DESIGN OF A REAL-TIME COMPUTING SYSTEM

H KOPETZ

Rapporteur: R de Lemos

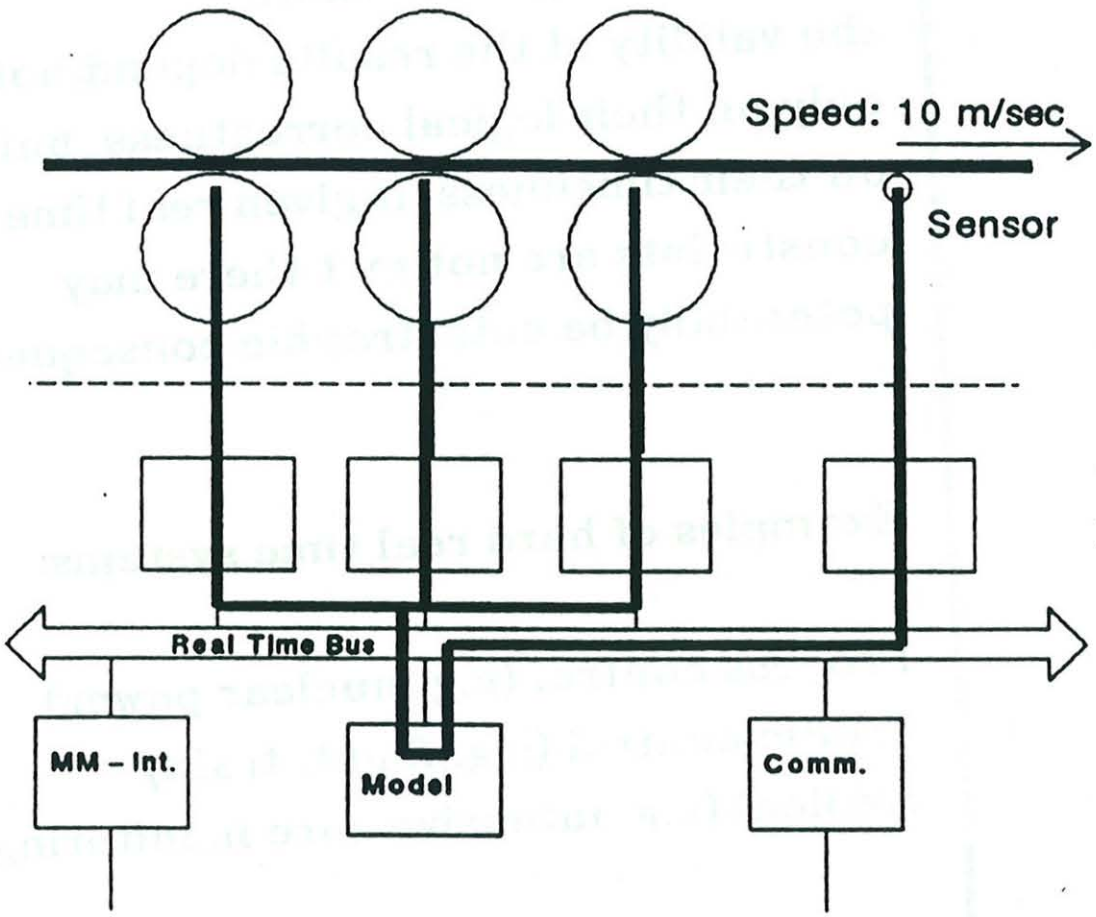
TU Wien		hard 1. 2

What is a "hard real time" system ?

A computer system, where
the validity of the results depend not
only on their logical correctness, but also
on their timeliness. If given real time
constraints are not met there may
potentially be catastrophic consequences.

Examples of hard real time systems:

Process control (e.g. nuclear power)
Traffic control (e.g. flight, train)
Medical (e.g. intensive care monitoring)



The "rolling mill" example

SOFT REAL TIME vs. HARD REAL TIME

Characteristic	Hard Real Time	On line
Response Time	hard	soft
Pacing	by the environm	by the computer
Peak load perform	predictable	degraded
Granularity of time	< 1 msec	about 1 sec
Basic com. service	end to end	transport service
Clustering	important	less important
data files	small to medium	large
data integrity	short time	long time
Safety	critical	not critical
Error detection lat	bounded by syst	responsibility of u
Redundancy	active	standby

In a typical hard real time system

* the Maximum Response time to a stimulus is determined by the environment
* an explicit flow control cannot be exercised over the environment
* the real time data is invalidated by the passage of real time
* the economic justification is related to the predictable performance under peak load condition
* Peak load is highly correlated

Flowcontrol:
Adjusting the speed of the sender such that the receiver will not be flooded by messages.

There is always an upper limit to the performance of a receiver!

Explicit Flowcontrol:
The receiver sends an acknowledgement message to the sender waiting for this message.
(The sender must be in the sphere of control of the receiver).

Implicit Flowcontrol
There is an agreed maximum messages rate for the sender. It is assumed that all receivers can follow this rate.

There is a potential conflict at the boundary between implicit and explicit flow control:

Is it possible to specify and guarantee the maximum sender rate?

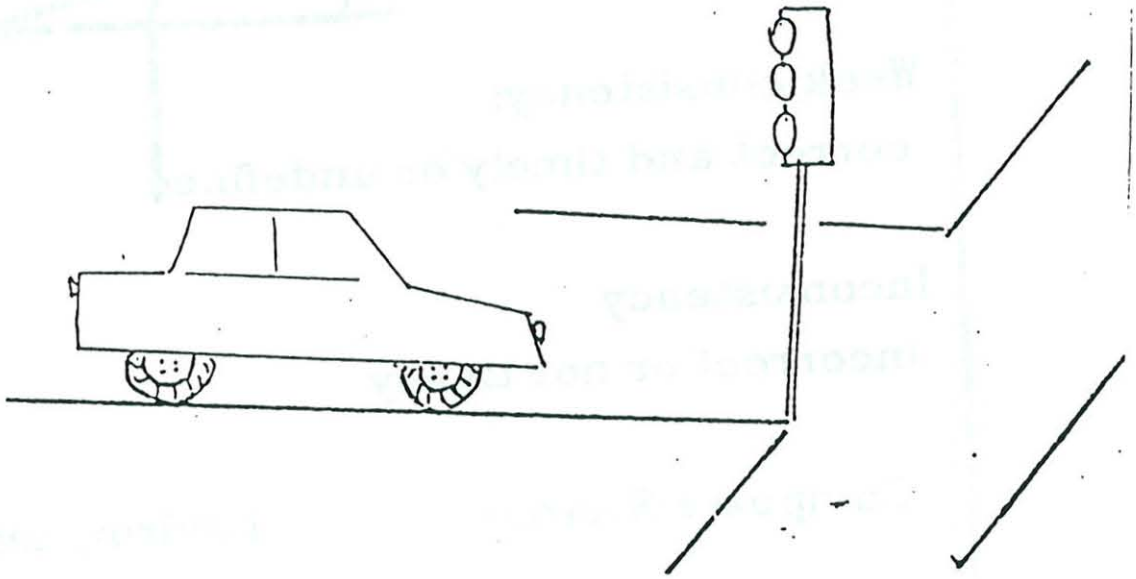
What happens, if a sender outside the sphere of control of the receiver sends more messages than the agreed limit?

How do we size the buffers?

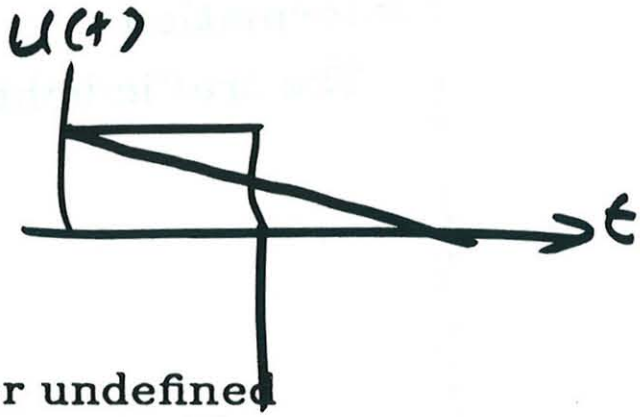
TU Wien		s1.15 1.4

What is the validity time of the information

The traffic light is green ?



Δt consistency between the real time data and the environment

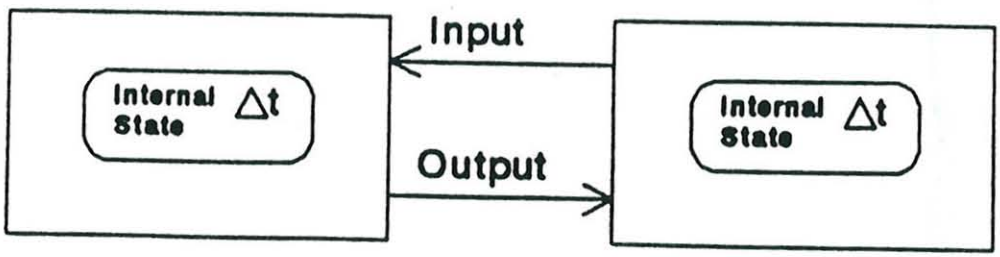


Strong consistency:
correct and timely

Weak consistency:
correct and timely or undefined

Inconsistency
incorrect or not timely

Computer System Environment



Peak Load

In many cases, the services of a hard real time system are needed most urgently under peak load conditions ("rare event"):

Examples:

- Ligthning stroke into a power grid
- Engine failure on takeoff of a plane
- Rupture of a pipe in a nuclear reactor

It is difficult to impossible to extrapolate from the behaviour under "normal situation" to the behaviour under "peak load situations" (rare events).

"Confidence by design"

Assumption about time base:

All nodes have access to a global time base of known synchronization accuracy:

- The time base is chronoscopic, i.e it does not contain any point of dicontinuity
- The metric of the time base is close to the metric of the time standard (TAI)
- The granularity is chosen in agreement with the synchronization accuracy.

Implementation issues of the time base are discussed in the chapter on real time.

Real Time (RT) entity:

Something of relevance for the given purpose which contains a time varying internal state.

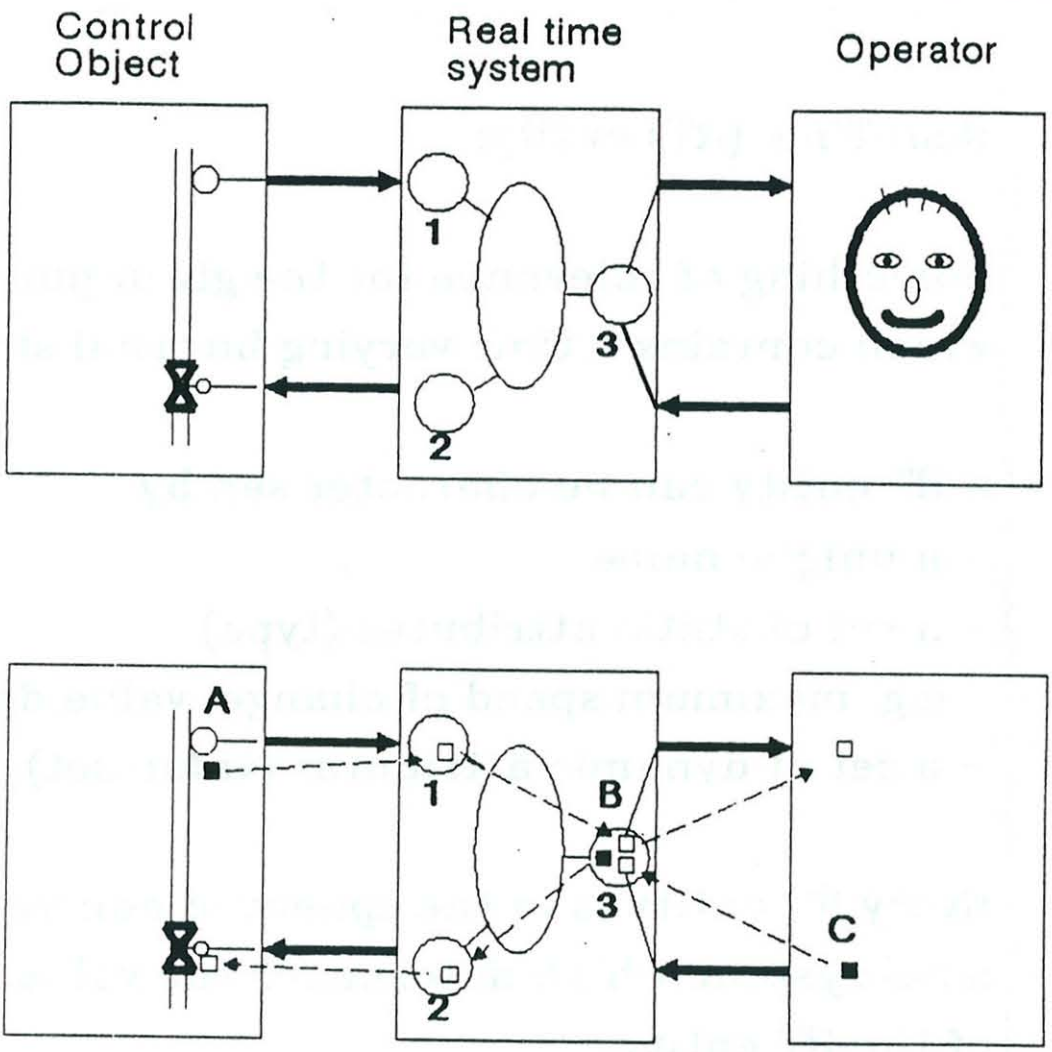
A RT entity can be characterised by
- a unique name
- a set of static attributes (type)
  e.g. maximum speed of change, value domain
- a set of dynamic attributes (value set)

Every RT entity is in the sphere of control of a subsystem, which determines the value set of the RT entity.

Examples of RT entities:
Temperature, setpoint, intended valve position

TU Wien

M 10



A Measured Value

B Intended Valve Position

C Setpoint

□ Observation

■ RT – entity /object

Message flow

Components

RT – entities and Observations

Relationship between entities and objects

In our model, the entities of the world of interest will be represented by objects.

Starting from the classical concept of an object we introduce

- Real Time objects and
- Distributed objects

TU Wien		object M 12

OBJECT (classic):

An object is an autonomous entity, containing an internal state and a set of associated operations.

An operation can be invoked by the receipt of a message and the object reacts by sending a message containing the result of the operation.

The external interface of an object is defined by the set of all messages it can receive or send (and its internal state space).

Related objects can be grouped together to form a class (and a superclass etc.).

REALTIME OBJECTS:

A realtime object is an object that

- contains the global time as part of its internal state and

- activates some of its internal operations whenever a predicate on the global time becomes true.

A realtime object is active, if output messages can be generated spontaneously.

A realtime object is passive, if an ouptut message can only be generated as a consequence of a request to do so by an input message.

Distributed real time object:

A distributed real time object is a
set of coequal realtime objects
located at different sites and
belonging to the same class.

Every local instance of a distributed
RT object provides a specified service
to the local site.

The quality of service of a distributed
RT object must be in conformance with
some specified consistency constraints:

Examples:
Clock synchronization within delta
Membership service

TU Wien		observ M15

Observation (State):

An observation is a specific message type containing information about the state of a RT object (and thus about the associated RT entity) at a particular point in time.

An observation can be represented by the following tuple:

<object name, value set, t_obs, t_val>

where

object name: name of the observed object

value set: the set of attribute values of the observed RT observed

t_obs: time of observation

t_val: validity time of the observation

Real time data base:

An observation is current at a point in time t_use if

$$t\_obs < t\_use < t\_val$$

An observation is archival at a point in time t_use if

$$t\_use >= t\_val$$

The real time data base is defined as the set of current observations about a specified set of RT-entities.

A subset of the archival observations of a specified set of RT-entities forms the archival data base.

State observation

Triggered by a periodic time signal
Contains full value set of the rt-entity

Event observation

Triggered by a state change
contains the differences between old state and new state

In a fault free system, the two observation techniques are functionally equivalent

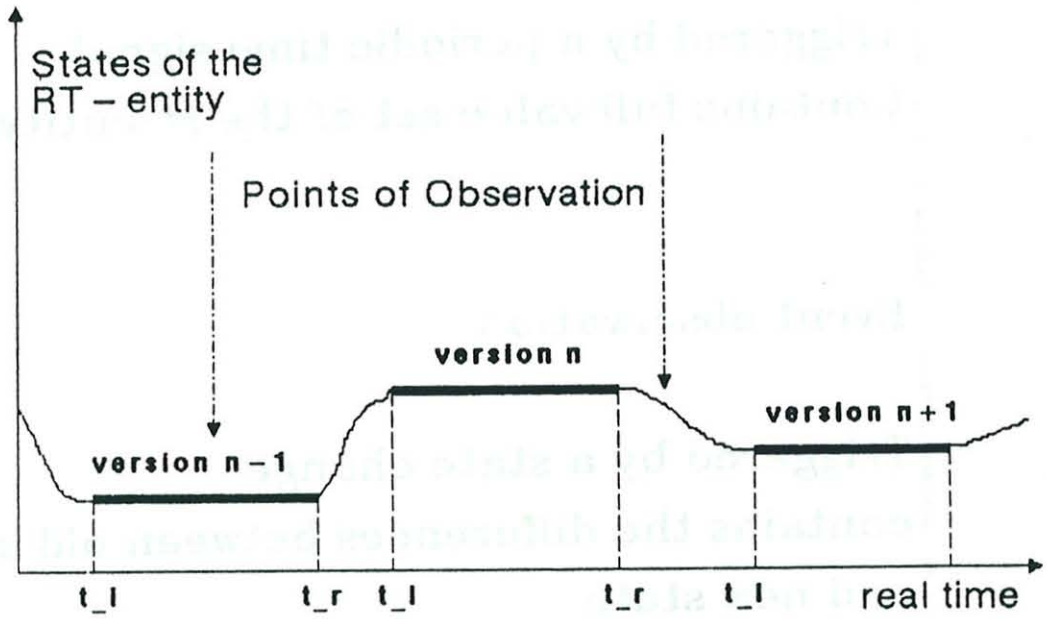
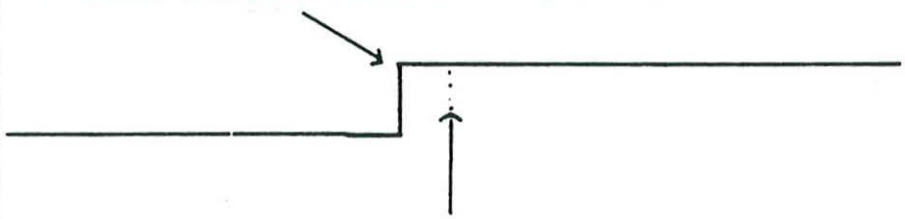


Fig 1.: Versions of a RT – entity and points of observation

RELATIONSHIP BETWEEN STATES AND EVENTS

Every change of a state is an event.

An event cannot be observed--
only the "new" state is observable.



Comparison of State and Event Observation

	State	Event
Size	large	small (varying)
Rate	constant	dynamic
Loss	not critical	critical
Flowcontrol	implicit	explicit
Protocols	simple	complex
Peak load	constant	???

TU Wien		rttrans M20

RT TRANSACTION

A real-time transaction is the time constrained execution of a set of communication and processing actions which transform the observed system from one consistent state (the start state) to another consistent state (the termination state).

The termination state can be the intended termination state or a safe exit state.

A real time transaction is started by a stimulus message and is terminated by a response message.

RT versus DB Transactions

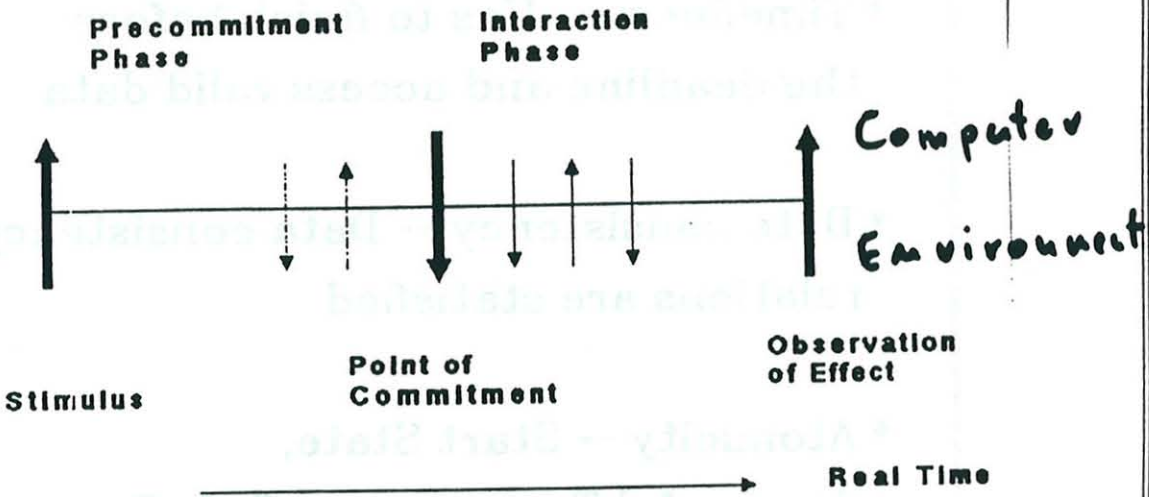
* RT Transaction is time constrained

* Point of Commitment within a RT Tansaction (not at the end)

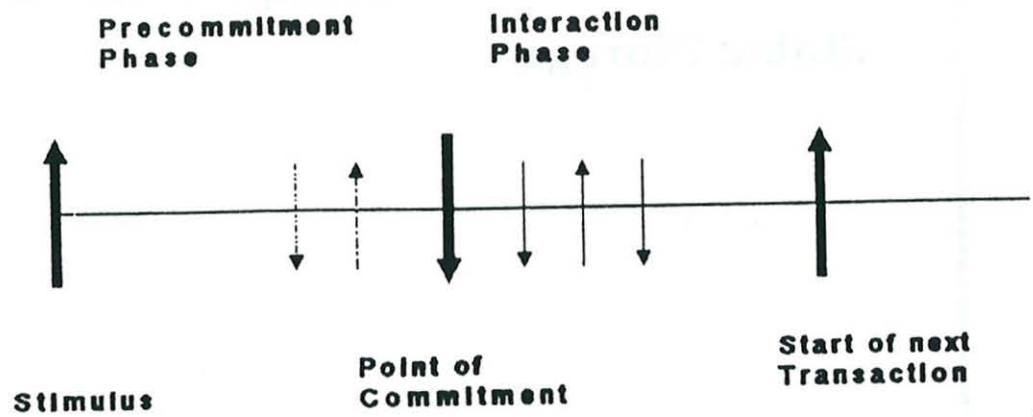
* Concurrency conflicts have to be resolved immediately in RT Transactions

* Viewed from the outside, there are three states (not two)

Real Time Transaction



Periodic RT – Transaction



Attributes of RT Transaction

* Timeliness -- Has to finish before the deadline and access valid data

* Data consistency -- Data consistency relations are statisfied

* Atomicity -- Start State, Successful Termination, Safe Exit

* Permanence -- Permanent Effects on Stable Storage

Relationship between realtime transactions and realtime objects:

Viewed from the outside, a real-time transaction can be considered as a RT-object the service of which is invoked by a request message and which returns the intended response message.

Viewed from the inside, a real-time transactions consists of the exchange of messages between lower level objects and the execution of the invoked operations by these lower level objects.

These two activities can be concurrent.

Thus RT transactions and RT objects are different views of the same phenomena.

The service request to a real time object, e.g readonly access to the internal state, can be quasi-independent from the execution of the the associated real time transaction.

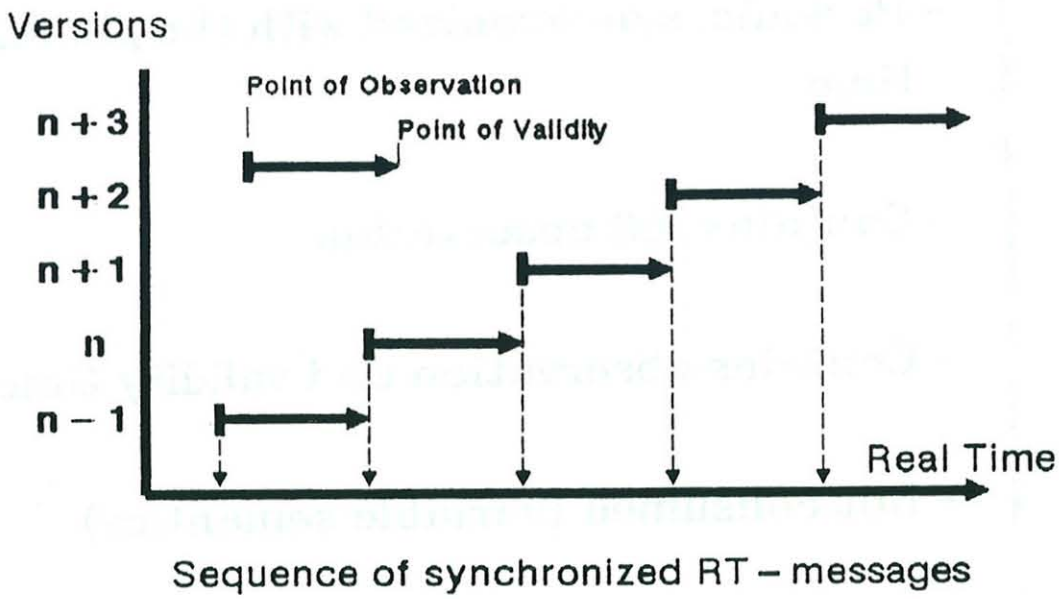
Triggered by a predicate on the time (which is part of the state of a real time object) the real time transaction can operate concurrently with the service request from the level above.

This is the fundamental difference to the concept of procedural abstraction or protocol layering, where a service request from a higher level initiates the activities of the lower level.

Real Time Message:

- Periodic, synchronized with the global time
- Contains full observation
- Contains observation and validity time
- Not consumed (variable semantics)
- New version overwrites previous version atomically

Quantization of the system behaviour in the domain of real time!



Sequence of synchronized RT – messages

DISCUSSION

Rapporteur: Rogério de Lemos

Professor Joseph inquired in what sense was the load on a computer highly correlated. Professor Kopetz replied that the load was highly correlated during emergency situations. For example, if a pipe bursts it will raise a lot of alarms, so it was very risky to base assumptions on the use of the Poisson distribution.

Professor Leveson asked what was meant by a contract. Professor Kopetz answered that in terms of the cluster design tool it was an agreement between two subsystems for the passage of data.

Professor Randell asked how in general one could identify, without knowing the internals of the system, what usage of a system could cause it to be overloaded. Professor Kopetz answered that in practice this issue was not difficult to resolve.

Professor Bron asked about the problems of estimating parameters needed for creating static transaction schedules, prior to completion of an implementation. Professor Kopetz replied that one used an estimate initially, but checked this estimate during detailed design and if necessary re-calculated the schedule.

Professor Randell asked whether it was reasonable to assume that one can usefully identify and classify all the various possible emergencies beforehand. Professor Kopetz answered that in his view it was essential for the designers to have fully adequate knowledge of the behaviour of the environment.

On the same subject Mr Waterworth queried about an environment that you thought you knew but at the end unexpected events could happen. At this point Professor Leveson intervened stating that you could protect yourself against that by using a fail-safe model for the situations you have not predicted. Professor Kopetz added that what they basically had was one emergency situation which includes all the others.

Professor Joseph wondered whether there was any limitation in checking the scheduling at the design time before having the code. Professor Kopetz answered that they knew the scheduling on the basis of the information about transactions and synchronization information. Later on they checked again from the code to see if the schedule which has been developed after the coding was in accordance with that generated.

Professor Nehmer asked how the resource conflicts would be resolved if all the tasks were scheduled in sequence. Professor Kopetz replied that only those branches were followed which were in agreement with the requirements specification, which is a precedence selection between the tasks which were free of resource conflicts. Those schedules which lead to resource conflicts were not considered and they were eliminated from the search. Professor Nehmer asked if semaphores were used. Professor Kopetz answered that they did not need semaphores in the operating system. They need a very simple schedule at the run-time because all the conflicts which could possibly occur would already have been considered in the compile time scheduling.

Professor Turski asked what could happen if something went slightly wrong with the scheduling at run-time, such as anything extending a little longer to cause the collapse of the designed scheduler. Professor Kopetz answered that they did not build the scheduler in a way that would just satisfy the execution time of the tasks within a slot. Instead safety margins were introduced for the purpose of fault-tolerance. Professor Turski argued that off-line scheduling was a numerical computation which was numerically unstable, and questioned Professor Kopetz as to whether he had investigated the numerical stability of the numerical analysis. At this point Professor Anderson intervened stating that Professor Kopetz had already responded by saying that if you had got an answer at least you could validate that answer, whatever it was. The alternatives involved a very dynamic use of semaphores.

Professor Mok asked whether the proposed scheme had any general run-time checks. Professor Kopetz answered that he had only timing checks.

Professor Nehmer inquired why Professor Kopetz did not extend the timing constraints by a value of δ to allow emergency transactions, in which case he would not have to change the schedules. Professor Kopetz replied by saying that they felt that the switch over was simple and they could switch back to the normal case.