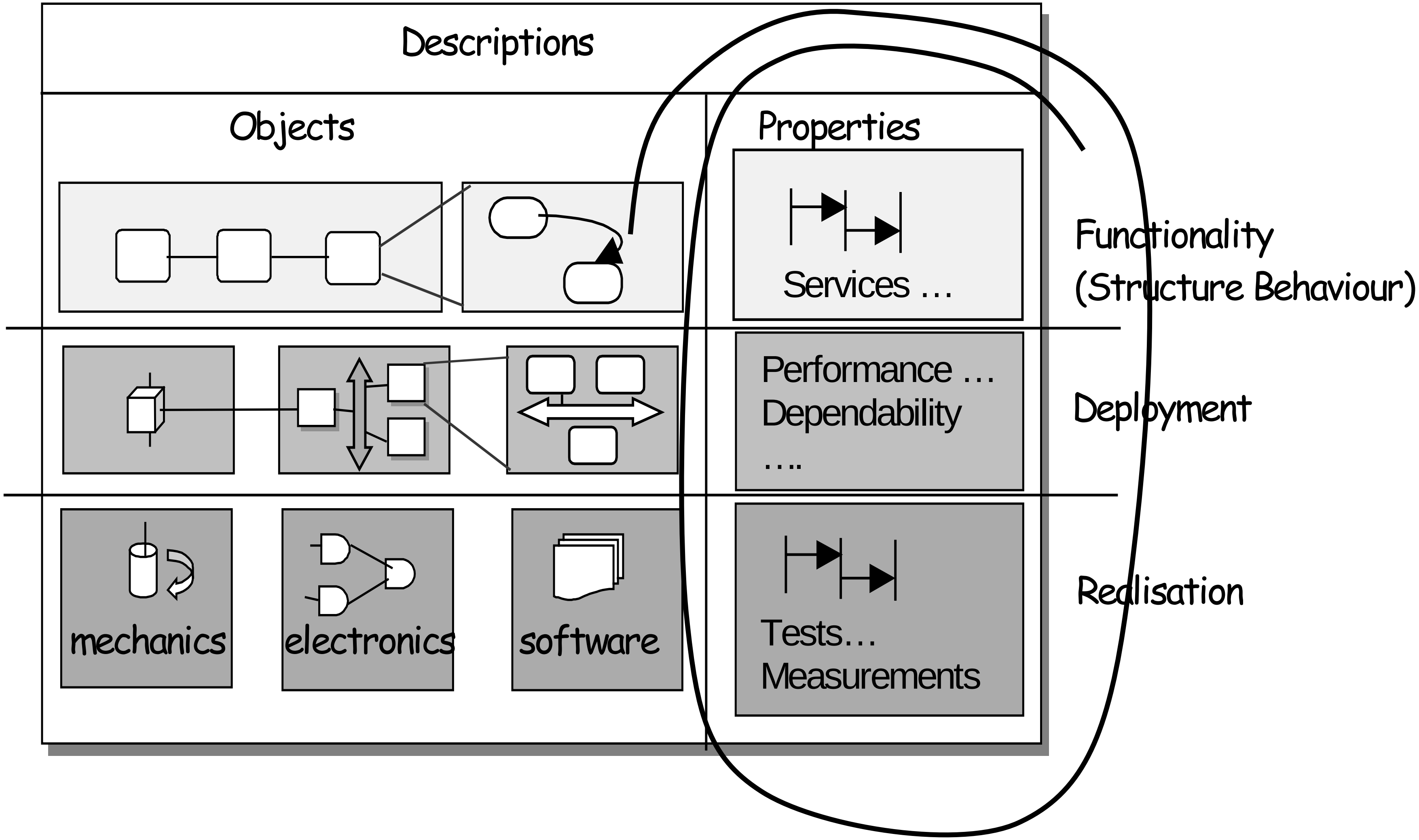
Simple MSC

Introduction to Message Sequence Charts

MSC for interaction behaviour properties


Descriptions

Objects

Properties

Services …

Performance …
Dependability
….

Tests…
Measurements

mechanics

electronics

software

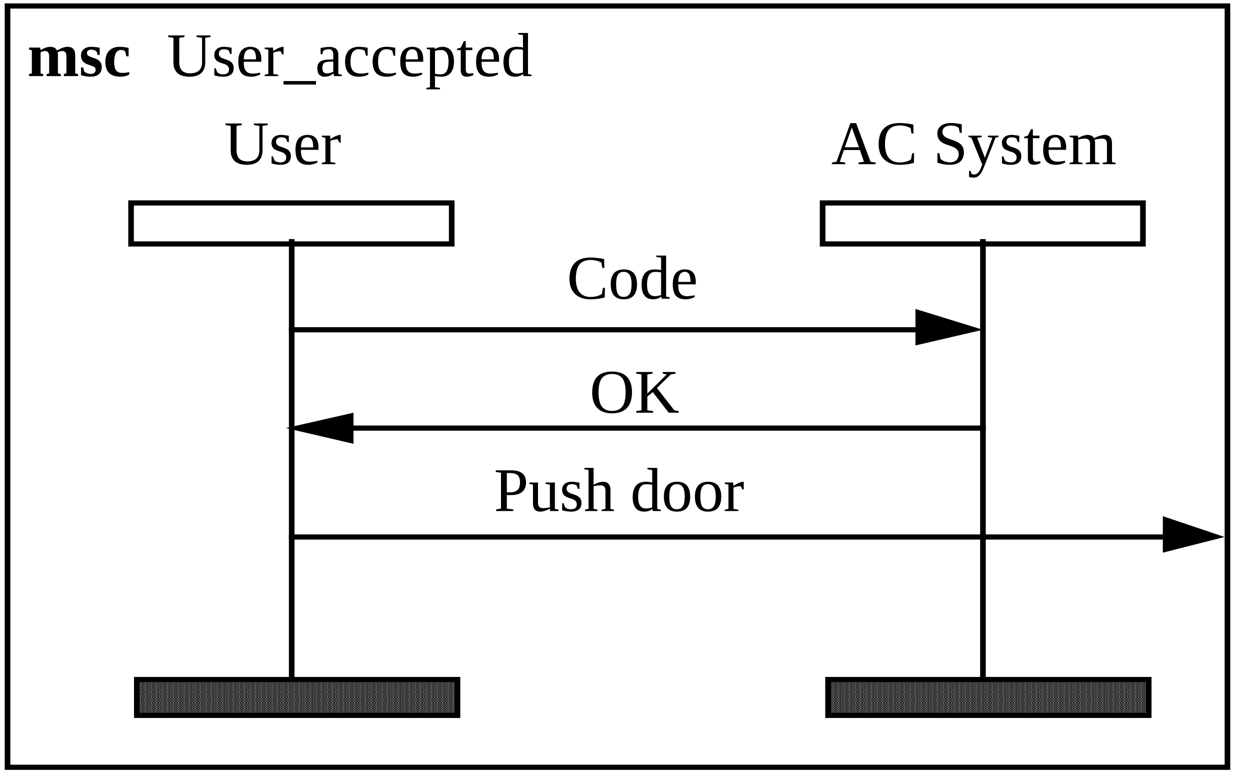
Functionality
(Structure Behaviour)

Deployment

Realisation

Basic MSC

- •Emphasizes interaction between instances (objects, actors)
- •Describes cases/traces of behaviour and (normally) not the total behavior



Look for MSC info at http://www.sdl-forum.org/

Introduction

Informal use:

- Long history in telecom and electrical engineering
- Dialects: Sequence diagrams (UML), Relay diagrams, message sequence diagrams…

Standardized:

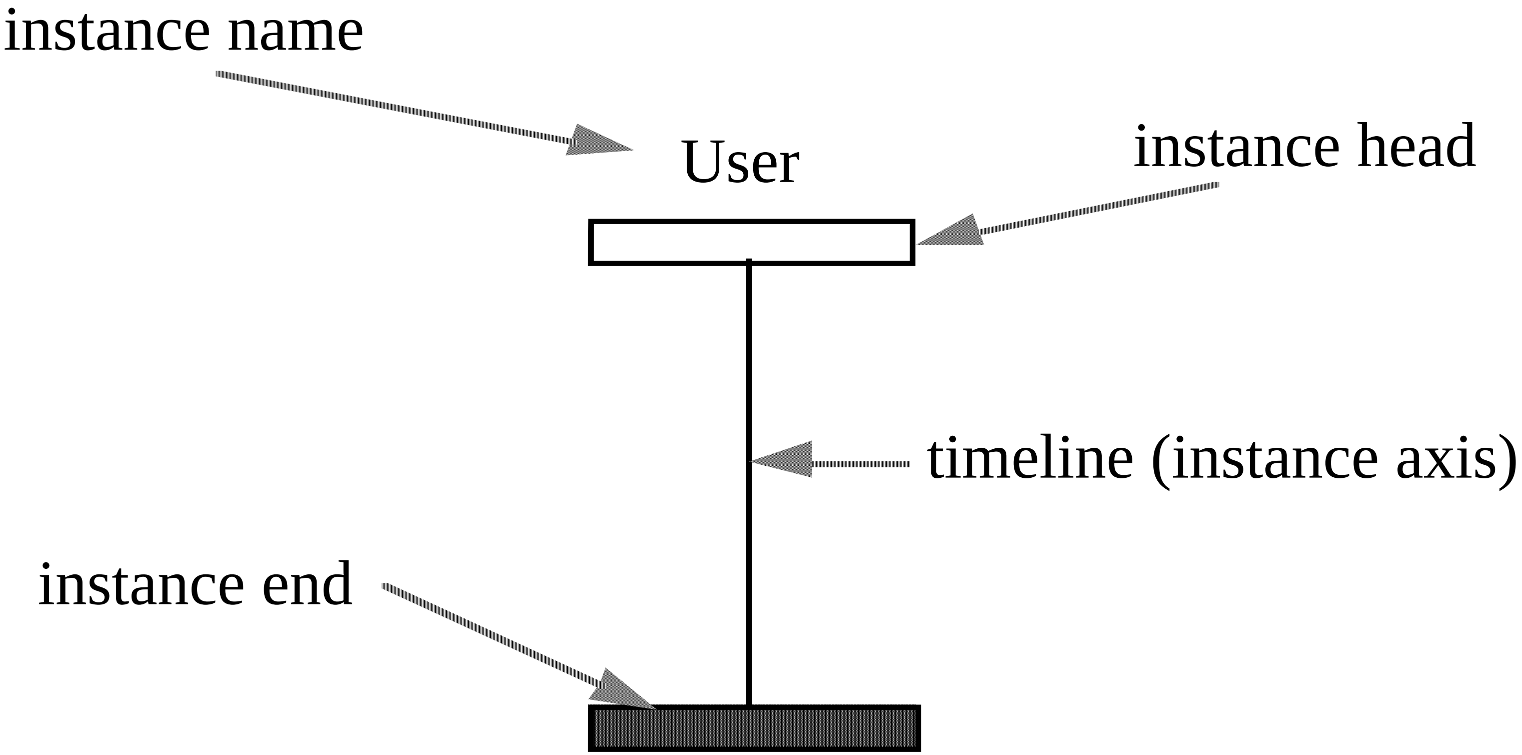
- MSC standardized by ITU in 1992 as Z-120 (“MSC-92”)
- Major revision and extension in 1996 and 2000 (“MSC-96”, “MSC-2000”)
- MSC has a formal semantics

Tools:

- MSC-tools as part of SDL-tools
- Stand-alone MSC-tools

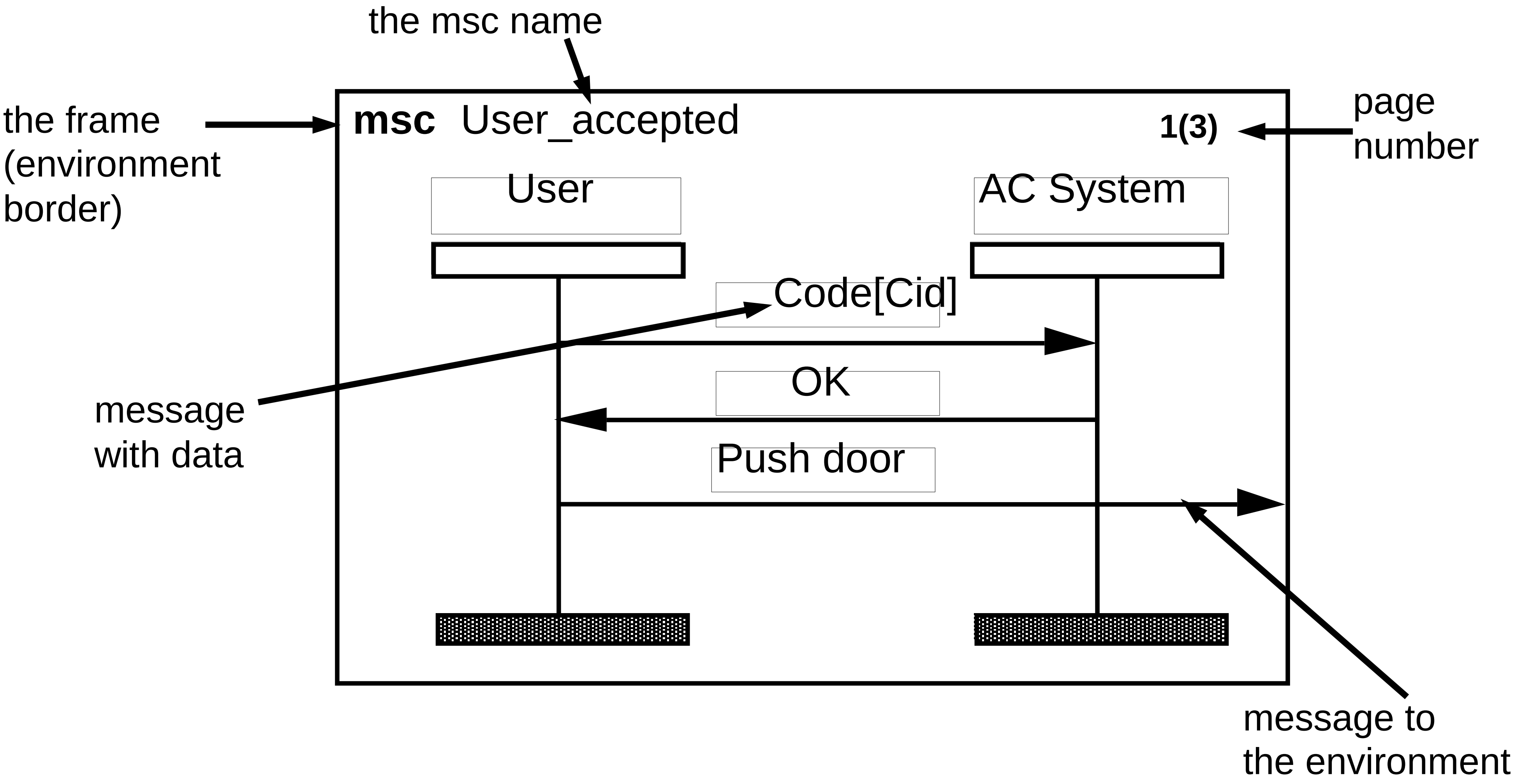
Instance

Instances are the actors of an MSC system



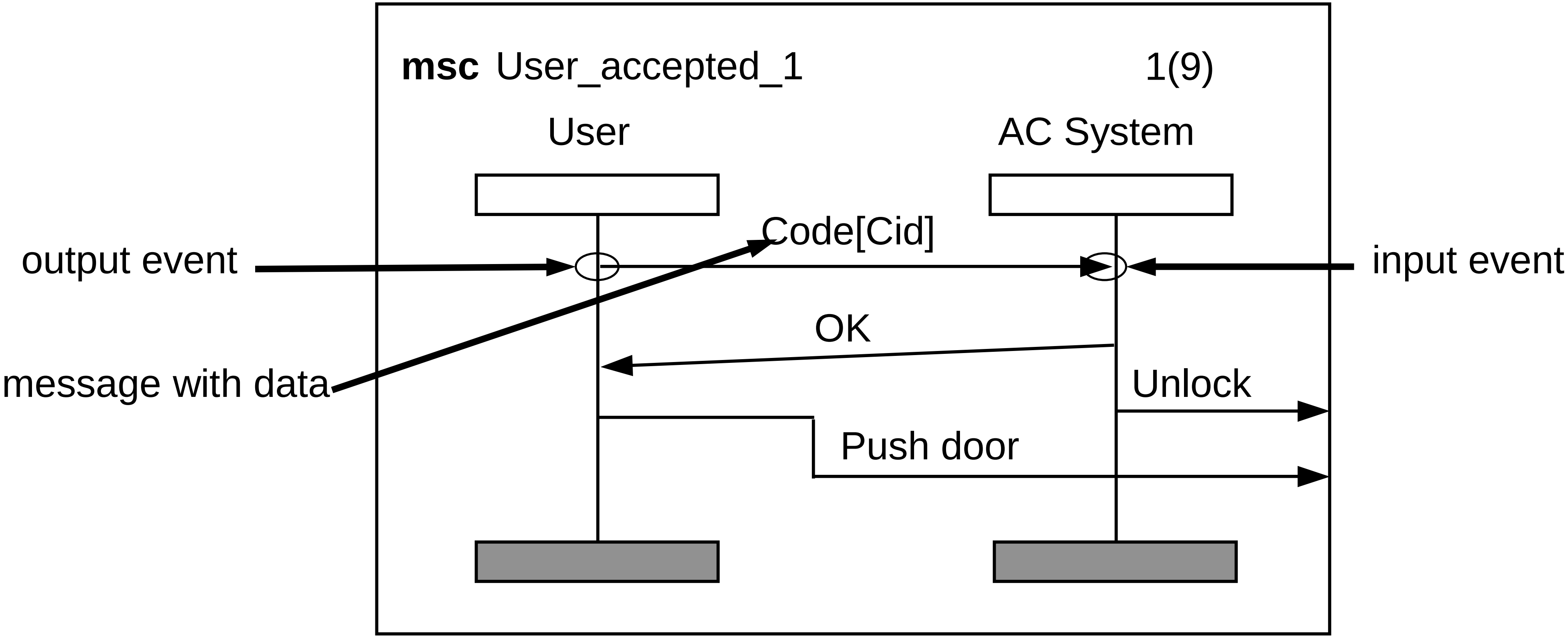
MSC diagram

- Message lines may be horizontal or with downward slope, and bended



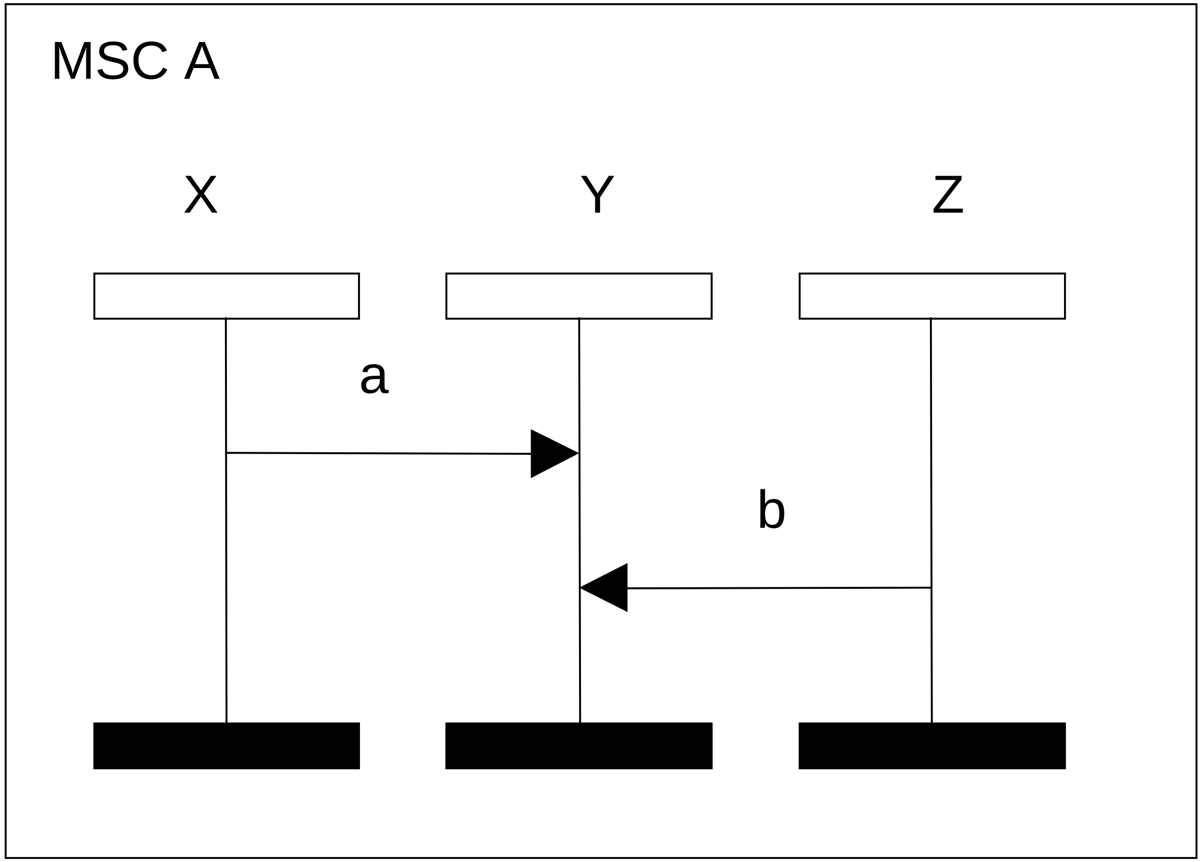
MSC semantics

- Messages have one output event, and one input event
- Input is normally interpreted as consumption of the message
- The output event must occur before the input event
- Events are strictly ordered along an instance’s timeline



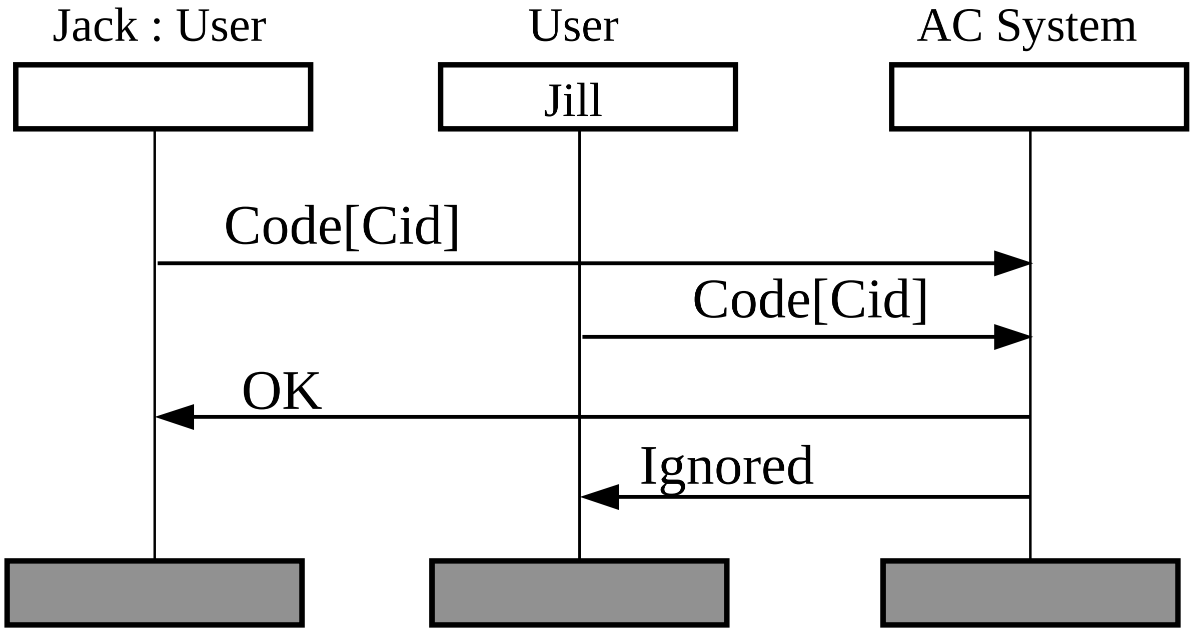
MSC semantics

- What event sequences are possible here?



Alternative instance naming

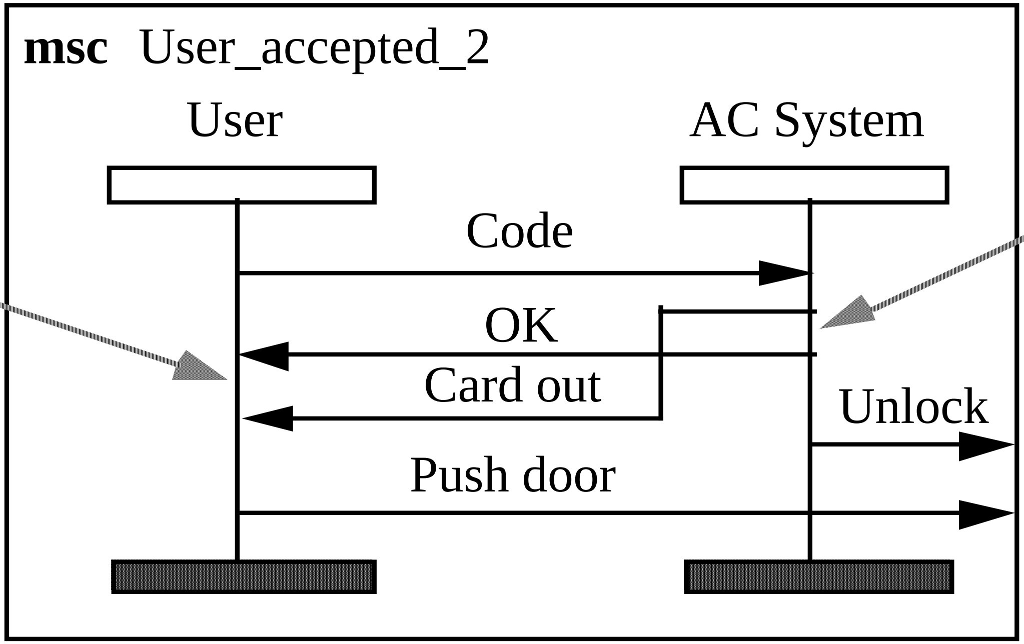
- Can also show *type* of instance.
- Instance name (excluding type) must be unique



Message Overtaking

- MSC describes asynchronous communication.
- When messages are asynchronous, it is important to be able to describe message overtaking i.e. that one message may be consumed before another event though the latter was output before the first one.

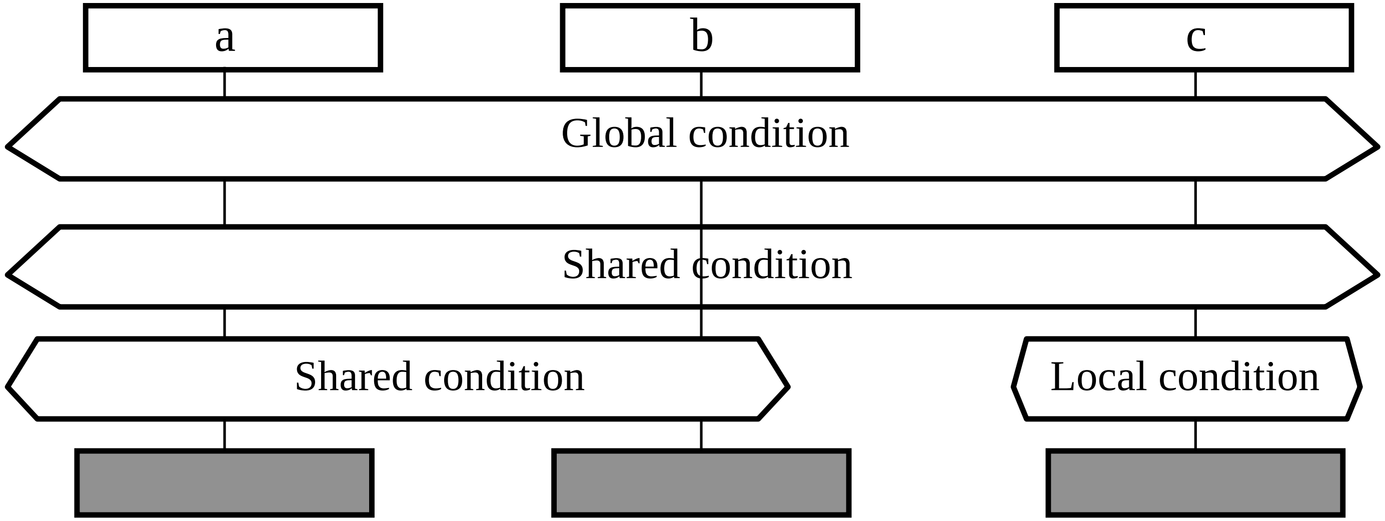
Input of *OK* comes before input of *Card out*



Output of *Card out* comes before output of *OK*

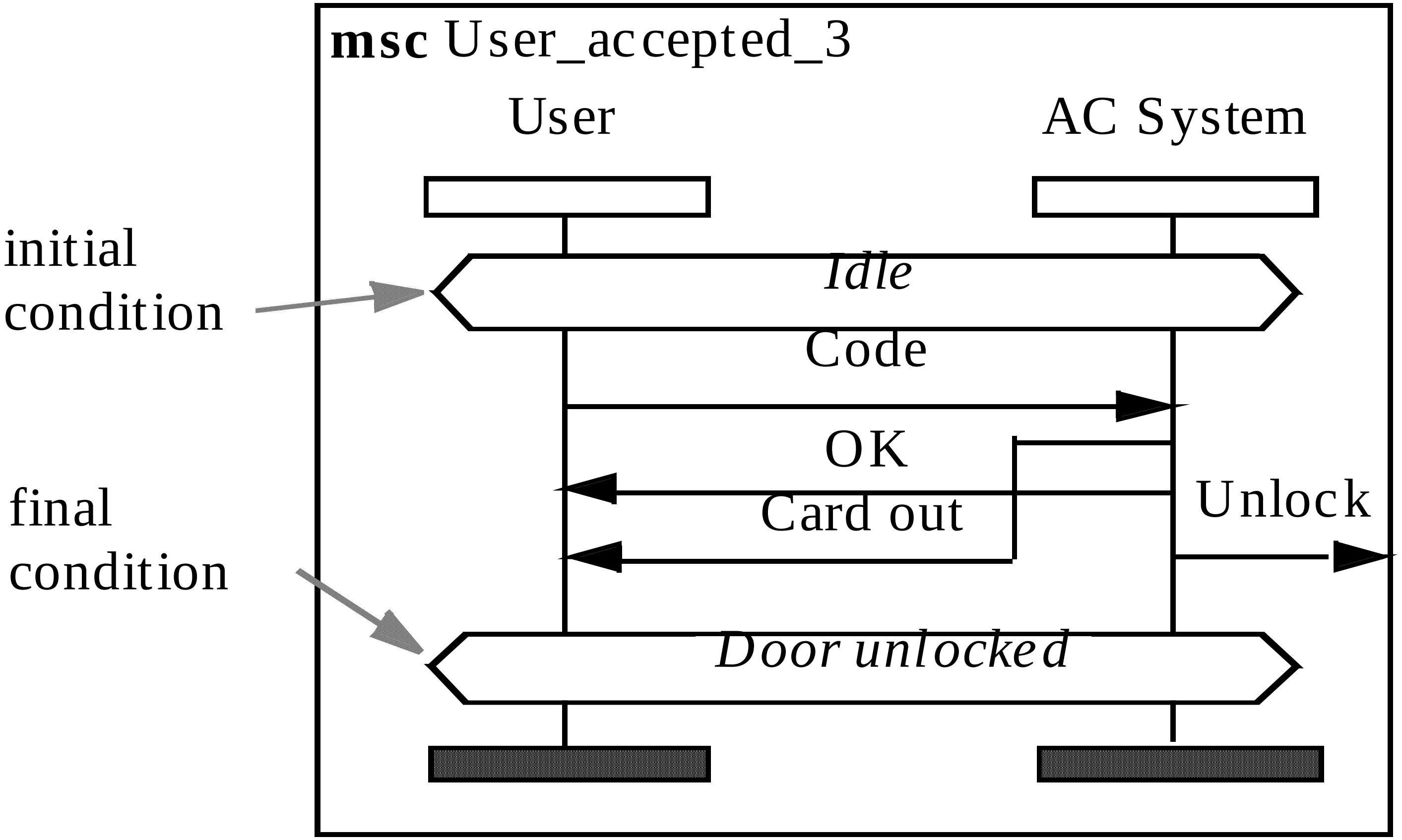
Condition

- A condition describes either a global system state (global condition) referring to all instances contained in the MSC, or a state referring to a subset of instances (shared condition). In the second case the condition may be local, i.e. attached to just one instance.
- The term “condition” is inspired by the Hoare logic (Hoare 1969), but there is no predicate logic behind the MSC term. The MSC condition is merely a label.



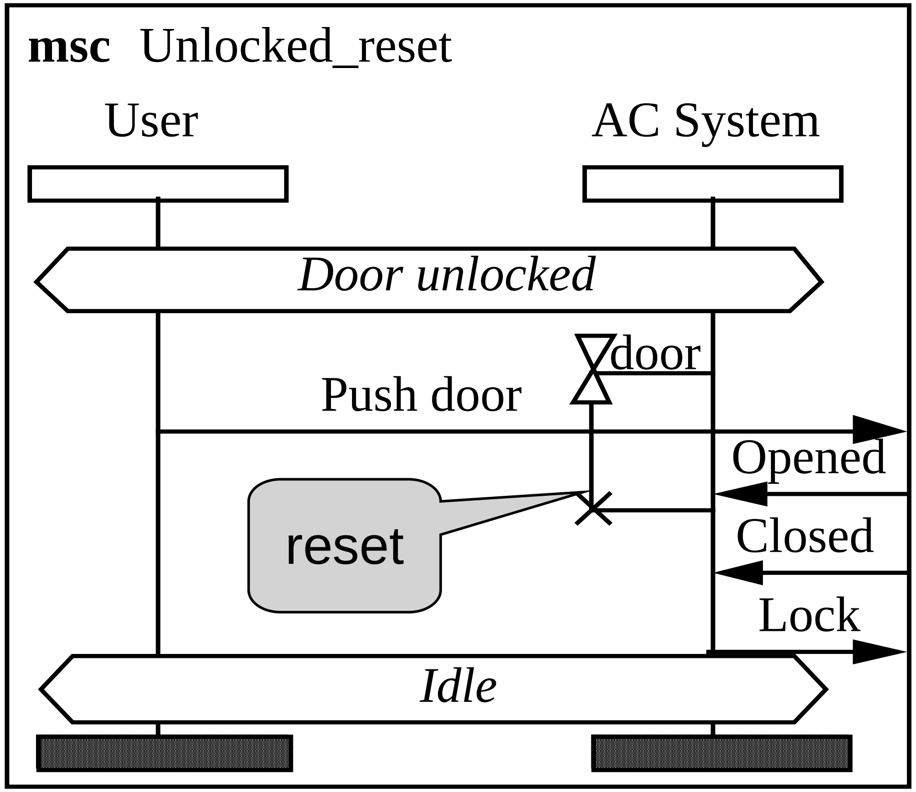
Condition example

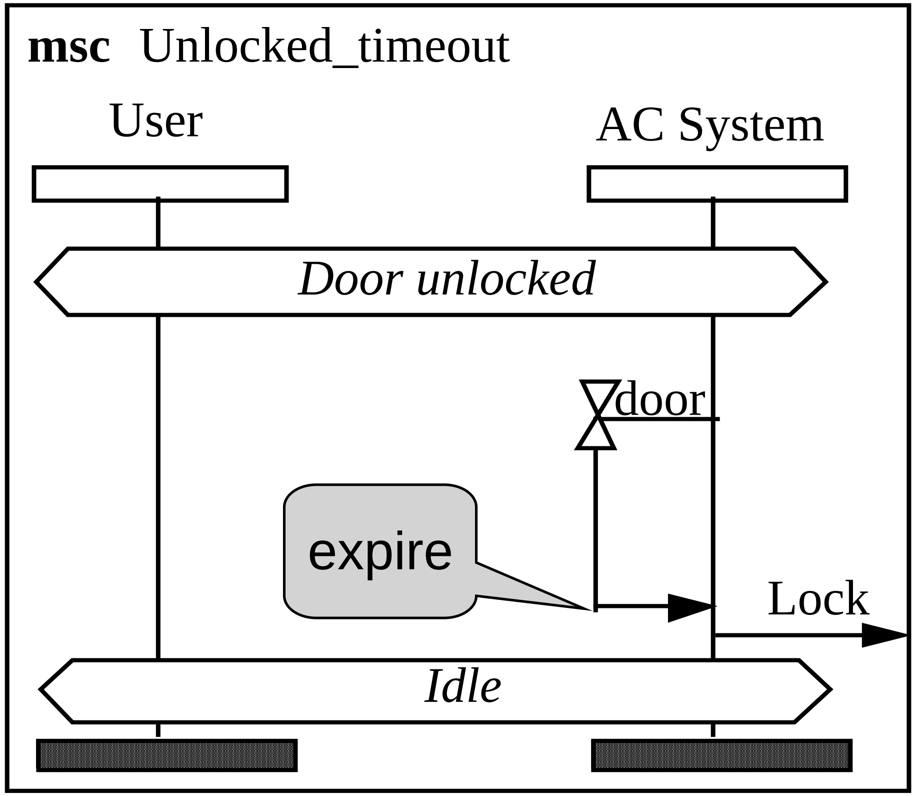
- The idea is that an MSC may have a start condition and an end condition. Combining two MSCs where the end condition of the first is equal to the start condition of the second is legal. Combining MSCs with unequal conditions is not legal.



Timers

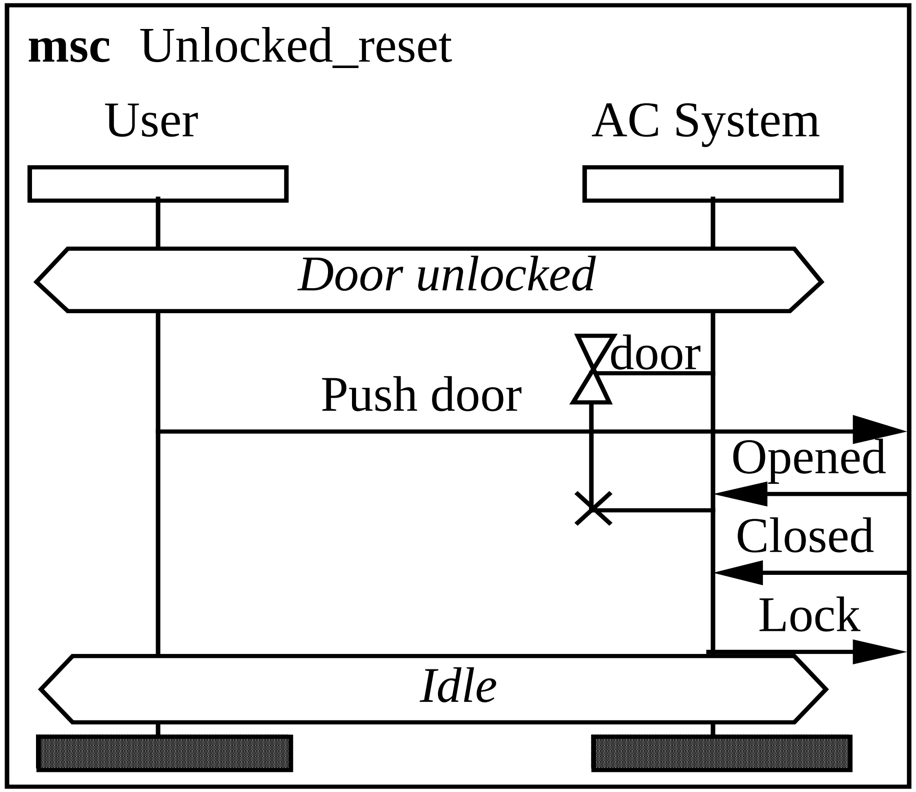
- Timers are messages which are sent by the instance to itself according to time.
- A timer can be set (started) and reset (terminated) by the instance. A timer may expire, which means an input event. Reset timers will not expire.

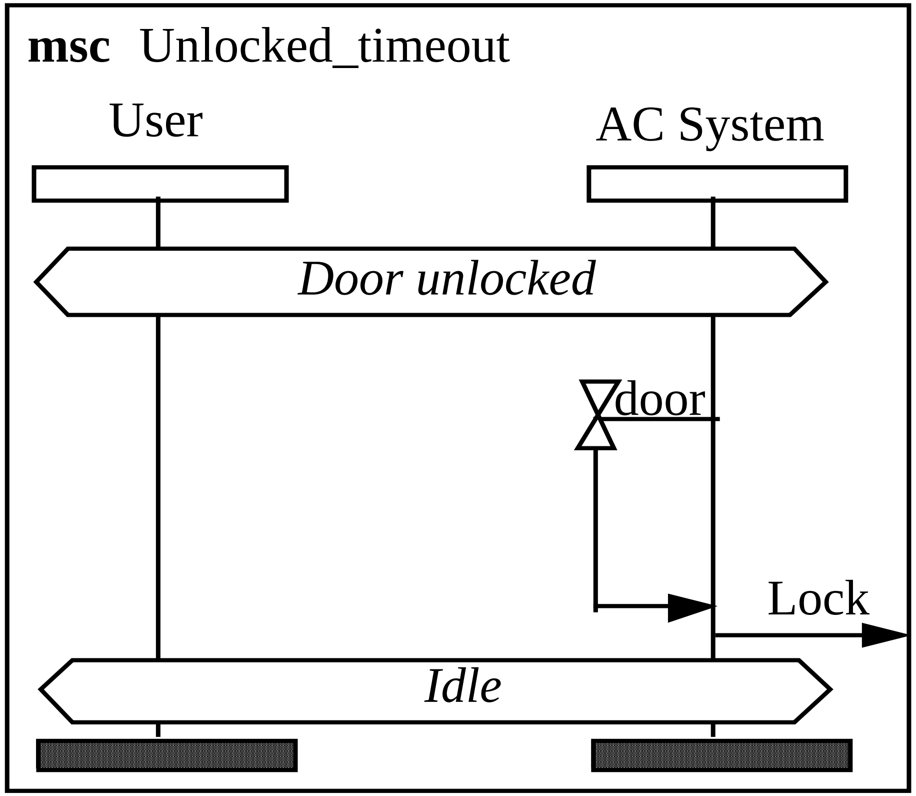




Alternatives and Iteration by conditions

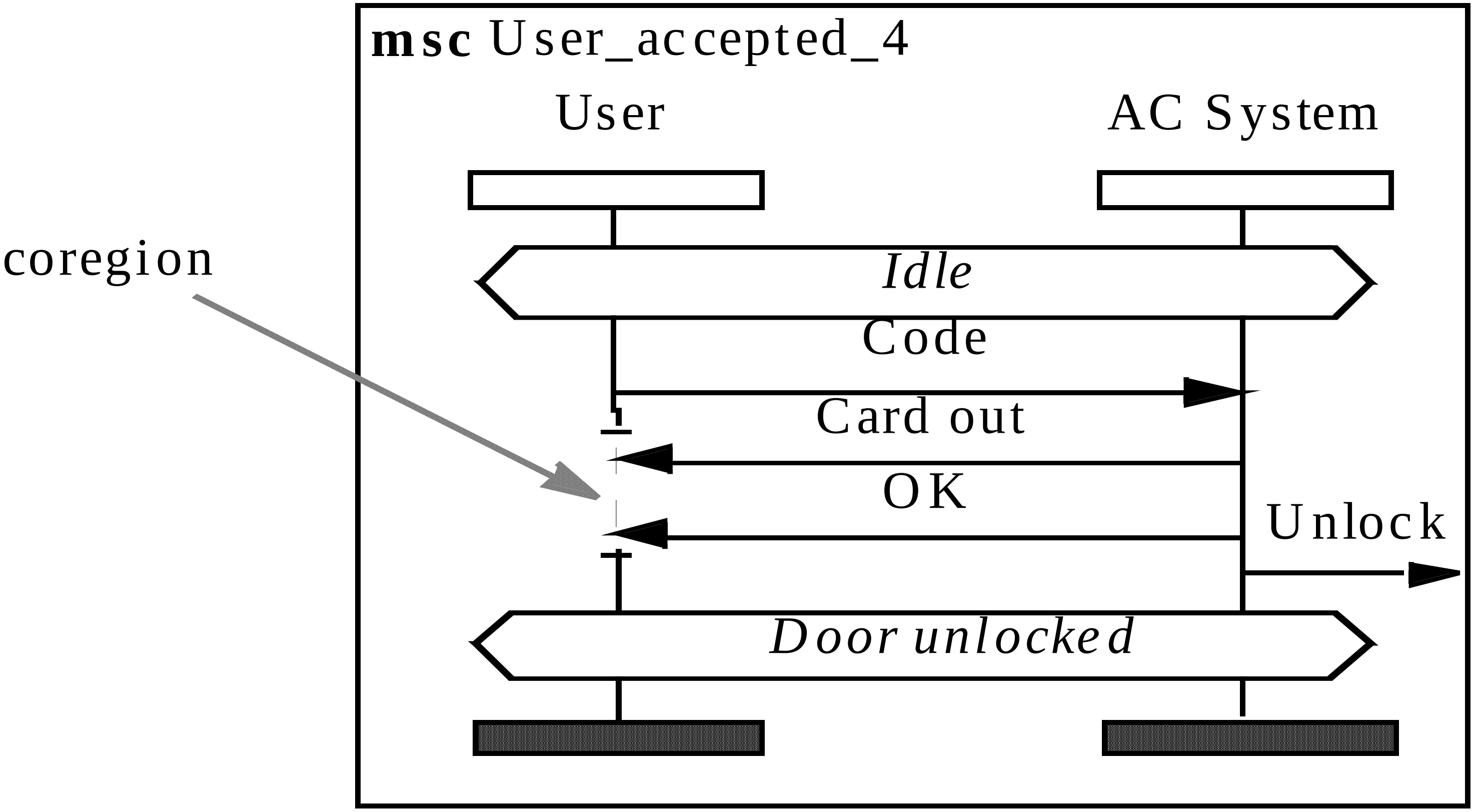
The two MSCs *Unlocked_reset* and *Unlocked_timeout* represent alternative courses of action from the state *Door Unlocked*. We also notice that they both end in *Idle* which is also the start condition of *User_accepted_3* on a former slide. This may be interpreted as describing an iterative situation.





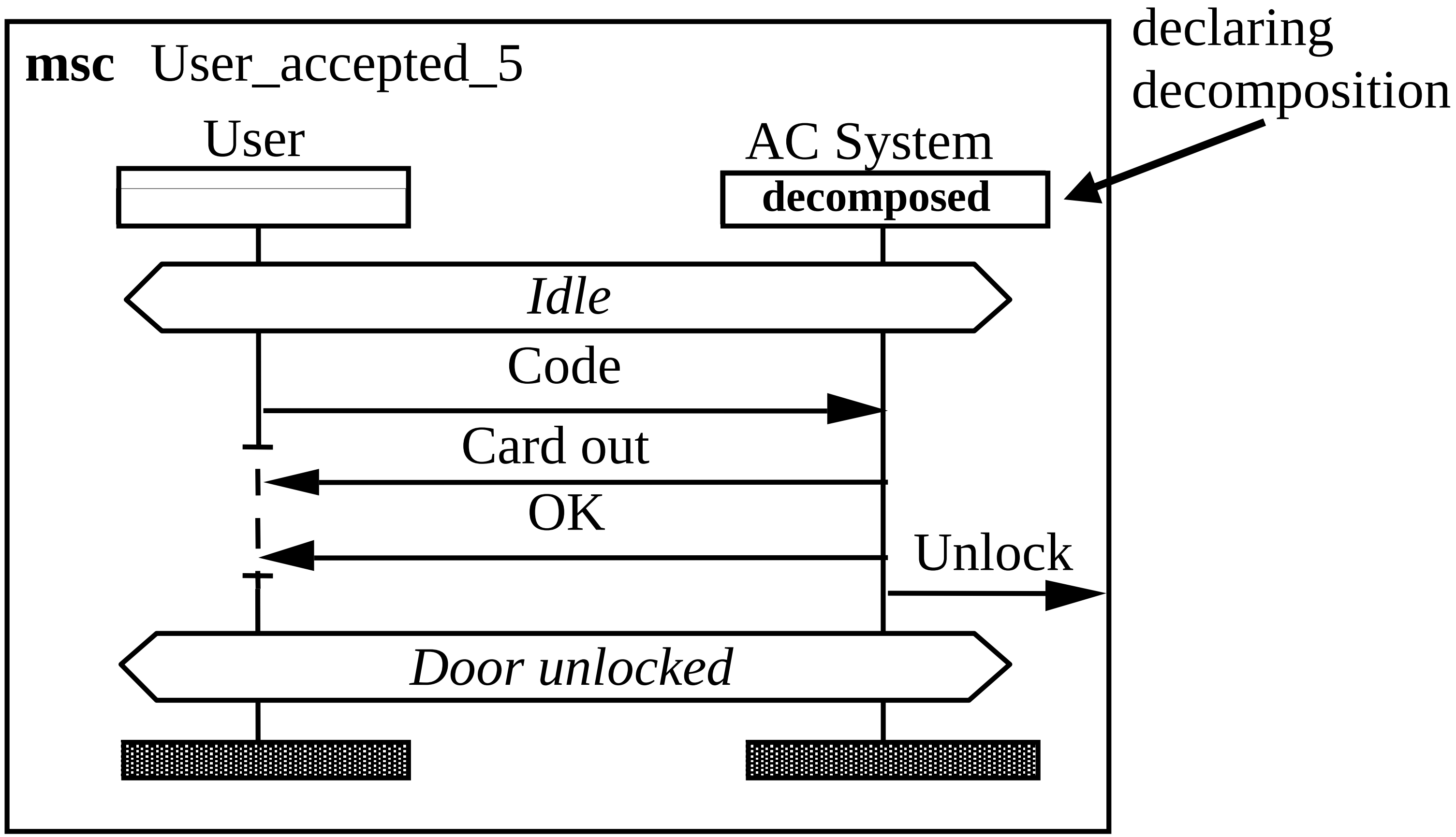
Coregion

- Coregion is a concept which is motivated by the fact that sometimes one does not care in which order a set of events occur



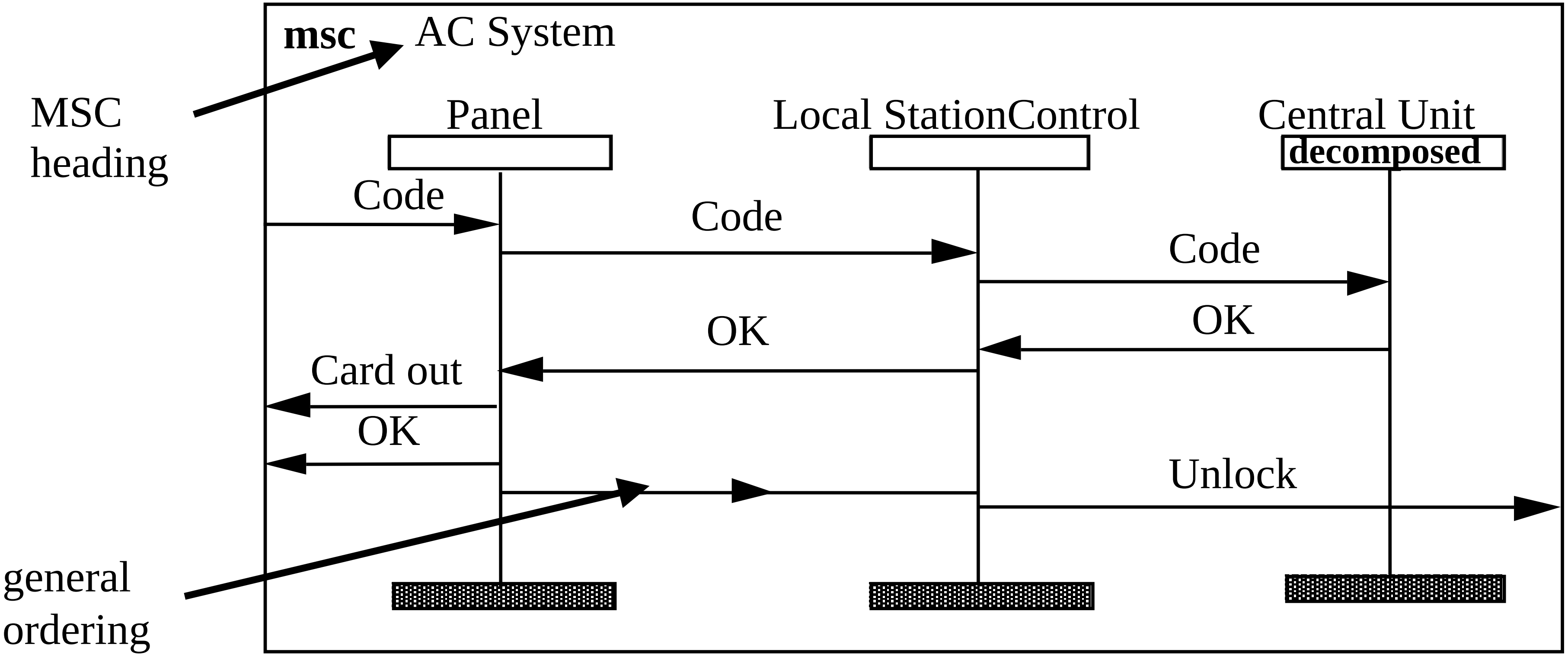
Decomposition

- When an instance contains an inner structure, the internal interactions may be described in a separate MSC diagram.
- This way specification MSCs are related to design MSCs



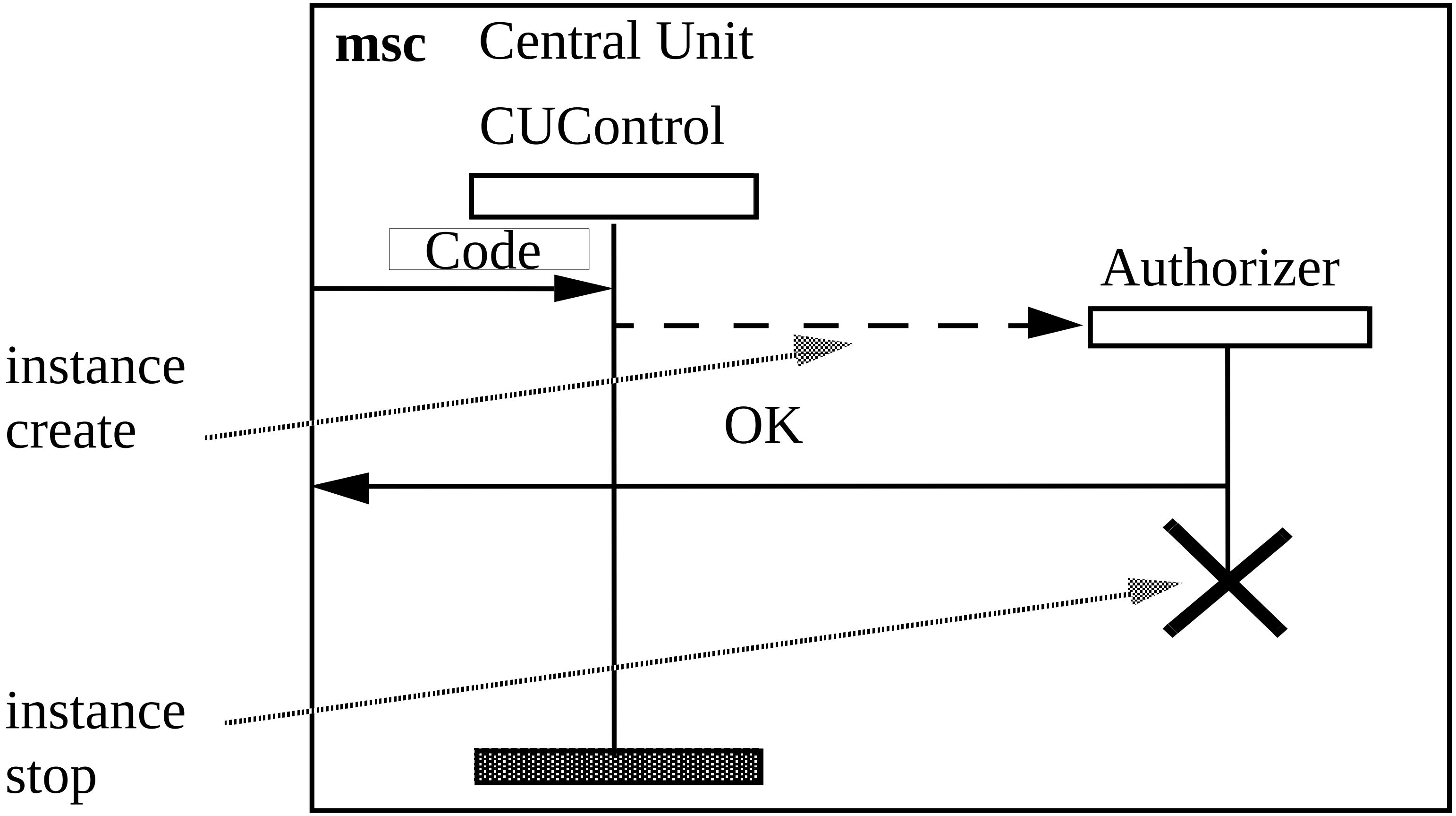
Instance decomposition

- The static requirement is that the interface should be exactly matching
- There cannot be more than one MSC with the same name in one document



Instance creation

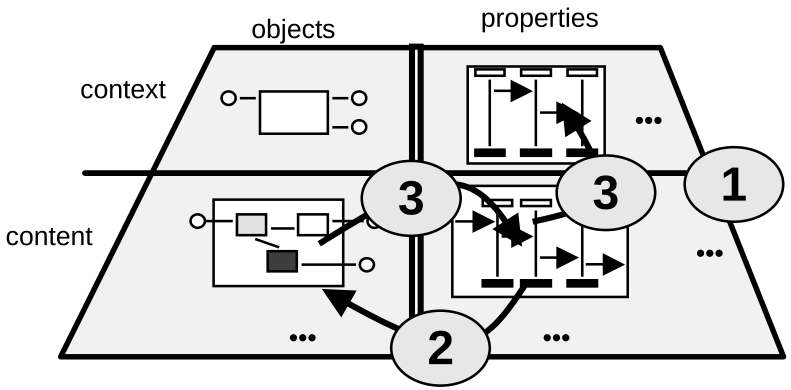
- The idea here (though rather far fetched) is that the *CUControl* needs to create a new process in the big mainframe computer to perform the task of authorizing the received *Code*. We see a situation where several Authorizers may work in parallel

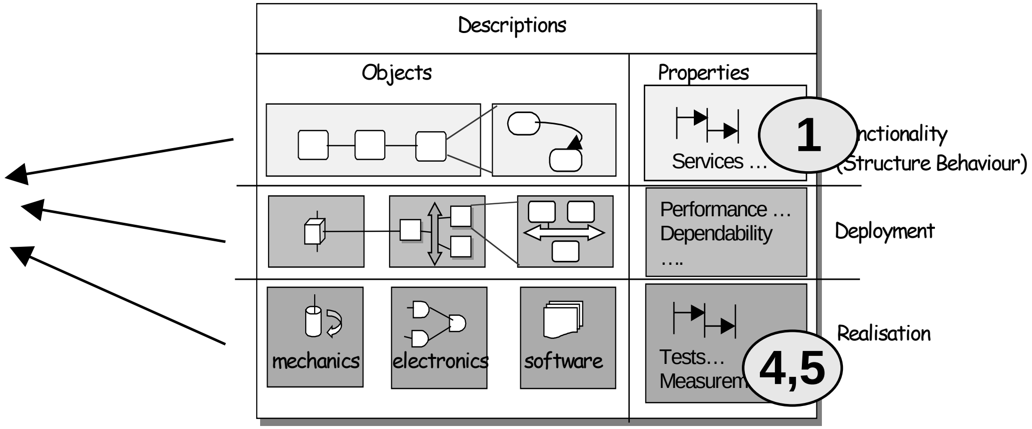


Basic MSC Summary

- Message are asynchronous, the output of one message must come before the corresponding input.
- The events on an instance’ timeline are strictly ordered (if it contains no coregion).
- The distance between events is not significant.
- An MSC document consists of a set of MSCs.
- Different MSCs within the same MSC document may be related by conditions.
- An instance (within an MSC) may be decomposed.
- In a coregion the events may come in any order.
- Dynamic creation and termination of instance

MSC is used to:





1. Precisely define behaviour properties such as:
   - use cases
   - interface behavior cases, protocol sequences
   - service behaviors
2. Partially synthesise designs
3. Verify that designs satisfy specified behaviour properties
4. Describe test cases
5. Document simulation traces
6. Generally improve understanding and communication about interaction problems

Structural features not covered here

- **MSC references** – such that MSCs may be referenced within other MSCs
- **MSC expressions** – combining MSCs to express alternatives, parallel merge and loops
- **Gates** – flexible connection points between references/expressions and their surroundings
- **HMSC** – High level MSC for better better overview of MSC documents
- **General ordering** – when neither strict order nor no order is the situation
- **Substitution** – generalizing MSCs wrt. message, instance and MSC names
- **MSC Document** – declaring a collection of MSC and their data
- **Decomposition** – decomposing instances