technische universität
dortmund

fakultät für informatik
informatik 12

Models of computation

Peter Marwedel
TU Dortmund
Informatik 12

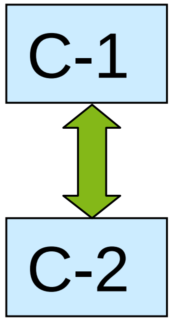
2012年 10 月 23 日

Models of computation

What does it mean, "to compute"?

Models of computation define:

- Components and an execution model for computations for each component
- Communication model for exchange of information between components.



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	(Not useful)		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	

Why not use von-Neumann (thread-based) computing (C, C++, Java, …) ?

Potential race conditions (☞inconsistent results possible)
☞ Critical sections = sections at which exclusive access to resource *r* (e.g. shared memory) must be guaranteed.

[redacted]

```
thread b {
  ..
  P(S)  //obtain lock
  ..    // critical section
  V(S)  //release lock
}
```

Race-free access to shared memory protected by S possible

This model may be supported by:

- mutual exclusion for critical sections
- special memory properties

Why not just use von-Neumann computing (C, Java, …) (2)?

Problems with von-Neumann Computing

- Thread-based multiprocessing may access global variables
- We know from the theory of operating systems that
  - access to global variables might lead to race conditions,
  - to avoid these, we need to use mutual exclusion,
  - mutual exclusion may lead to deadlocks,
  - avoiding deadlocks is possible only if we accept performance penalties.
- Other problems (need to specify total orders, …)

Consider a Simple Example

"*The Observer pattern defines a one-to-many dependency between a subject object and any number of observer objects so that when the subject object changes state, all its observer objects are notified and updated automatically*."

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns*, Addision-Wesley, 1995

Example: Observer Pattern in Java

```
public void addListener(listener) {…}

public void setValue(newvalue) {
    myvalue=newvalue;
    for (int i=0; i<mylisteners.length; i++) {
      myListeners[i].valueChanged(newvalue)
    }
}
```

Would this work in a multithreaded context?

Thanks to Mark S. Miller for the details of this example.

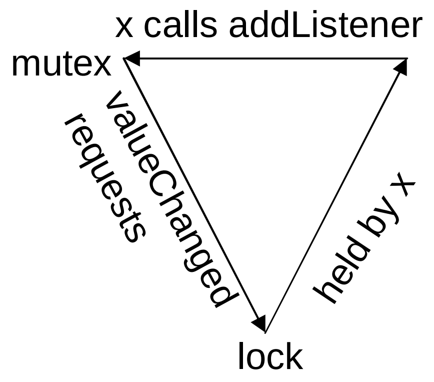
Example: Observer Pattern with Mutual Exclusion (mutexes)

```
public synchronized void addListener(listener) {…}

public synchronized void setValue(newvalue) {
    myvalue=newvalue;
    for (int i=0; i<mylisteners.length; i++) {
      myListeners[i].valueChanged(newvalue)
    }
}
```

Javasoft recommends against this.
What's wrong with it?

Mutexes using monitors are minefields

```
public synchronized void addListener(listener) {…}

public synchronized void setValue(newvalue) {
    myvalue=newvalue;
    for (int i=0; i<mylisteners.length; i++) {
      myListeners[i].valueChanged(newvalue)
    }
}
```



valueChanged() may attempt to acquire a lock on some other object and stall. If the holder of that lock calls addListener(): deadlock!

Simple Observer Pattern Becomes not so simple

```
public synchronized void addListener(listener) {…}

public void setValue(newValue) {
    synchronized (this) {
        myValue=newValue;
        listeners=myListeners.clone();
    }
    for (int i=0; i<listeners.length; i++) {
      listeners[i].valueChanged(newValue)
    }
}
```

while holding lock, make a copy of listeners to avoid race conditions

notify each listener outside of the synchronized block to avoid deadlock

This still isn't right.
What's wrong with it?

Simple Observer Pattern: How to Make it Right?

```
public synchronized void addListener(listener) {…}

public void setValue(newValue) {
    synchronized (this) {
        myValue=newValue;
        listeners=myListeners.clone();
    }
    for (int i=0; i<listeners.length; i++) {
      listeners[i].valueChanged(newValue)
    }
}
```

Suppose two threads call setValue(). One of them will set the value last, leaving that value in the object, but listeners may be notified in the opposite order. The listeners may be alerted to the value-changes in the wrong order!

Why are deadlocks possible?

We know from the theory of operating systems, that deadlocks are possible in a multi-threaded system if we have

- Mutual exclusion
- Holding resources while waiting for more
- No preemption
- Circular wait

Conditions are met for our example

A stake in the ground …

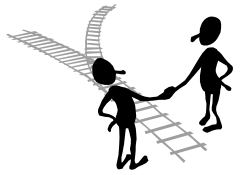
Nontrivial software written with threads, semaphores, and mutexes is incomprehensible to humans.

*"… **threads as a concurrency model are a poor match for embedded systems**. … they work well only … where best-effort scheduling policies are sufficient."*

Edward Lee: Absolutely Positively on Time, *IEEE Computer*, July, 2005

Ways out of this problem

- Looking for other options (“model-based design”)
- No model that meets all modeling requirements
- ☞ using compromises



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	(Not useful)		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	

technische universität
dortmund

fakultät für informatik
informatik 12

Early design phases

Peter Marwedel
TU Dortmund,
Informatik 12

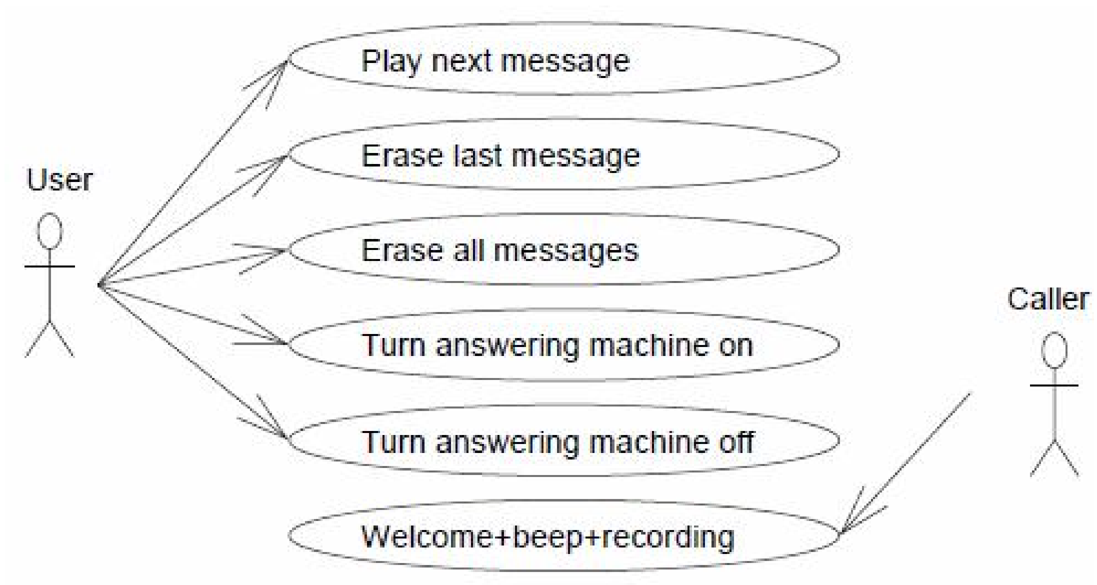
2012年 10 月 23 日

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

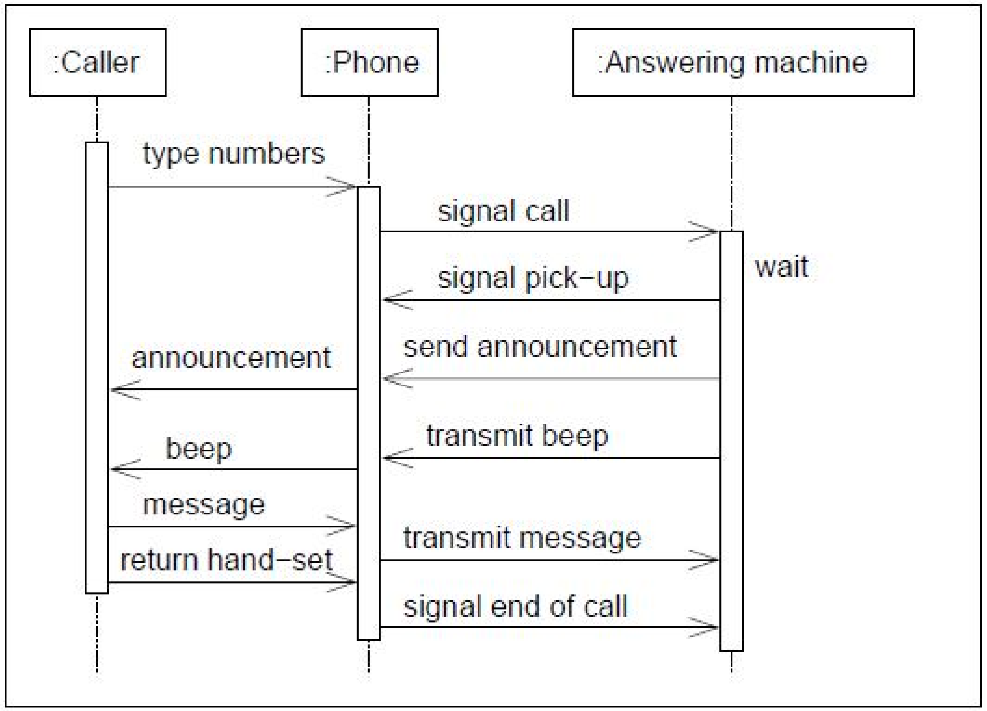


- Neither a precisely specified model of the computations nor a precisely specified model of the communication

(Message) Sequence charts

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

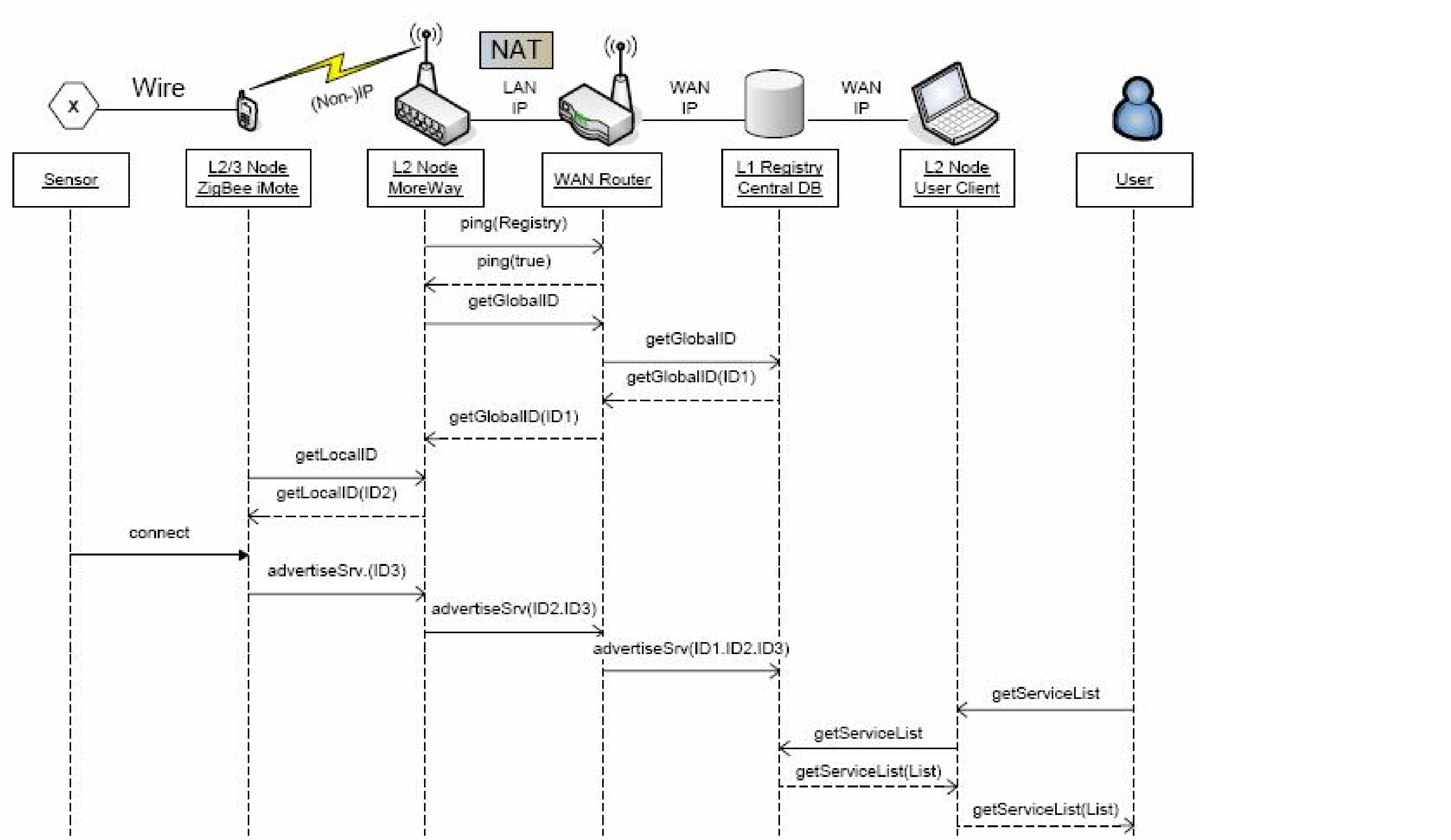
Example:



Calling an answering machine

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

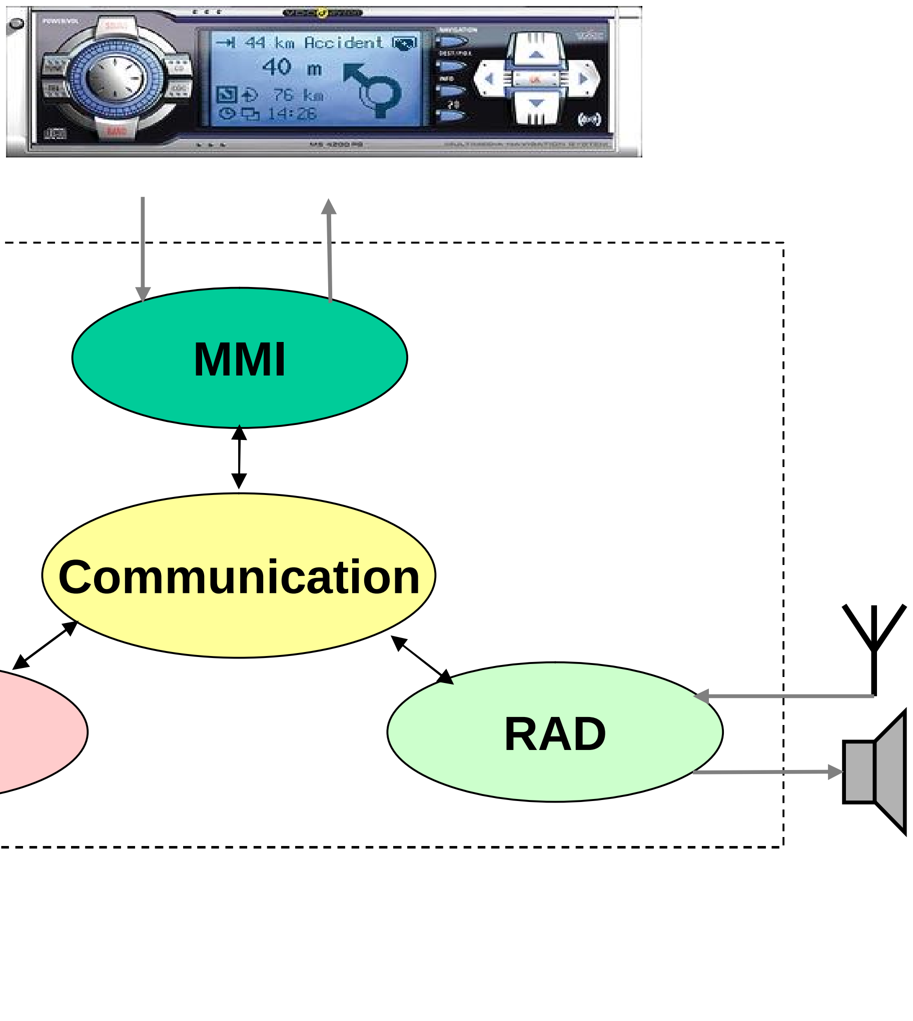
Example (2)


Wire
(Non-)IP
NAT
LAN IP
WAN IP
WAN IP
X
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)

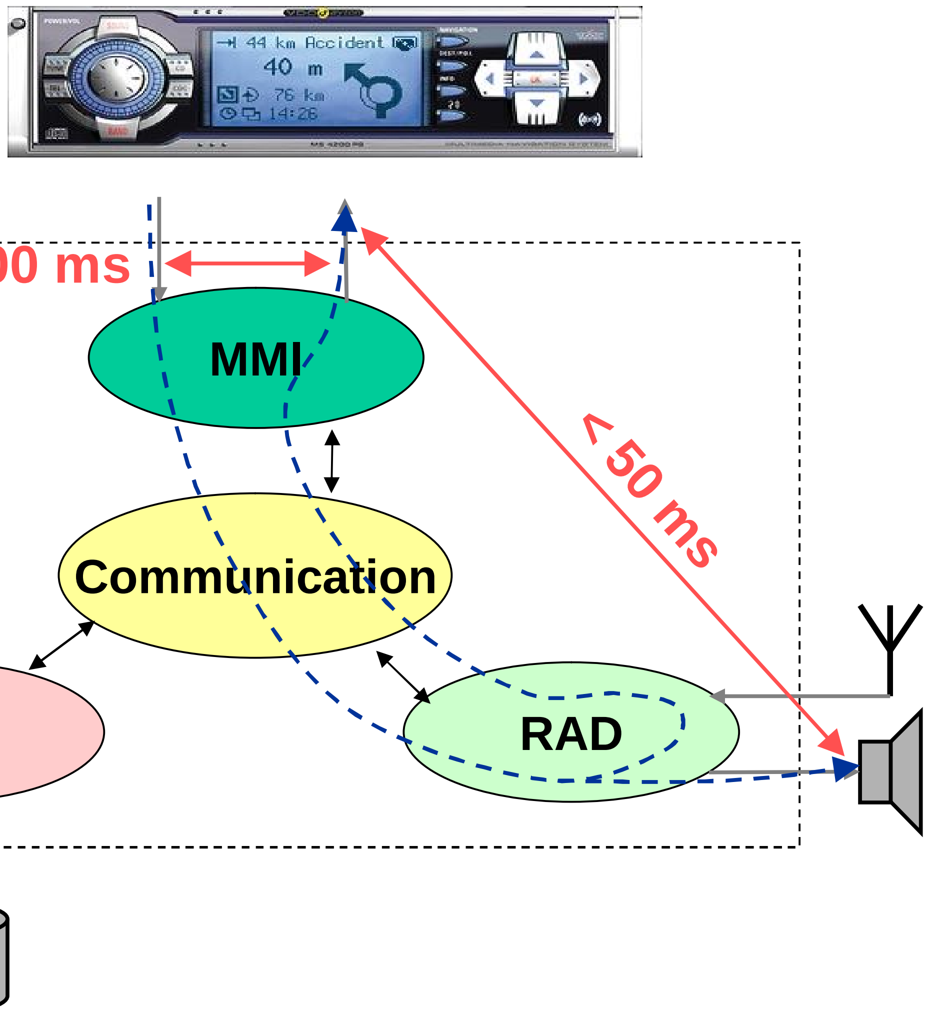
Application: In-Car Navigation System

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

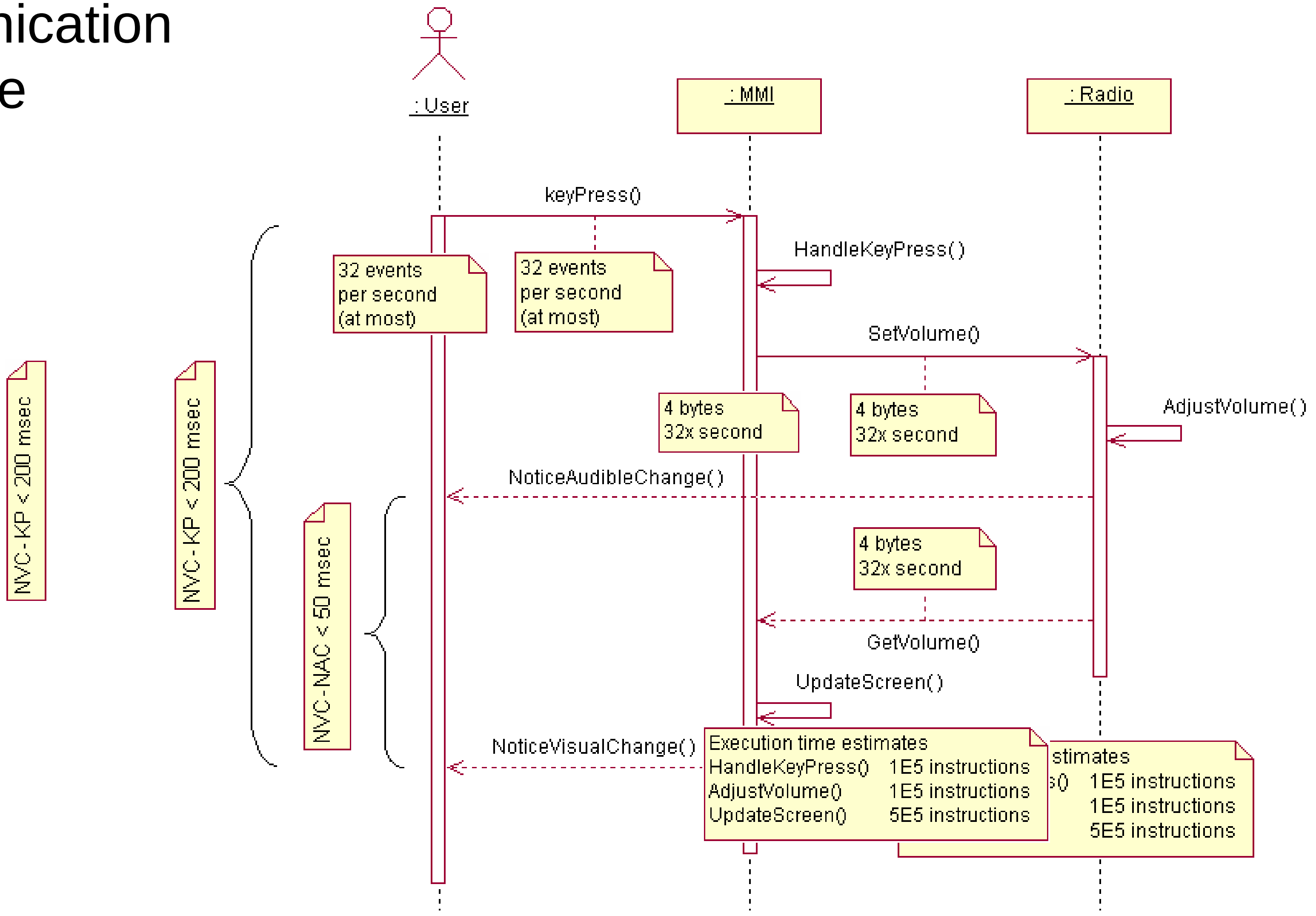


Use case 1: Change Audio Volume



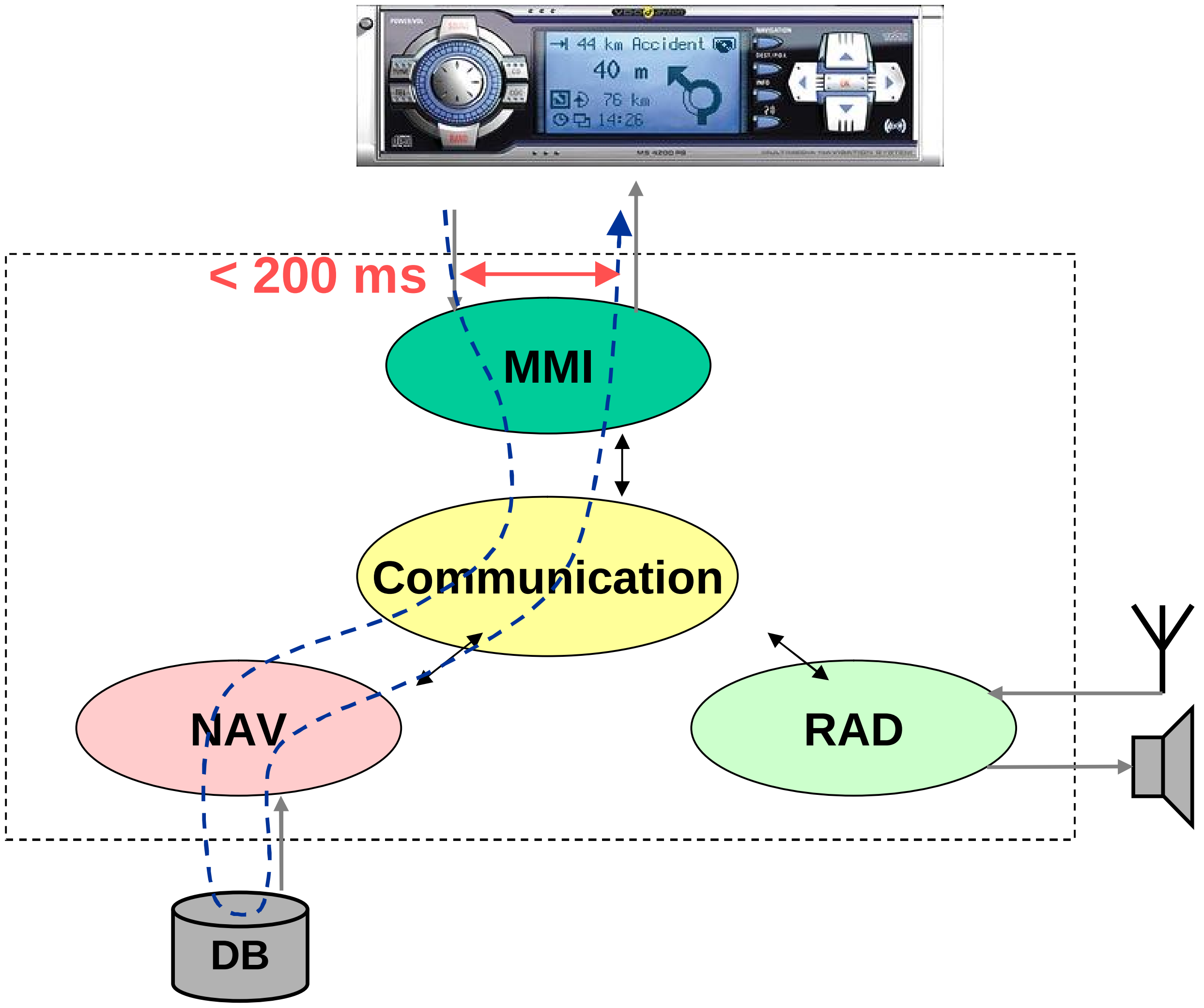
Use case 1: Change Audio Volume

Communication
Resource
Demand

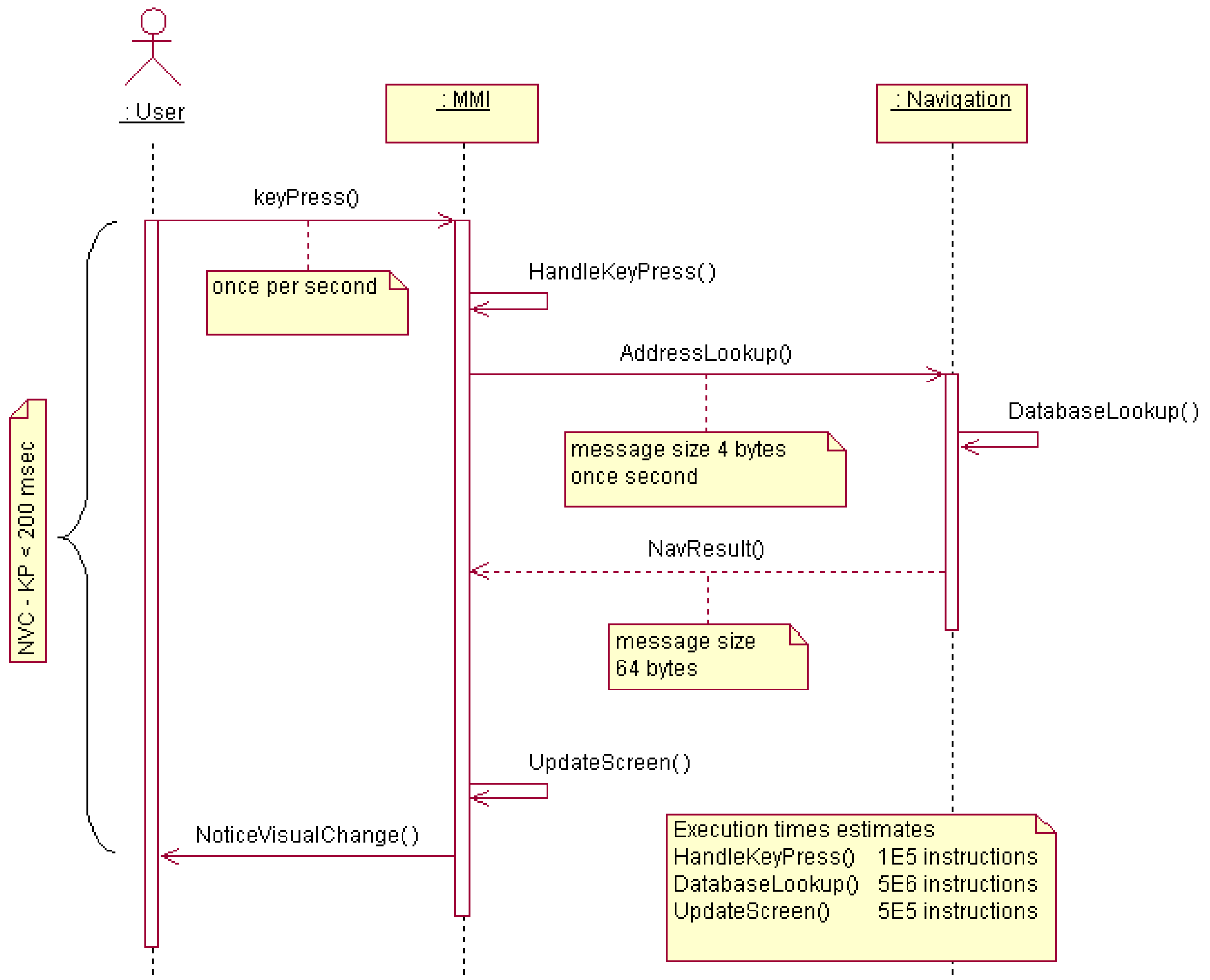


Execution time estimates
HandleKeyPress() 1E5 instructions
AdjustVolume() 1E5 instructions
UpdateScreen() 5E5 instructions

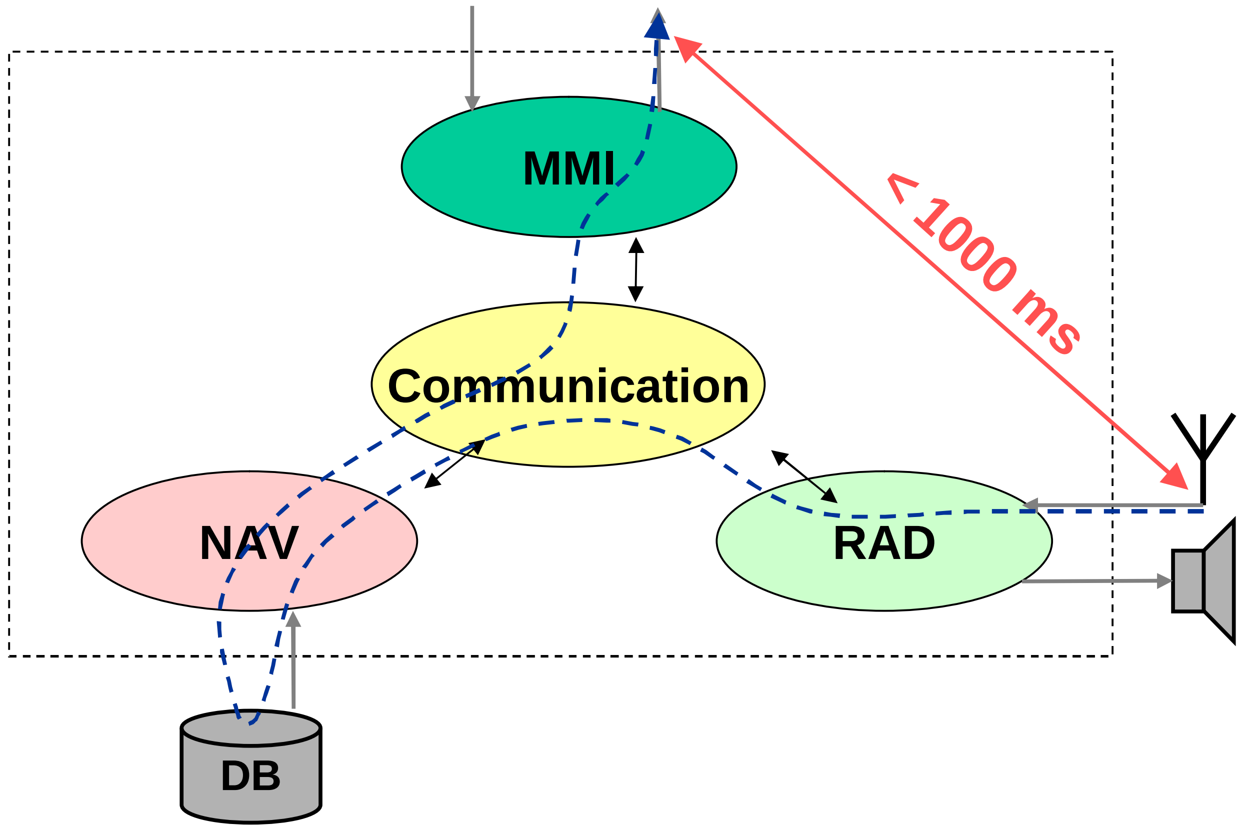
Use case 2: Lookup Destination Address



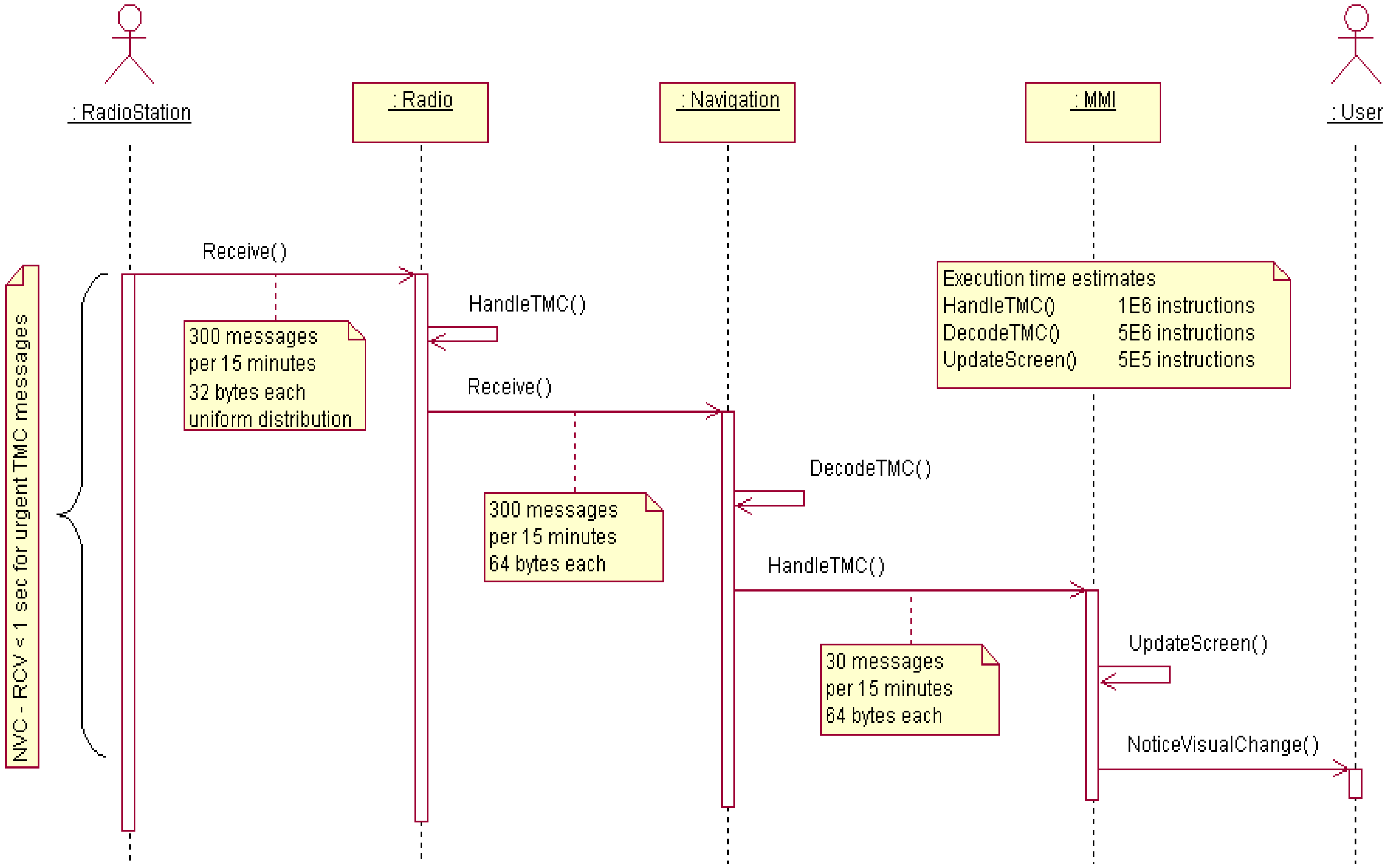
Use case 2: Lookup Destination Address



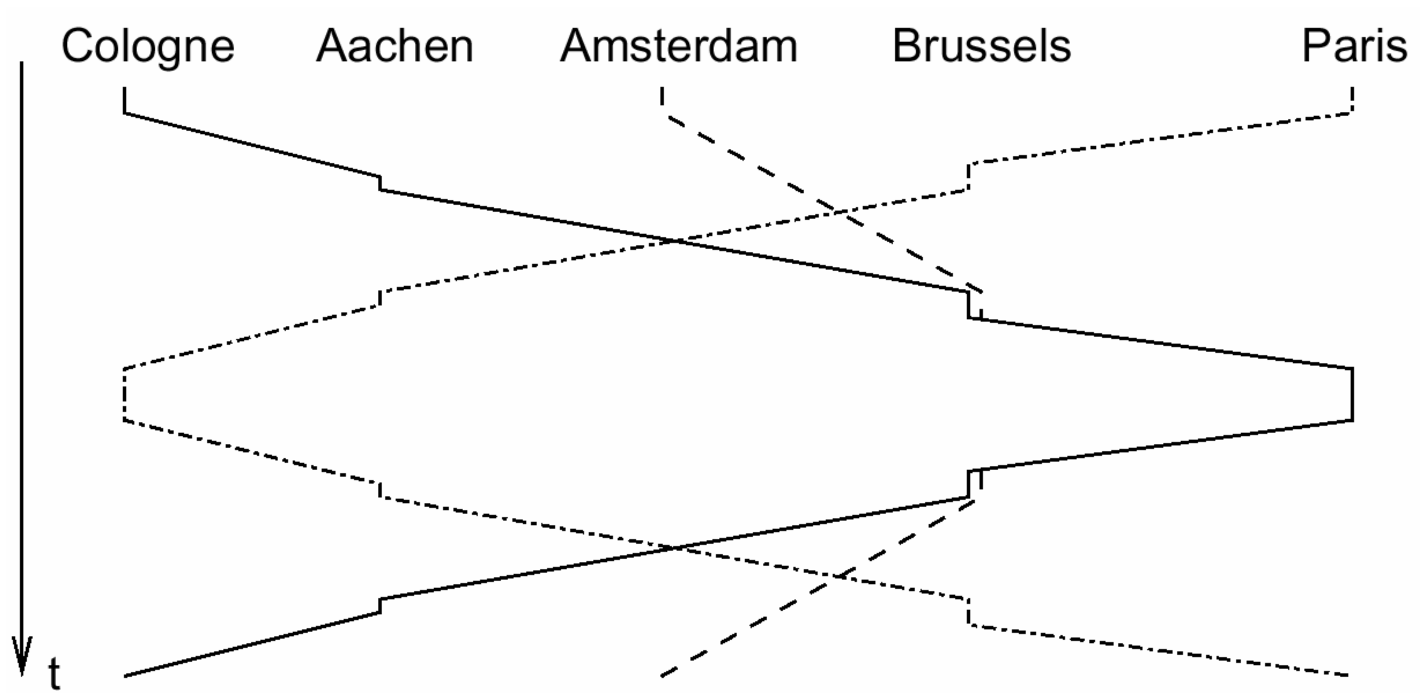
Use case 3: Receive TMC Messages



Use case 3: Receive TMC Messages

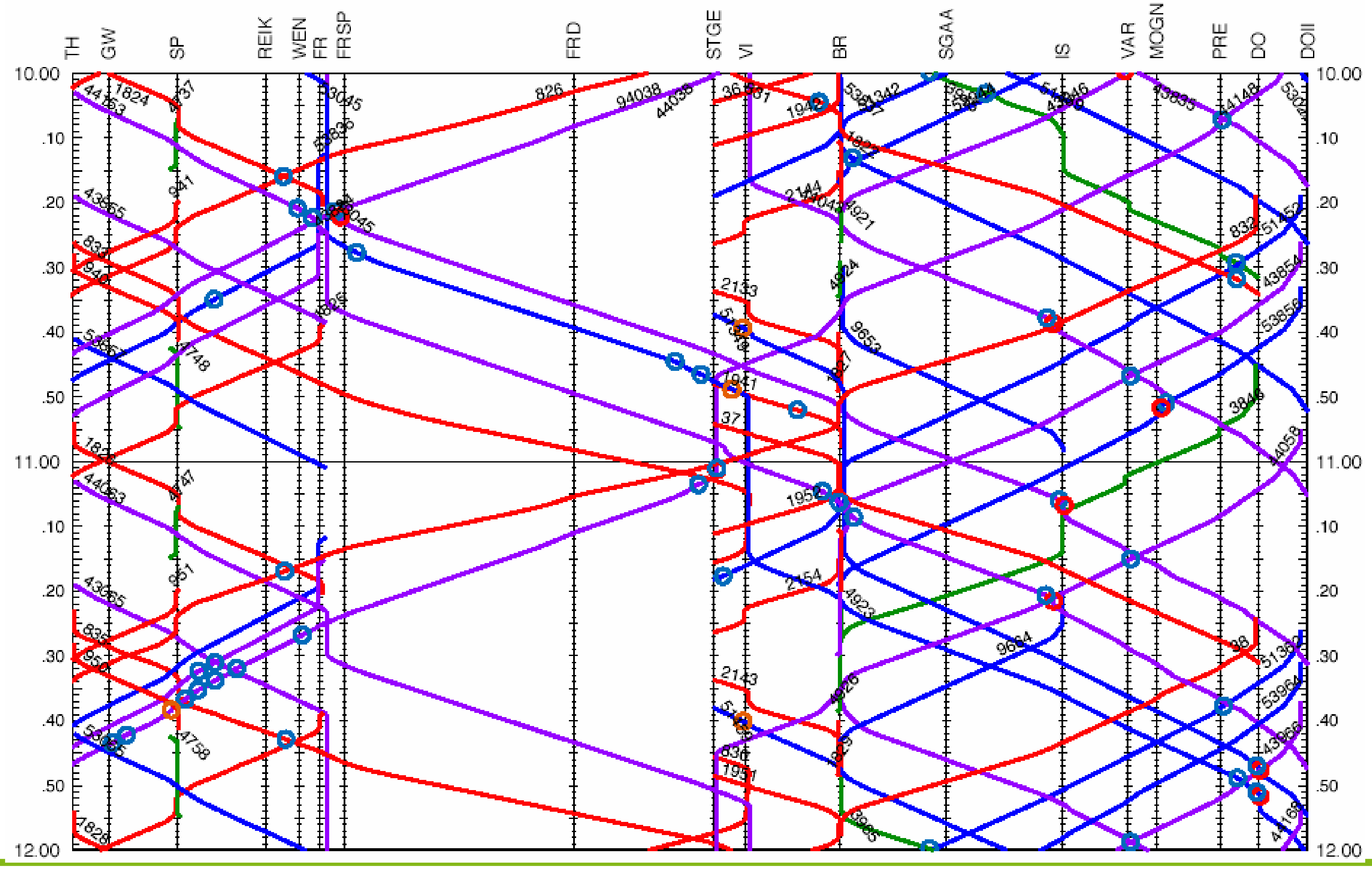


(Message) Sequence Charts (MSC)



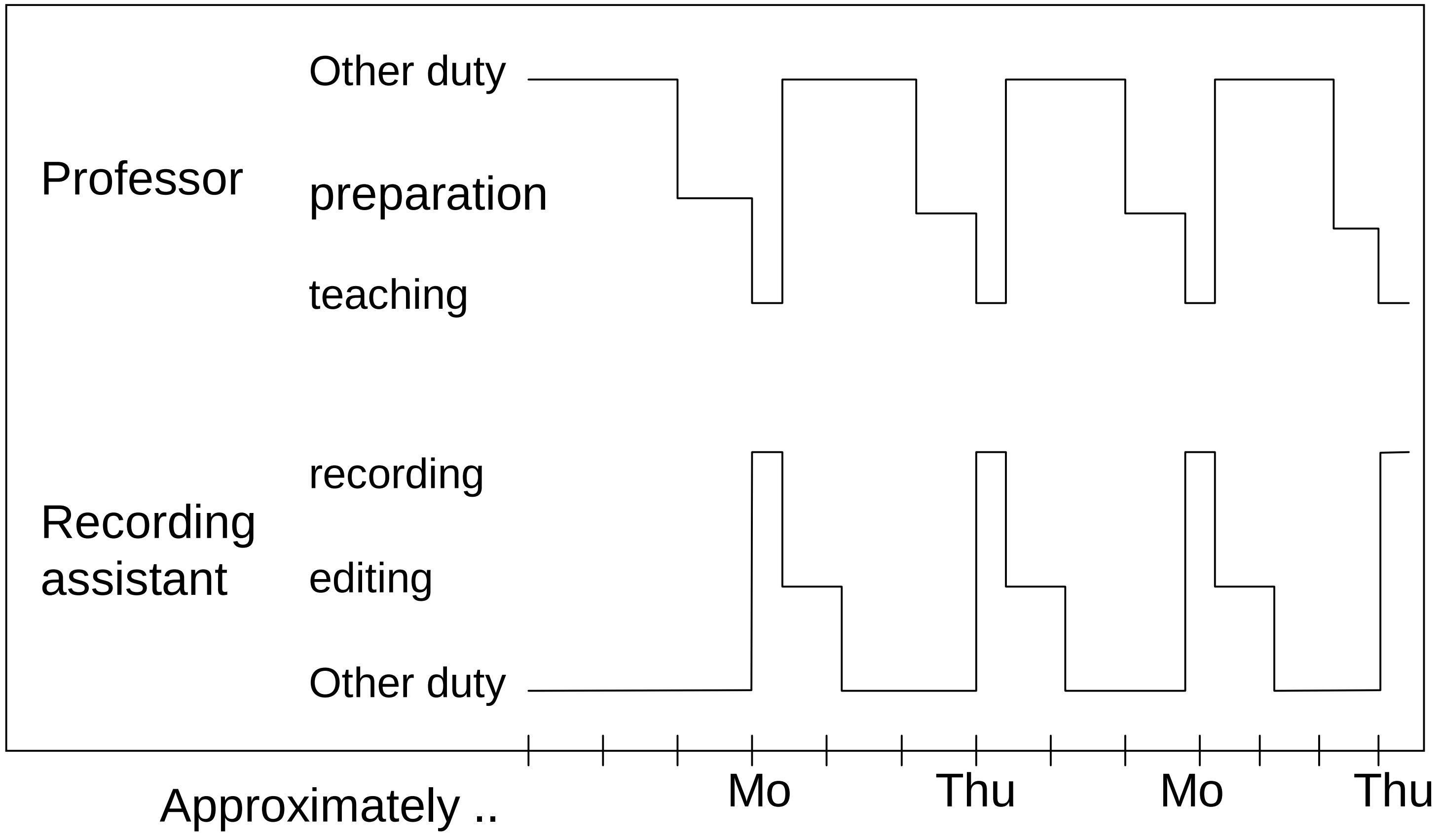
No distinction between accidental overlap and synchronization

Time/distance diagrams as a special case



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

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).

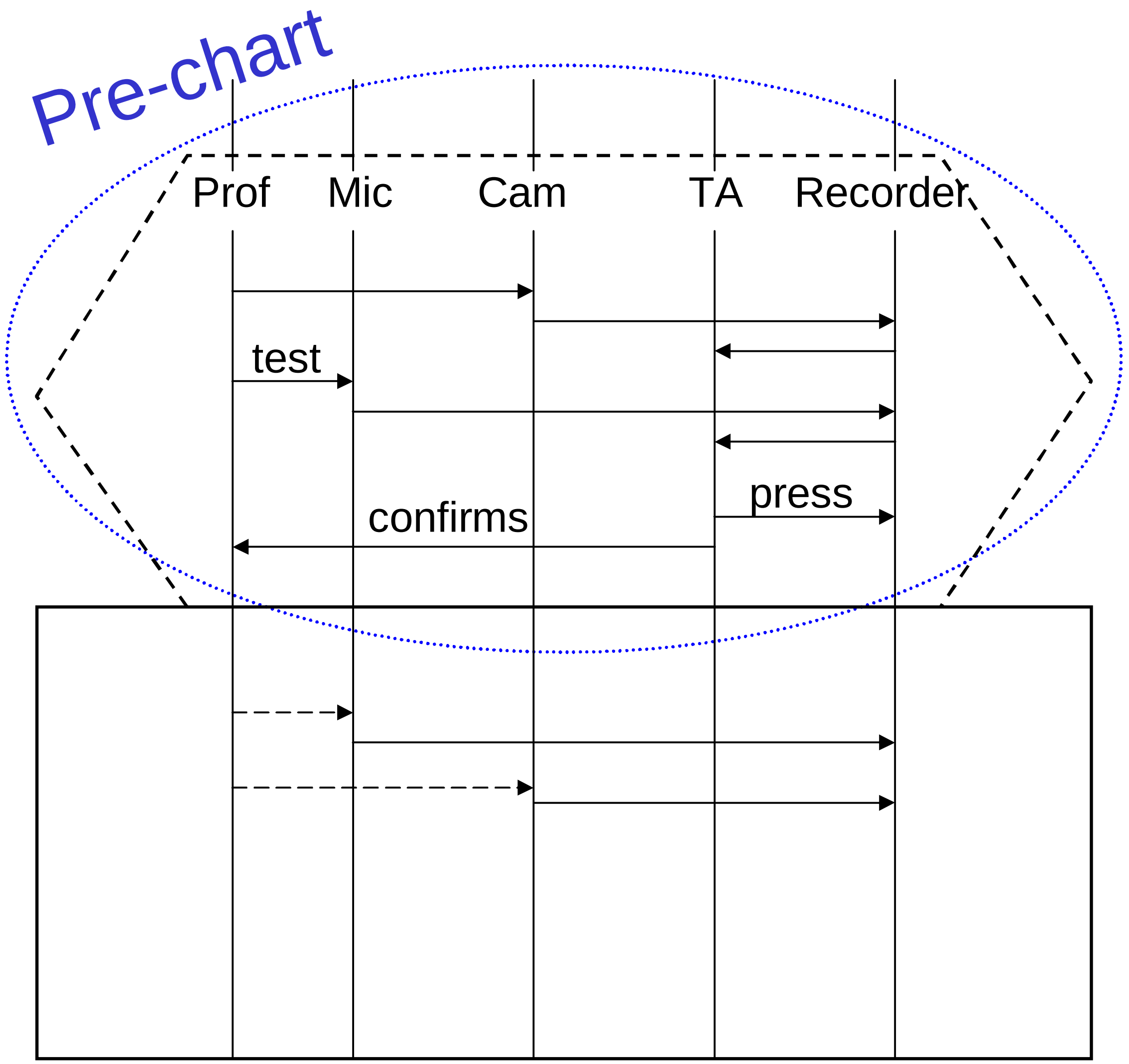
* W. Damm, D. Harel: LSCs: Breathing Life into Message Sequence Charts, *Formal Methods in System Design*, 19, 45–80, 2001

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.

technische universität
dortmund

fakultät für informatik
informatik 12

Communicating finite state machines

Peter Marwedel
TU Dortmund
Informatik 12

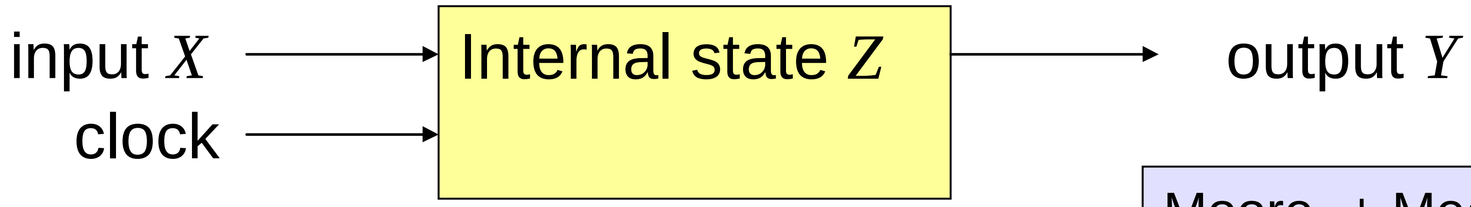
2012年 10 月 23 日

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	(Message) sequence charts
Communicating finite state machines	StateCharts		SDL
Data flow	(Not useful)		Kahn networks, SDF
Petri nets		C/E nets, P/T nets, …	C/E nets, P/T nets, …
Discrete event (DE) model	VHDL, Verilog, SystemC, …	Only experimental systems, e.g. distributed DE in Ptolemy	Only experimental systems, e.g. distributed DE in Ptolemy
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA	C, C++, Java with libraries

StateCharts: recap of classical automata

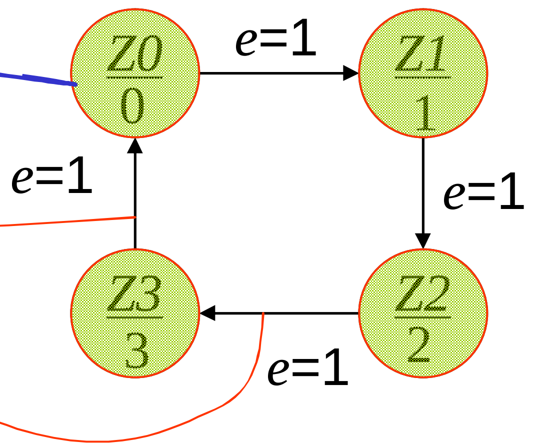
Classical automata:



Next state Z^+ computed by function δ
Output computed by function λ

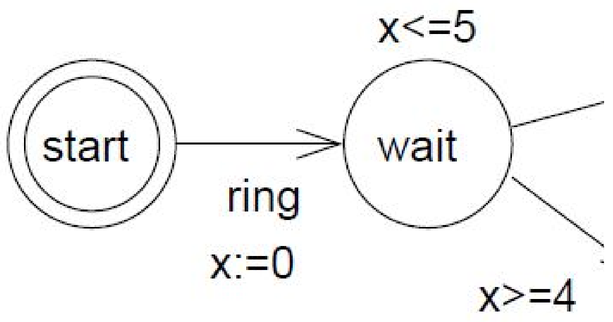
Moore- + Mealy automata=finite state machines (FSMs)

- Moore-automata:
  $Y = \lambda\ (Z); \quad Z^+ = \delta\ (X, Z)$
- Mealy-automata
  $Y = \lambda\ (X, Z); \quad Z^+ = \delta\ (X, Z)$



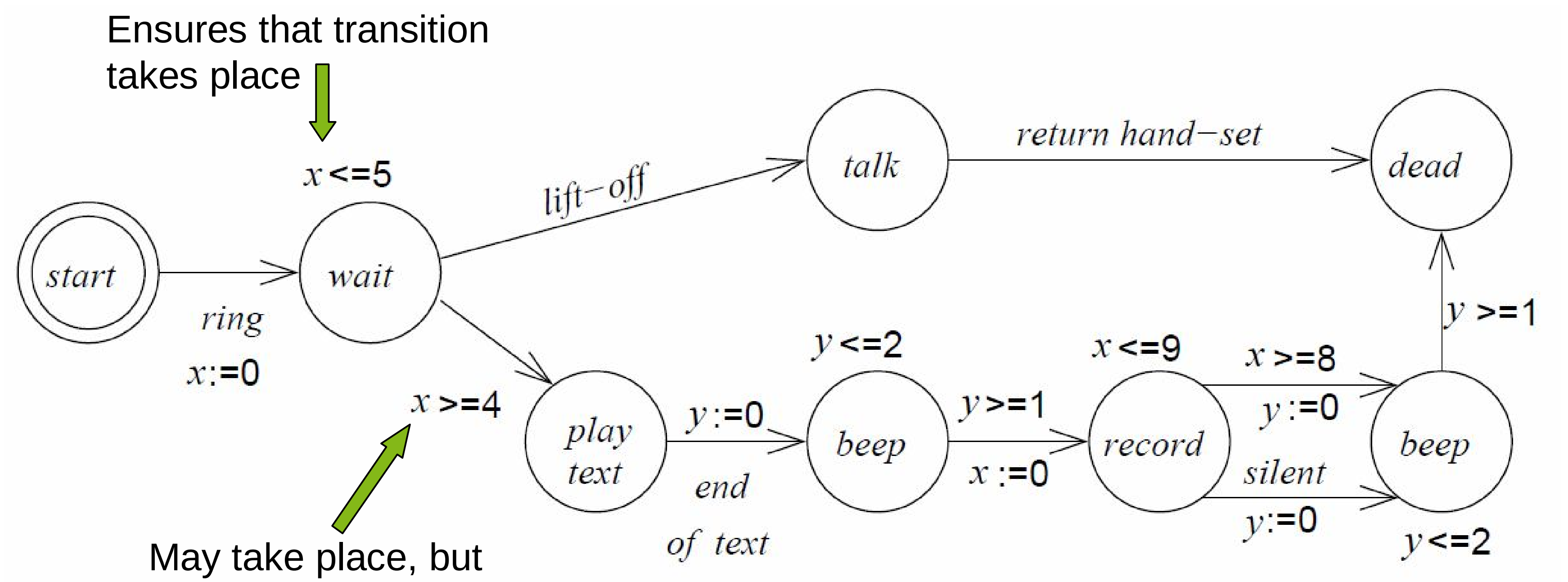
Timed automata

- Timed automata = automata + models of time
- *The variables model the logical clocks in the system, that are initialized with zero when the system is started, and then increase synchronously with the same rate.*
- *Clock constraints i.e. guards on edges are used to restrict the behavior of the automaton.*
  A transition represented by an edge can be taken when the clocks values satisfy the guard labeled on the edge.
- Additional invariants make sure, the transition is taken.
- *Clocks may be reset to zero when a transition is taken* [Bengtsson and Yi, 2004].



Example: Answering machine

Ensures that transition takes place



May take place, but does not have to

Definitions

Let C: real-valued variables C representing clocks.

Let Σ: finite alphabet of possible inputs.

Definition: A **clock constraint** is a conjunctive formula of atomic constraints of the form
$x \circ n$ or $x-y \circ n$ for $x, y \in C$, $\circ \in \{\leq,<,=,>,\geq\}$ and $n \in N$

Let $B(C)$ be the set of clock constraints.

Definition: A **timed automaton** A is a tuple (S, s_0, E, I) where

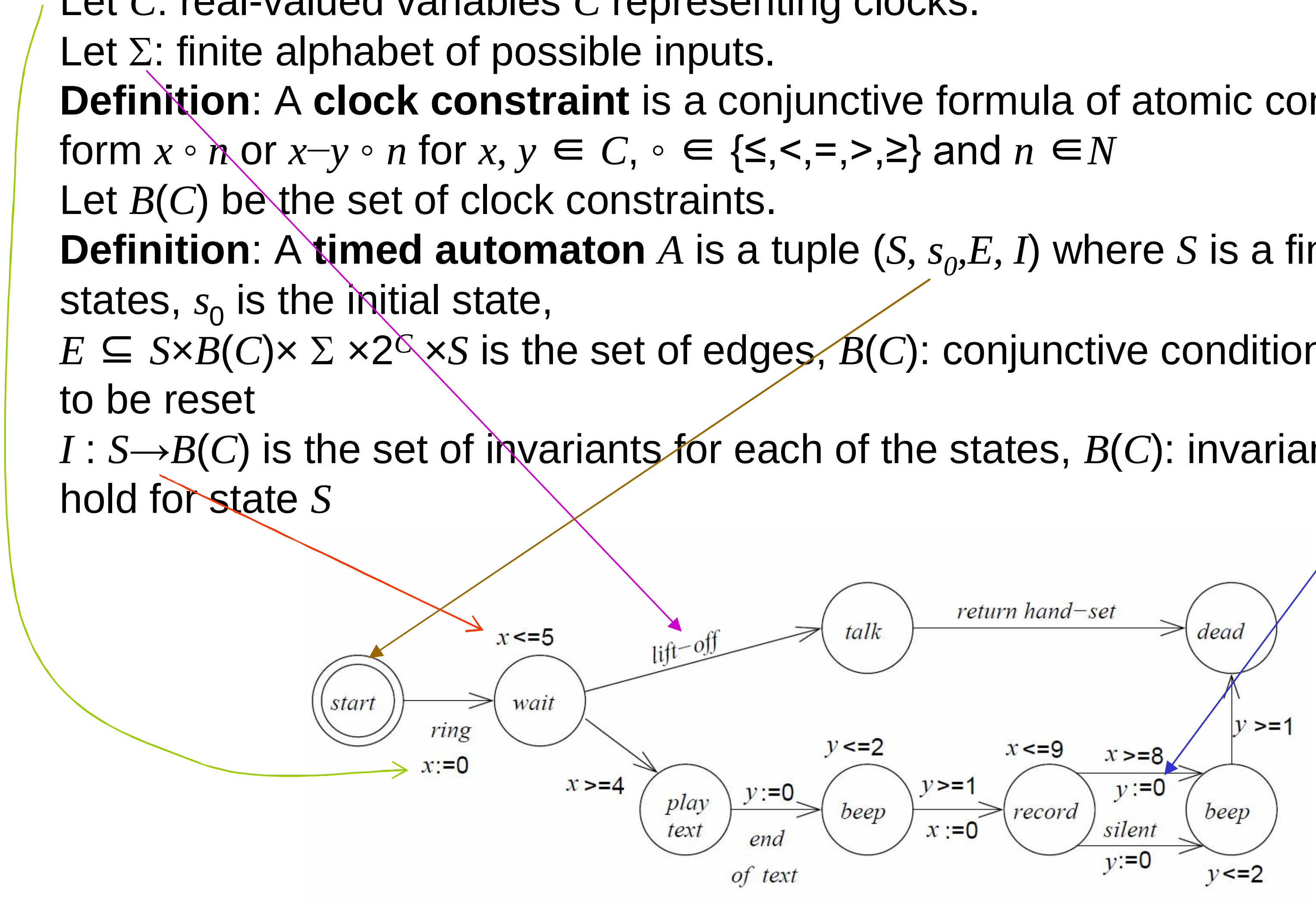
S is a finite set of states, s_0 is the initial state,

$E \subseteq S \times B(C) \times \Sigma \times 2^C \times S$ is the set of edges,
$B(C)$: conjunctive condition, 2^C: variables to be reset

$I : S \rightarrow B(C)$ is the set of invariants for each of the states
$B(C)$: invariant that must hold for state S

Definitions (2)

Let C: real-valued variables C representing clocks.
Let Σ: finite alphabet of possible inputs.
Definition: A **clock constraint** is a conjunctive formula of atomic constraints of the form $x \circ n$ or $x-y \circ n$ for $x, y \in C$, $\circ \in \{\leq,<,=,>,\geq\}$ and $n \in N$
Let $B(C)$ be the set of clock constraints.
Definition: A **timed automaton** A is a tuple (S, s_0, E, I) where S is a finite set of states, s_0 is the initial state,
$E \subseteq S \times B(C) \times \Sigma \times 2^C \times S$ is the set of edges, $B(C)$: conjunctive condition, 2^C: variables to be reset
$I : S \rightarrow B(C)$ is the set of invariants for each of the states, $B(C)$: invariant that must hold for state S



Summary

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