Chapter 5
Real-Time Entities and Images

Overview

- Real-time entities as significant state variables
- Observations, state and event observations
- Real-time images as current picture of real-time entity, and real-time objects
- Temporal accuracy and state estimation to improve real-time image accuracy
- Permanence in case of race conditions and idempotency with replicated messages
- Replica determinism to implement fault-tolerance by active redundancy

5.1 Real-Time Entities

A real-time (RT) entity is a state variable of relevance for the given purpose.

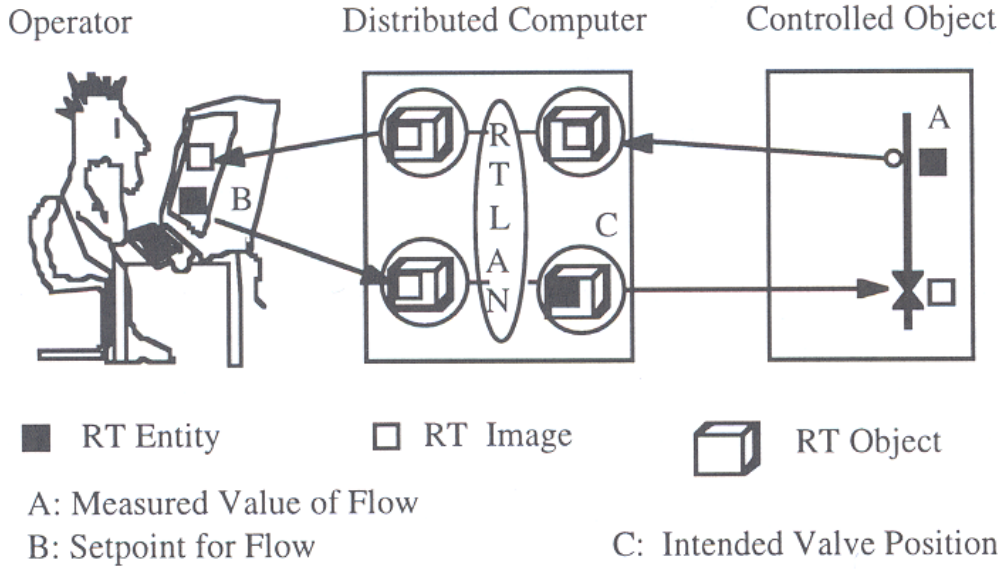
It is located either in the environment or in the computer system.

Examples: flow of liquid in pipe, pressure in vessel, setpoint of a control loop, intended position of a valve.

An RT entity has static and dynamic attributes:

- Static: name, type, maximum rate of change
- Dynamic: value, actual rate of change

Sphere of Control:



Each entity is in the sphere of control (SOC) of a subsystem that has the authority to set its value.

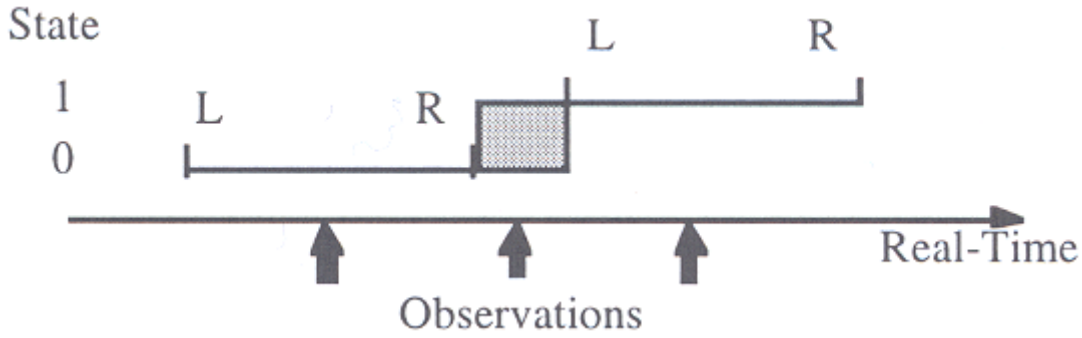
Example with three entities:

- Flow in pipe (in SOC of controlled object)
- Setpoint for flow (in SOC of operator)
- Intended valve position (in SOC of distributed system)

5.1 Real-Time Entities

Discrete and Continuous Real-Time Entities:

Discrete real-time entities: value set is discrete over time. It remains constant between an L_event and the next R_event. Between R_event and next L_event the set of values is undefined.



Continuous real-time entities: value set is always defined.

Example for discrete RT entity: garage door with “open” state and “closed” state.

5.2 Observations

Information about state of an RT entity at particular point in time we call an observation.

$$\text{Observation} = \langle \text{Name}, t_{obs}, \text{Value} \rangle$$

A continuous RT entity can be observed any time, while discrete RT entity can only be observed during L_event and R_event.

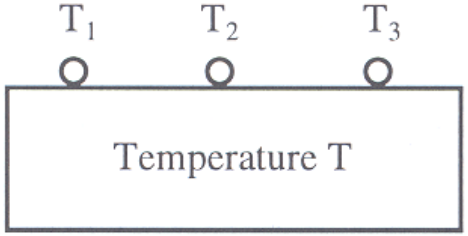
Untimed Observation

In a distributed system without global time, a timestamp can only be interpreted within the scope of the node that created it. In this case, the timestamp of the sender is meaningless for the receiver.

Often time of arrival at receiving node is taken as t_{obs}. This timestamp is imprecise because of delay and jitter.

Indirect Observation

Sometimes, a value of an RT entity cannot be observed directly, but must be inferred from direct observations. Example: temperature inside rod of steel.



State Observation

5.2 Observations

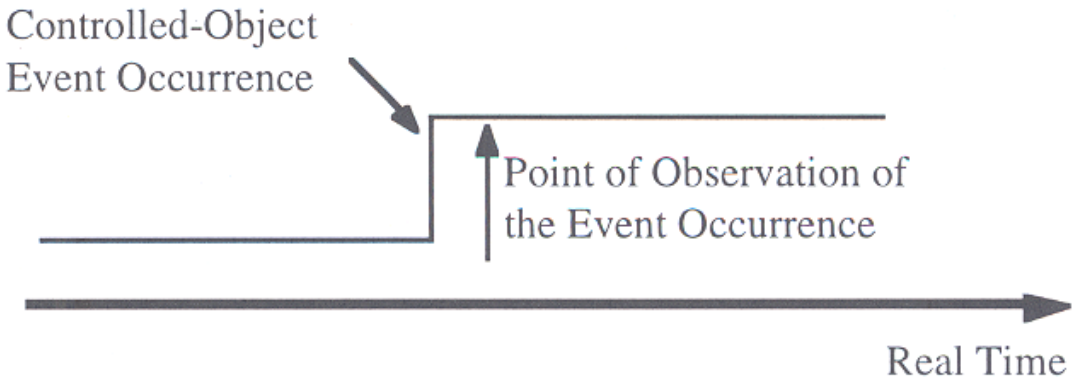
An observation is a state observation if the value of the observation contains the state of the RT entity. The time of the state observation refers to the point in real-time when RT entity was sampled.

Clients are typically only interested in the most recent value of the state variable.

Event Observation

An event is an occurrence (a state change) that happens at a point in time. Since an observation is also an event, it is not possible to observe an event.

Therefore, an observation is an event observation when it contains the change in value between the old and the new states. The time of the event observation is just a best estimate, typically the time of the L_event of the new state.



Problems with event observations:

5.2 Observations

- No precise time of the event occurrence (interrupt delay, sampling period in time-triggered approach).
- Value contains difference between old and new state (no absolute state). Loss or duplication of single event causes loss of state synchronization between state of observer and state of receiver.
- Event observations are only sent when RT entity changes its value. Thus latency for failure of observer node cannot be bounded; receiver just assumes there is no change in value.

On the other hand, event observations are more efficient than state observation when changes only occur infrequently.

5.3 Real-Time Images and Real-Time Objects

Real-Time Images

A real-time image is a current picture of a real-time entity.

An RT image is valid at a given point in time if it is an accurate representation of the corresponding RT entity.

Contrary to an observation, an RT image looses validity as time progresses.

RT images can be constructed from state observations and event observations. They can also be estimated by a technique called “state estimation”.

RT images are either stored inside the computer system or in the environment (e.g. in an actuator).

Real-Time Objects

A real-time object is a container within a node holding an RT image or an RT entity. With every RT object a real-time clock with a specified granularity is associated.

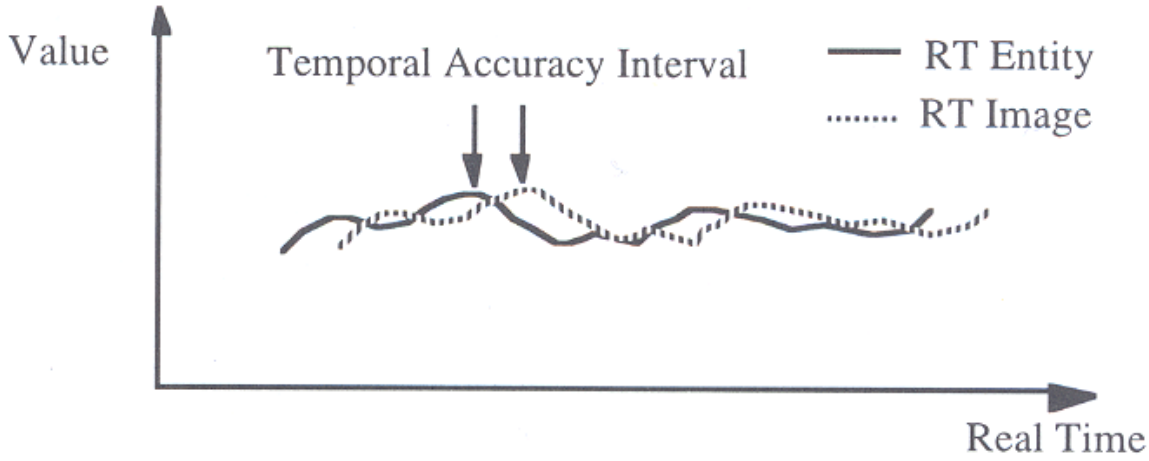
Whenever this object clock ticks, a temporal control signal is relayed to the object to activate an object procedure. If there is no other way to activate an object procedure, the object is called a synchronous RT object.

An RT object can be replicated in a distributed system, with each local site having its own version of the RT object.

Example: global time, each node has a local clock object representing global time.

5.4 Temporal Accuracy

Temporal accuracy d_{acc} is the delay between when a value occurs in the RT entity and when it appears in the corresponding RT image.



Temporal Accuracy Interval

The size of the permissible temporal accuracy interval is determined by the dynamics of the RT entity in the controlled object. The delay between observation of the RT entity and the use of the RT image causes an error:

$$error(t) = \frac{dv(t)}{dt} \left| z(t_{use}) - z(t_{obs}) \right|$$

5.4 Temporal Accuracy

If a temporally valid RT image is used, the worst case error is

$$error = \left\lceil \max_{\forall t} \frac{dv(t)}{dt} d_{acc} \right\rceil$$

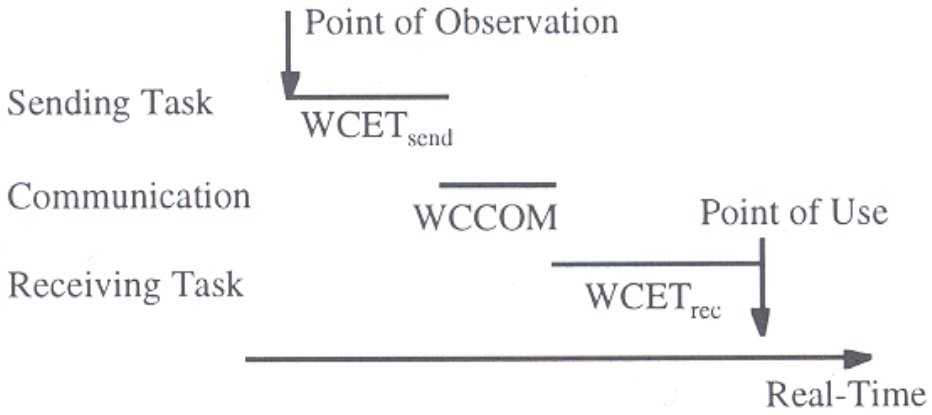
The worst case error should be in the same order of magnitude as the worst-case measurement error in the value domain (e.g. about 0.5% for analogue values measured with 8-bit precision).
For the result to be accurate, it must be based on a temporally accurate RT image, i.e.:

$$z(t_{obs}) \leq z(t_{use}) \leq z(t_{obs}) + d_{acc}$$

From this follows

$$z(t_{use}) - z(t_{obs}) \leq d_{acc}$$

Phased-aligned transaction



Example engine controller

5.4 Temporal Accuracy

Example for different temporal accuracy intervals in an automotive engine controller for an engine with a maximum of 6000 revolutions per minute (rpm).

RT image within computer	Max. change	Accuracy	d_{acc}
Position of piston within cylinder	6000 rpm	0.1°	3 µs
Position of accelerator pedal	100%/s	1%	10 ms
Engine load	50%/s	1%	20 ms
Temperature of the oil and the coolant	10%/min	1%	6 s

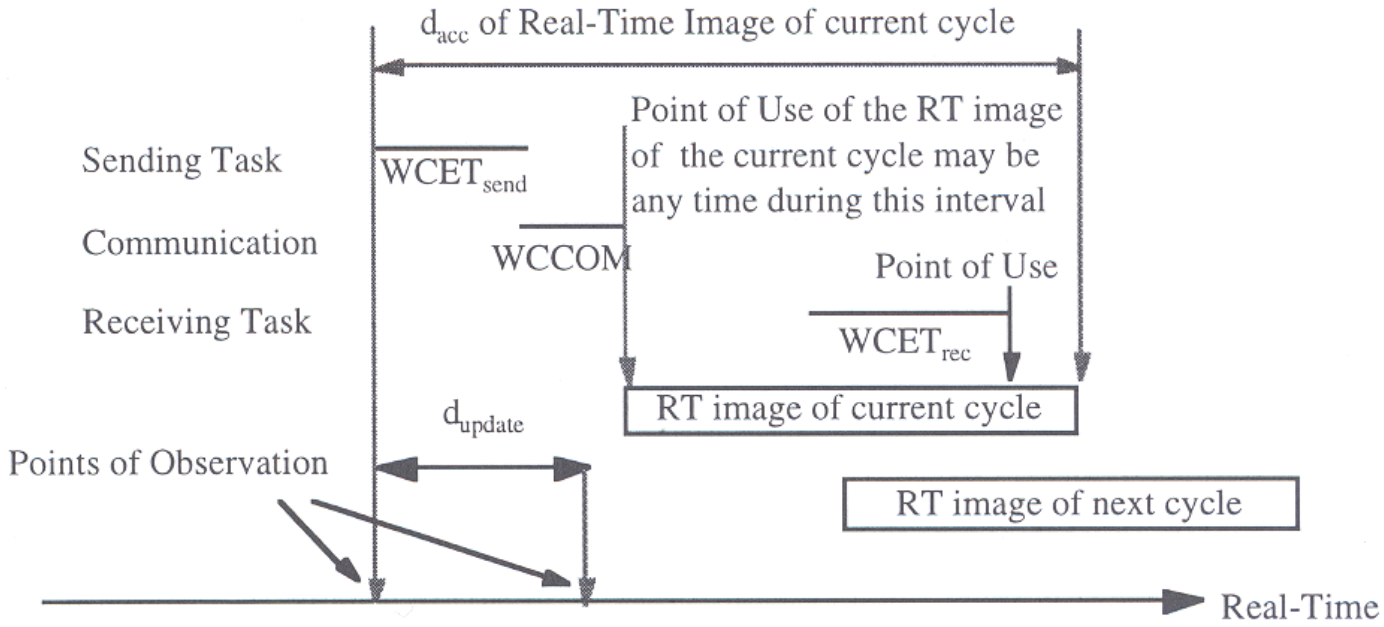
In this is example, it is clear that it will be difficult to obtain observations within an temporal accuracy of 3 µs. In this case a technique called “state estimation” is inevitable to solve this temporal accuracy problem.

Classification of Real-Time Images

Parametric RT image: Assume that RT image is updated periodically with an update period of d_{update} and that transaction is phase aligned at sender. The RT image is called parametric or phase insensitive if

$$d_{acc} > \left\lceil d_{update} + WCET_{send} + WCCOM + WCET_{rec} \right\rceil$$

5.4 Temporal Accuracy



A parametric RT image can be accessed at the receiver at any time without having to consider the phase relationship between the incoming observation message and the point of use of the data.

Example: the RT transaction that handles the position of the accelerator pedal (observation and preprocessing at sender, communication to receiver, processing at the receiver and output to the actuator) takes amount of time

$$WCET_{send} + WCCOM + WCET_{rec} = 4ms$$

Accuracy interval is 10 ms (see table on previous slide), thus messages sent with periods less than 6 ms make this RT image parametric.

5.4 Temporal Accuracy

Phase-sensitive RT image: assume an RT transaction that is phase aligned at sender. The RT image at the receiver is called phase sensitive if

$$d_{acc} \leq \lfloor d_{update} + WCET_{send} + WCCOM + WCET_{rec} \rfloor \text{ and}$$
$$d_{acc} > \lfloor WCET_{send} + WCCOM + WCET_{rec} \rfloor$$

In this case, the phase relationship between the moment at which the RT image is updated, and the moment at which the information is used, must be considered (e.g. update period of 8 ms in example above).

Phase-sensitive RT images impose additional constraints on the scheduling of real-time tasks that use this RT image.

Solution: increase update frequency of RT image, or deploy state estimation model to extend temporal accuracy interval.

State Estimation

With state estimation a model of an RT entity is built inside an RT object. With model the probable state of an RT entity at a future point in time can be computed, and the RT image updated accordingly. Using this technique, the temporal accuracy can be extended.

The control signal for execution of the model is derived from the tick of the real-time clock of the RT object.
Example: assume the crankshaft in an engine rotates with a rotational speed of 3000 rpm, i.e. 18 degrees per millisecond. If the time interval between point of observation t_{obs} and point of use t_{use} is 500 µs, we can advance the RT image by 9 degrees to arrive at an estimate of the position of the crankshaft at t_{use}.

5.4 Temporal Accuracy

Building a state estimation model of an RT entity requires that it is being governed by a well known and regular process, e.g. a physical or chemical process. If the behaviour of an RT entity is determined by chance events, state estimation cannot be used.

Input to the state estimation model: Most important dynamic input is length of time interval $[t_{obs}, t_{use}]$. Because t_{obs} and t_{use} are usually recorded at different nodes of a distributed system, a communication protocol with minimal jitter or a global time-base with a good precision is a prerequisite for state estimation.

If behaviour of RT entity can be described by a continuous, differentiable function v(t), the first derivative dv/dt is sometimes sufficient to obtain a reasonable estimate of the state of the RT entity at the point tuse:

$$v(t_{use}) \approx v(t_{obs}) + (t_{use} - t_{obs}) dv/dt$$

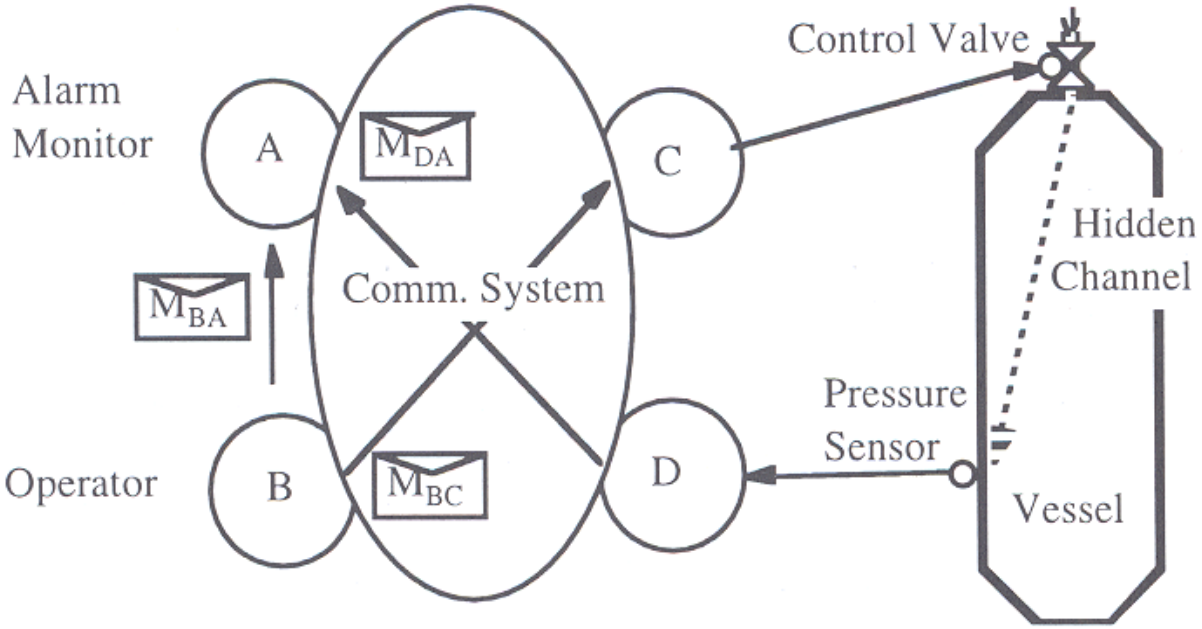
If precision does not suffice, a more elaborate series expansion around t_{obs} can be carried out, or another more detailed mathematical model can be employed.
These models must be computed on the fly, and may demand significant computing resources.

5.5 Permanence and Idempotency

Permanence

A particular message becomes permanent at a given node at that point in time when the node knows that all the messages that have been sent to it prior to the send time of this message have arrived or will never arrive.

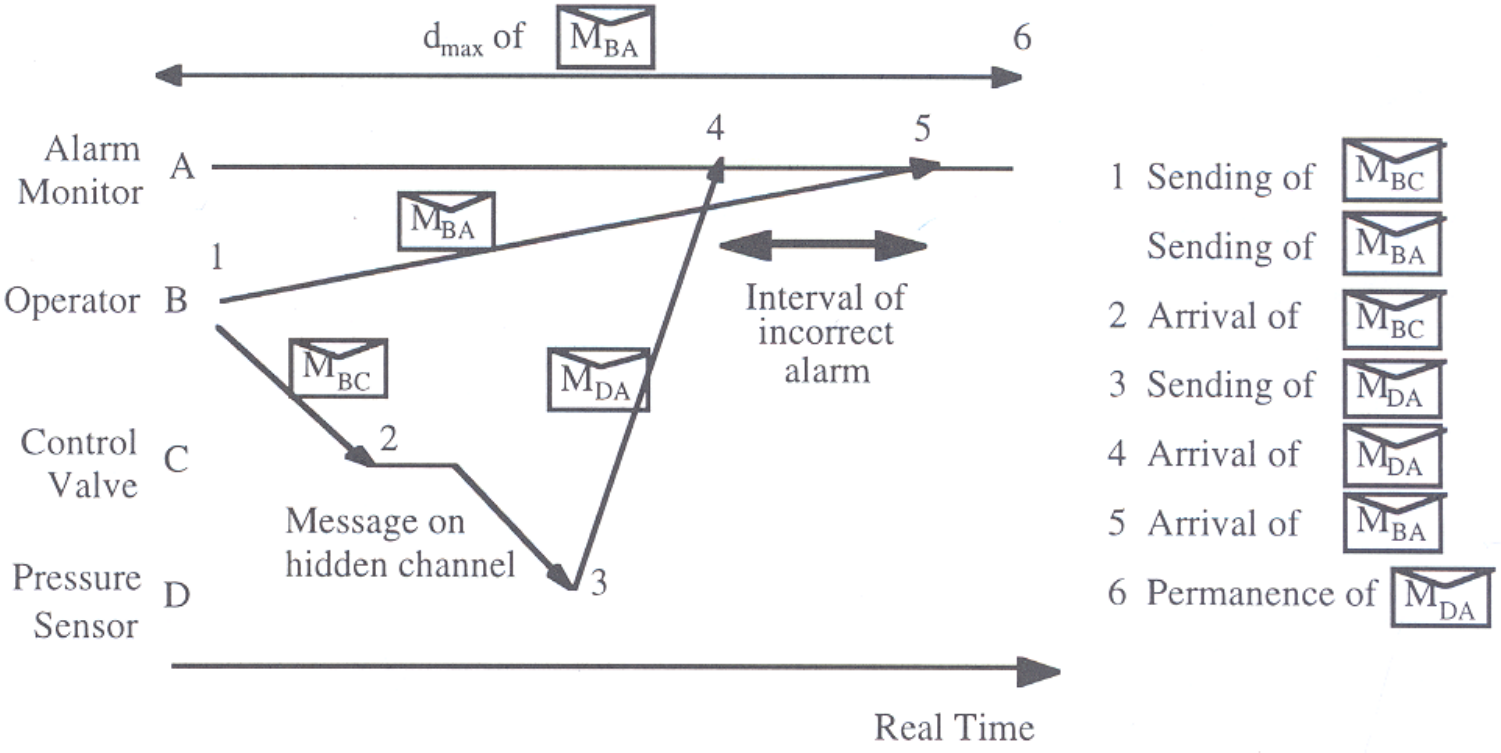
Example:



Pressure in vessel is monitored by distributed system. Alarm monitoring node A receives a message M_{DA} from the pressure sensor node D whenever there is a pressure change.

5.5 Permanence and Idempotency

If the pressure changes abruptly for no apparent reason, the alarm monitoring node A should raise an alarm. If operator at node B sends a message M_{BC} to node C to open the control valve in order to release pressure, this message should also go to node A so that node A will not raise an alarm due to the anticipated drop in pressure.



When communication system has jitter with minimum and maximum protocol execution times, MDA only becomes permanent at point 6. Alarms should not be raised until that point.

5.5 Permanence and Idempotency

Action delay: time interval between the start of transmission of a given message and the point in time when this message becomes permanent at the receiver.

The receiver must delay any action on the message until after the action delay has passed to avoid incorrect behaviour.

Irrevocable action: an action that cannot be undone. Example: activation of an airbag. It is important that an irrevocable action is triggered only after the action delay has passed.

Duration of the Action Delay

Systems with a global time: sent time t_{send} of the message can be part of the message, and can be interpreted by receiver. If receiver knows that maximum delay of communication system is $d_{\max}$, then receiver can infer that message will become permanent at

$$t_{permanent} = t_{send} + d_{\max} + 2g$$

where g is the granularity of the global time-base.

Systems without a global time: receiver does not know when the message has been sent. To be on the safe side, receiver must wait $d_{\max} - d_{\min}$ time units after the arrival of the message. Thus, in the worst case the receiver has to wait for an amount of time

$$t_{permanent} = t_{send} + 2d_{\max} - d_{\min} + g_l$$

where g_l is the granularity of the local time-base.

Accuracy Interval versus Action Delay

5.5 Permanence and Idempotency

An RT image may only be used if the message that transported the image is permanent, and the image is temporally accurate. Both conditions are only satisfied in time window $\left(t_{permanent}, t_{obs} + d_{acc}\right)$. If an implementation cannot meet the temporal requirements of the application, then state estimation may be only alternative to design a correct real-time system.

Idempotency

A set of replicated messages is idempotent if effect of receiving more than one copy of the message is the same as receiving a single copy. If messages are idempotent, implementation of fault tolerance by means of replicating message is simplified.

Typically, state messages are idempotent, while event messages are not.

5.6 Replica Determinism

A set of replicated RT objects is replica-determinate if all the members of this set have the same externally visible h-state, and produce the same output messages at points in time that are at most an interval of *d* time units apart.

A set of nodes is replica determinate, if all the nodes in this set contain the same externally visible h-state at their ground state, and produce the same output messages at points in time that are at most an interval of *d* time units apart.

In a fault-tolerant system, the time interval d determines the time it takes to replace a message or an erroneous message from a node by a correct message from redundant replicas.

Replica-determinism is required

- to implement fault-tolerance by active redundancy. Switchover from one replica to another must not upset the controlled object.
- to facilitate system test. A replica determinate system always produces identical results, in value domain and time domain, from the same input data presented at exactly the same relative points in time.

5.6 Replica Determinism

Basic Causes of Replica Non-determinism

Differing inputs: Due to digitalization errors when mapping from the continuous value domain onto a discrete value domain, e.g. for temperature, pressure. Same in the temporal domain, where external time is dense and internal time is discrete. If events that are occur on a dense time-base are observed in a different order by two replicas, then significantly different computational trajectories could develop.

Deviations of computational progress relative to physical time: The relationship between computational progress and physical time cannot be guaranteed if computer oscillator and real-time clock oscillator are driven by different resonators or if computer randomly takes different retry actions in case of minor hardware faults (e.g. hardware controlled instruction fetch retry). Whenever a program reads the real-time clock, different values are read, and thus different decision could be made.

Oscillator drift: No two physical oscillators have the same drift. If a decision involves local time (for example in case of a time-out), this could lead to a non-determinate outcome of replicated nodes.

Preemptive scheduling: Interrupts may be recognized at different points in time for different replicas. Thus the interrupting process may see different states at the two replicas at the point of interruption. This could lead to different results at a major decision point.

Nondeterministic language features: Some programming languages contain non-deterministic constructs, e.g. in ADA the SELECT statement. Since the language does not define which alternative is to be taken, the implementation of the language determines the course of action.

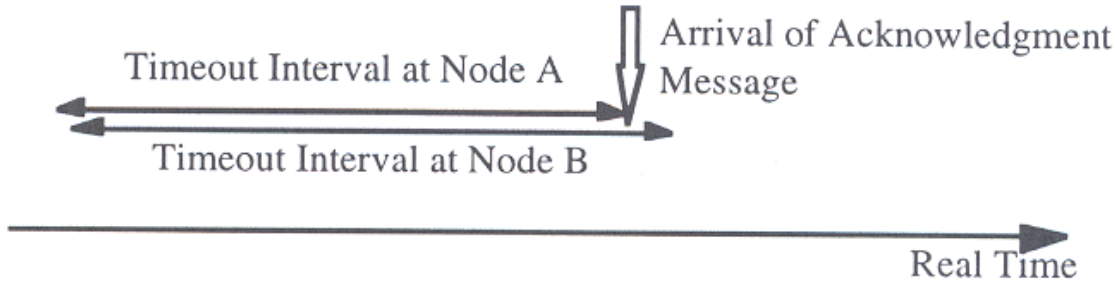
5.6 Replica Determinism

Race conditions: The wait statement in a C-task can also give rise to non-determinism, because of the uncertain outcome regarding the process which will win the race for a semaphore. The same holds for communication protocols that resolve media-access conflicts by reference to a random number generator, such as Ethernet, or by relying on the outcome of non-determinate temporal decisions, such as CAN.

Consistent comparison problem: If different software versions are deployed in replicas to tolerate design faults in the software, the order of operations in case of inexact algorithms such as for floating point arithmetic must be considered.

Building a Replica Determinate System

Sparse time-base: A sparse global time-base makes it possible to assign a significant event to the same global clock tick at all the replicas. Reference to a local real-time clock of a node can lead to replica non-determinism. This means that no local time-outs may be used in any part of the software, including application software, operating system, and the communication software.



Example: node B sees the acknowledge signal before its time-out, node A after its time-out.

5.6 Replica Determinism

Agreement on input: In case of redundant observation of an RT entity, an agreement protocol must be executed to reach a consistent view of the exact digital value of the observation, and the exact point in time on the sparse time-base.

Static control structure: Data-independent static control structure good choice for implementation of replica determinate software. No interrupts from the controlled object is allowed to occur, all inputs are periodically sampled by a trigger task. If interrupts cannot be avoided due to stringent timing requirements, all possibilities of task preemption must be statically analyzed to ensure replica determinism is maintained. Nonpreemptive dynamic scheduling avoids the problems of unpredictable task interference.

Deterministic algorithms: Any constructs that could lead to non-determinate results must be avoided. This involves dynamic synchronization constructs that could lead to race conditions, such as a semaphore wait operation, or floating point arithmetic in case of software diversity.

5.6 Replica Determinism

Points to Remember