Chapter 5
Temporal Relations

Overview The behavior of a real-time cluster must be based on timely information about the state of its physical environment and the state of other cooperating clusters. Real-time data is *temporally accurate* for a limited real-time interval only. If real-time data is used outside this application specific time interval, the system will fail. It is the objective of this chapter to investigate the temporal relations among state variables in the different parts of a cyber-physical system.

In this chapter, the notions of a *real-time* (*RT*) *entity*, a *real-time* (*RT*) *image*, and a *real-time* (*RT*) object are introduced and the concept of *temporal validity* of an RT image is established. The temporal validity of the RT image can be extended by state estimation. A real-time clock is associated with every RT object. The object's clock provides periodic temporal control signals for the execution of the object procedures, particularly for state estimation purposes. The granularity of the RT object's clock is aligned with the dynamics of the RT entity in the controlled object that is associated with the RT object. The notions of parametric and phase-sensitive observations of RT entities are introduced, and the concept of *permanence* of an observation is discussed. The duration of the action delay, which is the time interval between the transmission of a message and the instant when this message becomes *permanent*, is estimated.

The final section of this chapter is devoted to an elaboration of the concept of determinism. Determinism is a desired property of a computation that is needed if fault-tolerance is to be achieved by the replication of components. Determinism is also helpful for testing and for understanding the operation of a system. *A set of replicated RT objects is replica determinate* if the objects visit the same state at approximately the same future point in time. The main causes for a loss of determinism are failures that the fault-tolerance mechanisms are intended to mask and *non-deterministic design constructs* which must be avoided in the design of deterministic systems.

H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Real-Time Systems Series, DOI 10.1007/978-1-4419-8237-7_5,

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 of the computer system or in the computer system itself. Examples of RT entities are the flow of a liquid in a pipe, the setpoint of a control loop that is selected by the operator, or the intended position of a control valve. An RT entity has static attributes that do not change during the lifetime of the RT entity, and dynamic attributes that change with the progression of real-time. Examples of static attributes are the name, the type, the value domain, and the maximum rate of change. The value set at a particular instant is the most important dynamic attribute. Another example of a dynamic attribute is the rate of change at a chosen instant.

5.1.1 Sphere of Control

Every RT entity is in the sphere of control (SOC) of a subsystem that has the authority to set the value of the RT entity [Dav79]. Outside its SOC, the RT entity can only be observed, but the *semantic content* of the RT entity cannot be modified. At the chosen level of abstraction, syntactic transformations of the representation of the value of an RT entity that do not change its *semantic content* (see Sect. 2.2.4) are disregarded.

Example: Figure 5.1 shows another view of Fig. 1.8 and represents the small control system that controls the flow of a liquid in a pipe according to a setpoint selected by the operator. In this example, there are three RT entities involved: the *flow in the pipe* is in the SOC of the controlled object, the *setpoint* for the flow is in the SOC of the operator, and the *intended position* of the control valve is in the SOC of the computer system.

5.1.2 Discrete and Continuous Real-Time Entities

An RT entity can have a discrete value set (*discrete RT entity*) or a continuous value set (*continuous RT entity*). If the timeline proceeds from left to right, then the

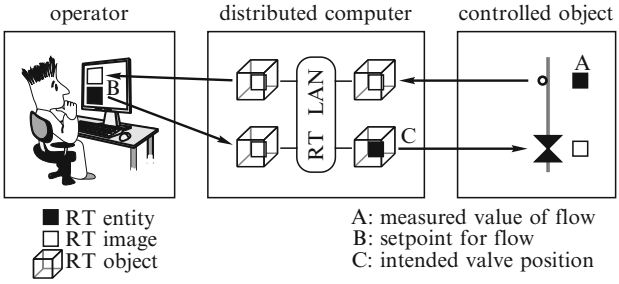

Fig. 5.1 RT entities, RT images, and RT objects

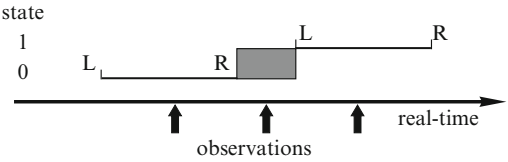

Fig. 5.2 Discrete RT entity

value set of a discrete RT entity is constant during an interval that starts with a left event (*L_event*) and ends with a right event (*R_event*) – see Fig. 5.2.

In the interval between an R_event and the next L_event, the set of values of a discrete RT entity is undefined. In contrast, the set of values of a continuous RT entity is always defined.

Example: Consider a garage door. Between the defined states specified by *door closed* and *door open*, there are many intermediate states that can be classified neither as *door open* nor as *door closed*.

5.2 Observations

The information about the state of an RT entity at a particular instant is captured by the notion of an *observation*. An observation is an *atomic data structure*

$$Observation = <Name,\ t_{obs},\ Value>$$

consisting of the name of the RT entity, the instant when the observation was made (t_{obs}), and the observed value of the RT entity. A continuous RT entity can be observed at any instant while the observation of a discrete RT entity gives a meaningful value only in the interval between a *L_event* and an *R_event* (see Fig. 5.2).

We assume that an *intelligent sensor node* is associated with a *physical sensor* to capture the physical signal, to generate the timestamp, and to transform the physical signal to meaningful digital technical units. An observation should be transported in a single message from this sensor node to the rest of the system because the message concept provides for the needed atomicity of an observation message.

5.2.1 Untimed Observation

In a distributed system without global time, a timestamp can only be interpreted within the scope of the node that created the timestamp. The timestamp of a sender that made an observation is thus meaningless at the receiver of the observation message if no global time is available. Instead, the time of arrival of an untimed observation message at the receiver node is often taken to be the time of observation t_{obs}. This timestamp is imprecise because of the delay and the jitter between the

instant of observation and the arrival instant of the message at its destination. In a system with a significant jitter of the execution time of the communication protocol (in comparison to the median execution time) and without access to a global time-base it is not possible to determine the instant of observation of an RT entity precisely. This imprecision of time measurement can reduce the quality of the observation (see Fig. 1.6).

5.2.2 Indirect Observation

In some situations, it is not possible to observe the value of an RT entity directly. Consider, for example, the measurement of the internal temperature within a slab of steel. This internal temperature (the value of the RT entity) must be measured indirectly.

The three temperature sensors T_1, T_2, and T_3 measure the change of temperature of the surface (Fig. 5.3) over a period of time. The value of the temperature T within the slab and the instant of its relevance must be inferred from these surface measurements by using a mathematical model of heat transfer.

5.2.3 State Observation

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 the RT entity was sampled (observed). Every reading of a state observation is self-contained because it carries an *absolute value*. Many control algorithms require a sequence of equidistant state observations, a service provided by periodic time-triggered readings.

The semantics of state observations matches well with the semantics of the state messages introduced in Sect. 4.3.4. A new reading of a state observation replaces the previous readings because clients are normally interested in the most recent value of a state variable.

5.2.4 Event Observation

An *event* is an occurrence (a state change) that happens at an instant. Because an observation is also an event, it is not possible to observe an event in the controlled

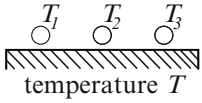

Fig. 5.3 Indirect measurement of an RT entity

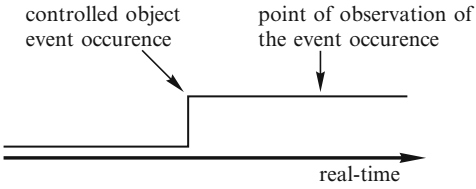

Fig. 5.4 Observation of an event

object directly. It is only possible to observe the *consequences of the controlled object's event* (Fig. 5.4), i.e., the subsequent state. An observation is an *event observation* if it contains the *change in value* between the *old* and the *new* states. The instant of the event observation denotes the best estimate of the instant of the occurrence of this event. Normally, this is the time of the L-event of the new state.

There are a number of problems with event observations:

1. Where do we get the precise time of the event occurrence? If the event observation is event-triggered, then, the time of event occurrence is assumed to be the rising edge of the interrupt signal. Any delayed response to this interrupt signal will cause an error in the timestamp of the event observation. If the event observation is time-triggered, then, the time of *event occurrence* can be at any point within the sampling interval.
2. Since the value of an event observation contains the *difference* between the old state and the new state (and not the absolute state), the loss or duplication of a single event observation causes the loss of state synchronization between the state of the observer and the state of the receiver. From the point of view of reliability, event observations are more fragile than state observations.
3. An event observation is only sent if the RT entity changes its value. The latency for the detection of a failure (e.g., a *crash*) of the observer node cannot be bounded because the receiver assumes that the RT entity has not changed its value if no new event message arrives.

On the other hand, event observations are more *data-efficient* than state observations in the case where the RT entity does not change frequently.

5.3 Real-Time Images and Real-Time Objects

5.3.1 Real-Time Images

A real-time (RT) image is a *current* picture of an RT entity. An RT image is *valid* at a given instant if it is an accurate representation of the corresponding RT entity, both in the value and the time domains. The notion of *temporal accuracy* of an RT image will be discussed in detail in the next section. While an observation records a fact that remains valid forever (a statement about an RT entity that has been observed at a particular instant), the validity of an RT image is *time-dependent*

and thus invalidated by the progression of real-time. RT images can be constructed from up-to-date state observations or from up-to-date event observations. They can also be estimated by a technique called *state estimation* that will be discussed in Sect. 5.4.3. RT images are stored either inside the computer system or in the environment (e.g., in an actuator).

5.3.2 Real-Time Objects

A real-time (RT) object is a *container* within a node of the distributed computer system that holds an RT image or an RT entity [Kop90]. A real-time clock with a specified granularity is associated with every RT object. Whenever this object clock ticks, a temporal control signal is relayed to the object to activate an object procedure [Kim94].

Distributed RT Objects. In a distributed system, an RT object can be replicated in such a manner that every local site has its own version of the RT object to provide the specified service to the local site. The quality of service of a distributed RT object must conform to some specified consistency constraints.

Example: A good example of a distributed RT object is *global time*; every node has a local clock object that provides a synchronized time service with a specified precision Π (quality of service attribute of the internal clock synchronization). Whenever a process reads its local clock, it is guaranteed that a process running on another node that reads its local clock at the same instant will get a time value that differs by at most one tick.

Example: Another example of a distributed RT object is a *membership service* in a distributed system. A *membership service* generates consistent information about the state (operational or failed) of all nodes of the system at agreed instants (*membership points*). The length and the jitter of the interval between a membership point and the instant when the consistent membership information is known at the other nodes are *quality of service parameters* of the membership service. A responsive membership service has a small maximum delay between the instant of a relevant state change of a node (failure or join), and the instant at which all other nodes have been informed in a consistent manner of this state change.

5.4 Temporal Accuracy

Temporal accuracy denotes the temporal relationship between an *RT entity* and its associated *RT image*. Because an RT image is stored in an RT object, the temporal accuracy can also be viewed as a relation between an RT entity and an RT object.

5.4.1 Definition

The temporal accuracy of an RT image is defined by referring to the *recent history* of observations of the related RT entity. A recent history RH_i at time t_i is an ordered

set of instants $\{t_i, t_{i-1}, t_{i-2}, \ldots, t_{i-k}\}$, where the length of the recent history, $d_{acc} = z(t_i) - z(t_{i-k})$, is called the *temporal accuracy interval* or the *temporal accuracy* d_{acc} ($z(e)$ is the timestamp of event e generated by the reference clock z; see Sect. 3.1.2). Assume that the RT entity has been observed at every instant of the recent history. An RT image is temporally accurate at the present time t_i if

$$\exists\, t_j \in RH_i : Value\ (RT\ image\ at\ t_i) = Value\ (RT\ entity\ at\ t_j)$$

The current value of a temporally accurate RT image is a member of the set of observations in the recent history of the corresponding RT entity. Because the transmission of an observation message from the observing node to the receiving node takes some amount of time, the RT image lags behind the RT entity (see Fig. 5.5).

Example: Let us assume, that the temporal accuracy interval of a temperature measurement is 1 min. An RT-image is temporally accurate if the value contained in the RT image has been observed at most a minute ago, i.e., it is still in the recent history of the corresponding RT-entity.

Temporal Accuracy Interval. The size of the admissible temporal accuracy interval d_{acc} is determined by the dynamics of the RT entity in the controlled object. The delay between the observation of the RT entity and the use of the RT image causes an $error(t)$ of the RT image that can be approximated by the product of the gradient of the value v of the RT entity multiplied by the length of the interval between the instant of observation and the instant of its use (see also Fig. 1.6):

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

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

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

is given by the product of the maximum gradient and the temporal accuracy d_{acc}. In a balanced design, this worst-case error caused by the temporal delay is in the same order of magnitude as the worst-case measurement error in the value domain and is typically a fraction of a percentage point of the full range of the measured variable.

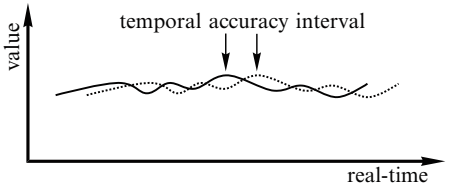

Fig. 5.5 Time lag between RT entity and RT image

If the RT entity changes its value quickly, a short accuracy interval must be maintained. Let us call t_{use} the instant when the result of a computation using an RT image is applied to the environment. For the result to be accurate, it must be based on a temporally accurate RT image, i.e.:

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

where d_{acc} is the accuracy interval of the RT image. If this important condition is transformed, it follows that:

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

Phase-Aligned Transaction. Consider the case of an RT transaction consisting of the following *phase synchronized tasks*: the computational task at the sender (observing node) with a worst-case execution time $WCET_{send}$, the message transmission with a worst-case communication delay *WCCOM*, and the computational task at receiver (actuator node) with a worst-case execution time $WCET_{rec}$ (Fig. 5.6, see also Fig. 3.9). Such a transaction is called a *phase-aligned transaction*.

In such a transaction, the worst-case difference between the point of observation and the point of use,

$$(t_{use} - t_{obs}) = WCET_{send} + WCCOM + WCET_{rec},$$

is given by the sum of the worst-case execution time of the sending task, the worst-case communication delay, and the worst-case execution time of the receiving task that uses the data in the output of a setpoint to the actuator in the controlled object. If the temporal accuracy d_{acc} that is required by the dynamics of the application is smaller than this sum, the application of a new technique, *state estimation*, is inevitable in solving the temporal accuracy problem. The technique of state estimation is discussed in Sect. 5.4.3.

Example: Let us analyze the required temporal accuracy intervals of the RT images that are used in a controller of an automobile engine (Table 5.1) with a maximum rotational speed of 6,000 revolutions per minute (rpm). There is a difference of more than six orders of magnitude in the temporal accuracy intervals of these RT images. It is evident that the d_{acc} of the first data element, namely the position of the piston within the cylinder, requires the use of state estimation.

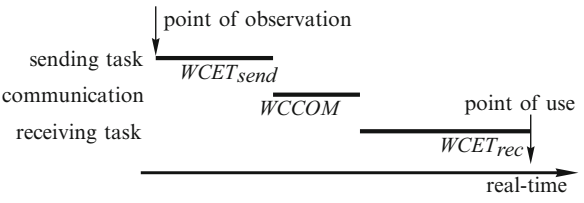

Fig. 5.6 Synchronized actions

5.4.2 Classification of Real-Time Images

Parametric RT Image. Assume that an RT image is updated periodically by a state observation message from the related RT entity with an update period d_{update} (Fig. 5.7) and assume that the transaction is phase aligned at the sender. If the temporal accuracy interval d_{acc} satisfies the condition

$$d_{acc} > (d_{update} + WCET_{send} + WCCOM + WCET_{rec}),$$

then we call the RT image *parametric* or *phase insensitive*.

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 the receiver, processing at the receiver and output to the actuator) takes an amount of time:

$$WCET_{send} + WCCOM + WCET_{rec} = 4 \text{ ms}.$$

Because the accuracy interval of this observation is 10 ms (Table 5.1), messages sent with periods less than 6 ms will make this RT image *parametric*.

If components are replicated, then care must be taken that all replicas access the same version of a parametric RT image, otherwise the *replica determinism* (see Sect. 5.6) will be lost.

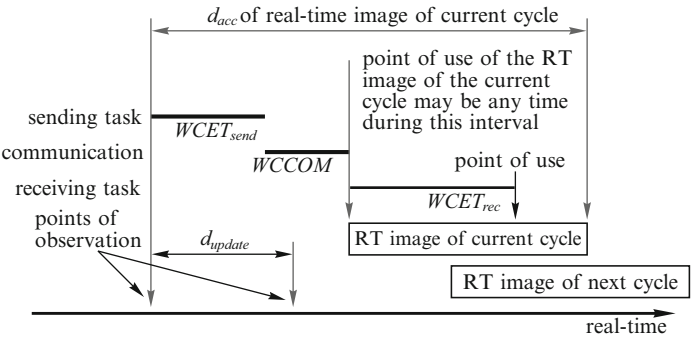

Fig. 5.7 Parametric real-time image

Table 5.1 Temporal accuracy intervals in engine control

RT image within computer	Max. change	Accuracy	d_{acc}
Position of piston within cylinder	6,000 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

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

$$d_{acc} \leqslant (d_{update} + WCET_{send} + WCCOM + WCET_{rec})$$

and

$$d_{acc} > (WCET_{send} + WCCOM + WCET_{rec})$$

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. In the above example, an update period of more than 6 ms, e.g., 8 ms, would make the RT image phase sensitive.

Every phase-sensitive RT image imposes an additional constraint on the scheduling of the real-time task that uses this RT image. The scheduling of a task that accesses phase-sensitive RT images is thus significantly more complicated than the scheduling of tasks using parametric RT images. It is good practice to minimize the number of RT images that are phase-sensitive. This can be done, within the limits imposed by d_{update}, by either increasing the update frequency of the RT image, or by deploying a state-estimation model to extend the temporal accuracy of the RT image. While an increase in the update frequency puts more load on the communication system, the implementation of a state-estimation model puts more load on the processor. A designer has the choice to find a tradeoff between utilizing communication resources and processing resources.

5.4.3 State Estimation

State estimation involves the building of a model of an RT entity inside an RT object to compute the probable state of an RT entity at a selected future instant and to update the corresponding RT image accordingly. The state estimation model is executed periodically within the RT object that stores the RT image. The control signal for the execution of the model is derived from the tick of the real-time clock that is associated with the RT object (see Sect. 5.3.2). The most important future instant where the RT image must be in close agreement with the RT entity is t_{use}, the instant where the value of the RT image is used to deliver an output to the environment. State estimation is a powerful technique to extend the temporal accuracy interval of an RT image, i.e., to bring the RT image into better agreement with the RT entity.

Example: Assume that the crankshaft in an engine rotates with a rotational speed of 3,000 rpm, i.e., 18°/ms. If the time interval between the instant of observation, t_{obs}, of the position of the crankshaft and the instant of use, t_{use}, of the corresponding RT image is 500 μs, we can update the RT image by 9° to arrive at an estimate of the position of the crankshaft at t_{use}. We could improve our estimate if we also consider the angular acceleration or deceleration of the engine during the interval $[t_{obs}, t_{use}]$.

An adequate state estimation model of an RT entity can only be built if the behavior of the RT entity is governed by a known and regular process, i.e., a well-specified physical or chemical process. Most technical processes, such as the above-mentioned control of an engine, fall into this category. However, if the behavior of the RT entity is determined by chance events, then, the technique of state estimation is not applicable.

Input to the State Estimation Model. The most important dynamic input to the state estimation model is the precise length of the time interval $[t_{obs}, t_{use}]$. Because t_{obs} and t_{use} are normally 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. This prerequisite is an important requirement in the design of a field bus.

If the behavior of an RT entity can be described by a continuous and differentiable function $v(t)$, the first derivative dv/dt is sometimes sufficient in order to obtain a reasonable estimate of the state of the RT entity at the instant t_{use} in the neighborhood of the instant of observation:

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

If the precision of such a simple approximation is not adequate, a more elaborate series expansion around t_{obs} can be carried out. In other cases a more detailed mathematical model of the process in the controlled object may be required. The execution of such a mathematical model can demand considerable processing resources.

5.4.4 Composability Considerations

Assume a time-triggered distributed system where an RT entity is observed by the sensor node, and the observation message is then sent to one or more nodes that interact with the environment. The length of the relevant time interval $[t_{obs}, t_{use}]$ is thus the sum of the delay at the sender, given by the length $[t_{obs}, t_{arr}]$, and the delay at the receiver, given by the length $[t_{arr}, t_{use}]$, (the communication delay is subsumed in the sender delay). In a time-triggered architecture, all these intervals are static and known a priori (Fig. 5.8).

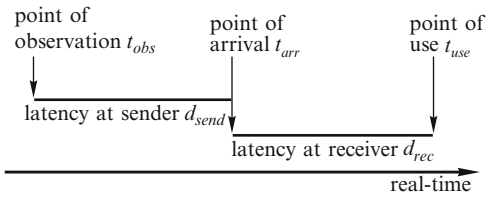

Fig. 5.8 Latency at sender and receiver

If the state estimation is performed in the RT object at the receiver, then any modification in the delay at the sender will cause a modification of the time interval that must be compensated by the state estimation of the receiver. The receiver software must be changed if a latency change takes place inside the sender node. To decrease this coupling between the sender and the receiver, the state estimation can be performed in two steps: the sender performs a state estimation for the interval $[t_{obs}, t_{arr}]$ and the receiver performs a state estimation for the interval $[t_{arr}, t_{use}]$. This gives the receiver the illusion that the RT entity has been observed at the point of arrival of the observation message at the receiver. The point of arrival is then the implicit timestamp of the observation, and the receiver is not affected by a schedule change at the sender. Such an approach helps to unify the treatment of sensor data that are collected via a field bus as well as directly by the receiving node.

5.5 Permanence and Idempotency

5.5.1 Permanence

Permanence is a relation between a particular message arriving at a node and the set of all related messages that have been sent to this node before this particular message. A particular message becomes *permanent* at a given node at that point in time when the node knows that all related messages that have been sent to it prior to the send time of this message have arrived (or will never arrive) [Ver94].

Example: Consider the example of Fig. 5.9, where the pressure in a vessel is monitored by a distributed system. The alarm-monitoring node (node A) receives a periodic message M_{DA} from the pressure-sensor node (node D). If the pressure changes abruptly for no apparent reason, the alarm-monitoring node A should raise an alarm. Suppose that the operator node B sends a message M_{BC} to node C to open the control valve in order to release the pressure. At the same time, the operator node B sends a message M_{BA} to node A, to inform node A about the opening of the valve, so that node A will not raise an alarm due to the anticipated drop in pressure.

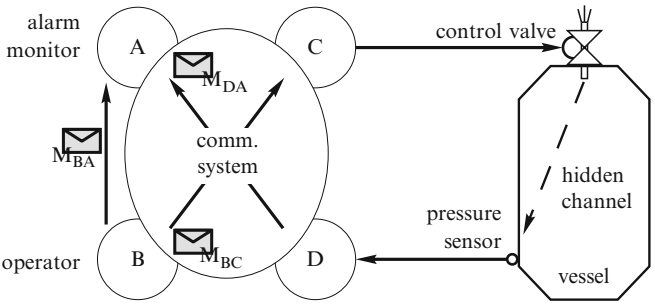

Fig. 5.9 Hidden channel in the controlled object

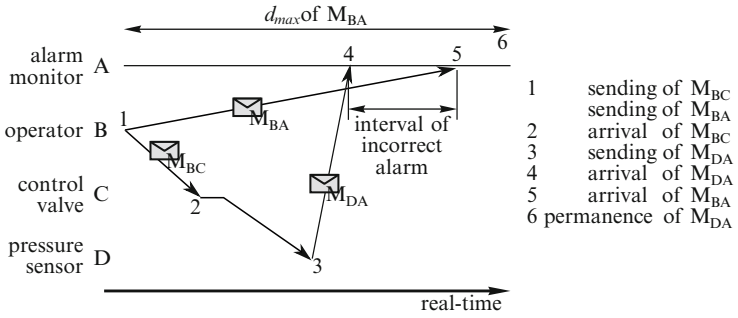

Fig. 5.10 Permanence of messages

Assume that the communication system has a minimum protocol execution time d_{min} and a maximum protocol execution time d_{max}, i.e., a jitter $d_{jit} = d_{max} - d_{min}$. Then the situation depicted in Fig. 5.10 could occur. In this figure, the message M_{DA} from the pressure sensor node arrives at the alarm monitoring node A before the arrival of the message M_{BA} from the operator (that informs the alarm monitoring node A of the anticipated drop in pressure). The transmission delay of the *hidden channel* in the controlled object between the opening of the valve and the changing of the pressure sensor is shorter than the maximum protocol execution time. Thus, to avoid raising any false alarms, the alarm-monitoring node should delay any action until the alarm message M_{DA} has become *permanent*.

Action Delay. The time interval between the start of transmission of a given message and the point in time when this message becomes permanent at the receiver, is called the *action delay*. The receiver must delay any action on the message until *after* the action delay has passed to avoid an incorrect behavior.

Irrevocable Action. An *irrevocable action* is an action that cannot be undone. An irrevocable action causes a lasting effect in the environment. An example of an irrevocable action is the activation of the firing mechanism on a firearm. It is particularly important that an irrevocable action is triggered only after the action delay has passed.

Example: The pilot of a fighter aircraft is instructed to eject from the airplane (irrevocable action) immediately after a critical alarm is raised. Consider the case where the alarm has been raised by a message that has not become permanent yet (e.g., event 4 in Fig. 5.10). In this example, the hidden channel, which was not considered in the design, is the cause for the loss of the aircraft.

5.5.2 Duration of the Action Delay

The duration of the action delay depends on the jitter of the communication system and the temporal awareness of the receiver (see also Table 7.2). Let us assume the position of the omniscient outside observer who can see all significant events.

Systems with a Global Time. In a system with global time, the send time t_{send} of the message, measured by the clock of the sender, can be part of the message, and can be interpreted by the receiver. If the receiver knows that the maximum delay of the communication system is d_{max}, then, the receiver can infer that the message will become permanent at $t_{permanent} = t_{send} + d_{max} + 2g$, where g is the granularity of the global time-base (see Sect. 3.2.4 to find out where the $2g$ comes from).

Systems without a Global Time. In a system without global time, the receiver does not know when the message has been sent. To be on the safe side, the receiver must wait $d_{max} - d_{min}$ time units after the arrival of the message, even if the message has already been d_{max} units in transit. In the worst case, as seen by the outside observer, the receiver thus has to wait for an amount of time

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

before the message can be used safely (where g_l is the granularity of the local time-base). Since in an *event-triggered communication system* ($d_{max} - d_{min} + g_l$) is normally much larger than $2g$, where g is the granularity of the global time, a system without a global time-base is significantly *slower* than a system with a global time-base. (In this case, the implementation of a time-triggered communication system is not possible, since we operate under the assumption that no global time base is available).

5.5.3 Accuracy Interval Versus Action Delay

An RT image may only be used if the message that transported the image is permanent, and the image is temporally accurate. In a system without state estimation, both conditions can only be satisfied in the time window ($t_{permanent}$, $t_{obs} + d_{acc}$). The temporal accuracy d_{acc} depends on the dynamics of the control application while ($t_{permanent} - t_{obs}$) is an implementation-specific duration. If an implementation cannot meet the temporal requirements of the application, then, state estimation may be the only alternative left in order to design a correct real-time system.

5.5.4 Idempotency

Idempotency is the relationship among the members of a set of replicated messages arriving at the same receiver. A set of replicated messages is *idempotent* if the effect of receiving more than one copy of a message is the same as receiving only a single copy. If messages are idempotent, the implementation of fault tolerance by means of replicating messages is simplified. No matter whether the receiver receives one or more of the replicated messages, the result is always the same.

Example: Let us assume that we have a distributed system without synchronized clocks. In such a system, only untimed observations can be exchanged among nodes, and the time of arrival of an observation message is taken as the time of observation. Assume a node observes an RT entity, e.g., a valve, and reports this observation to other nodes in the system. The receivers use this information to construct an updated version of the local RT image of the RT entity in their RT objects. A state message might contain the absolute value *position of valve at* 45°, and will replace the old version of the image. An event message might contain the relative value *valve has moved by* 5°. The contents of this event message are added to the previous contents of the state variable in the RT object to arrive at an updated version of the RT image. While the state message is idempotent, the event message is not. A loss or duplication of the event message results in a permanent error of the RT image.

5.6 Determinism

Determinism is a *property* of a computation that makes it possible to predict the future result of a computation, given that the *initial state* and all *timed inputs* are known. A given computation is either *determinate* or *not determinate*.

Example: Consider the case of a fault-tolerant brake-by-wire system in a car. After agreement on the sensor inputs (e.g., brake pedal position, speed of the car, etc.), three independent but synchronized channels process *identical* inputs. The three outputs are presented to four smart voting actuators (Figs. 9.8 and 9.9), one at the brake-cylinder of each one of the four wheels of the car. After the earliest possible arrival of a correct output message at a voting actuator, an *acceptance window* is opened. The duration of the acceptance window is determined by the differences in the execution speeds and the jitter of the communication system of the three channels, provided they operate correctly. Every correct deterministic channel will deliver the same result before the end of the acceptance window. If one channel fails, one of the three arriving result messages will contain a value that is different from the other two (*value failure*) or only two (identical) result messages will arrive during the *acceptance window* (*timing failure*). By selecting the majority result at the end of the *acceptance window*, the voter will mask a failure of any one of the three channels. The *end point* of the acceptance window is the *significant event* when the voting actions can be performed and the result can be transmitted to the environment. If the computations and the communication system have a large jitter, then this end point of the acceptance window is far in the future and the responsiveness of the computer system is reduced.

5.6.1 Definition of Determinism

In Sect. 2.5 the *principle of causality* has been introduced. Causality refers to the unidirectional relationship that connects an *effect* to a *cause* [Bun08]. If this relationship is one of *logical and temporal entailment*, we speak of *determinism*, which we define as follows: *A physical system behaves deterministically if, given an initial state at instant t and a set of future timed inputs, then the future states and the values and times of future outputs are entailed*. The words *time* and *instants* refer to the progression of dense (physical) time. Many natural laws of physical systems conform to this definition of determinism. In a digital computer model of a physical

system there is no dense time. In a deterministic distributed computer system, we must assume that all events, e.g., the observation of the initial state at instant t and the timed inputs, are *sparse events* on a *sparse* global time base (see Sect. 3.3) in order that the temporal properties of and the relations (such as *simultaneity*) among the events that occur in the different nodes of the distributed system can be precisely specified despite the finite precision of the clock synchronization and the discrete time-base. This transformation of dense events in the physical world to sparse events in the cyber world (the distributed computer system), performed by an agreement protocol (see Sect. 3.3.1), reduces the *faithfulness* of the computer model, since events that are closer than the granularity of the time-base cannot be ordered consistently.

In a real-time context, the concept of determinism requires that the behavior of a system is predictable in the domains of *values* and *time*. Neglecting the temporal dimension leads to a reduced notion of determinism – we call it *logical* (*L*) *determinism*. *L-determinism* can be defined as follows: *A system behaves L-deterministically if, given an initial state and a set of ordered inputs, then the subsequent states and the values of subsequent outputs are entailed.*

The use of the word *determinism* in everyday language relates the future behavior of a system as a consequence of its present state. Since in a time-less system the concept of *future* does not exist, *L-determinism* does not capture the everyday meaning of the word *determinism*.

> **Example:** In the above example of a braking system, it is not sufficient for the establishment of correctness to demand that the braking action will *eventually* take place. The maintenance of an upper real-time bound for the start of the braking action (the end point of the acceptance window), e.g., *that the braking action will start 2 ms after the brake pedal has been pressed*, is an integral part of correct behavior.

Deterministic behavior of a component is desired for the following reasons:

- An *entailment relation* between initial state, input, output and time simplifies the *understanding* of the real-time behavior of the component (see also Sect. 2.1.1).
- Two replicated components that start from the same initial state and receive the same timed inputs will produce the same results at about the same time. This property is important if the results of a faulty channel are to be masked (outvoted) by the correct results of two correct channels (see Sect. 6.4.2) as exemplified in the above example on the braking system of a car.
- The testability of the component is simplified, since every test case can be reproduced, eliminating the appearance of spurious *Heisenbugs* (see Sect. 6.1.2).

Determinism is a *desired property* of behavior. The implementation of a computation will achieve this desired property with an *estimated probability*.

An implementation can fail to meet this *desired property of determinism* for the following reasons:

1. The initial states of the computations are not precisely defined.
2. The hardware fails due to a random physical fault.

3. The notion of *time* is unclear.
4. The system (software) contains design errors or *Non-Deterministic Design Constructs* (NDDC) that lead to unpredictable behavior in the value domain or in the temporal domain.

From the point of view of fault-tolerance, every loss of determinism of a replicated channel is tantamount to a failure of that channel that eliminates the further fault-masking capability of the fault-tolerant system.

In order to realize replica-deterministic behavior in an implementation of a fault-tolerant distributed real-time computer system, we must ensure that:

- The initial state of all involved computations is defined consistently. It is *impossible* to build a replica-deterministic distributed real-time system without the establishment of some sort of a sparse global time base for the consistent time-stamping of the events in order to be able to determine whether an event is included in the initial state or not. Without a sparse global time base and sparse events, simultaneity cannot be resolved consistently in a distributed system, possibly resulting in an inconsistent temporal order of the replicated messages that report about these simultaneous events. Inconsistent ordering results in the loss of *replica determinism*.
- The assignment of events to a sparse global time-base can be established at the system level by the generation of *sparse events* or at the application level by the execution of agreement protocols which assign consistently dense events to sparse intervals.
- The message transport system among the components is *predictable*, i.e., the instants of message delivery can be foreseen and the temporal order of the received messages is the same as the temporal order of the sent messages *across all* channels.
- The computer system and the observer (user) agree on a precise notion of real-time.
- All involved computations are *certain*, i.e., there are no program constructs in the implementation that produce arbitrary results or contain NDDCs, and that the final result of a computation will be available during the anticipated acceptance window.

If any one of the above conditions is not satisfied, then the fault-masking capability of a fault-tolerant system may be reduced or lost.

5.6.2 Consistent Initial States

Correct replicated channels that are introduced to mask a failure will only produce identical results if they start from the same initial state and receive the same inputs at the same instants.

According to Sect. 4.2.1, the *state* of a component can only be defined if there is a consistent separation of *past events* from *future events*. The sparse time model, introduced in Sect. 3.3, provides for such a consistent separation of past events from future events and makes it possible to define the instants where the initial state of a distributed system is consistently defined. Without a sparse global time, the establishment of a consistent initial state of replicated components of a distributed system is difficult.

A sensor is a physical device that will eventually fail. In order to mask the failure of a sensor, multiple sensors must be provided in a fault-tolerant system that measure, either directly or indirectly, the same physical quantity. There are two reasons why redundant observations of a physical quantity by replicated sensors will deviate:

1. It is impossible to build *perfect* sensors. Every *real sensor* has a finite measurement error that limits the accuracy of the observed value.
2. The quantities in the physical world are normally *analog values*, but their representations in cyber-space are *discrete values*, leading to a *discretization error*.

It is therefore necessary to execute agreement protocols at the boundary between the physical world and cyber space in order that all replicated channels receive the *consistent* (*exactly the same*) *agreed input data* (see Sect. 9.6). These agreement protocols will present the same set of values at the same sparse time interval to all replicated channels.

5.6.3 Non-deterministic Design Constructs

A distributed computation that starts from a well-defined initial state can fail to reach the envisioned goal state for the following reasons:

1. A hardware fault or design error causes the computation to crash, to deliver an incorrect result, or to delay the computation beyond the end of the agreed temporal acceptance window. It is the goal of a fault-tolerant design to mask these kinds of failures.
2. The communication system or the clocking system fails.
3. A non-deterministic design constructs (NDDC) destroys the determinism. A loss of determinism, caused by an NDDC, eliminates the fault-masking capability of a fault-tolerant system.

The undesired effect of an NDDC can be in the *value domain* or in the *temporal domain*. A basic assumption in the design of a fault-tolerant system that masks failures by comparing the results of replica-determinate channels is the statistical independence of failures in different channels. This assumption is violated if an NDDC is the cause of the loss of determinism, because the same NDDC may appear in all replicated channels. This leads to dangerous correlated failures of the replicated channels.

The following list is indicative of constructs that can lead to a loss of determinism in the value domain (i.e., *L-determinism*):

1. *Random number generator.* If the result of a computation depends on a random number that is different for each channel, then the determinism is lost. Communication protocols that resolve a media-access conflict by reference to a random number generator, such as the bus-based CSMA/CD Ethernet protocol, exhibit *non-determinism.*
2. *Non-deterministic Language Features.* The use of a programming language with non-deterministic language constructs, such as the SELECT statement in an ADA program, can lead to the loss of replica determinism. Since the programming language does not define which alternative is to be taken at a decision point, it is left up to the implementation to decide the course of action to be taken. Two replicas may take different decisions.
3. *Major decision point.* A *major decision point* is a decision point in an algorithm that provides a choice between a set of significantly different courses of action. If the replicated components select different computational trajectories at a major decision point, then the states of the replicas will start to diverge.

   Example: Consider the case where the *result of a timeout check* determines whether a process continues or backtracks. This is an example for a major decision point.

4. *Preemptive scheduling.* If dynamic preemptive scheduling is used then the points in the computations where an external event (interrupt) is recognized may differ at the different replicas. Consequently, the interrupting processes see different states at the two replicas at the point of interruption. They may reach different results at the next major decision point.
5. *Inconsistent message order.* If the message order in the replicated communication channels is not identical, then the replicated channels may produce different results.

Most of the above constructs can also cause a loss of determinism in the temporal domain. Additionally, the following mechanisms and inadequacies must be considered that can cause a loss of the *temporal dimension of determinism*, even if the system is *L-deterministic*:

1. *Task preemption and blocking.* Task preemption and blocking extend the execution time of tasks and may delay a result until the acceptance window has been closed.
2. *Retry mechanisms.* Any retry mechanism in hardware or software leads to an extension of the execution time and can cause an unacceptable delay of a value-correct result.
3. *Race conditions.* A *semaphore wait operation* can give rise to non-determinism, because of the uncertain outcome regarding the process that will win the race for the semaphore. The same argument applies to communication protocols that resolve the access conflict by relying on the outcome of non-determinate temporal decisions, such as CAN or, to a lesser degree, ARINC 629.

In some designs where NDDCs are present, an attempt is made to reestablish replica determinism by explicitly coordinating the decisions that could lead to a loss of determinism among the replicas, e.g., by distinguishing between a *leader* process and a *follower* process [Pow91]. If at all possible, inter-replica coordination should be avoided because it compromises the independence of the replicas, and requires additional time and additional communication bandwidth.

5.6.4 Recovery of Determinism

A loss of determinism in an L-deterministic system can be avoided if the *acceptance window* is extended such that the probability of a *deadline miss* (i.e., that the result is available at the end of the acceptance window) is reduced to an acceptably low value. This technique is often used to reestablish determinism at the macro-level, even if the precise temporal behavior at the micro-level cannot be predicted. The main disadvantage of this technique is the increased delay until a result is delivered, which causes an increase in the *dead-time* of control loops and the reaction time of reactive systems.

Example: Many *natural laws* at the level of Newtonian physics are considered to be *deterministic*, although the underlying quantum-mechanical processes at the micro-level are *non-deterministic*. The abstraction of *deterministic behavior at the macro-level* is possible because the large number of involved particles and the large time-spans at the macro-level, relative to the duration of the processes at the micro-level, makes it highly improbable that non-deterministic behavior can be observed at the macro-level.

Example: In a server farm of a cloud, where more than 100,000 *L-deterministic* Virtual Machines (VM) can be active at any instant, a failed VM can be reconfigured and restarted such that the intended result is still made available within the *specified acceptance window*. Such a system will have a deterministic behavior at the *external level* (see the Four Universe Model in Sect. 2.3.1), although the implementation at the lower *informational level* behaves non-deterministically.

The recovery of determinism at the *external level* (see the *Four Universe Model* in Sect. 2.3.1) of systems that behave non-deterministically at the level of the implementation is an important strategy when developing an understandable model of the behavior of a system-of-systems at the user level.

Points to Remember

- An observation of an RT entity is an atomic triple $< Name,\ t_{obs},\ Value >$ consisting of the name of the RT entity, the point in real time when the observation was made (t_{obs}), and the observed value of the RT entity. A continuous RT entity has always a defined value set and can be observed at any instant, whereas a discrete RT entity can only be observed between the L_event and the R_event.

- An observation is a *state observation* if the value of the observation contains the absolute state of the RT entity. The time of the state observation refers to the point in real time when the RT entity was sampled.
- An observation is an *event observation* if it contains information about the *change of value* between the *old state* and the *new state*. The time of the event observation denotes the best estimate of the instant of this event.
- A *real-time* (*RT*) *image* is a *current* picture of an RT entity. An RT image is *valid* at a given instant if it is an accurate representation of the corresponding RT entity, both in the value domain and time domain.
- A *real-time* (*RT*) *object* is a container within a node of the distributed computer system holding an RT image or an RT entity. A real-time clock with a specified granularity is associated with every RT object.
- The present value of a temporally accurate RT image is a member of the set of values that the RT entity had in its recent history.
- The delay between the observation of the RT entity and the use of the RT image can cause, in the worst-case, a maximum error $error(t)$ of the RT image that can be approximated by the product of the maximum gradient of the value v of the RT entity multiplied by the length of the accuracy interval.
- Every phase-sensitive RT image imposes an additional constraint on the scheduling of the real-time task that uses this RT image.
- *State estimation* involves the building of a model of an RT entity inside an RT object to compute the probable state of an RT entity at a selected future instant, and to update the corresponding RT image accordingly.
- If the behavior of an RT entity can be described by a continuous and differentiable variable $v(t)$, the first derivative dv/dt is sometimes sufficient to get a reasonable estimate of the state of the RT entity at the point t_{use} in the neighborhood of the point of observation.
- To decrease the coupling between sender and receiver the state estimation can be performed in two steps: the sender performs a state estimation for the interval $[t_{obs}, t_{arr}]$, and the receiver performs a state estimation for the interval $[t_{arr}, t_{use}]$.
- A particular message becomes *permanent* at a given node at that instant when the node knows that all the related messages that were sent to it, prior to the send time of this message, have arrived (or will never arrive).
- The time interval between the start of transmission of a message and the instant when this message becomes permanent at the receiver is called the *action delay*. To avoid incorrect behavior, the receiver must delay any action on the message until *after* the action delay has passed.
- An RT image may only be used if the message that transported the image has become permanent, and the image is temporally accurate. In a system without state estimation, both conditions can be satisfied only in the time window $[t_{permanent}, t_{obs} + d_{acc}]$.
- No matter whether the receiver receives one or more out of set of replicated *idempotent* messages, the result will always be the same.

- *Determinism* is a *desired property of a computation* that enables the prediction of the output at a future instant on the basis of a given initial state and timed inputs.
- The basic causes for replica non-determinism are: inconsistent inputs, a difference between the computational progress and the progress of the physical time in the replicas (caused by differing oscillator drifts), and NDDCs.
- If at all possible, inter-replica coordination should be avoided because it compromises the independence of the replicas, and requires additional time and additional communication bandwidth.

Bibliographic Notes

The concept of temporal accuracy of a real-time object has been introduced in the real-time object model presented in [Kop90]. Kim has extended this model and analyzed the temporal properties of real-time applications using this model [Kim94]. The problem of replica determinism has been extensively studied in [Pol95]. An interesting philosophical treatment of the topics of causality and determinism can be found in [Bun08] and [Hoe10].

Review Questions and Problems

5.1 Give examples of RT entities that are needed to control an automotive engine. Specify the static and dynamic attributes of these RT entities, and discuss the temporal accuracy of the RT images associated with these RT entities.
5.2 What is the difference between a *state observation* and an *event observation*? Discuss their advantages and disadvantages.
5.3 What are the problems with event observations?
5.4 Give an informal and a precise definition of the concept of *temporal accuracy*. What is the *recent history*?
5.5 What is the difference between a *parametric* RT image and a *phase-sensitive* RT image? How can we create parametric RT images?
5.6 What are the inputs to a state estimation model? Discuss state estimation in a system with and without a global time-base.
5.7 Discuss the interrelationship between state estimation and composability.
5.8 What is a *hidden* channel? Define the notion of *permanence*.
5.9 Calculate the action delay in a distributed system with the following parameters: $d_{max} = 20$ ms, $d_{min} = 1$ ms:
(a) No global time available, and the granularity of the local time is 10 µs
(b) Granularity of the global time 20 µs
5.10 What is the relationship between *action delay* and *temporal accuracy*?
5.11 Define the notion of *determinism*! What is *L-determinism*?

5.12 Give an example that shows that a local time-out can lead to replica non-determinism.
5.13 What mechanisms may lead to replica non-determinism?
5.14 How can we build a replica-determinate system?
5.15 Why should explicit inter-replica coordination be avoided?
5.16 Calculate the *action delay* in the system of Fig. 5.9, considering that the nodes are connected by an AFDX protocol with the temporal parameters of Table 7.2.