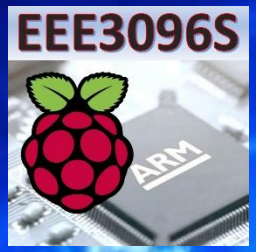
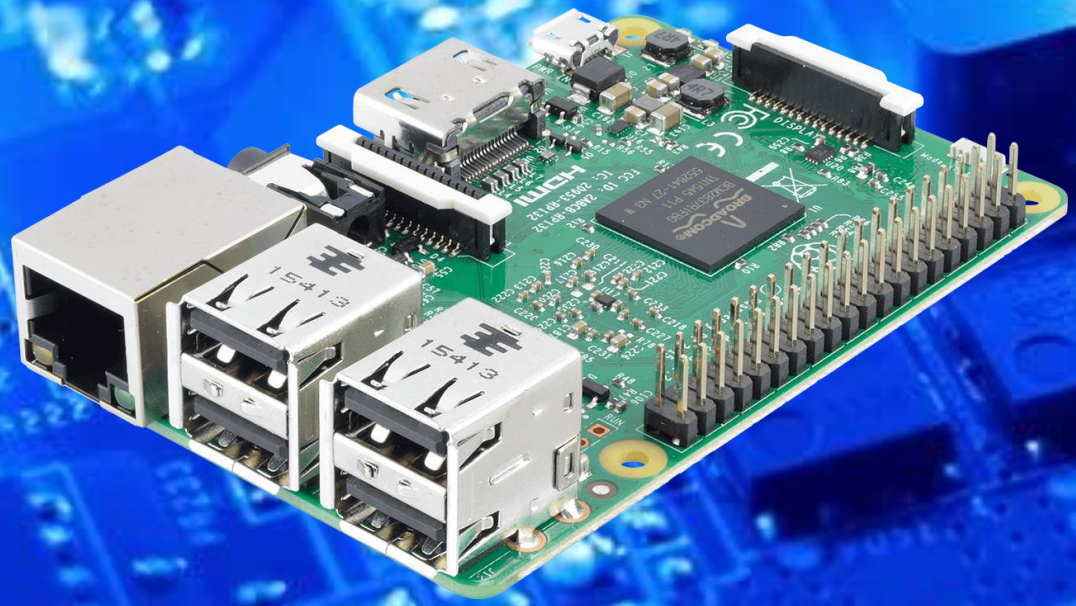




Specifications and Modeling (2)

Section 2.3 of textbook relates

L6

Embedded Systems II

Dr Simon Winberg

Electrical Engineering
University of Cape Town

Outline of Lecture

- Programming Considerations
  - Critical sections
  - Von Neumann problems of parallel operation
  - Observer Pattern
  - Threads – to be use wisely if needed
- Modelling
  - (Message) Sequence Charts (MSC)
  - Timing Distance Diagrams (TDD)
  - UML Timing Diagram

Programming

Threads

Critical Section

Observer Pattern

Wise use of Threads

What is a Thread?

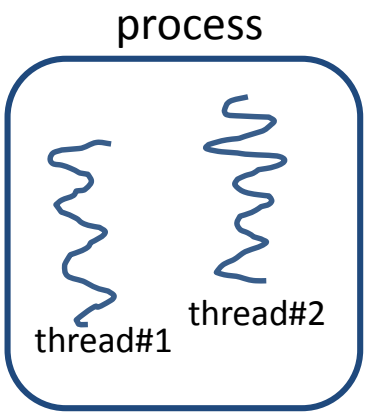
Thread of execution = the smallest sequence of programmed instructions that can be managed independently by a scheduler

Generally, a thread is a component of a process. Multiple threads can exist within one process, and they may run concurrently.

(technically, thread ≠ task… a task could be implemented by one of more threads)

A task is a “basic unit of programming”. This unit of programming may be an entire program. It is often consider to be a ‘body of work’ with a related purpose. One program may use other utility programs, the utility programs may also be considered tasks (or subtasks).

A task may correspond to a use case (e.g. banking use cases or tasks could be: check balance, make deposit, do transfer, etc.)

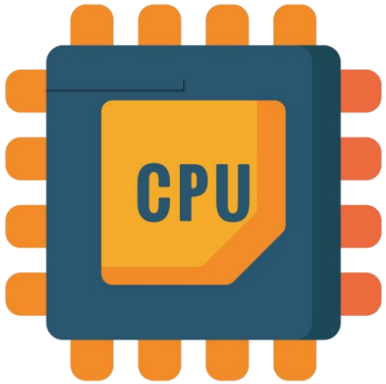


Thread 1 and 2 might run concurrently depending on the design and loading of the CPU

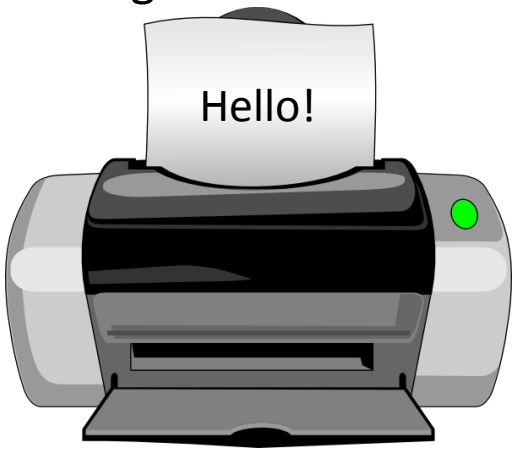
Critical Section

Consider this example of outputting to a printer…

A critical section of code is one in which exclusive access needs to be given to a device, such as a output port controlling a device.



t1
output port



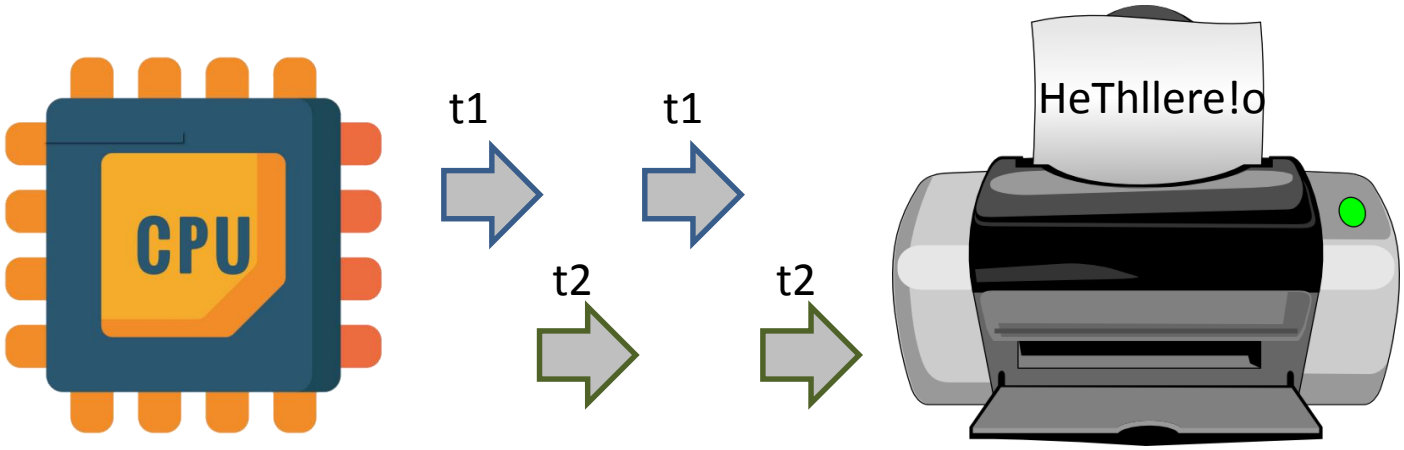
Thread 1:

```
void printtext (char* t1)
{
  int i=0;
  while (t1[i]!=0)
    outp(PRINT,t1[i++]);
}
printtext (“Hello!”);
```

Critical section

BUT this code example here is not giving exclusive access!

Unprotected Critical Section



Thread 1:

```
void printtext (char* t1)
{
  int i=0;
  while (t1[i]!=0)
     outp(PRINT,t1[i++]);
}

printtext (“Hello”);
```

Thread 2:

```
void printtext2 (char* t1)
{
  int i=0;
  while (t1[i]!=0)
     outp(PRINT,t1[i++]);
}

printtext (“There!”);
```

So this is what could happen if you have not protected your critical section…

This has caused a ‘race condition’, which one needs to prevent

Dangers of von-Neumann (thread-based) computing (C, C++, Java, …) ?

Potential race conditions (☞ potential inconsistent results)

☞ Critical sections = code sections where exclusive access to a resource *r* (e.g. shared memory) must be guaranteed.

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

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

Race-free access to shared memory protected by semaphore (S) possible

This model may be supported by:

- mutual exclusion for critical sections
- special memory properties

Example critical section: writing output, e.g. sending control sequence to a peripheral. Tight control of device.

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

Problems with von-Neumann (sequential) Computing

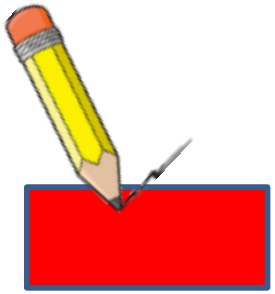
- Thread-based multiprocessing may access global variables
- We know from the theory of operating systems that
  - access to global variables might lead to race conditions,
  - to avoid these, we need to use mutual exclusion,
  - mutual exclusion may lead to deadlocks,
  - avoiding deadlocks is possible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

A race condition: two or more threads to change shared memory at the same time. i.e. both threads are "racing" to access/change the data and it may be non-determinate behaviour. Embedded Systems designers don't like non-determinate behaviour!

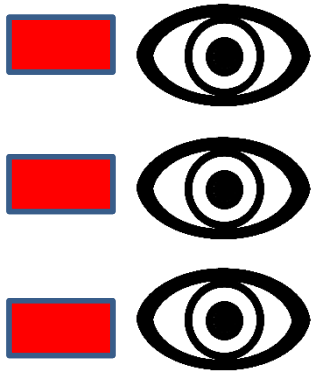
Consider a Simple Example

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

This helps to Explain: …



subject object



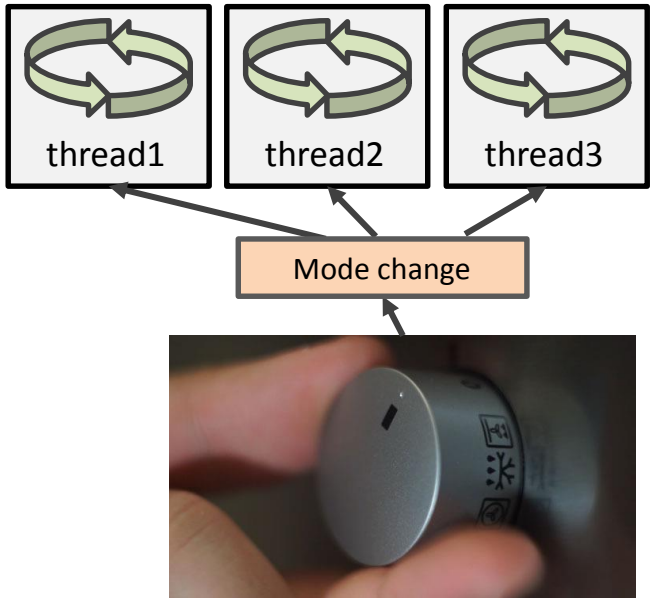
observer objects

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

Observer Pattern in Embedded Systems

- The observer pattern is found in distributed event handling systems, or "event driven" software.
- This approach is often needed in threaded embedded systems applications

```
public void irq_knob () {
   int x=readinput();
   for (int i=0; i<listeners.length; i++) {
     listeners[i].notify(x)
   }
```



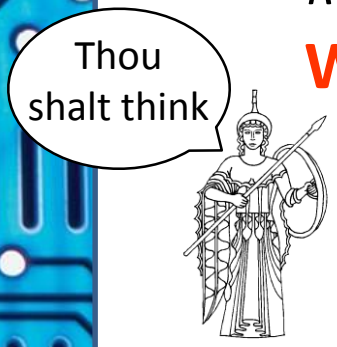
Example: Observer Pattern in Java

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

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

Reliability is one of the crucial design constraints for an ES

Would this be reliable in a multithreaded context?



No, it would not be reliable. A thread could utilize its myvalue variable before this forloop completes. You need a way to ensure other threads don't try to use the value while this operation is busy.

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.

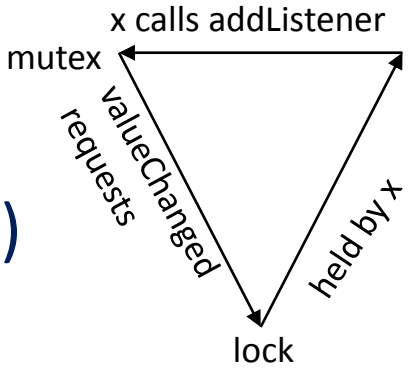
What's wrong with it?

Answer …

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: 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
- Listeners may be notified in the opposite order. The listeners may be alerted to the value-changes in the wrong order! (i.e. old value first)

Functions with the **synchronized** keyword can only have one instance invoked at once, a synchronized function can't start until any other instance has ended.

Simple Observer Pattern: How to Make it Right? Better Result

General psudocode solution:

Interrupt:

- Que.enque(input)
- Notify

Ensures the input is stored in the right order.

Notify handler:

- If (busy) then wait();
- S(busy = 1)
- Add any new listeners
- x = Que.deque();
- For i=1 to len(listeners)
  - listener.update(x);
- busy = 0

By wait() one could assume come back later

Technically need the if to have an atomic (!busy?busy=1) operation

Ensures the input is handled in the right order. Note that it is fine if more inputs are generated at any stage in this function.

Any listeners to be added should not be done while distributing input to listeners

Simple Observer Pattern: More appropriate result for ES…

General psudocode solution:

Interrupt:

- latest_input=input
- Notify

Notify handler:

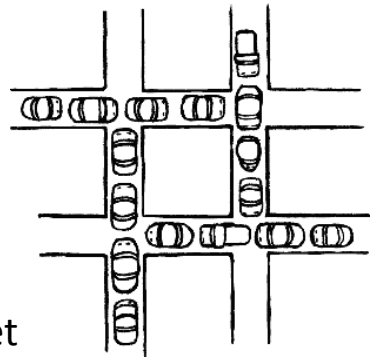
- If (busy) then wait();
- S(busy = 1)
- Add any new listeners
- x = latest_input;
- For i=1 to len(listeners)
  - listener.update(x);
- busy = 0

In some applications there is no use in storing an input history, the system needs to respond to the latest input; it may be perfectly safe throwing intermediate results away. For example consider a oven control dial, you may turn the knob and pass various functions you don't want to activate.

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

(Deadlock can sometimes happen even with preemption where tasks get stuck swapping resources, never holding all the ones they need at once)



traffic deadlock

A stake in the ground …

General thoughts on threads for embedded systems…

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

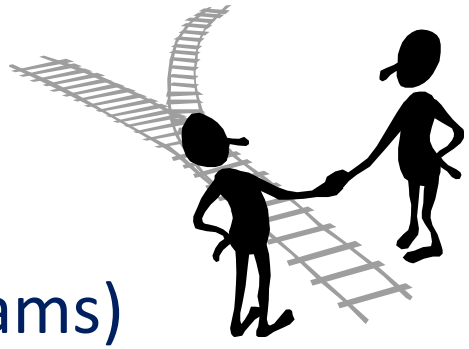
*"… **threads as a concurrency model are a poor match for embedded systems**. … they work well only … where best-effort scheduling policies are sufficient." **

General advice: avoid them if you don't need them!!

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

Ways out of thread problems

- Looking for other options ("model-based design"), to get a clear view of what your program is going
- No model that meets all modelling requirements
- → use compromises
  (e.g. structured programming vs. object-oriented event driven programs)



There is much motivation for using a plain simple structured programming approach to develop embedded software (e.g. [1]), or using protothreads [2] as a compromise.

[1] Kamal, Raj. *Embedded systems: architecture, programming and design*. Tata McGraw-Hill Education, 2011.

[2] Dunkels, Adam, et al. "Protothreads: Simplifying event-driven programming of memory-constrained embedded systems." Proceedings of the 4th international conference on Embedded networked sensor systems. Acm, 2006.

Early Design Phase

A bit more UML

→ Sequence Charts

Thoughts

- Why do systems and their software need to be designed?
- Can't we just start writing software and design as we go along?
- What do we want to achieve in the design process?

(hint) you might just happen to see this kind of a question in a test…

Models of computation in this course

Communication/ local computations	**Shared memory**	**Message passing** Synchronous | Asynchronous	
Undefined components	Plain text ✓, use cases ✓ | **(Message) Sequence Charts**, ICD		
Communicating finite state machines	StateCharts		SDL
Data flow	Scoreboarding + Tomasulo Algorithm (☞ Comp.Archict.)		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 (Ptolemy only discussed briefly)	
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |	

* Classification based on implementation with centralized data structures
SystemC will not be delved into detail. Only brief flavour of VHDL and Verilog given

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.
- But even from this stage it is good to bring in tools to help manage this information

Expectations for tools and requirements:

Machine-readable (even if essays / notes)

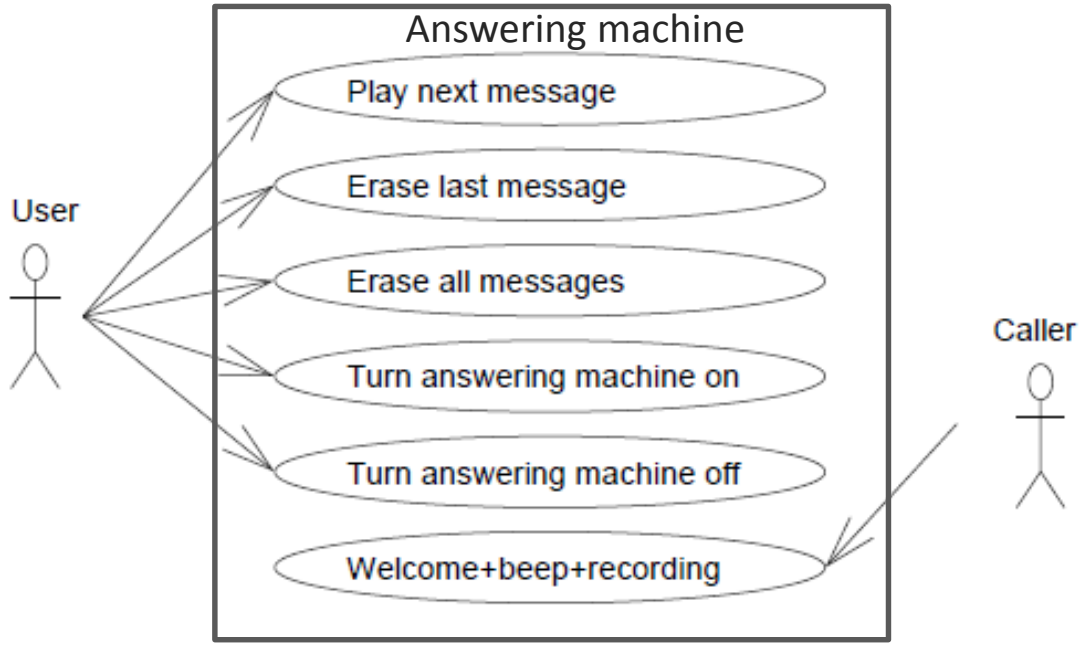
Version management

Dependency analysis (which features depend on others)

Example: DOORS® [Telelogic/IBM]

Use cases (brief reminder)

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



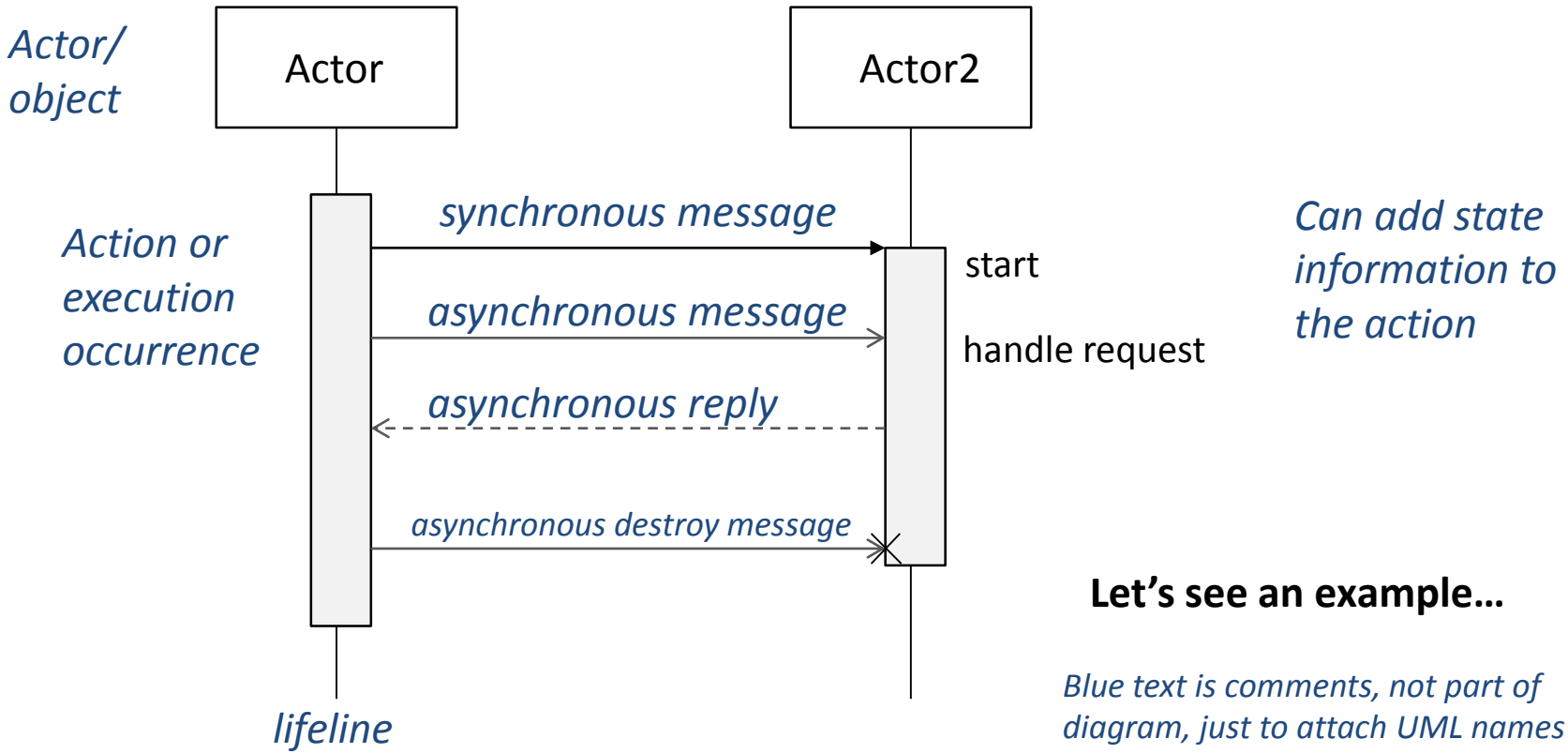
Some tools might use directed links to emphasise the user initiates the operation. The standard approach is not to use directed links.

Neither a precisely specified model of the computations nor a precisely specified model of the communication (intentionally so!)

Sequence charts

Sequence Chart: Used to show both order of actions/methods for objects and messages/data sent between them

How to draw one…



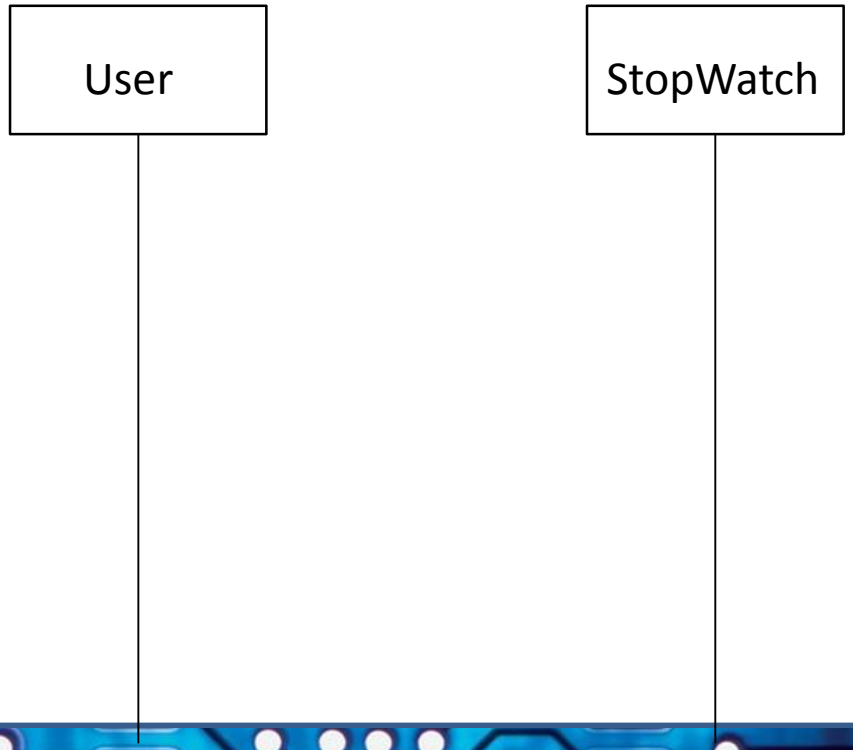
Let's see an example…

Blue text is comments, not part of diagram, just to attach UML names

Do Me A Chart…

Draw a quick sequence diagram for the count mode execution for below scenario.

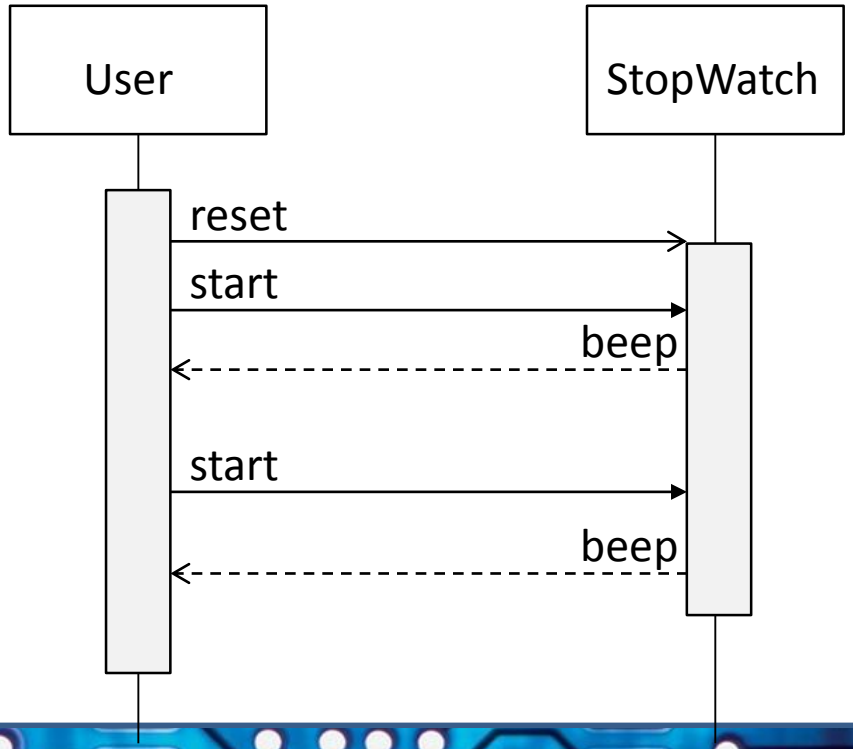
Operations: Initially stopwatch is idle. User presses reset which activates count mode, and clears the counter, setting s=0 and ds=0, and beeps. If user presses start, the watch increments ds for each 100ms that elapses. Each time ds gets to 10, s increments and ds is reset to 0. If user presses start then count mode stops and watch makes a beep.
Scenario: User presses reset, waits 500ms. Then presses start. Waits 10s200ms and presses start.



Do Me A Chart…

Draw a quick sequence diagram for the count mode execution for below scenario.

Operations: Initially stopwatch is idle. User presses reset which activates count mode, and clears the counter, setting s=0 and ds=0, and beeps. If user presses start, the watch increments ds for each 100ms that elapses. Each time ds gets to 10, s increments and ds is reset to 0. If user presses start then count mode stops and watch makes a beep.
Scenario: User presses reset, waits 500ms. Then presses start. Waits 10s200ms and presses start.

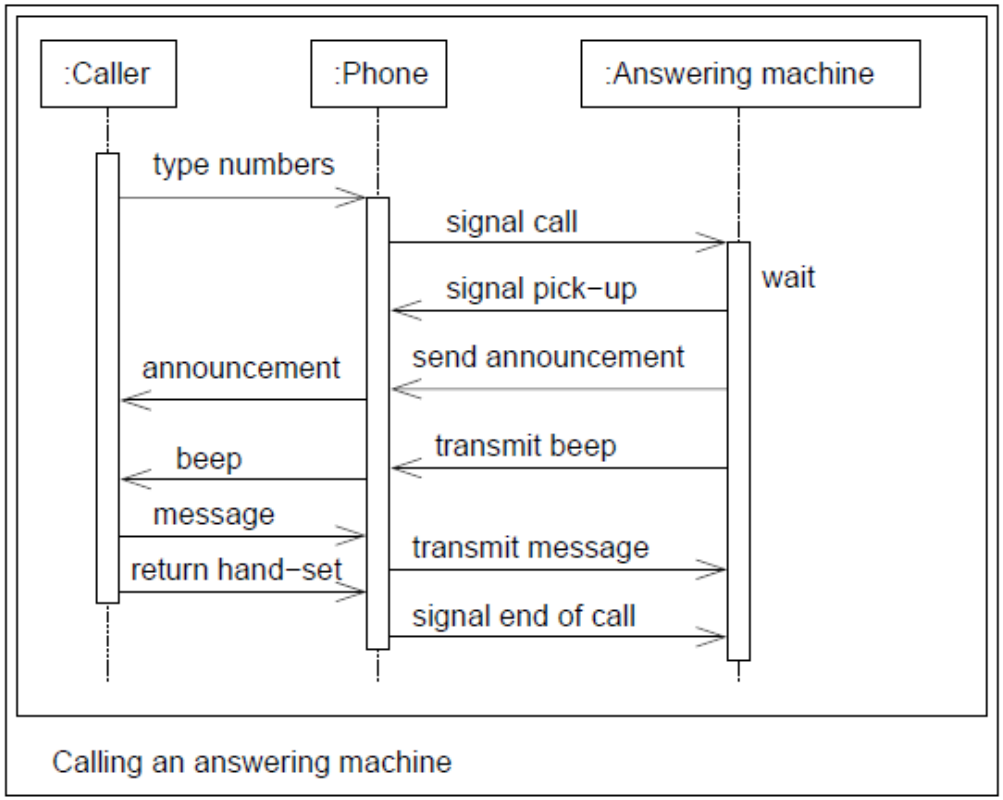


I've made the start messages synchronous to emphasise that the user even and watch response must be in synch.

(Message) Sequence charts (MSC)

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

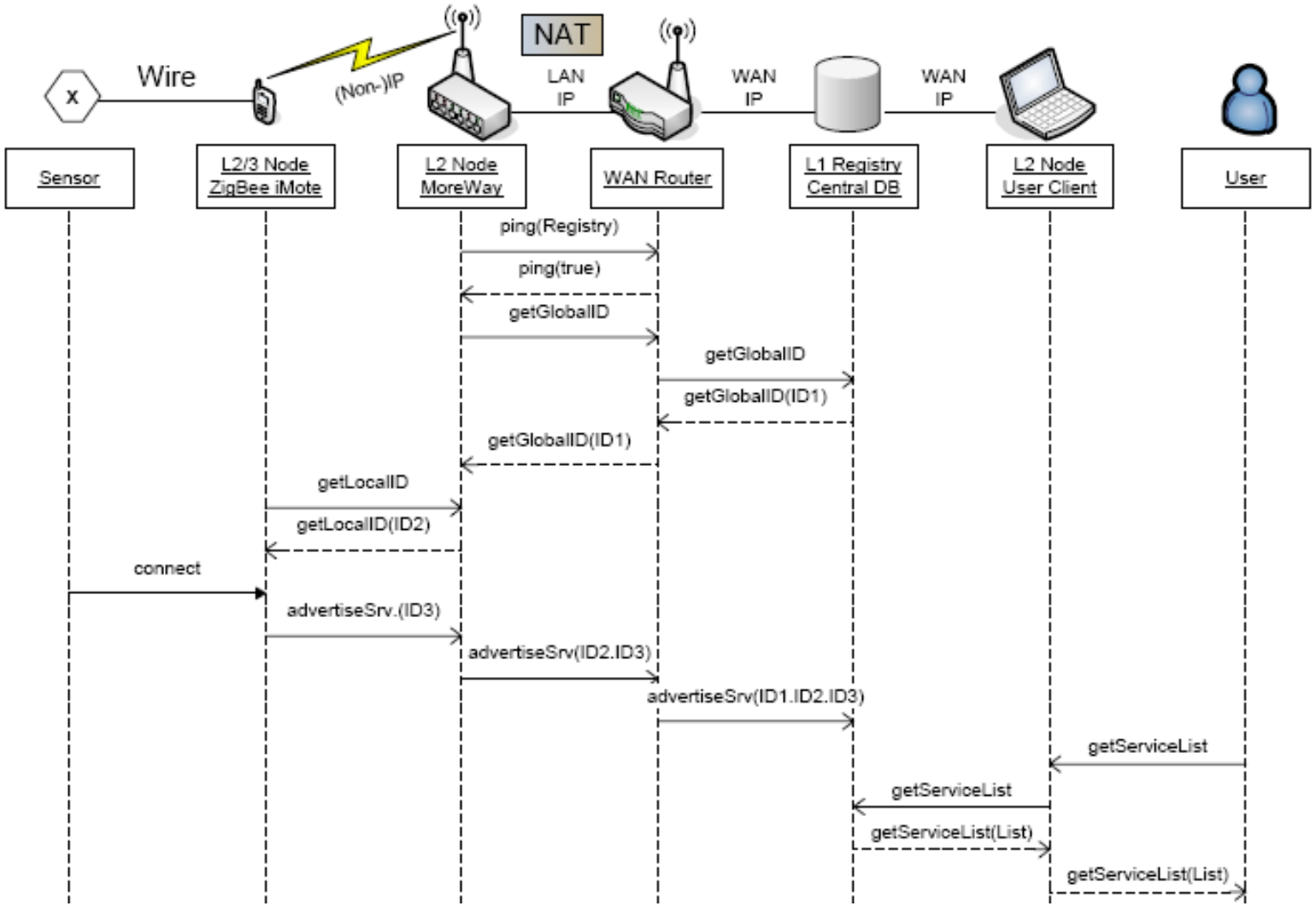
Example:



Calling an answering machine

- Included in UML
- Previously called Message Sequence Charts (MSC)
- Now mostly called Sequence Charts

Example (2)



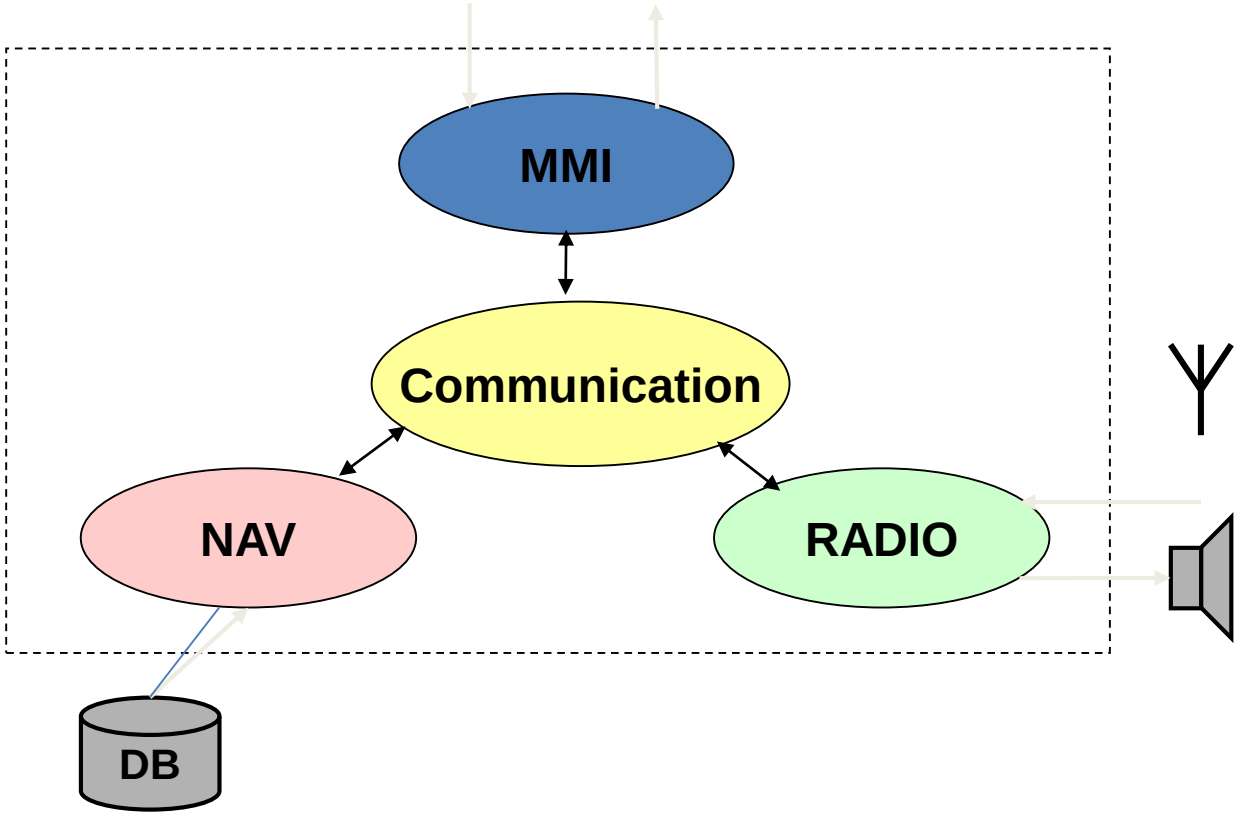
You can embellish them with colors and 'glyphs'

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
  (indicated by red outlines…)

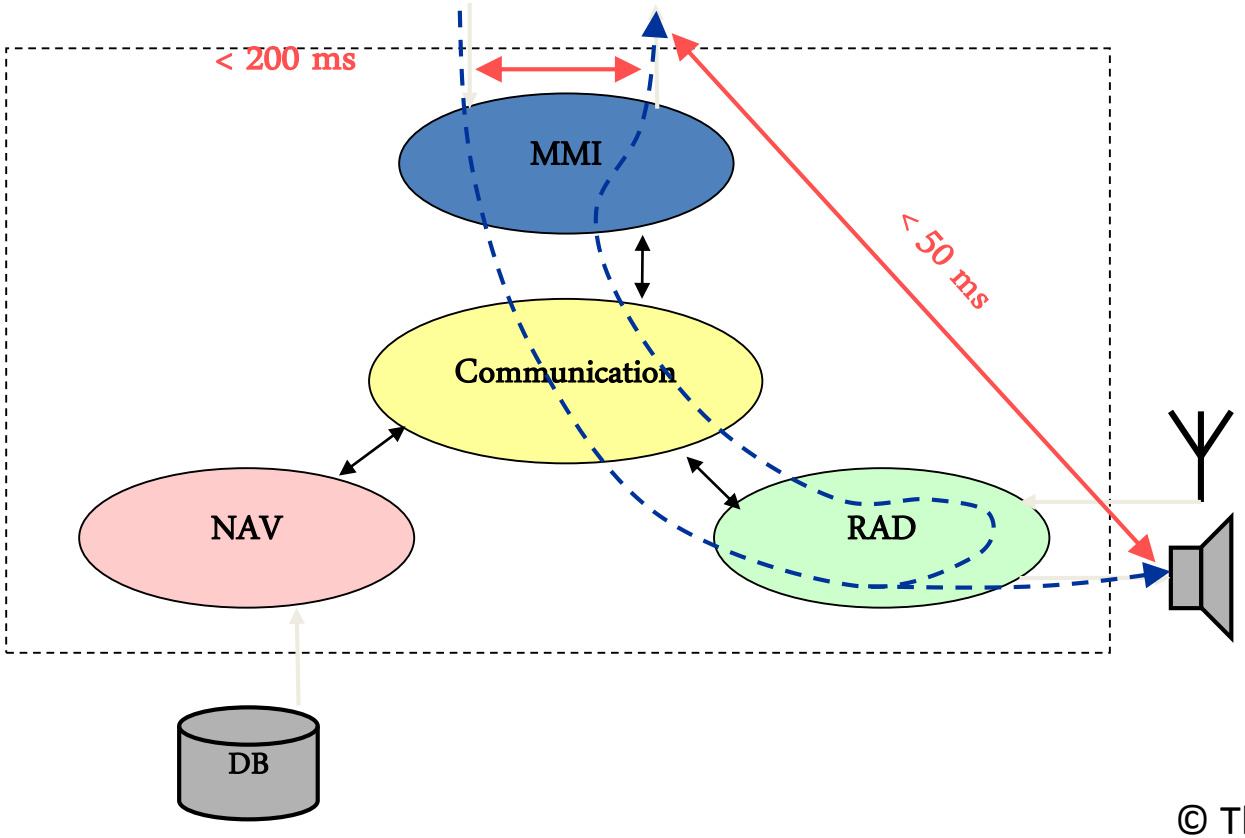
System Overview

Audi Music Media Interface (MMI)



Use case 1: Change Audio Volume

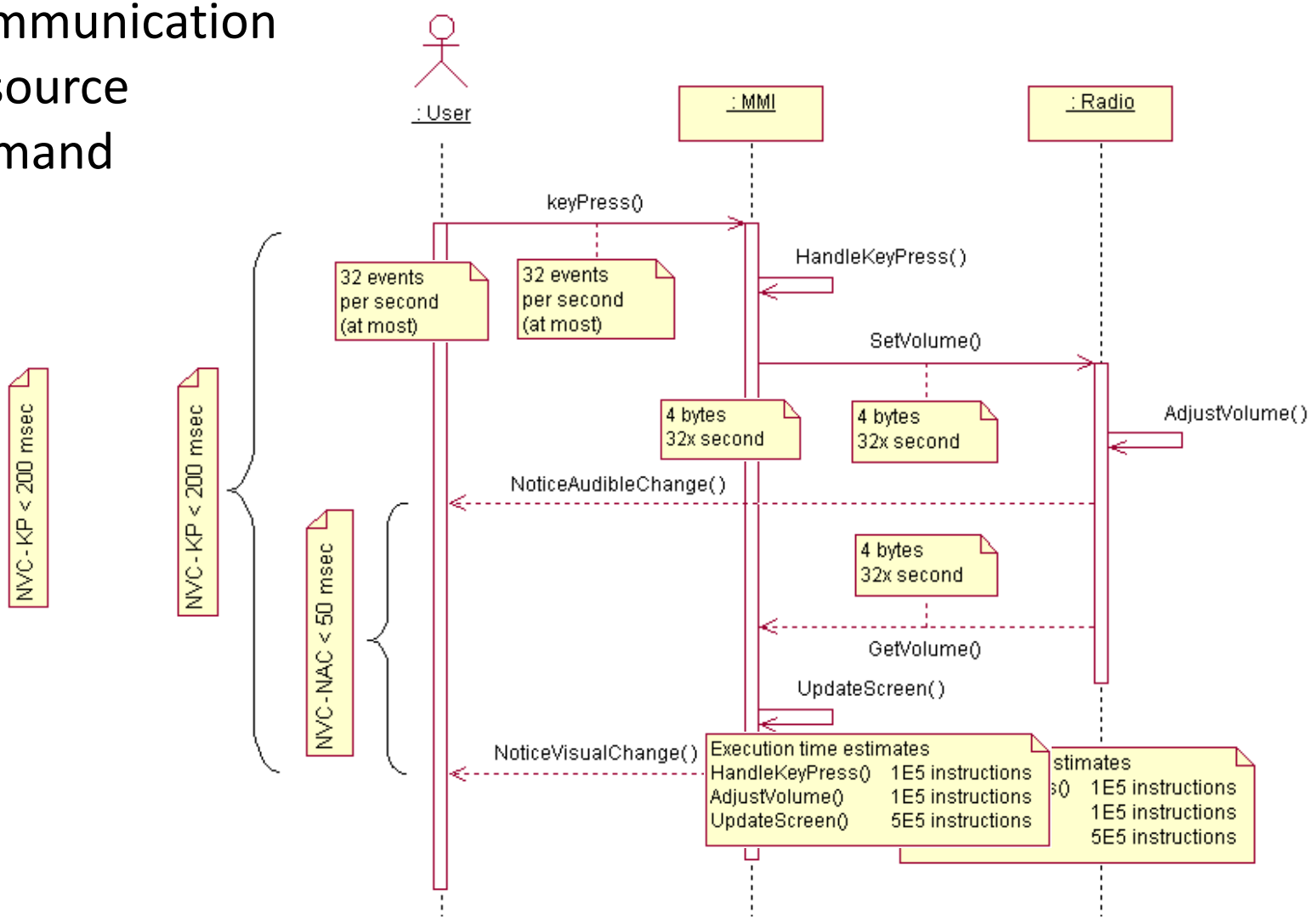
Mapping out timing constrains – an important aspect of requirement analysis for embedded systems



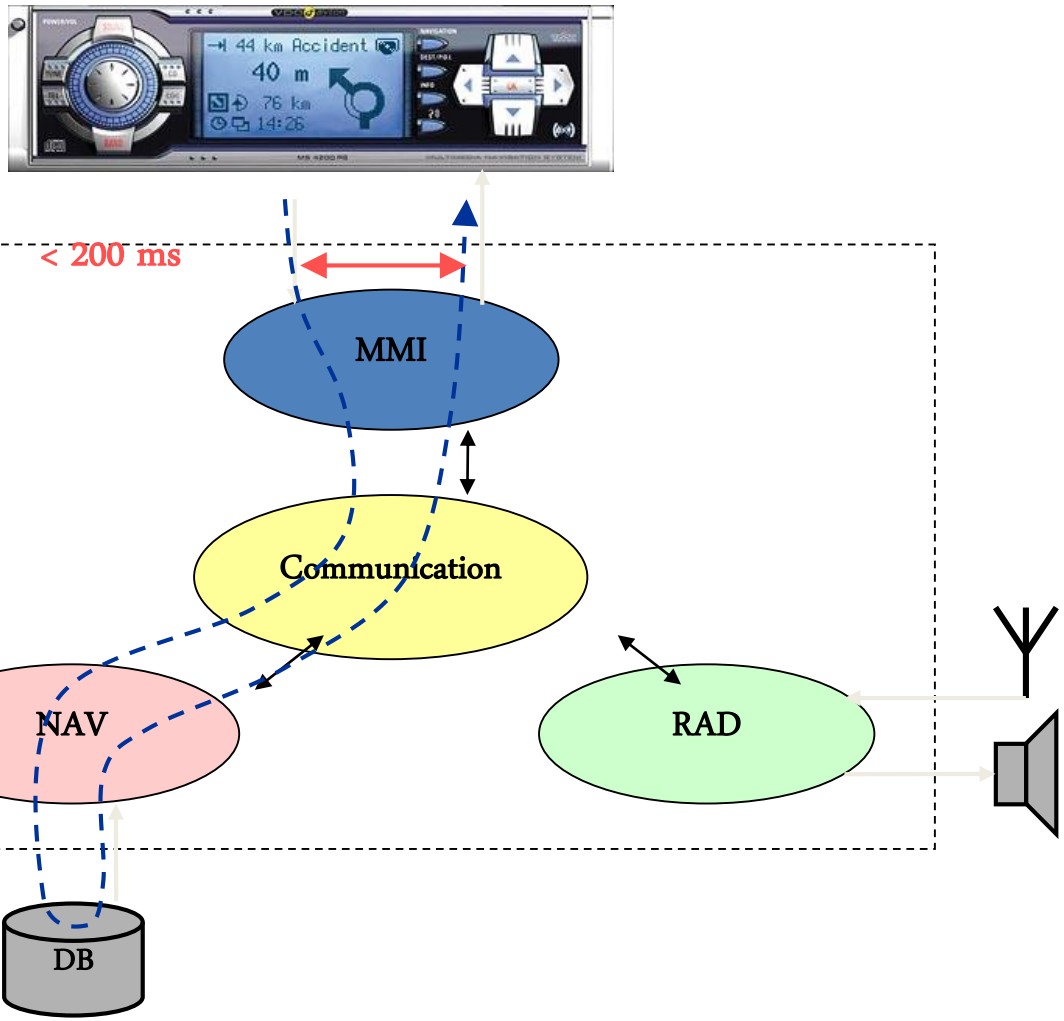
Putting timing into a sequence diagram

Use case 1: Change Audio Volume

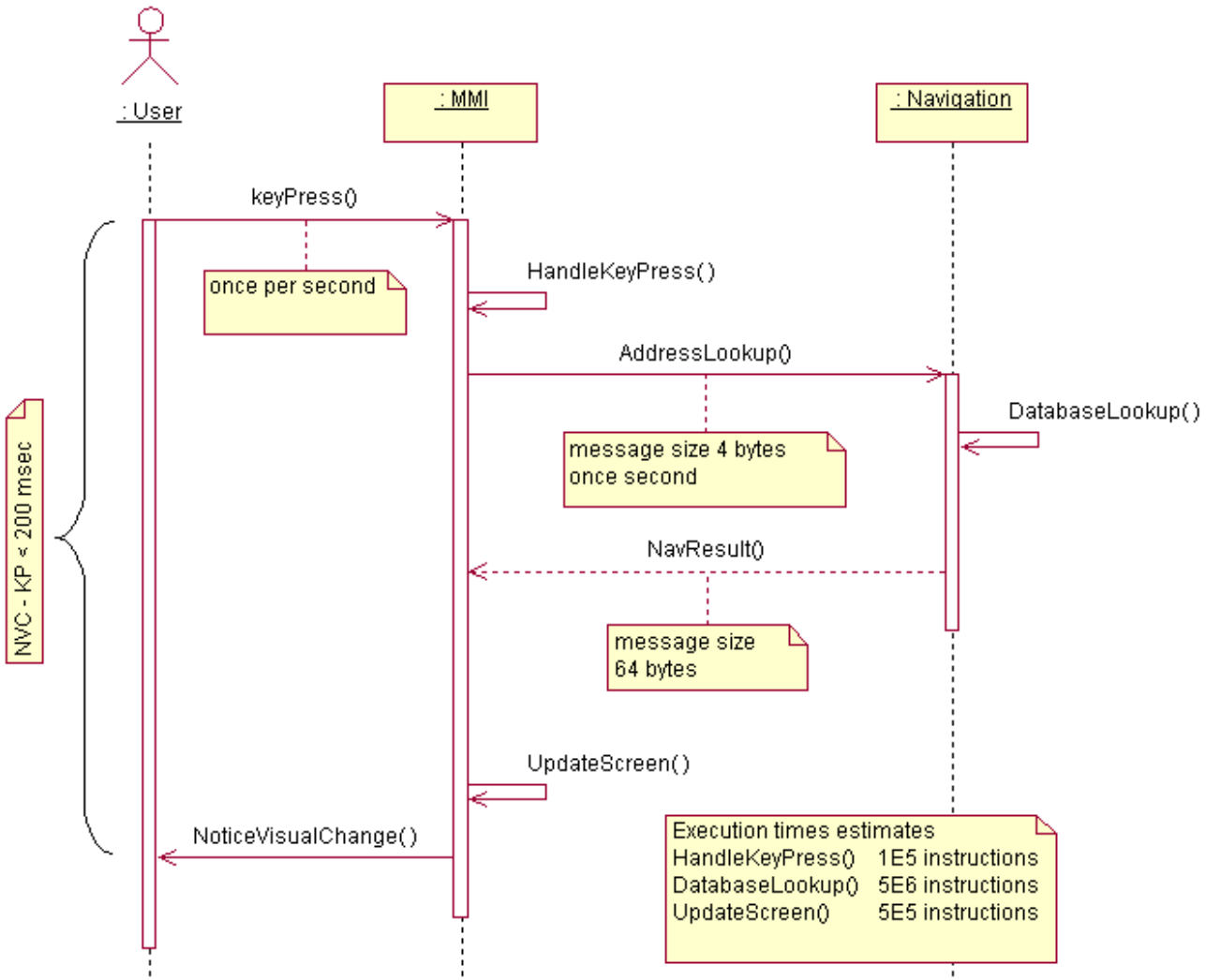
Communication
Resource
Demand



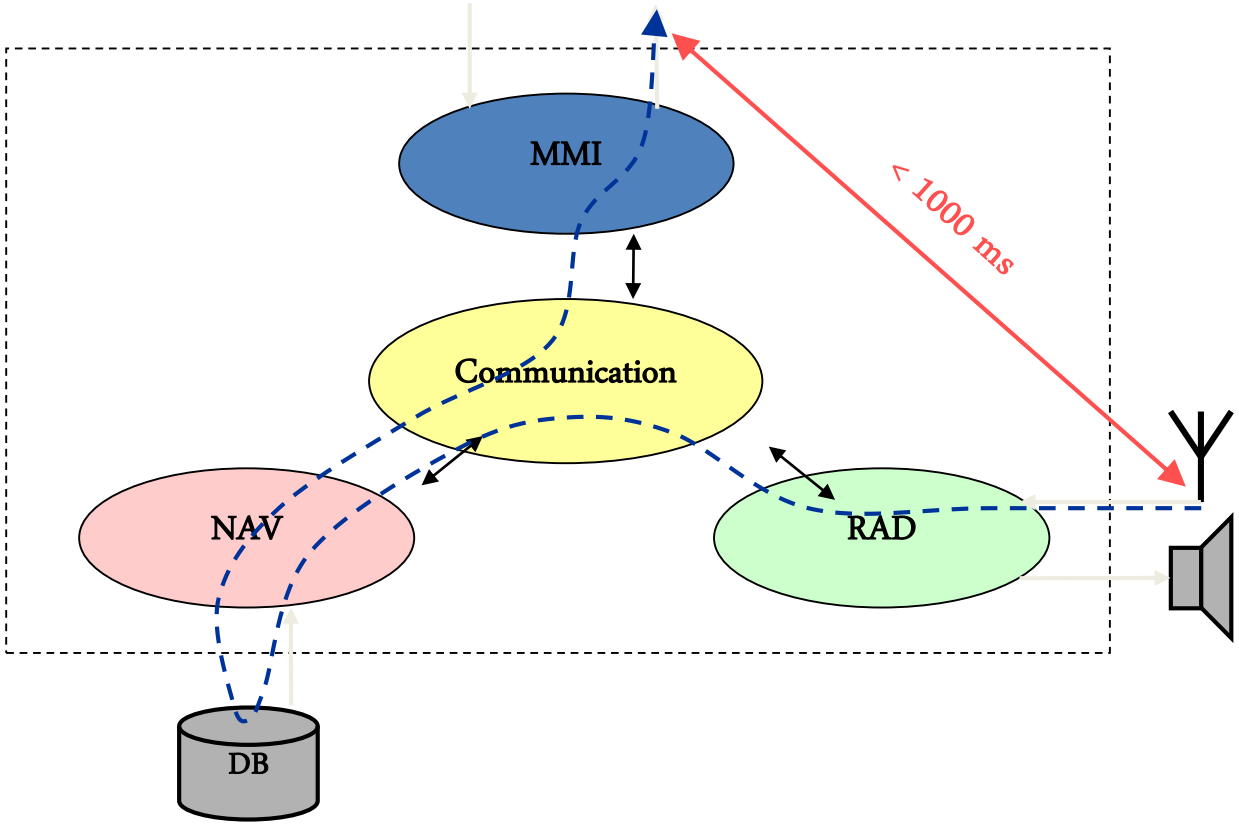
Use case 2: Lookup Destination Address



Use case 2: Lookup Destination Address

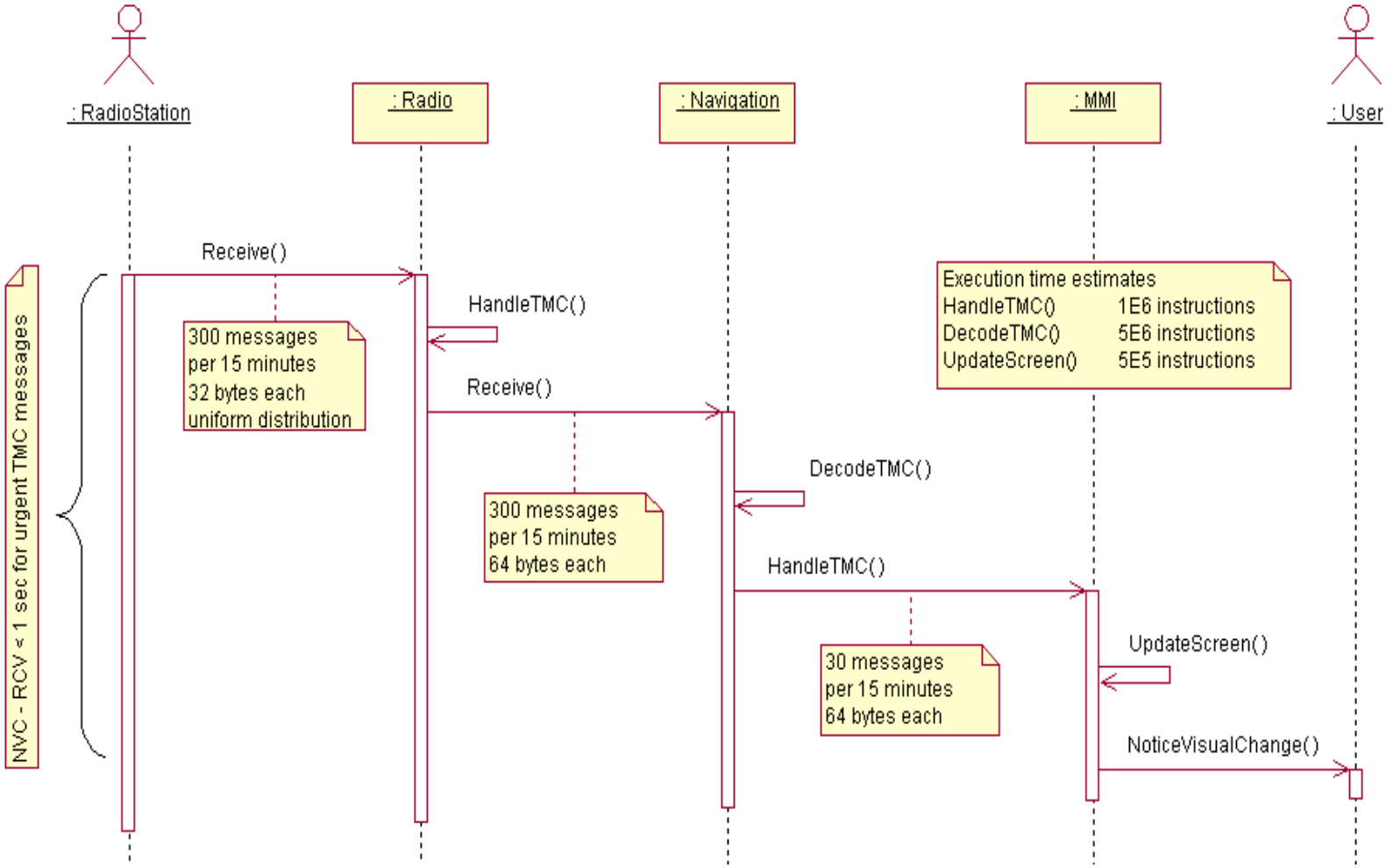


Use case 3: Receive TMC Messages



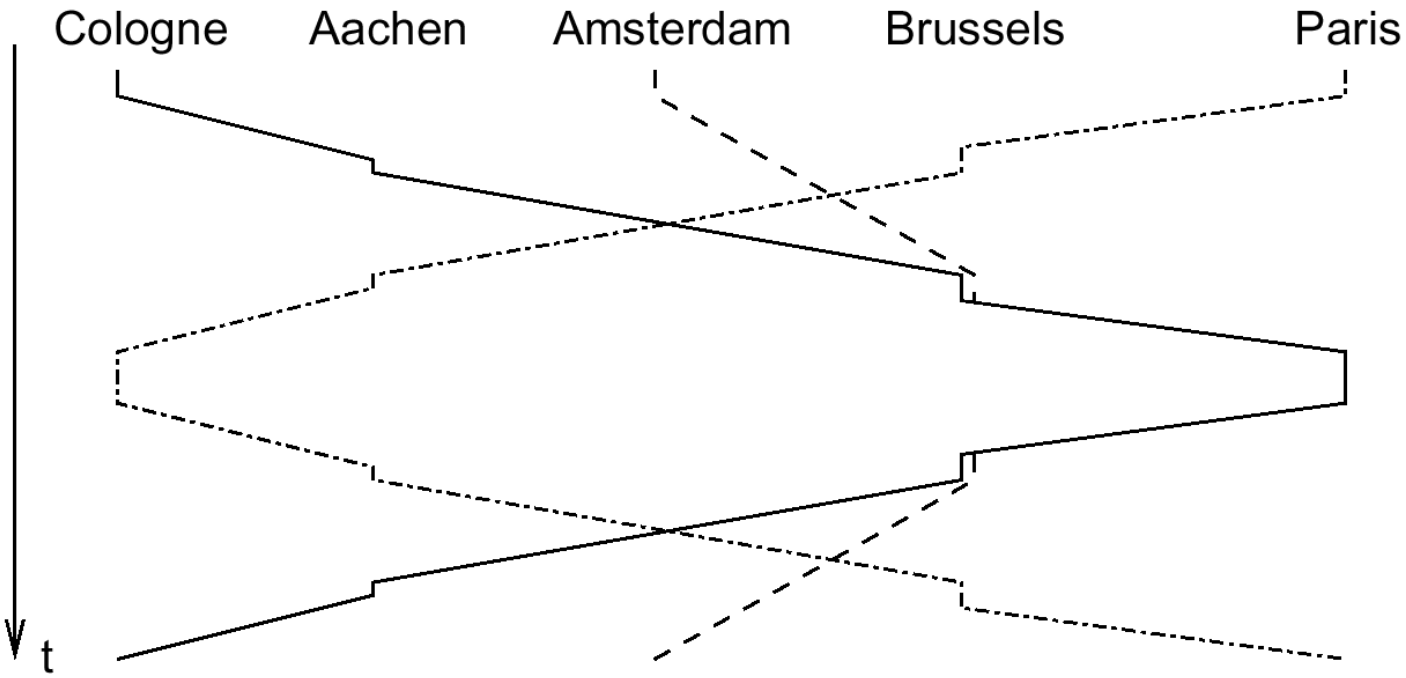
Traffic Message Channel (TMC): technology for delivering traffic & travel info to drivers.

Use case 3: Receive TMC Messages



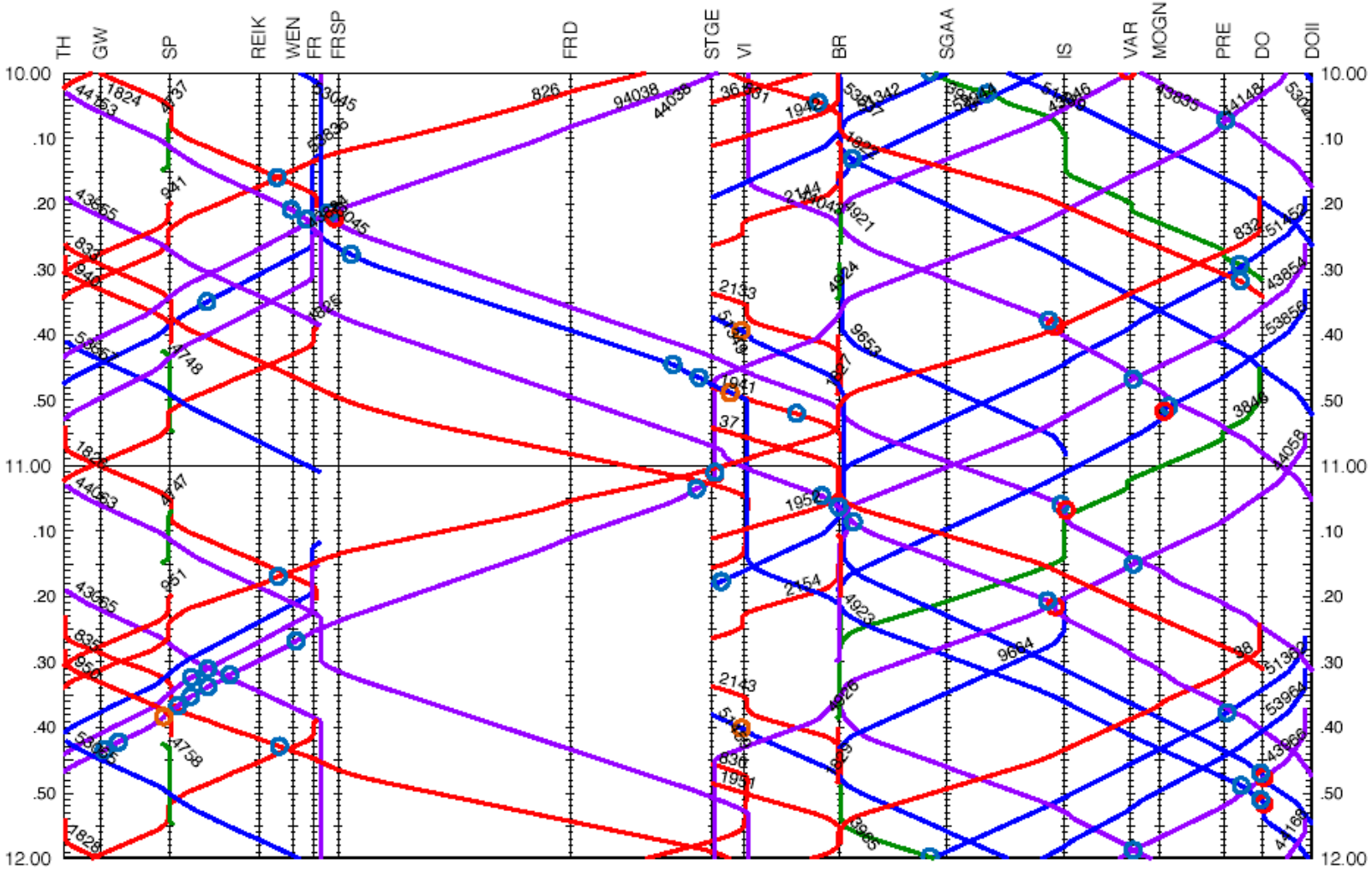
Time/Distance Diagrams (TDD)

These diagrams are more concise than sequence diagrams, just shows time that actions take in relation to others. This example uses shows trains moving from one station to another, but you could think of it as processing operations moving between states / results from one object to another.



No distinction between accidental overlap and synchronization

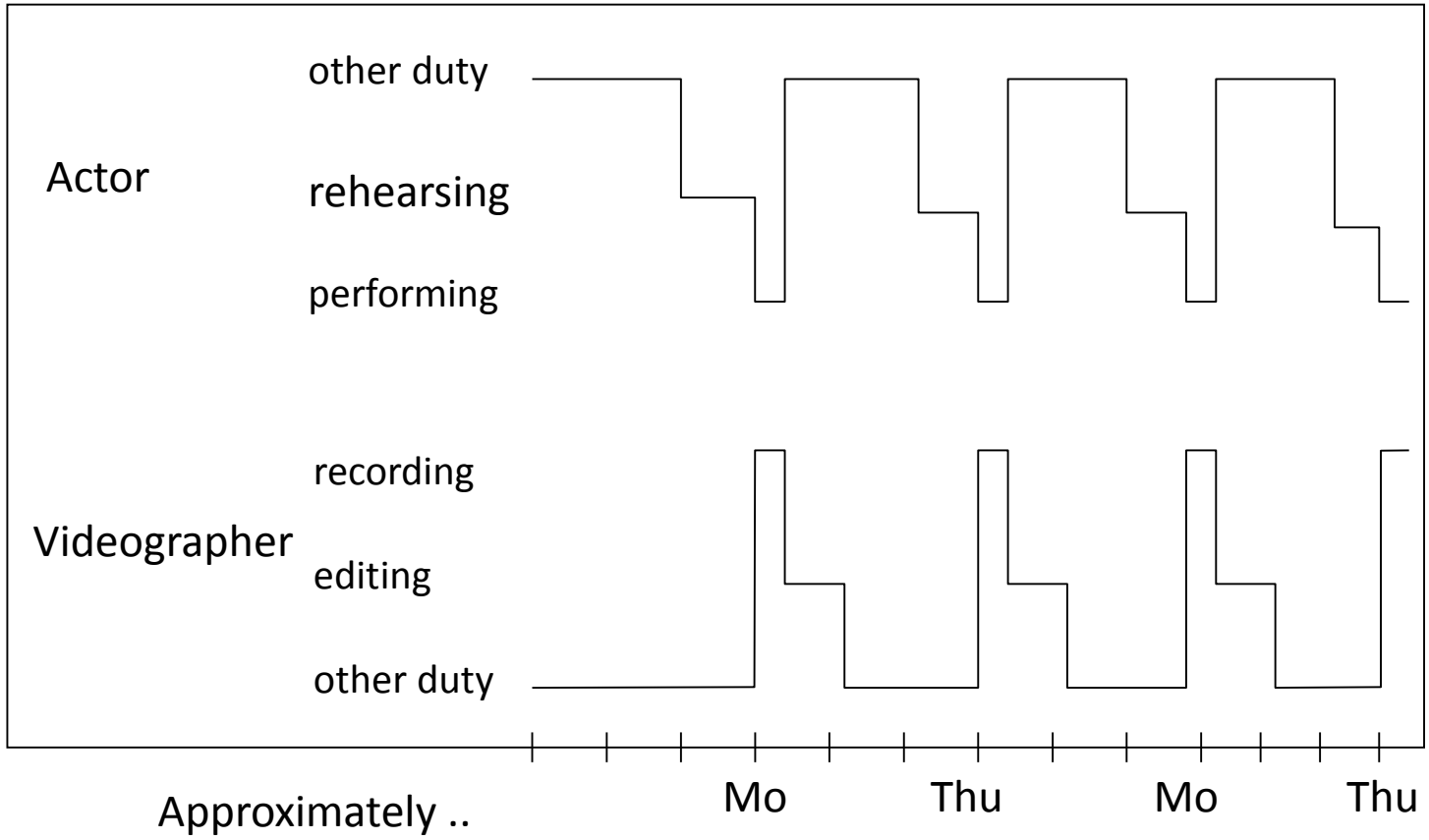
Clearly these can have limited use as the system expands. This TDD shows simulated Swiss railway traffic in Lötschberg area.



Levi-TDD

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 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)

See further discussion in textbook, essential aspect is:
pre-charts: Pre-charts describe conditions that must hold for the main chart to apply.

---

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

(Message) Sequence Charts (MSC)

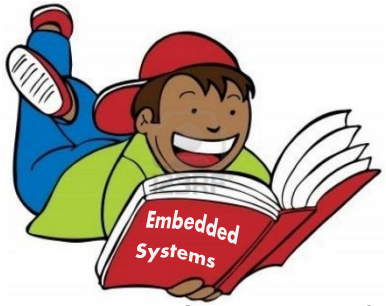
- **PROs:**
  - Appropriate for visualizing schedules
  - Proven method for representing schedules in transportation.
  - Standard defined: *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC*), ITU-TS, Geneva, 1996.
  - Semantics also defined: *ITU-TS Recommendation Z.120: Message Sequence Chart (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'9*7, Chapman and Hall, 1997.

The Next Episode…

Lecture L07

L07: Communicating finite state machines, UML State Charts, Customer Tours



Reminder: Read section 2.4