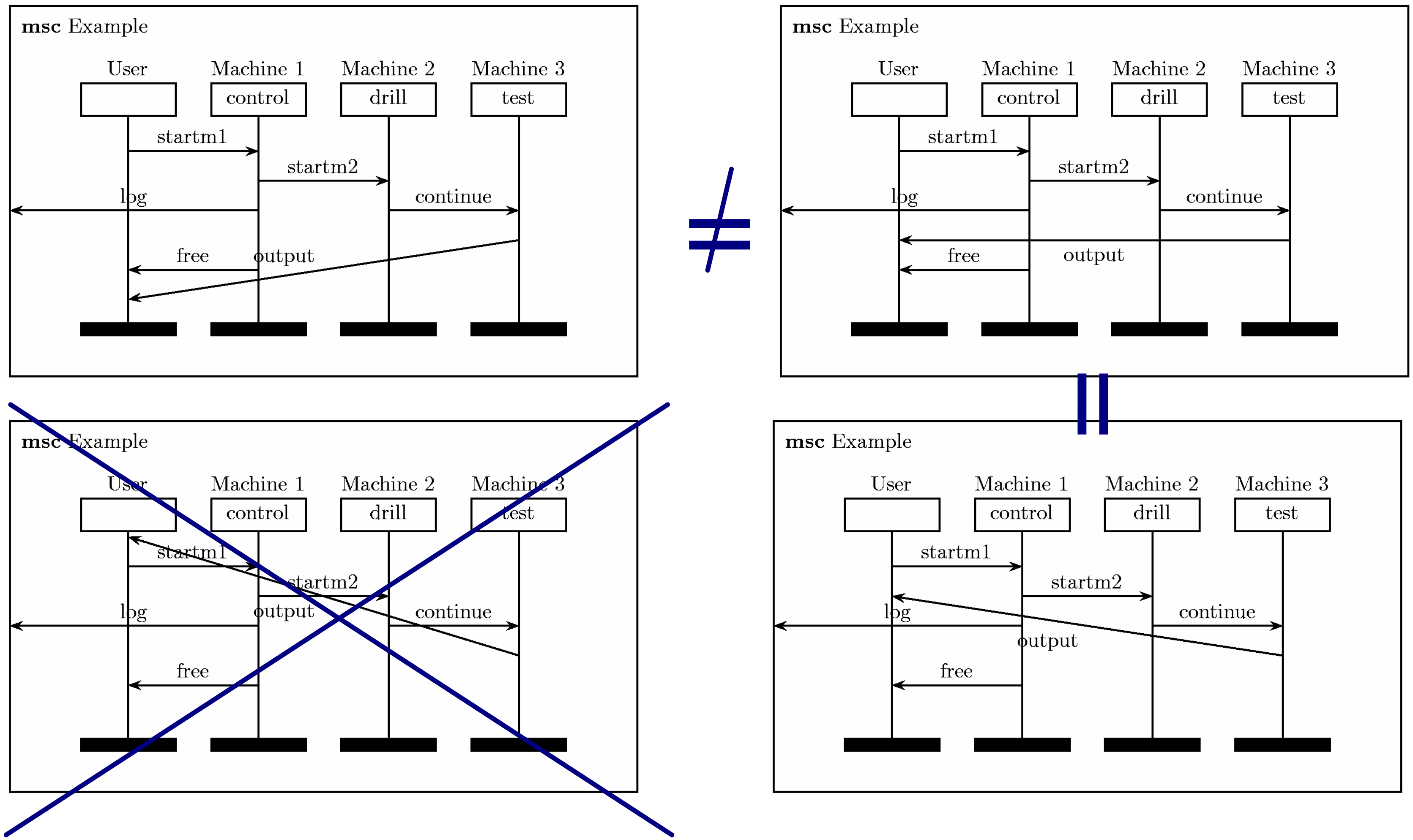






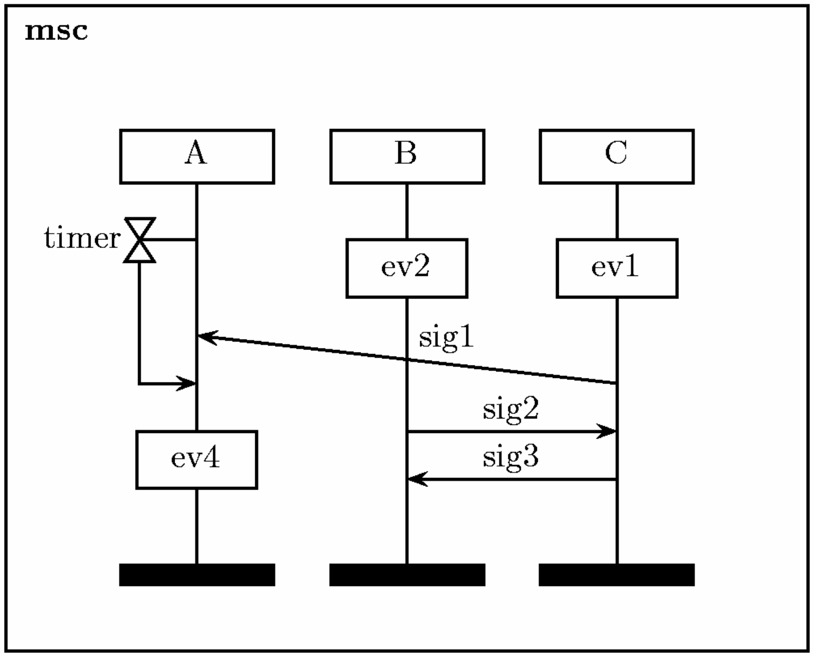
Traces?

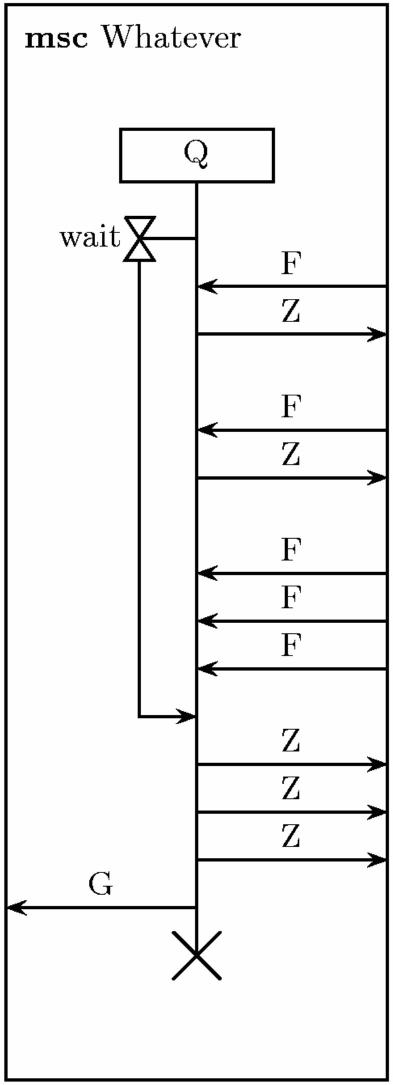
MSC: Message Sequence Charts



Traces?

MSC: timers


msc
A
B
C
timer
ev2
ev1
sig1
sig2
sig3
ev4


msc Whatever
Q
wait
F
Z
F
Z
F
F
F
Z
Z
Z
G

Types of events (1)

Start modelling processes with identifying events in and around your system:

- External Events: events that occur outside the system, usually initiated by the environment.
  Naming events: Include the name of the external agent in the name
  E.g. events: “Customer places order”, “Management checks order status”, “Customer reports change of address”
- External events usually have their counterparts (responses) within the system.

Types of events (2)

- Temporal Events: events that occur as a result of reaching a point in time.
  E.g. payroll systems produce a paycheck every two weeks (or once a month),
  reports to management are generated regularly.
  System automatically produces reports etc. at right time (so no external agent needed)

Types of events (3)

- State Events: events that occur when something happens inside the system that triggers the need for processing.
  E.g. the sale of a product results in an adjustment in the inventory (event “Reorder point reached”)
  This state change might occur as a result of external events or of temporal events (so could be similar to temporal event, but point in time can’t be defined)

Technology-Dependent Events

- System controls: checks or safety procedures put in place to protect the integrity of the system
  Logging on to a system (for security reasons)
  Controls for keeping integrity of a database
- To help decide which events apply to controls we assume that technology is perfect (never the case!)
- During analysis we should focus on events that are required under "perfect" conditions – "perfect technology assumption"
- It is during the design phase that we deal with other issues and events from a "non-perfect world" point of view, e.g. events like "Time to back up the database".

Describing events

Event Table: A table that lists events in rows and key pieces of information about each event in columns

- The trigger: an occurrence that tells the system that an event occurs (the arrival of data needing processing or of a point in time)
- The source: an external agent or actor that supplies data to the system
- The activity: behavior that the system performs when an event occurs
- The response: an output, produced by the system, that goes to a destination
- The destination: An external agent or actor that receives data from the system

Components/objects

- By analysing use cases, you can identify components/objects of your system,
- Think who and when creates/destroys these components, whether it is done statically or dynamically,
- List communication partners of each component,
- Define interfaces of each component and check their consistency with interfaces of other components,
- List main tasks of each component,
- Start modelling components.

Modelling dynamic process creation

- In Spin (and many others): directly,
  run *process name*
  you just create a new process instance according to a given specification,
- In Petri nets: define a process pattern for every type of processes; by sending tokens to initial places, you can start up a new process.

Interprocess communication

Concurrent processes: collection of two or more sequential processes in operation executing concurrently.

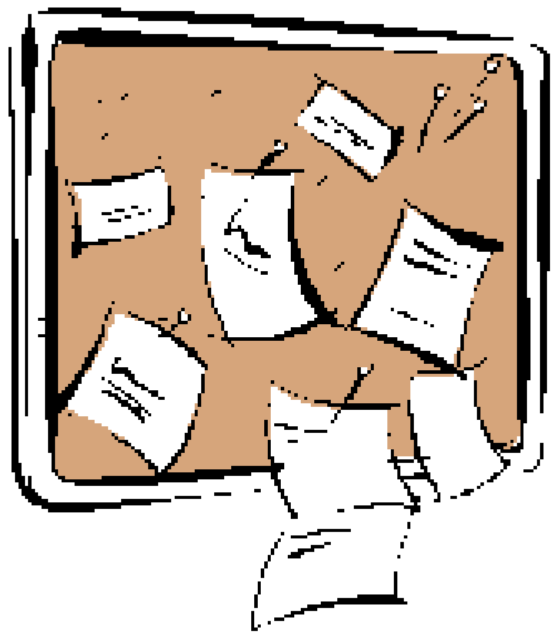
Interprocess communication: the transfer of information between processes.

Note that the mere occurrence of communication can be informative (synchronisation).

Communication via:

- Shared variables
- Message passing

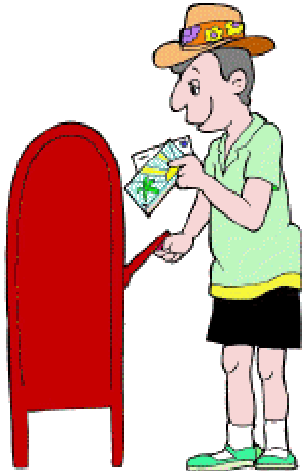
Shared variables



- An attempt to model distributed systems at the most primitive level.
- Time-independent: a process cannot refer to the amount of time it has been waiting for a variable to change.
- Naturally asynchronous.
- Not appropriate for modelling protocols.
- Failure-free communication between processes, while in the real world messages can become garbled or lost.

Types of message passing

- Asynchronous:



- Synchronous:

Types of interprocess connection (1)

- Names: single point for receiving messages per process

- Entries: several reception points per process

Types of interprocess connection (2)

- Ports: target messages not necessarily addressed to a single process

- Broadcasting: distributes a message to many “appropriate” receivers

To know more

1.6, 1.7 in Berard et al. “Systems and Software Verification”

What do you have by now?

- Requirements,
- Use cases
- You have identified
  - main events,
  - main components involved,
  - their interfaces,
  - the way you model the communication between components.

Homework for this week

Read chapter 5 (Timed automata), pp. 59-68, from B. Berard et al., Systems and Software Verification. Model Checking Techniques and Tools.

Carry out Part 1 of Assignment 3

Next week:

Timed Systems