Early design phases

Jian-Jia Chen
(slides are based on
Peter Marwedel)
TU Dortmund,
Informatik 12

© Springer, 2010

2014年 10 月 14日

These slides use Microsoft clip arts.
Microsoft copyright restrictions apply.

Models of computation considered in this course

Communication/ local computations	Shared memory	Message passing Synchronous | Asynchronous	
Undefined components	Plain text, use cases (Message) sequence charts		
Communicating finite state machines	StateCharts		SDL
Data flow			Kahn networks, SDF
Petri nets		C/E nets, P/T nets, …	
Discrete event (DE) model	VHDL*, Verilog*, SystemC*, …	Only experimental systems, e.g. distributed DE in Ptolemy	
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |	

Capturing the requirements as text

- In the very early phases of some design project, only descriptions of the system under design (SUD) in a natural language such as English or Japanese exist.
- Expectations for tools:
  - Machine-readable
  - Version management
  - Dependency analysis
  - Example: DOORS® [Telelogic/IBM]

Use cases

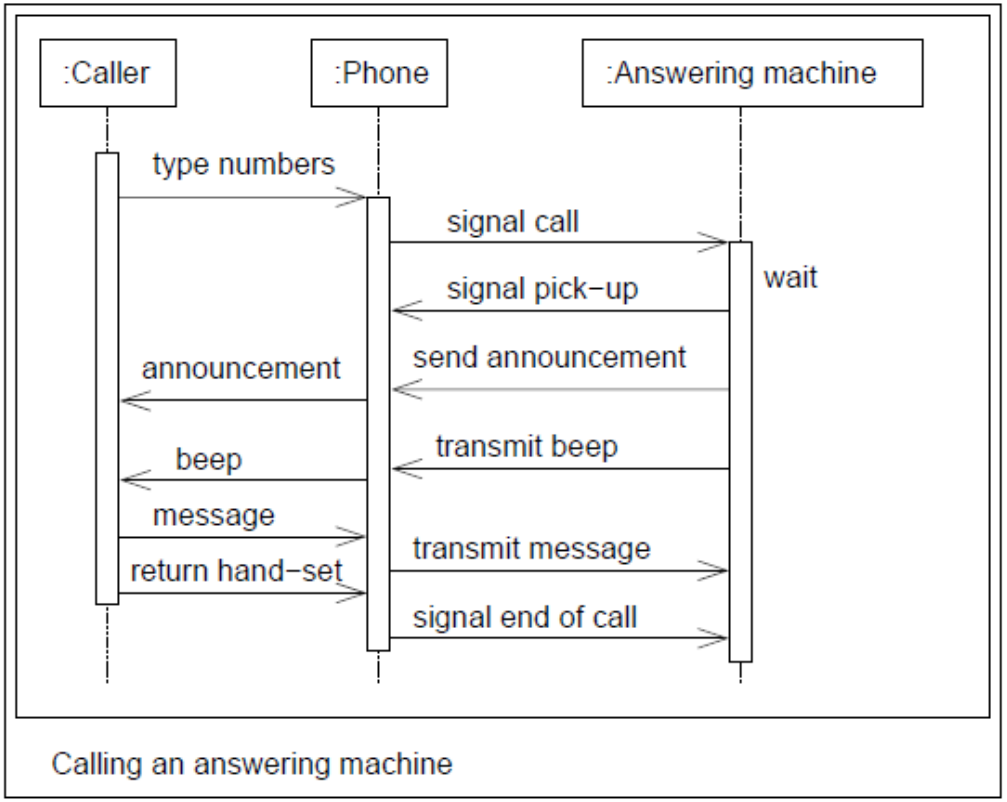
- Use cases describe possible applications of the SUD
- Included in UML (Unified Modeling Language)
- Example: Answering machine

- Neither a precisely specified model of the computations nor a precisely specified model of the communication

(Message) Sequence charts

- Explicitly indicate exchange of information
- One dimension (usually vertical dimension) reflects time
- The other reflects distribution in space

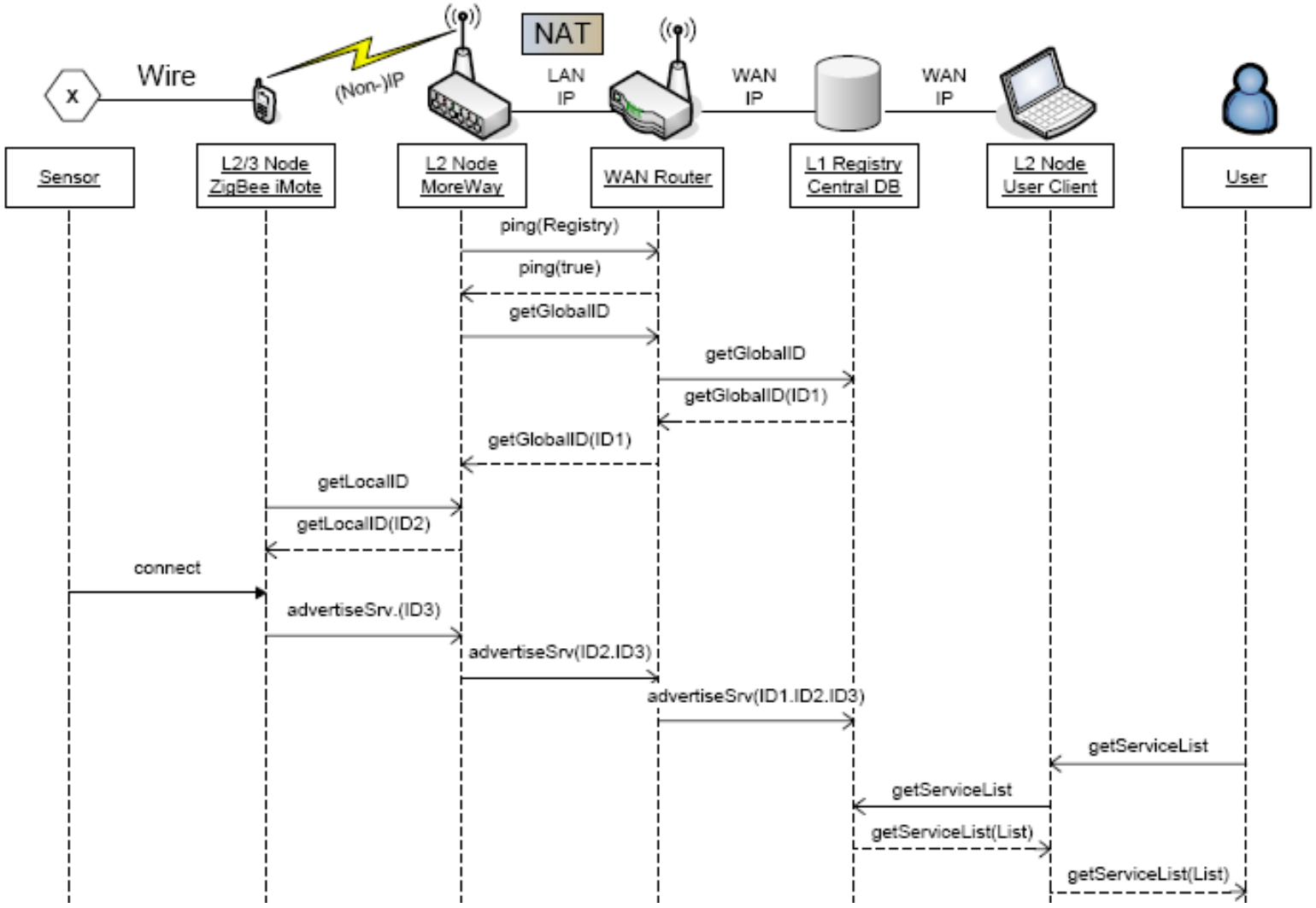
Example:



Calling an answering machine

- Included in UML
- Earlier called Message Sequence Charts, now mostly called Sequence Charts

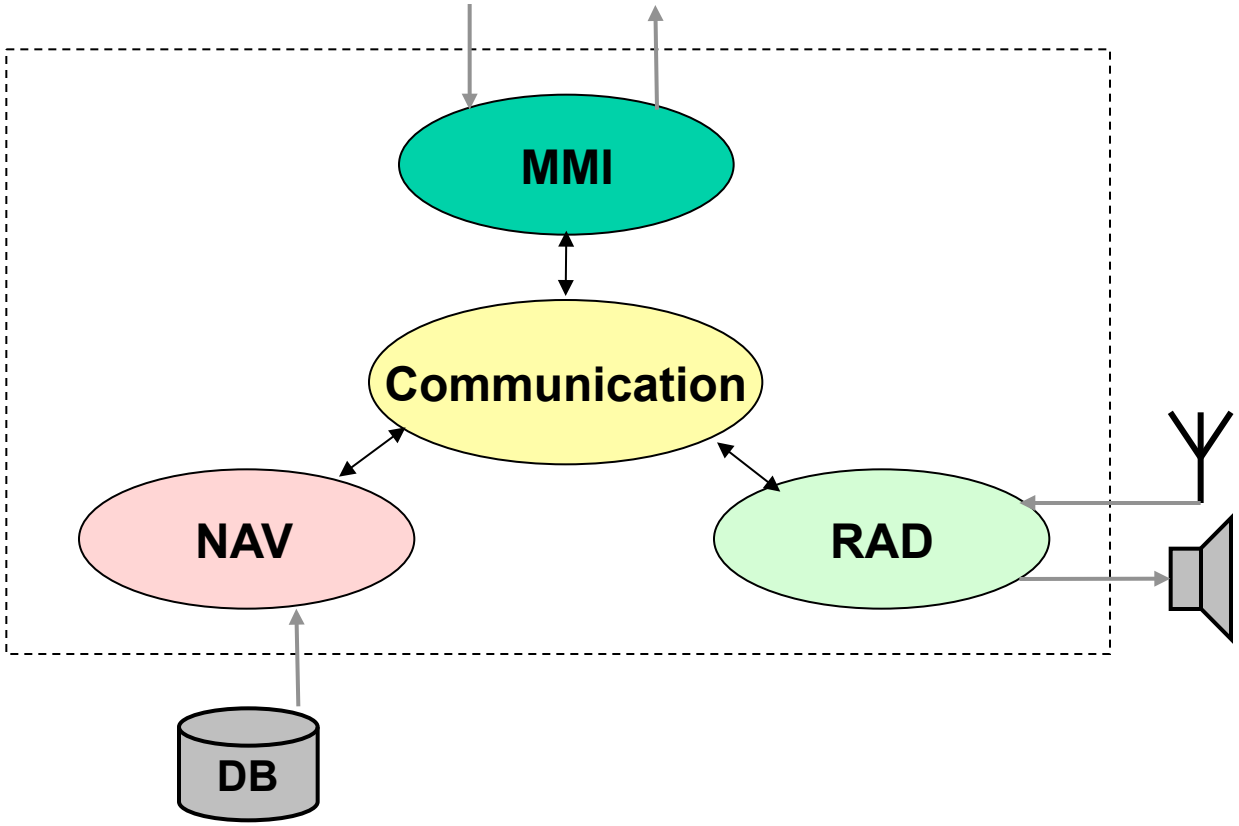
Example (2)


Wire
(Non-)IP
NAT
LAN IP
WAN IP
WAN IP
Sensor
L2/3 Node ZigBee iMote
L2 Node MoreWay
WAN Router
L1 Registry Central DB
L2 Node User Client
User
ping(Registry)
ping(true)
getGlobalID
getGlobalID
getGlobalID(ID1)
getGlobalID(ID1)
getLocalID
getLocalID(ID2)
connect
advertiseSrv.(ID3)
advertiseSrv(ID2.ID3)
advertiseSrv(ID1.ID2.ID3)
getServiceList
getServiceList
getServiceList(List)
getServiceList(List)

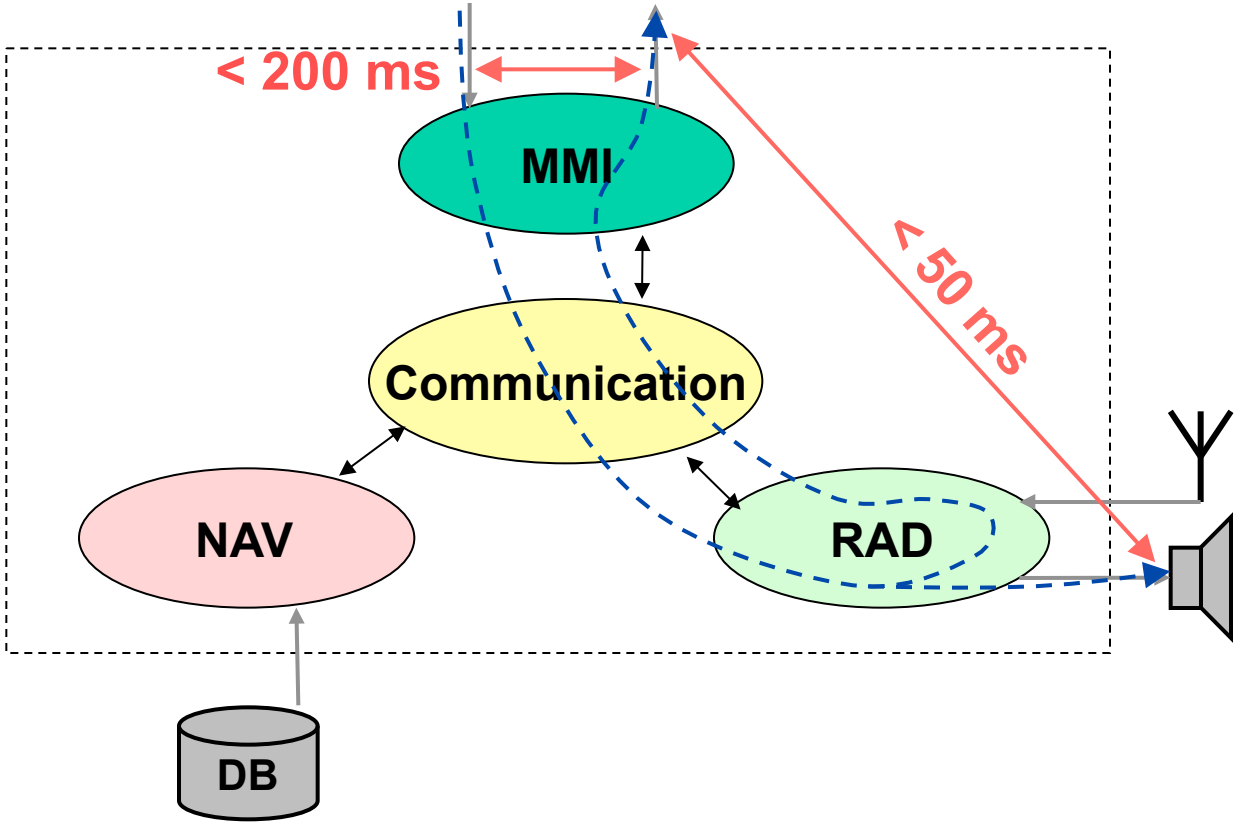
Application: In-Car Navigation System

Car radio with navigation system
User interface needs to be responsive
Traffic messages (TMC) must be processed in a timely way
Several applications may execute concurrently

System Overview

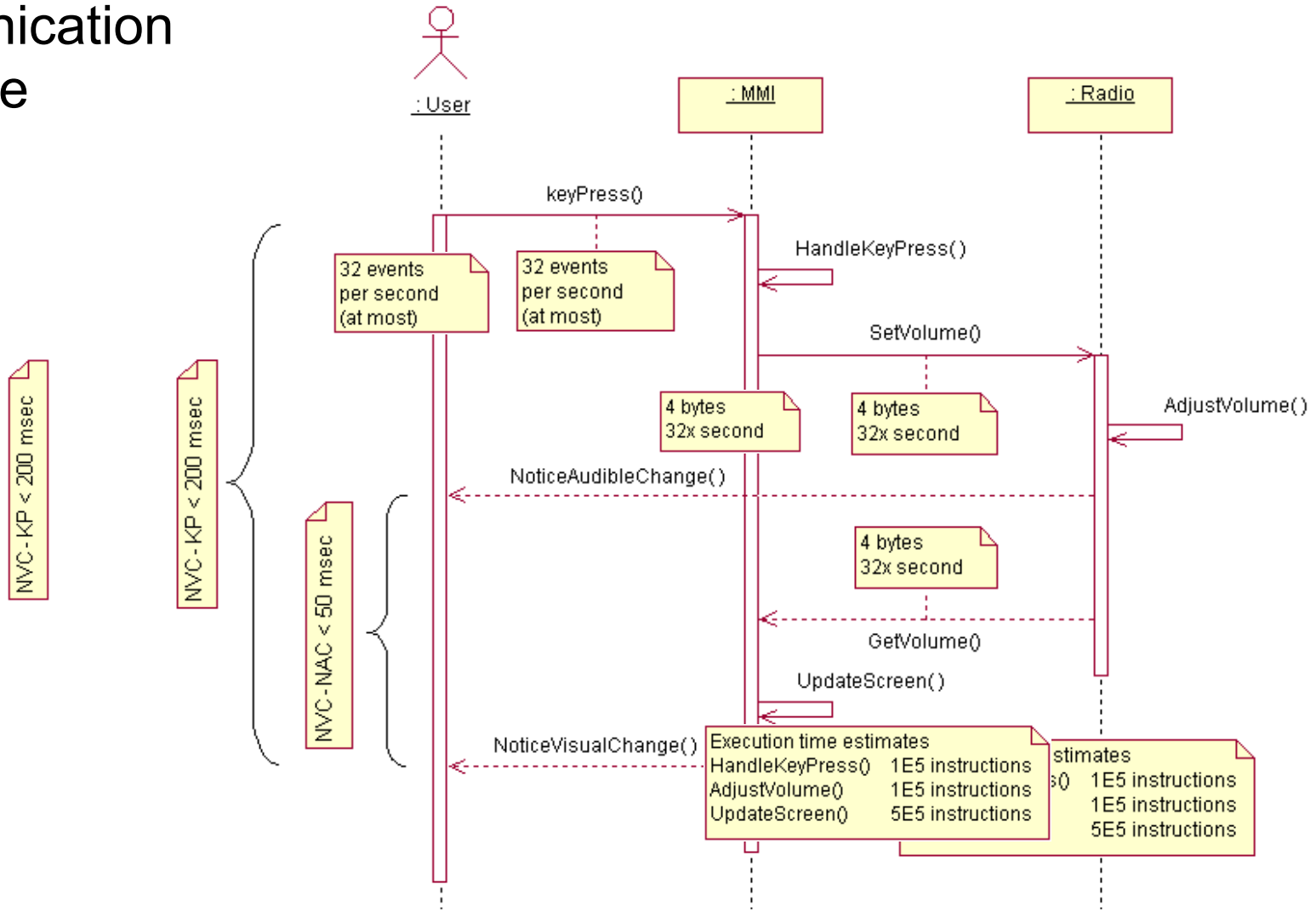


Use case 1: Change Audio Volume

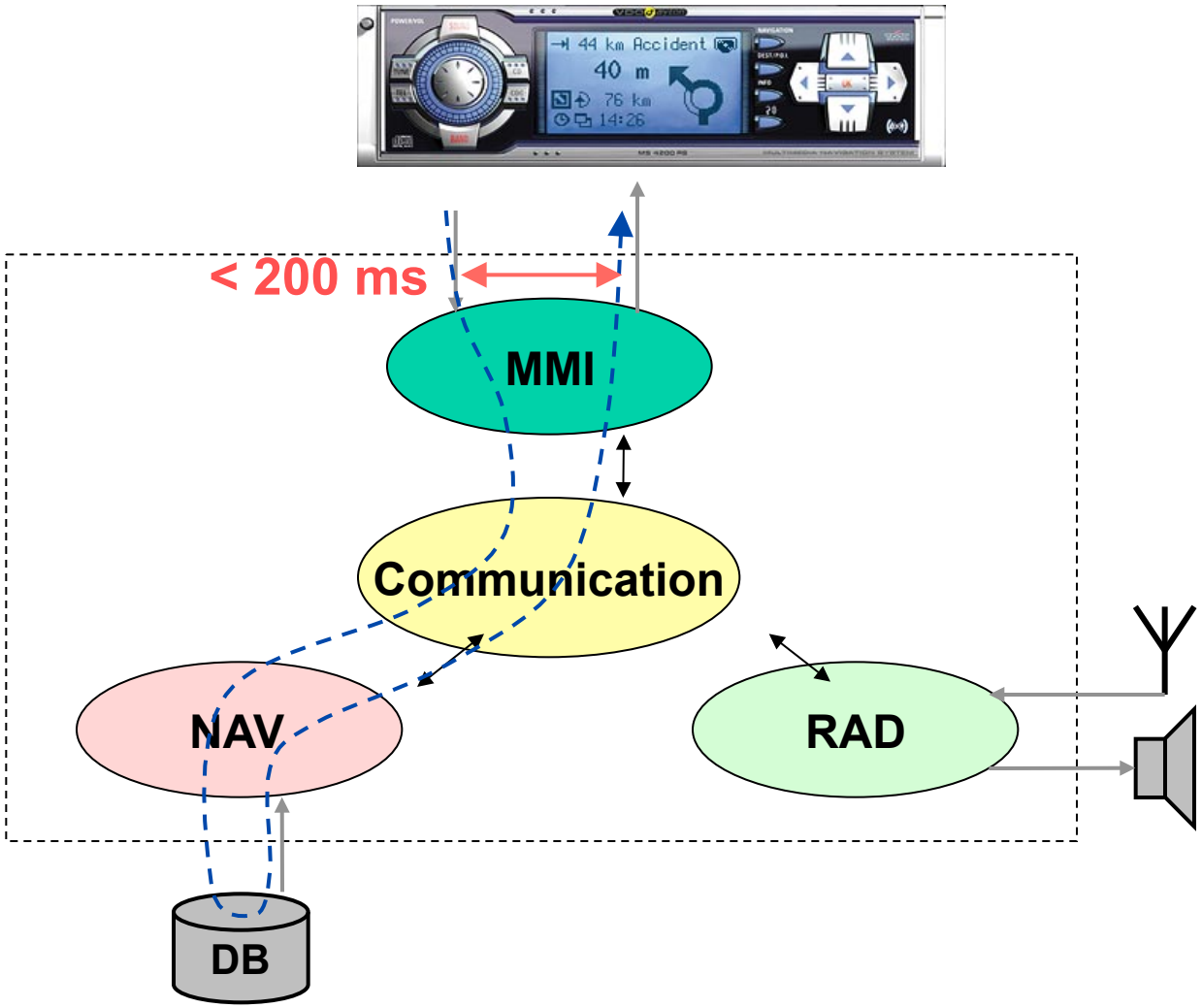


Use case 1: Change Audio Volume

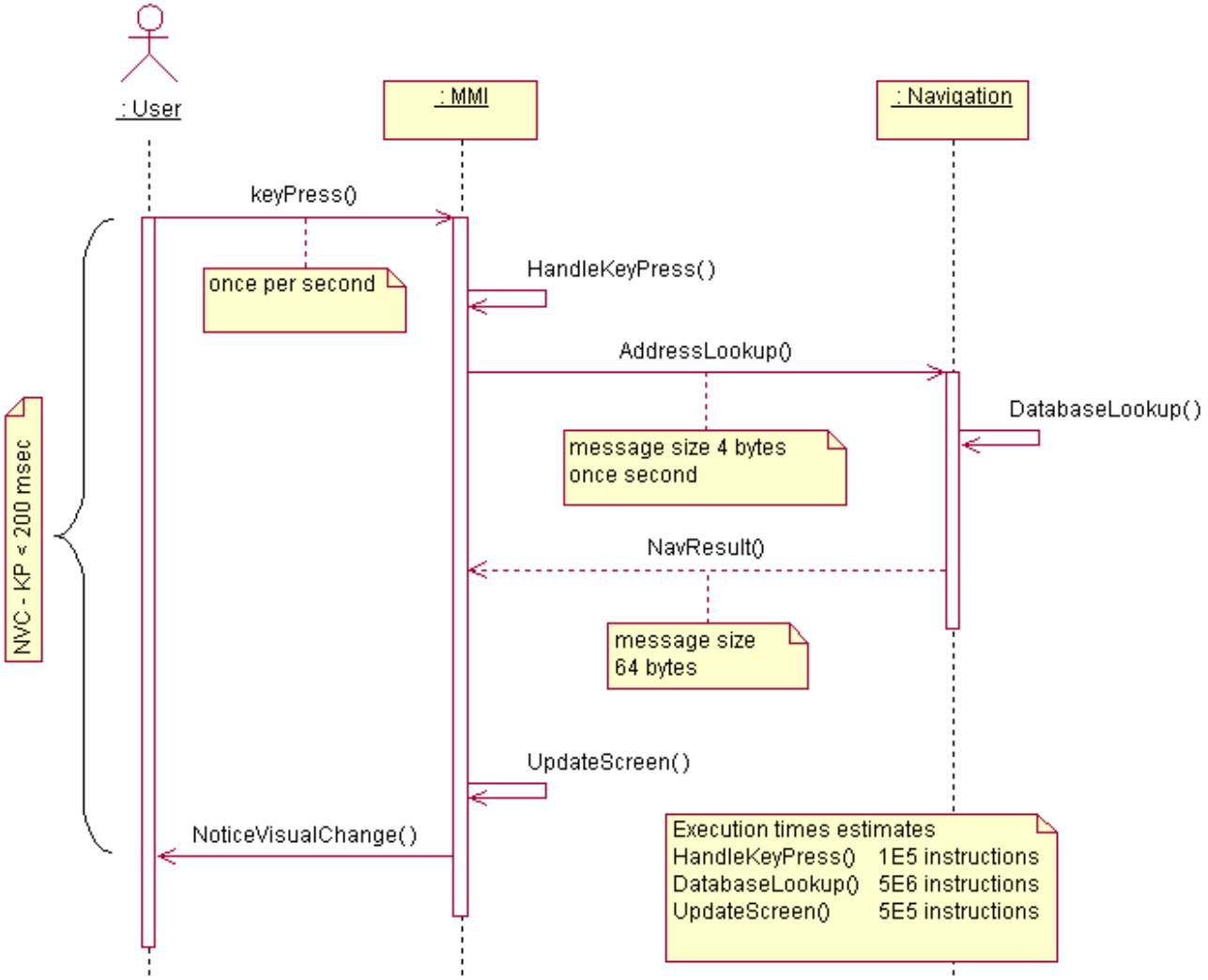
Communication
Resource
Demand



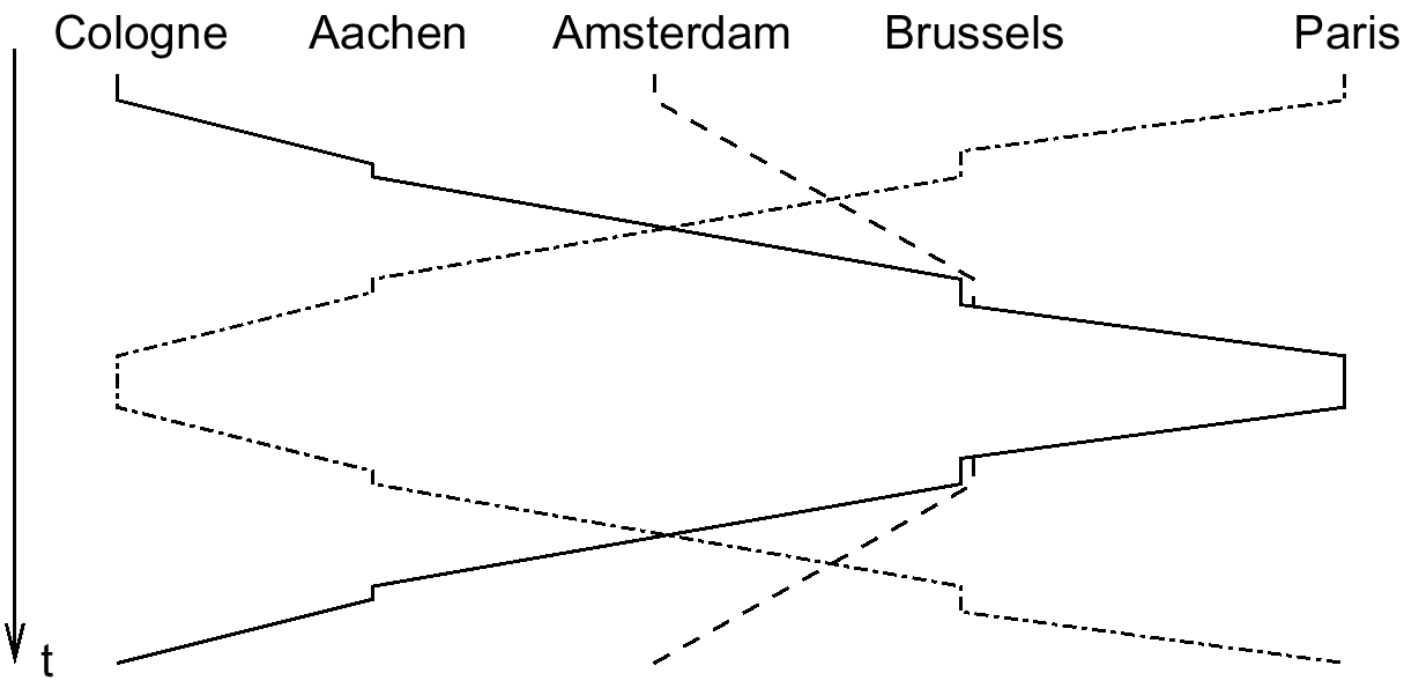
Use case 2: Lookup Destination Address



Use case 2: Lookup Destination Address

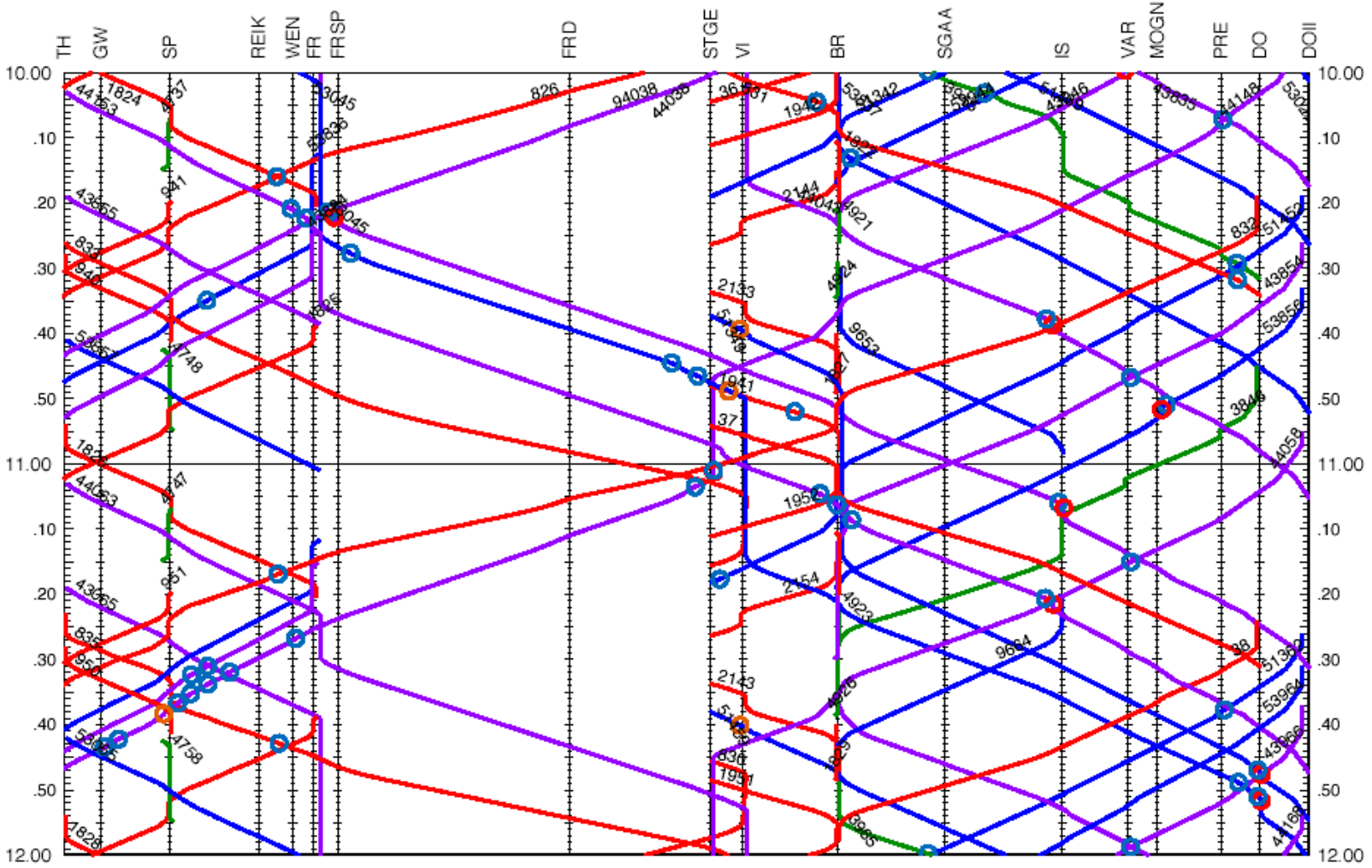


Time/distance diagrams as a special case



No distinction between accidental overlap and synchronization

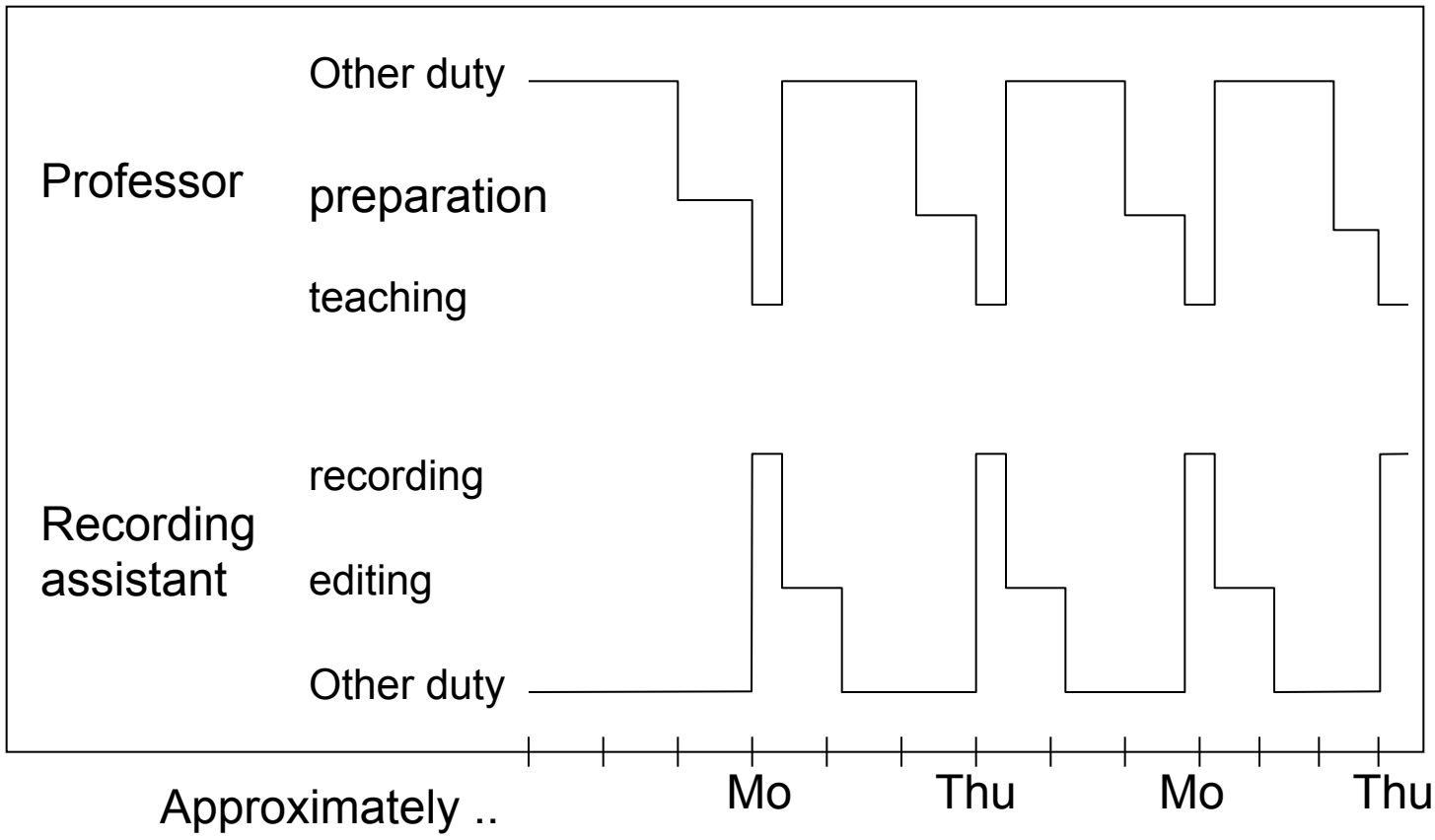
Time/distance diagrams as a special case



© www.opentrack.ch

UML: Timing diagrams

Can be used to show the change of the state of an object over time.



Based on Scott Ambler, Agile Modeling, //www.agilemodeling.com, 2003

Summary

- Motivation for non-von Neumann models
- Support for early design phases
  - Text
  - Use cases
  - (Message) sequence charts

StateCharts and StateMates

Jian-Jia Chen
(slides are based on Peter Marwedel)
TU Dortmund,
Informatik 12

2014年 10 月 14 日

© Springer, 2010

These slides use Microsoft clip arts.
Microsoft copyright restrictions apply.

Models of computation considered in this course

Communication/ local computations	Shared memory	Message passing Synchronous | Asynchronous	
Undefined components	Plain text, use cases (Message) sequence charts		
Communicating finite state machines	**StateCharts**		SDL
Data flow			Kahn networks, SDF
Petri nets		C/E nets, P/T nets, …	
Discrete event (DE) model	VHDL*, Verilog*, SystemC*, …	Only experimental systems, e.g. distributed DE in Ptolemy	
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |	

StateCharts

Classical automata not useful for complex systems (complex graphs cannot be understood by humans).

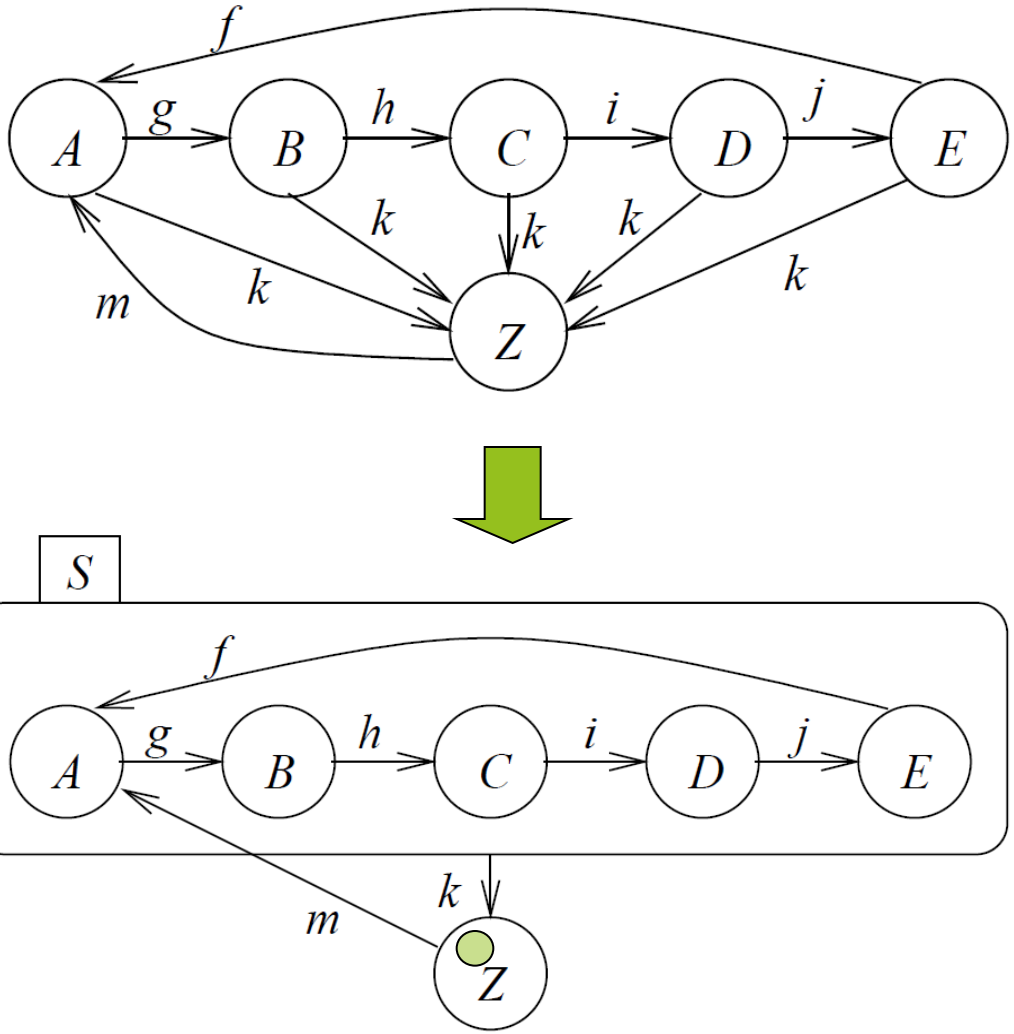
☞ Introduction of hierarchy ☞ StateCharts [Harel, 1987]

StateChart = *the only unused combination of „flow“ or „state“ with „diagram“ or „chart“*

Used here as a (prominent) example of a model of computation based on shared memory communication.

☞ appropriate only for local (non-distributed) systems

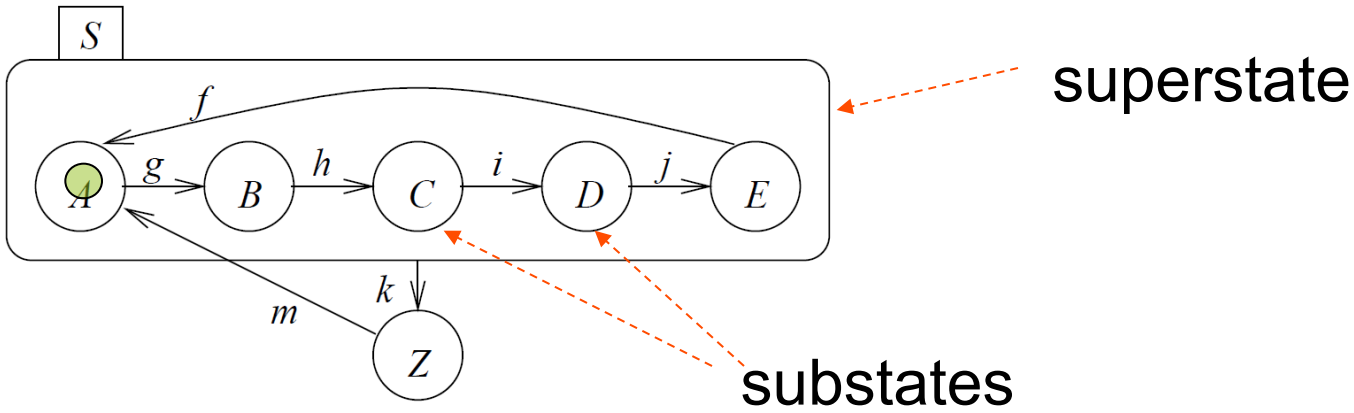
Introducing hierarchy



FSM will be **in** exactly one of the substates of S if S is **active**
(either in A or in B or ..)

Definitions

- Current states of FSMs are also called **active** states.
- States which are not composed of other states are called **basic states.**
- States containing other states are called **super-states**.
- Super-states S are called **OR-super-states**, if exactly one of the sub-states of S is active whenever S is active.



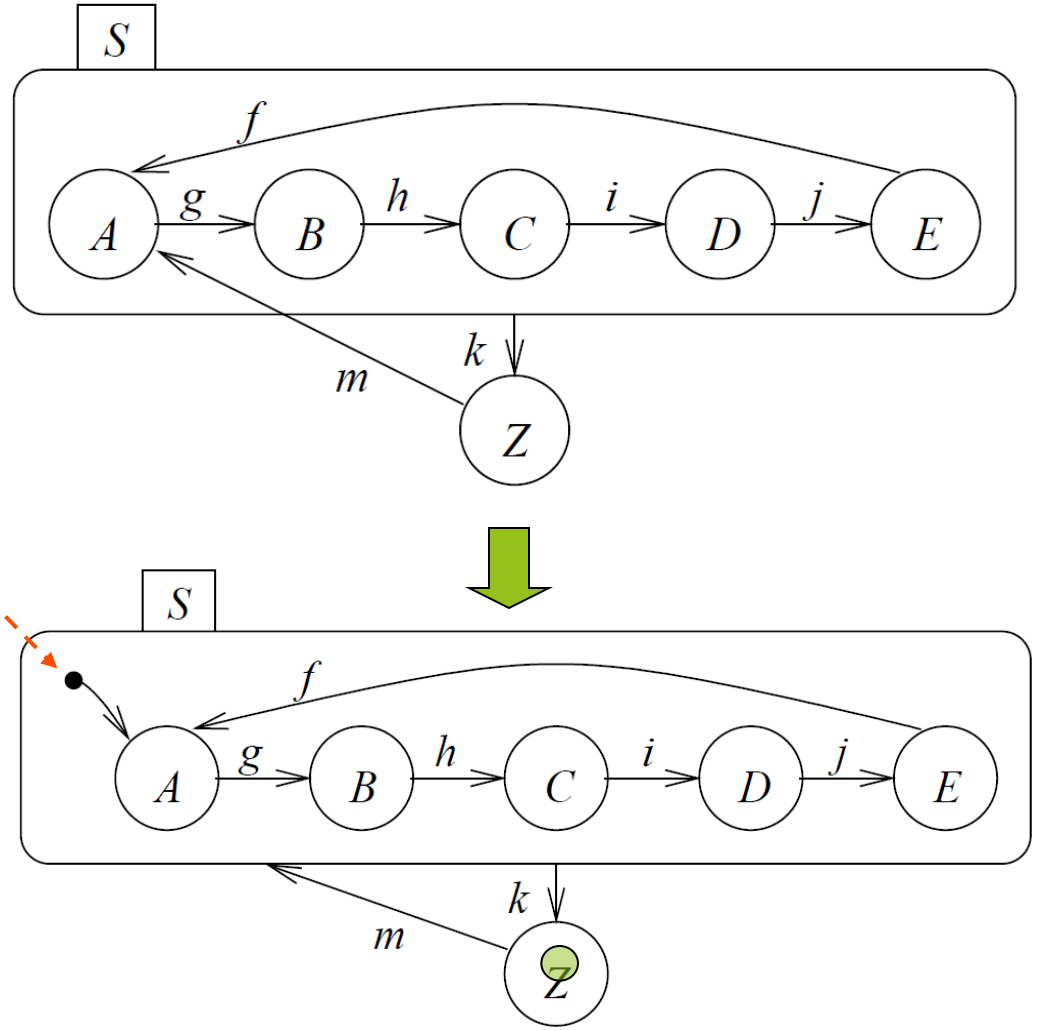
Default state mechanism

Try to hide internal structure from outside world!

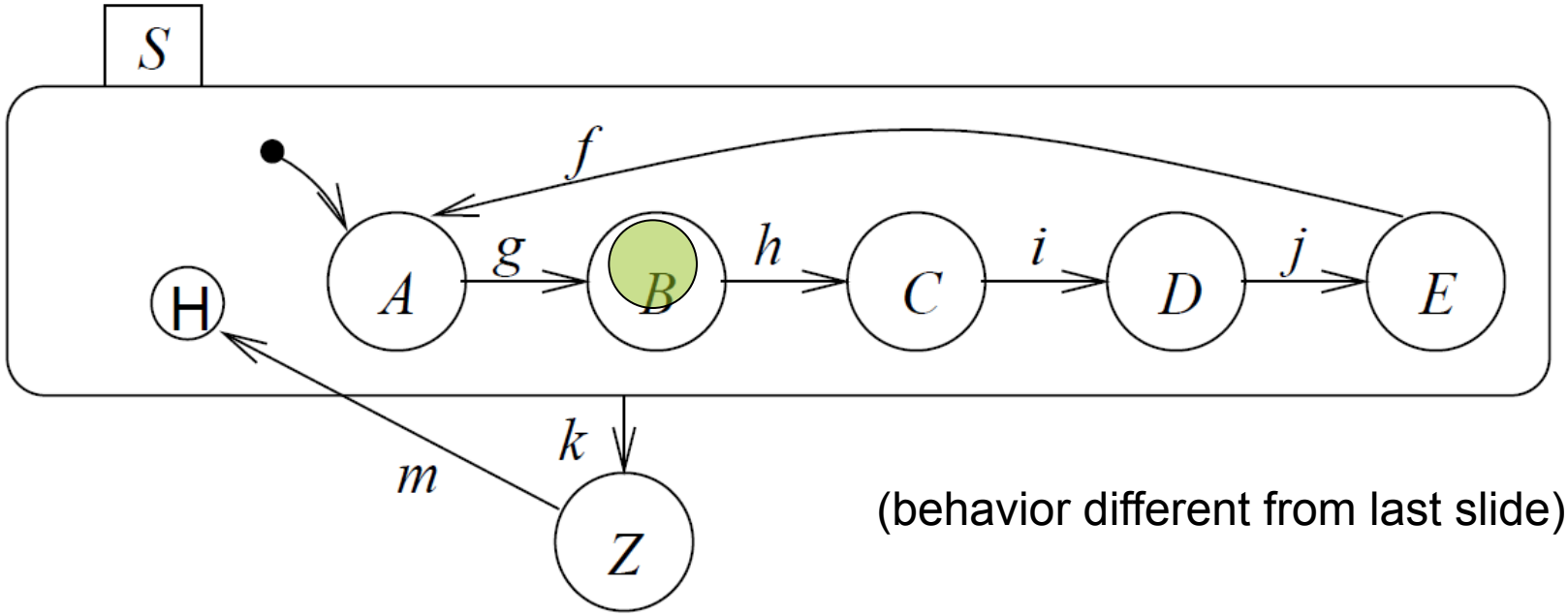
☞ Default state

Filled circle indicates sub-state entered whenever super-state is entered.

Not a state by itself!



History mechanism

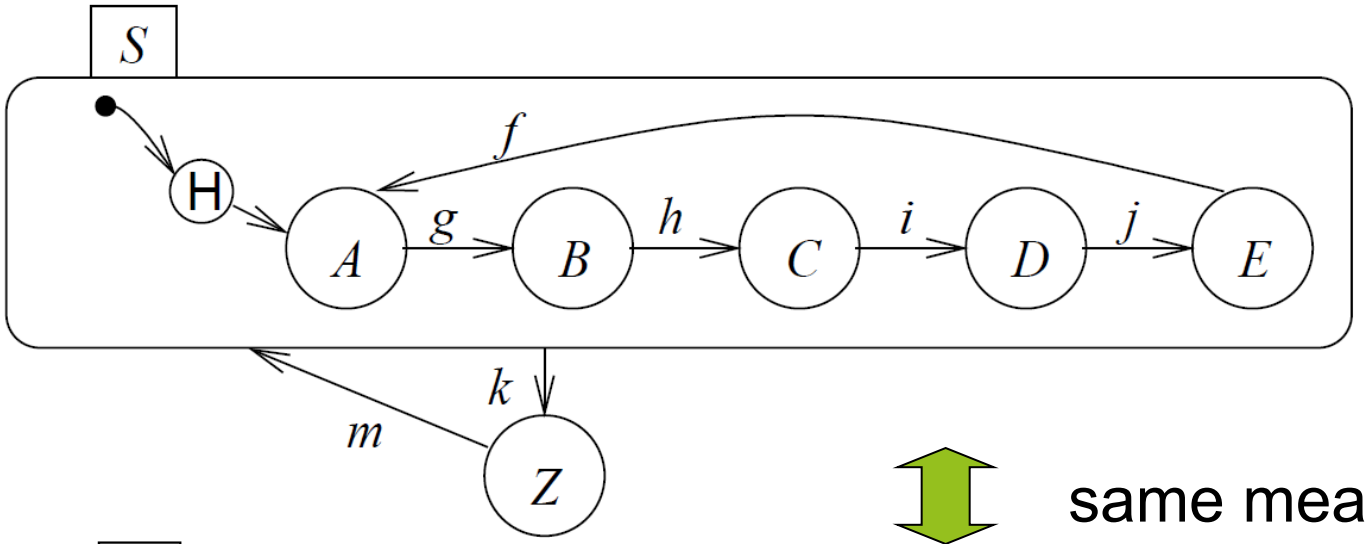


(behavior different from last slide)

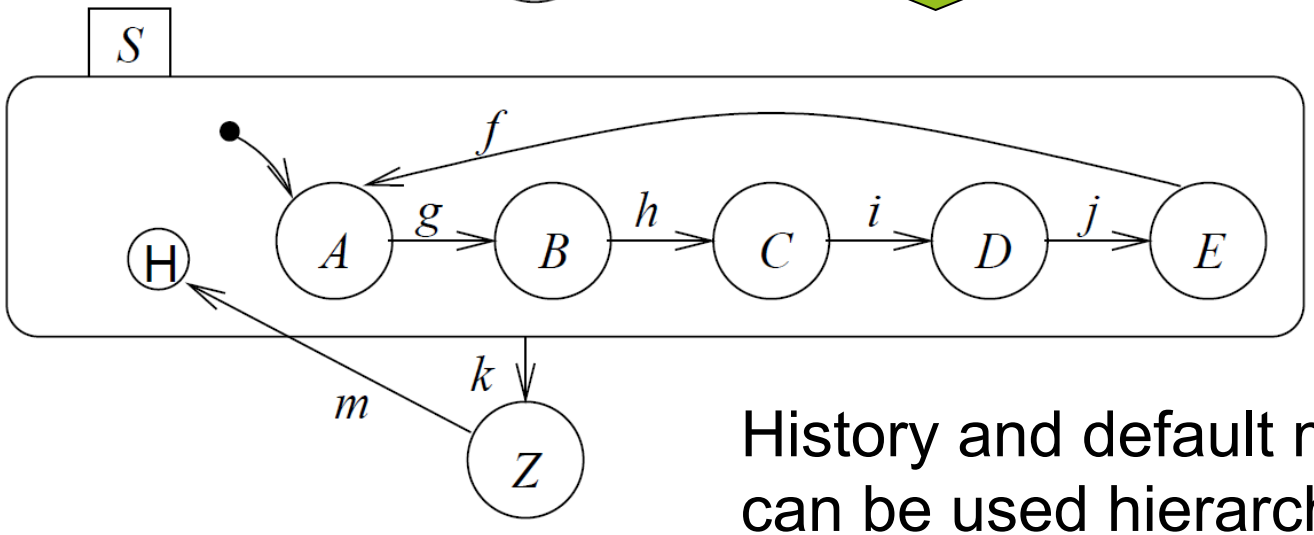
For input m, S enters the state it was in before S was left (can be A, B, C, D, or E).

If S is entered for the first time, the default mechanism applies.

Combining history and default state mechanism



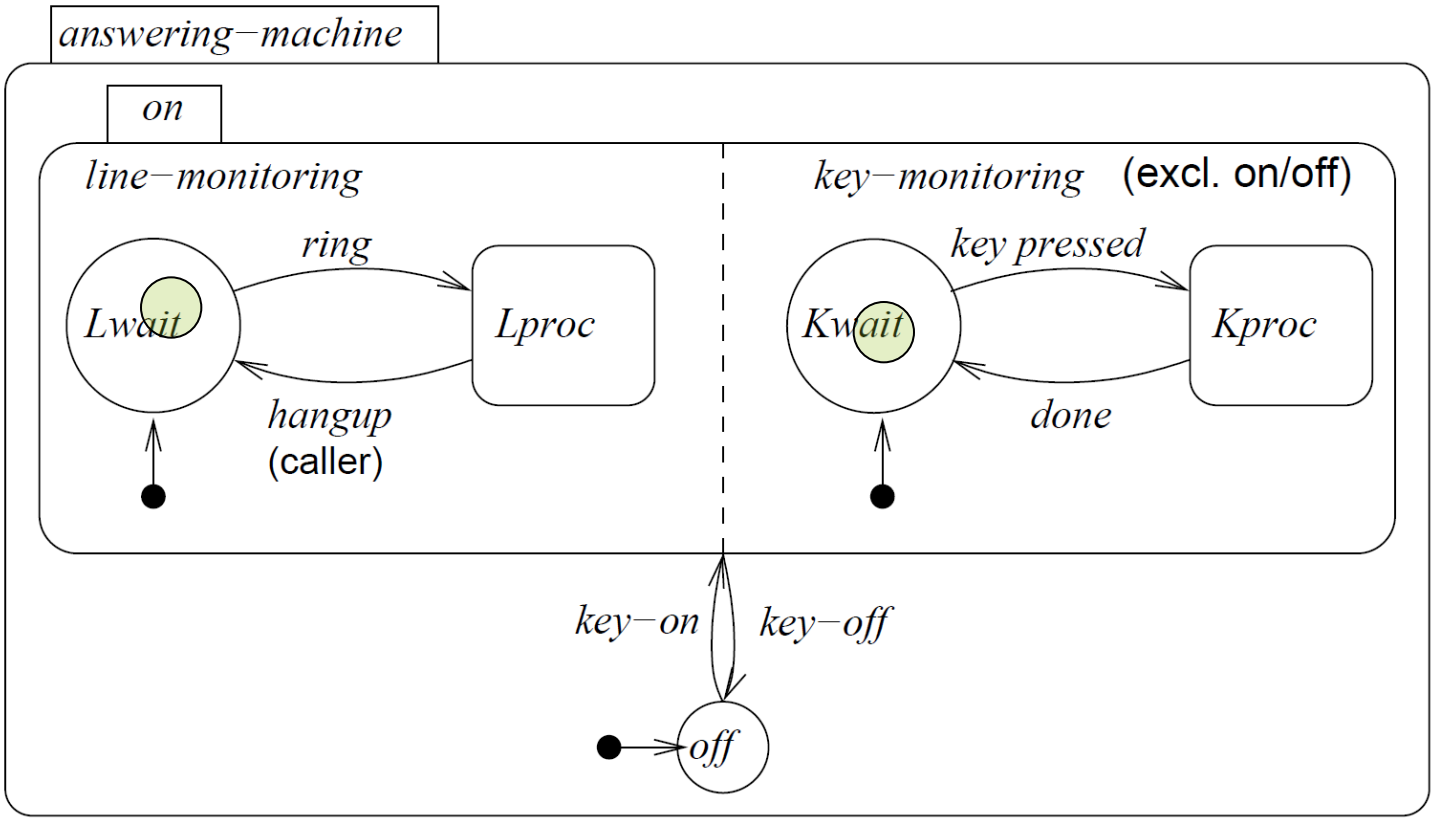
same meaning



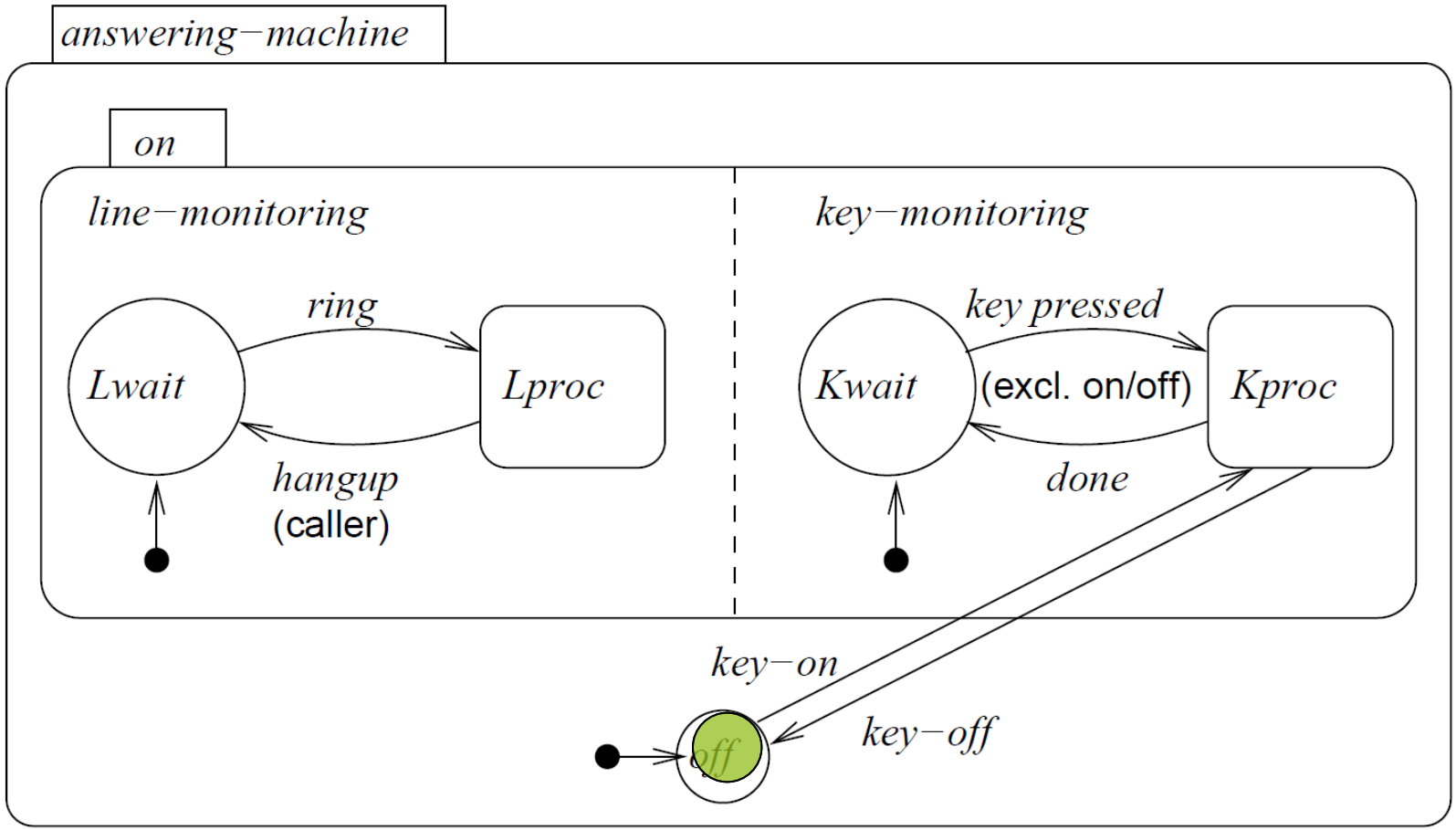
History and default mechanisms can be used hierarchically.

Concurrency

Convenient ways of describing concurrency req.
AND-super-states: FSM is in **all** (immediate) sub-states of a super-state;
Example:



Entering and leaving AND-super-states



Line-monitoring and key-monitoring are entered and left, when service switch is operated.

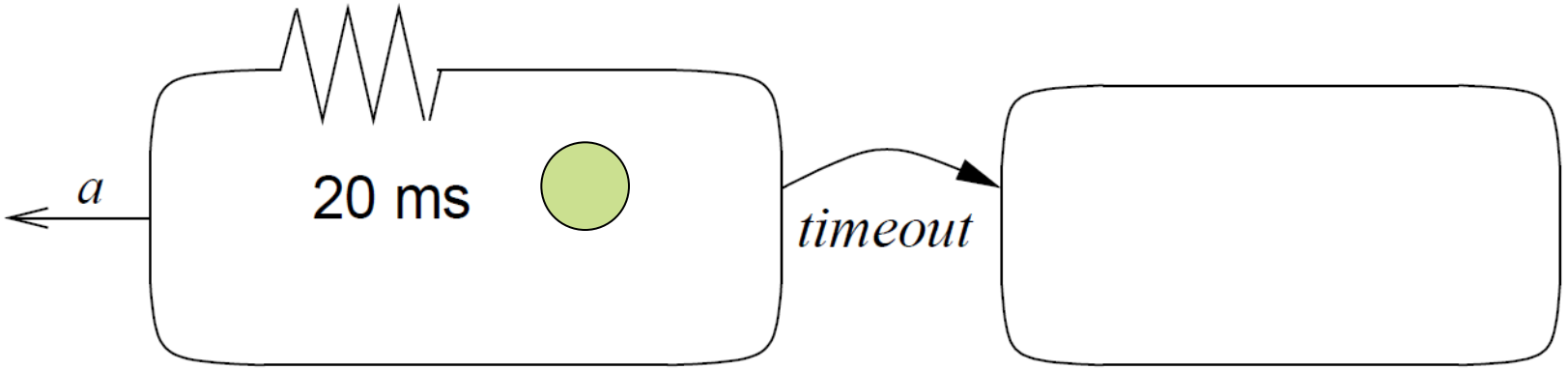
Types of states

In StateCharts, states are either

- **basic states, or**
- **AND-super-states, or**
- **OR-super-states.**

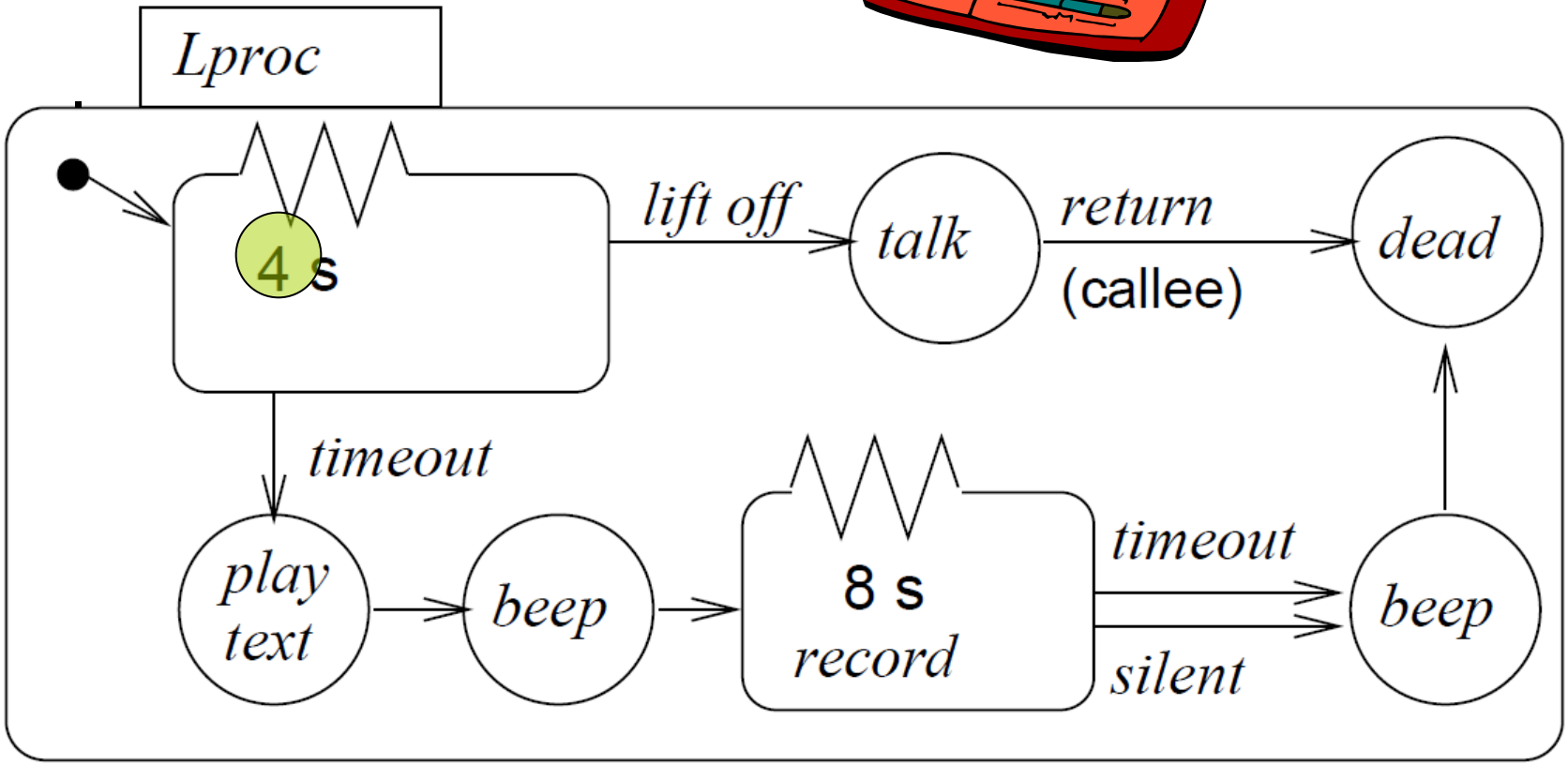
Timers

Since time needs to be modeled in embedded & cyber-physical systems, timers need to be modeled.
In StateCharts, special edges can be used for timeouts.



If event a does not happen while the system is in the left state for 20 ms, a timeout will take place.

Using timers in an answering machine



General form of edge labels

Events:

- Exist only until the next evaluation of the model
- Can be either internally or externally generated

Conditions:

- Refer to values of variables that keep their value until **they are reassigned**

Reactions:

- Can either be assignments for variables
- or creation of events

Example:

- *service-off* [not in *Lproc*] / *service*:=0

The StateCharts simulation phases (StateMate Semantics)

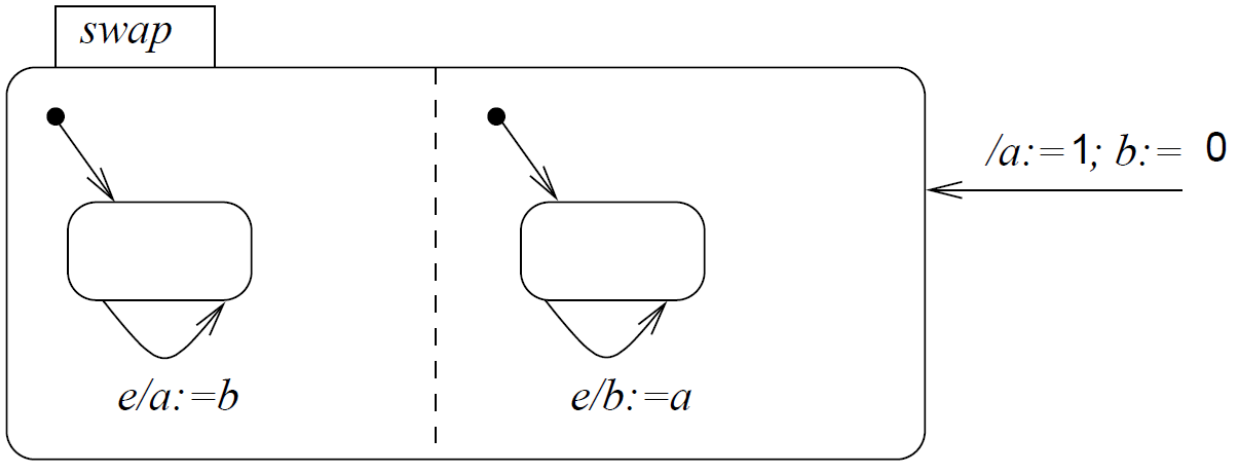
How are edge labels evaluated?

Three phases:

1. Effect of external changes on events and conditions is evaluated,
2. The set of transitions to be made in the current step and right hand sides of assignments are computed,
3. Transitions become effective, variables obtain new values.

Separation into phases 2 and 3 enables a resulting unique ("determinate") behavior.

Example



In phase 2, variables a and b are assigned to temporary variables:

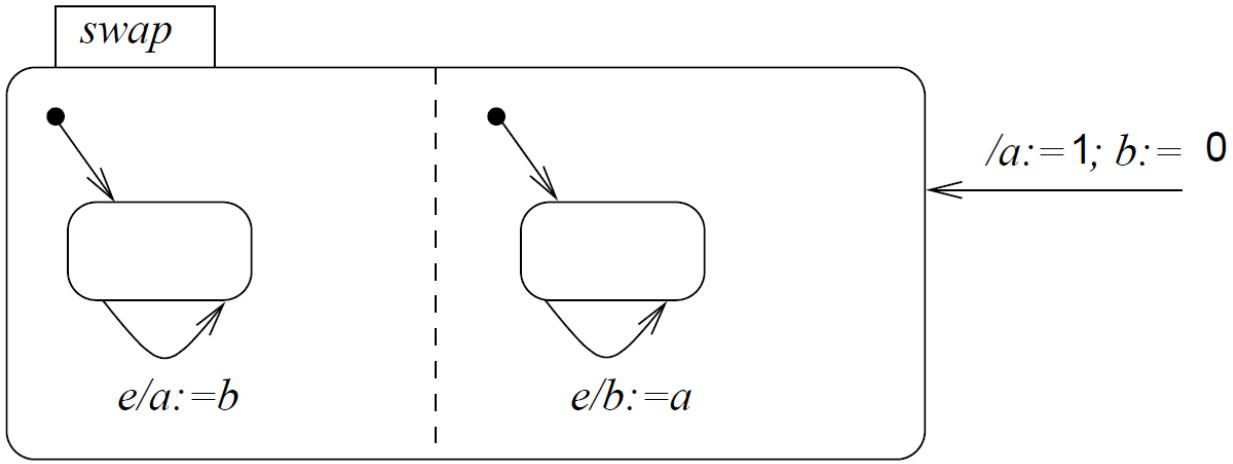
a’ := b, b’ := a;

In phase 3, these are assigned to a and b.

a := a’, b := b’;

As a result, variables a and b are swapped.

Example (2)



In a single phase environment, executing the left state first would assign the old value of b (=0) to a and b:

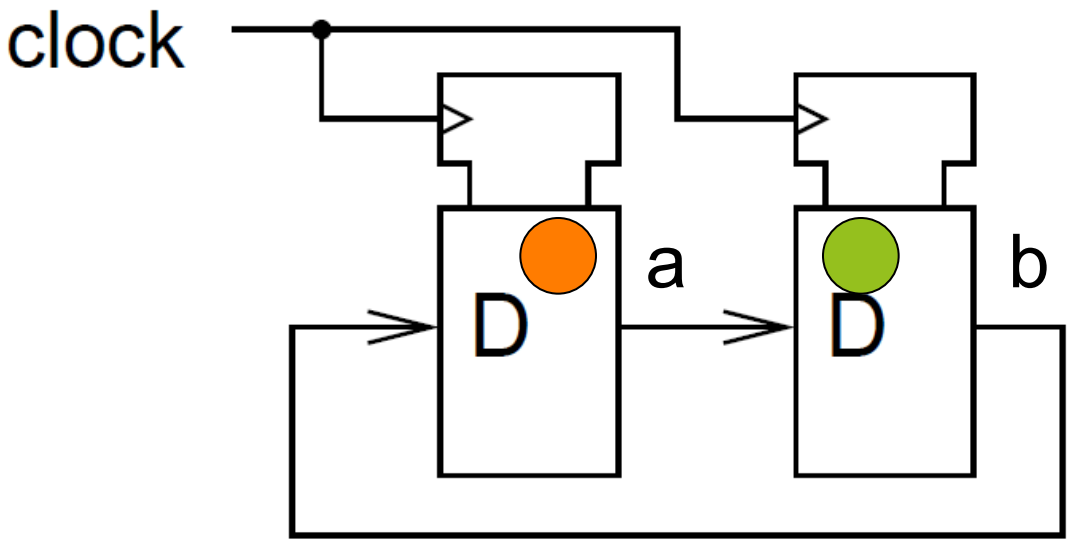
a := 0, b := 0;

Executing the right state first would assign the old value of a (=1) to a and b.

b := 1, a := 1;

The result would depend on the execution order.

Reflects model of clocked hardware

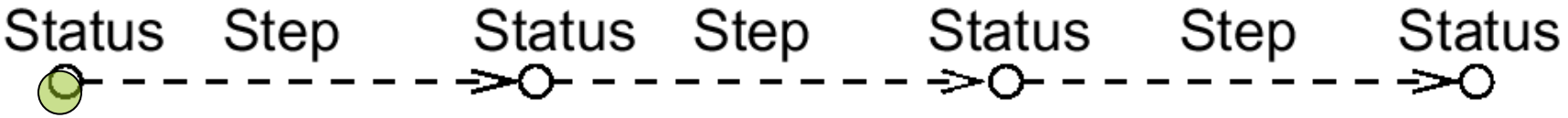


In an actual clocked (synchronous) hardware system, both registers would be swapped as well.

Same separation into phases found in other languages as well, especially those that are intended to model hardware.

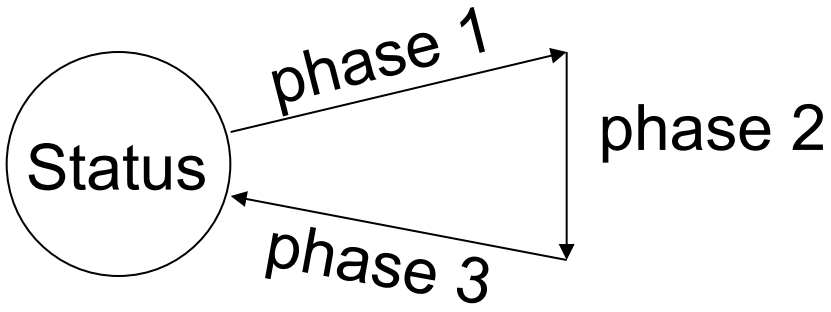
Steps

Execution of a StateMate model consists of a sequence of (status, step) pairs



Status= values of all variables + set of events + current time

Step = execution of the three phases (StateMate semantics)



Other implementations of StateCharts do not have these 3 phases (and hence could lead to different results)!

Lifetime of events

Events live until the step following the one in which they are generated (“one shot-events“).

Other semantics

Several other specification languages for hierarchical state machines (UML, dave, …) do not include the three simulation phases.

These correspond more to a SW point of view with no synchronous clocks.

Some systems allow turning the multi-phased simulation on and off.

Broadcast mechanism

Values of variables are visible to all parts of the StateChart model.

New values become effective in phase 3 of the current step and are obtained by all parts of the model in the following step.

!

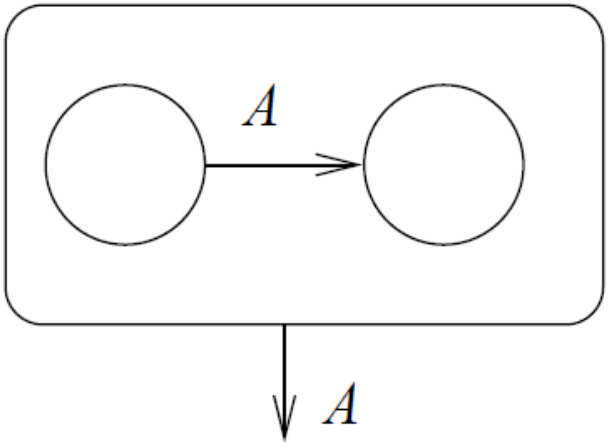
- StateCharts implicitly assumes a **broadcast** mechanism for variables
  (→ implicit **shared memory communication**
  –other implementations would be very inefficient -).
- StateCharts is appropriate for local control systems (☺), but not for distributed applications for which updating variables might take some time (☹).

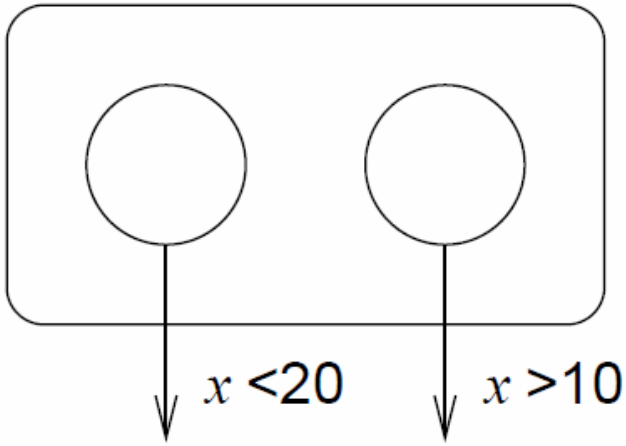
Determinate vs. deterministic

- Kahn (1974) calls a system **determinate** if we will always obtain the same result for a fixed set (and timing) of inputs
- Others call this property **deterministic**
  However, this term has several meanings:
  - Non-deterministic finite state machines
  - Non-deterministic operators
    (e.g. + with non-deterministic result in low order bits)
  - Behavior not known before run-time
    (unknown input results in non-determinism)
  - In the sense of determinate as used by Kahn

In order to avoid confusion, we use the term “determinate“ in this course.

Conflicts





Techniques for resolving these conflicts wanted

StateCharts determinate or not?

Must all simulators return the same result for a given input?

- Separation into 3 phases a required condition
- Semantics ≠ StateMate semantics may be non-determinate

Potential other sources of non-determinate behavior:

- Choice between conflicting transitions resolved arbitrarily: Tools typically issue a warning if such a situation could exist

→ **Determinate behavior for StateMate semantics if transition conflicts are resolved and no other sources of undefined behavior exist**

Evaluation of StateCharts (1)

Pros (👍):

- Hierarchy allows arbitrary nesting of AND- and OR-super states.
- (StateMate-) Semantics defined in a follow-up paper to original paper.
- Large number of commercial simulation tools available (StateMate, StateFlow, BetterState, ...)
- Available “back-ends“ translate StateCharts into SW or HW languages, thus enabling software or hardware implementations.

Backups

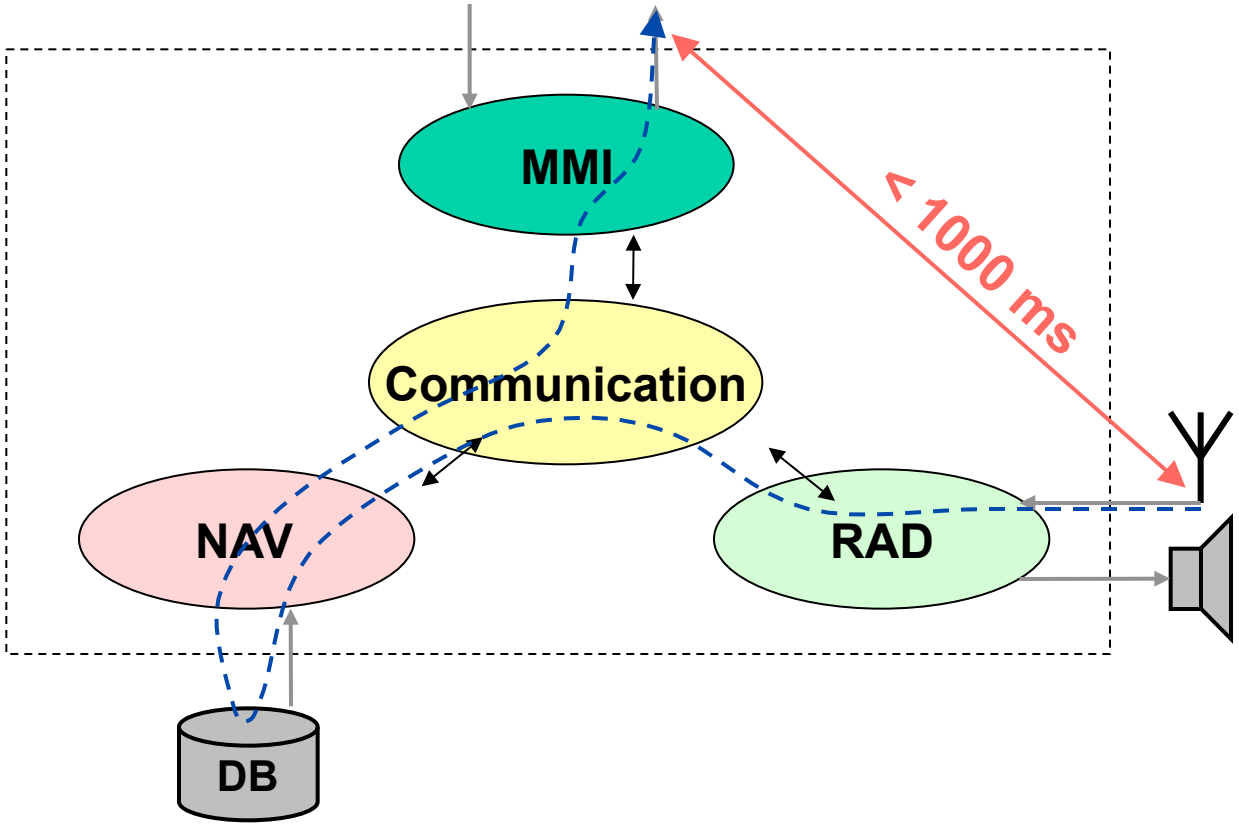
Jian-Jia Chen
(slides are based on
Peter Marwedel)
TU Dortmund,
Informatik 12

© Springer, 2010

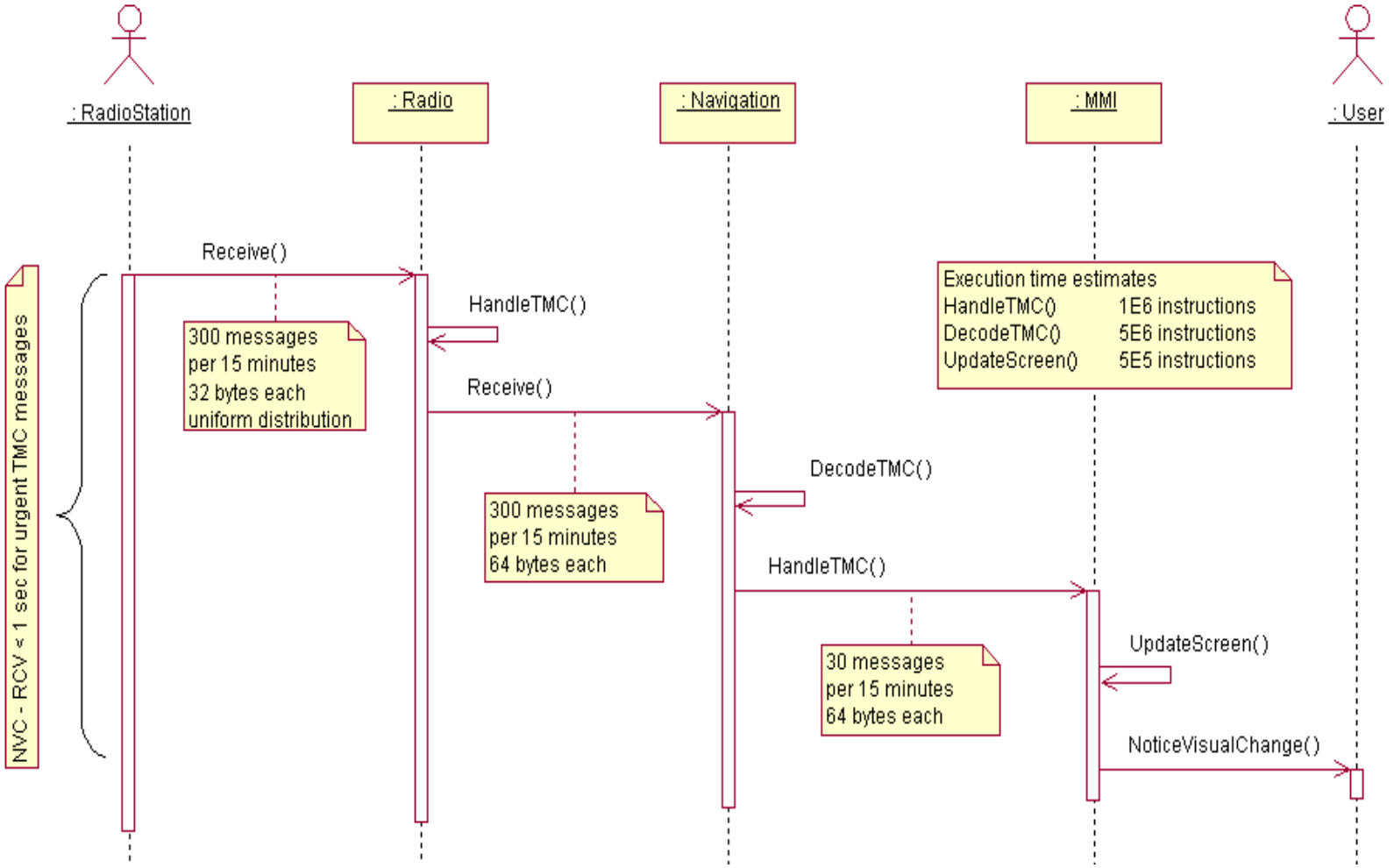
2014年 10 月 14日

These slides use Microsoft clip arts.
Microsoft copyright restrictions apply.

Use case 3: Receive TMC Messages



Use case 3: Receive TMC Messages



Life Sequence Charts* (LSCs)

Key problems observed with standard MSCs:

During the design process, MSC are initially interpreted as

"what could happen"

(existential interpretation, still allowing other behaviors).

Later, they are frequently assumed to describe

"what must happen"

(referring to what happens in the implementation).

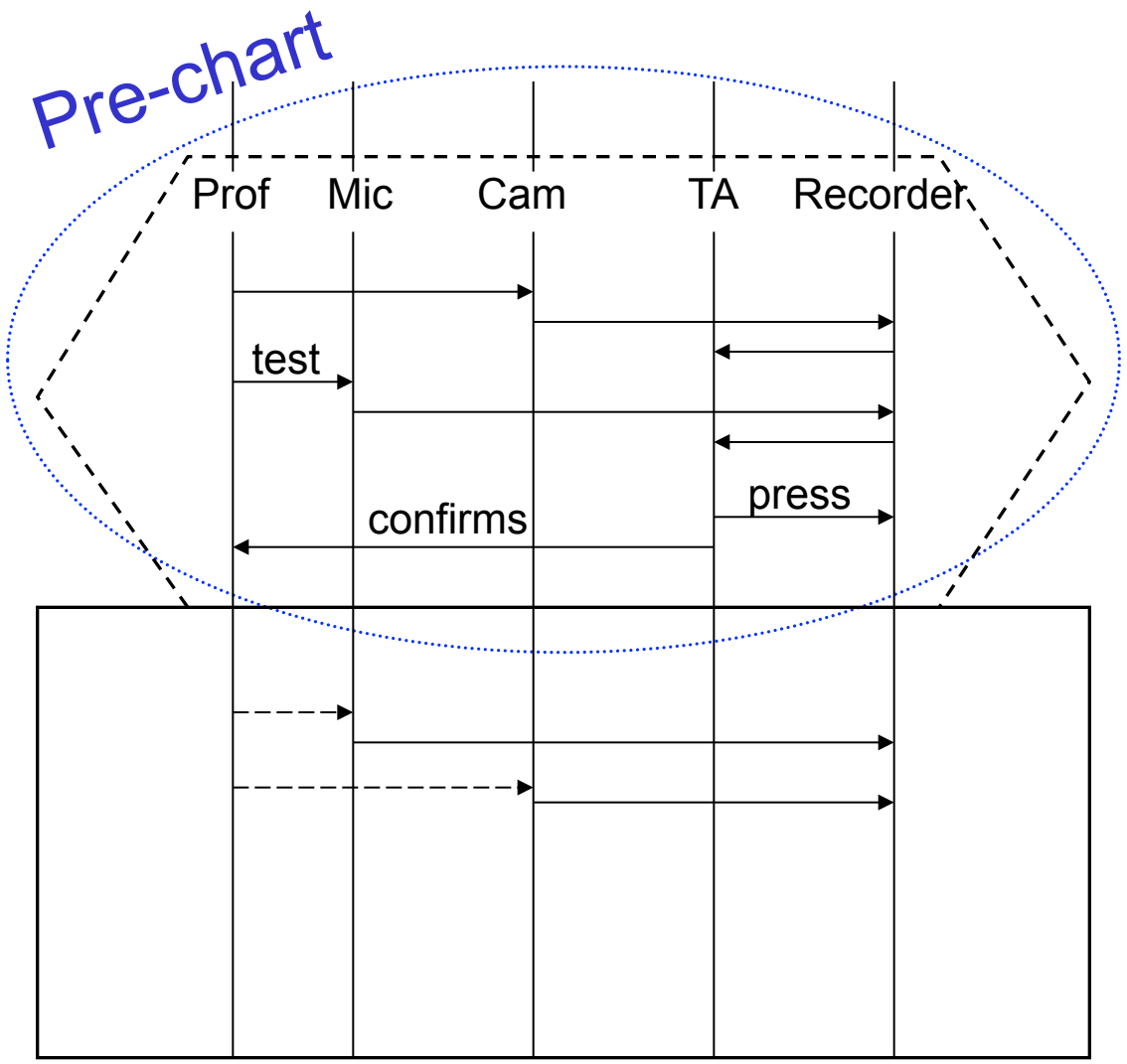
* W. Damm, D. Harel: LSCs: Breathing Life into Message Sequence Charts, *Formal Methods in System Design*, 19, 45–80, 2001

Extensions for LSCs (1)

Extension 1:

Introduction of **pre-charts:**
Pre-charts describe conditions that must hold for the main chart to apply.

Example:



Extensions (2)

Extension 2: Mandatory vs. provisional behavior

Level	Mandatory (solid lines)	Provisional (dashed lines)
Chart	All runs of the system satisfy the chart	At least one run of the system satisfies the chart
Location	Instance must move beyond location/time	Instance run need not move beyond loc/time
Message	If message is sent, it will be received	Receipt of message is not guaranteed
Condition	Condition must be met; otherwise abort	If condition is not met, exit subchart

(Message) Sequence Charts

PROs:

- Appropriate for visualizing schedules,
- Proven method for representing schedules in transportation.
- Standard defined: *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*, ITU-TS, Geneva, 1996.
- Semantics also defined: *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)—Annex* B: *Algebraic Semantics of Message Sequence Charts*, ITU-TS, Geneva.

CONS:

- describes just one case, no timing tolerances: "What *does an MSC specification mean: does it describe all behaviors of a system, or does it describe a set of sample behaviors of a system?"* *

* H. Ben-Abdallah and S. Leue, "Timing constraints in message sequence chart specifications," in *Proc. 10th International Conference on Formal Description Techniques FORTE/PSTV'97*, Chapman and Hall, 1997.