15th Triennial World Congress, Barcelona, Spain

# TIME-TRIGGERED REAL-TIME COMPUTING

**H. Kopetz** *

* *Institut für Technische Informatik*
*TU Vienna, Austria*
*email: hk@vmars.tuwien.ac.at*

Abstract: Time-triggered (TT) distributed real-time computing systems are moving into the mainstream for the implementation of safety-critical applications in the aerospace and automotive sectors. This paper introduces the basic principles of a time-triggered real-time computing system, and elaborates on the benefits that can be gained from the availability of a global time base in general, and in specifying the linking interfaces of components in particular. It describes the concept of a temporal firewall that forms a fully specified operational interface of a component. The most important contribution of the TT-paradigm is the capability to precisely specify operational interfaces in the temporal domain and thus establish a sound basis for the composability of a design and the reuse of components in distributed real-time systems. *Copyright c 2002 IFAC*

Keywords: real-time, linking interface, time-triggered, global time, distributed system, composability, component reuse

## 1. INTRODUCTION

Hard real-time computer systems must produce the intended results at the intended points in real time. A failure to produce the intended result at the intended instant can entail catastrophic consequences. The correctness of a real-time system thus depends on both, its proper behavior in the value domain and in the temporal domain. It follows that any computer architecture or design methodology for real-time systems must be concerned with both issues, the issue of temporal correctness and the issue of value correctness. *Non-real-time* computer systems deal with value correctness and performance only. In these systems, temporal correctness is a *non-issue*. The focus of this paper is on distributed hard real-time computer systems.

In many computer systems a new process (either a computational process or a communication process) — we call it a *task* — is started whenever the *environment* delivers a *service request*. Such a service request can be the transmission of a message by an operator from a terminal or the generation of an interrupt by a physical device. In a distributed system, such a service request will normally result in the execution of a sequence of computational and communication tasks, where each subsequent task is started whenever the temporally preceding task has progressed beyond a certain point or finished. If we abstract from the inner logic of the tasks (the detailed data transformation logic or the detailed protocol execution) and focus only on the start instant and the termination instant of each task in a task sequence, then we get the *temporal control structure* of the task sequence (Kopetz, 1997) p.82. If the start of a task is triggered by an external event (e.g., an interrupt) from the environment or by the progress (termination) of the preceding task, then the task sequence is called *event-triggered*. If the start of the task is triggered by the progression of a global notion of time, i.e., when the global time

reaches a specified value, then the task sequence is called *time-triggered*. In a time-triggered system every task will periodically observe the state of its environment (e.g., by sampling) to determine whether a particular computational activity has to be performed. Event-triggered systems excel in flexibility, whereas time-triggered systems excel in temporal predictability.

Time-triggered computing systems have been around for many years, not only in safety critical applications where temporal predictability is of primary concern. The idea of using a periodic clock signal as the trigger for the initiation of a task is widely used in control systems, and in communication systems (e.g., time-division multiple access — TDMA — as a media access strategy). However, in many of these systems — even if they are distributed — the time-triggers are generated locally, such that each subsystem has its own (uncoordinated) time base. In recent years, the interest in system-wide synchronized time-triggers has been growing. In these distributed computing systems a (fault-tolerant) global time-base is established and used for the generation of synchronized time triggers throughout the system. We will use the term *time-triggered system* to describe a distributed computing system where a system-wide global time base of known precision is provided at every node to trigger the execution of significant computational and communication tasks.

The present paper is organized as follows. In Section 2 the requirements for distributed hard real-time computer systems are elaborated. Section 3 introduces a set of basic concepts that are useful for the understanding of time-triggered computing systems. Of particular importance are the concepts of a *sparse time base*, of *state information* versus *event information*, and of a *linking interface* . Section 4 describes the principles of operation of a time-triggered system and elaborates on the concept of a temporal firewall as a precisely specified operational interface. Section 5 discusses further benefits that can be accrued from the existence of a global time in a distributed computing system. Section 6 shows how the requirements established in Section 2 can be fulfilled within a properly designed time-triggered architecture. The paper terminates with a conclusion in Section 7.

## 2. REAL-TIME REQUIREMENTS

In this Section we discuss the important requirements that must be satisfied by a distributed hard real-time computing system.

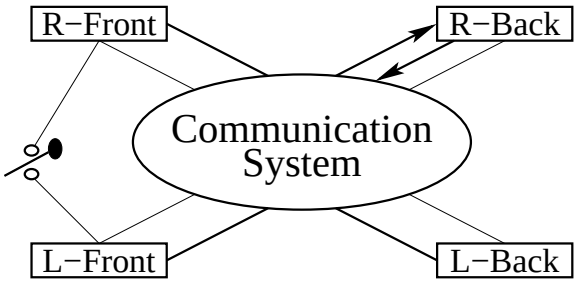


Fig. 1. Simple "brake-by-wire" application

### *2.1 Correctness and Predictability*

The foremost requirements of a hard real-time control system are value correctness and temporal correctness. Whereas value correctness of computations is the topic of many scientific contributions since a long time (e.g., (Boyer and Moore, 1981)) and will not be investigated further in this paper, the issues of temporal correctness have not received as much attention. The temporal correctness of an implementation can only be ascertained if the specification contains precise statements about the intended temporal behavior. The temporal specification of the behavior of a distributed computer system is simplified, if all nodes of the system have access to a common global notion of time.

### *2.2 Consistent Distributed Computing Base*

An important requirement of a distributed real-time computing system is to provide a consistent distributed computing base to all correct nodes in order that reliable distributed applications can be built with manageable effort (Rushby, 2001). If a node cannot be certain that every other node works on exactly the same data, then the design of distributed algorithms becomes very cumbersome (Lamport *et al.*, 1982), because the intricate agreement problem has to be solved at the application level.

Example: The simple "brake-by-wire" system in a car (Fig. 1) demonstrates the importance of a consistent view in a distributed real-time application. In this application the four nodes that control the brakes at the four wheels of a car are connected by a fault-tolerant communication system. The *R-Front* and the *L-Back* node accept the brake pedal pressure from one fail-silent brake pedal sensor, the *L-Front* and *R-Back* node accept the brake pedal pressure from the other fail-silent brake pedal sensor. Every wheel node informs all other nodes about its view of the brake pedal sensors, performs a distributed algorithm to allocate the brake force to each wheel and controls the brake at its local wheel. The brake is assumed to be designed in such a way that the brake

autonomously visits a defined state, e.g., *wheel free running — no brake force applied* in case the wheel node crashes or the electric or mechanic mechanism in the local brake fails. As soon as the other three wheels learn about the failure at one wheel, they redistribute the brake force to the remaining operational brakes in order that the car is stopped safely with three braking wheels. The time interval between the instant of a wheel brake failure and the instant of redistribution of the brake-force is a *safety critical parameter of this application*. During this error detection interval the braking system is in an inconsistent state. We conjecture that there is a potential for a fatal accident if this inconsistent state is not detected and corrected within at most a few sampling intervals. Consider the scenario where the R-back node has an outgoing link failure. In this scenario the other three nodes will assume the R-back node has failed (since they don't receive any message from the R-back node), but the R-back node will think it is operating correctly, since it receives all messages form the other nodes. This scenario illustrates the need for the prompt detection and elimination of safety relevant inconsistencies in a distributed real-time control system.

### *2.3 Composability*

The concept of composability refers to the *ease of composing a system-of-systems out of component systems*. In a distributed real-time environment we view the nodes (including their application software) as the *component systems* and the distributed system as a whole as the *system-of-systems*. In a composable real-time architecture the *linking interfaces (LIFs)* (Jones *et al.*, 2001) of the nodes must be specified in the temporal domain and in the value domain (see also Section 3.4). The reasoning about the emergent properties of the system-of-systems can then be based solely on the LIF specifications without any knowledge about the internal implementation of the components.

### *2.4 Validation*

Today, the effort required to validate the behavior of a large distributed real-time system is very significant. What are needed are architecture-based approaches, where a constructive validation is supported such that the results of the component-system validation can be used for the system-of-systems'validation. Constructive validation is thus closely related to the requirement of composability, introduced above. If, in addition to validation, the certification of a system in a safety critical application is demanded, a systematic validation methodology is even more important.

### *2.5 Fault Tolerance*

In fail-operational applications safety-critical real-time systems require the provision of fault-tolerance in the control system in order to tolerate the failure of any component. Fault tolerance will become more important in near future, since smaller VLSI structures are causing an increase in the occurrence of transient and intermittent hardware failures (Constantinescu, 2002). Ideally, an arbitrary failure of every component (possibly implemented on a single chip) of the distributed control system should be tolerated without any impact on safety critical system services. The software required to implement the fault tolerance should be strictly separated from the application software in order not to increase the complexity of the application software.

### *2.6 Extensibility*

Every successful real-time system will be modified and extended over its lifetime. The effort required to change existing functions and add new functions is captured by the notion of *extensibility*. There are two steps required to effect a change, the implementation of the new or modified function and the validation of the value and temporal properties of the modified system.

## 3. BASIC CONCEPTS

In this Section some of the basic concepts needed to describe time-triggered systems are developed. The fundamental characteristic of a time-triggered system is the strict separation of the temporal domain from the value domain of a process. The instants when a task or a protocol execution starts, is assumed to be independent from the involved data. It is further assumed that the worst-case execution time (WCET) of a task or a protocol execution is known *a priori*. It follows that the worst-case termination instant of a task or protocol execution is also known *a priori*. These instants, the start instant and the worst-case termination instant, determine the temporal control structure. The specification of this temporal control structure requires a model of time that supports the unambiguous identification of significant instants at every node of a distributed system.

### *3.1 Sparse Time*

For most applications, a model of time based on Newtonian physics is adequate. In this model,

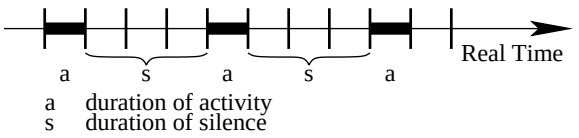


Fig. 2. Sparse time base

real-time progresses along a *dense* timeline, consisting of an infinite set of *instants*, from the past to the future. A *duration* (or *interval*) is a section of the timeline, delimited by two instants. A happening that occurs at an instant (i.e., a cut of the timeline) is called an *event*. An *observation* of the state of the world at an instant is thus an event. The *time-stamp* of an event is established by assigning the state of the local clock of the observer to the event immediately after the event occurrence. Due to the impossibility of synchronizing clocks perfectly and the denseness property of real time, there is always the possibility of the following sequence of events occurring: clock in node $j$ ticks, event $e$ occurs, clock in node $k$ ticks. In such a situation, the single event $e$ is time-stamped by the two clocks $j$ and $k$ with a difference of one tick. The finite precision of the global time-base and the digitalization of the time make it impossible in a distributed system to order events consistently on the basis of their global time-stamps based on a *dense* time. This problem can be solved by the introduction of a *sparse time base* (Kopetz, 1997), p.55. In the sparse-time model the continuum of time is partitioned into an infinite sequence of alternating durations of *activity* and *silence* as shown in Figure 2. The activity intervals form a synchronized system-wide *action lattice*.

From the point of view of temporal ordering, all events that occur within a duration of activity of the action lattice are considered to happen *at the same time*. Events that happen in the distributed system at different nodes at the same global clock-tick are thus considered *simultaneous*. Events that happen during different durations of activity (at different points of the action lattice) and are separated by the required interval of silence (the duration of this silence interval depends among others, on the precision of the clock synchronization (Kopetz, 1992)) can be temporally ordered on the basis of their global timestamps. The architecture must make sure that significant events, such as the sending of a message, or the observation of the environment, occur only during an interval of activity of the action lattice. The time-stamps of events that are based on a sparse time based can be mapped on the set of positive integers. It is then possible to establish the temporal order of events by integer arithmetic.

The timestamps of events that are outside the control of the distributed computer system (and therefore happen on a dense timeline) must be

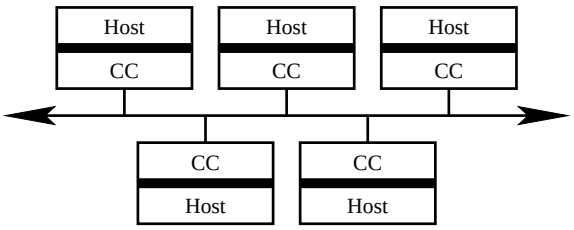


Fig. 3. Distributed real-time system with five nodes

assigned to an agreed lattice point of the action lattice by an *agreement protocol*. Agreement protocols are also needed to come to a system-wide consistent view of analogue values that are digitized by more than one analogue-to-digital converter.

### *3.2 RT Entity and RT Image*

A distributed time-triggered computer system can be depicted by a set of nodes that are interconnected by a real-time communication system as shown in Figure 3. All nodes have access to a global time of known precision. A node consists of a *communication controller* (CC) and a *host computer*. The common boundary between the communication controller and the host computer within a node is called the *communication network interface* CNI (thick black line in Figure 3).

The dynamics of a real-time application is modeled by a set of relevant state variables, the *real-time* (RT) entities that change their state as time progresses. Examples of RT entities are the flow of a liquid in a pipe, the setpoint of a control loop 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 has dynamic attributes that change with 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.

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 a discrete RT entity can only be observed when the state of this RT is not changing.

A *real-time (RT) image* is a *temporally accurate* picture of an RT entity at instant $t$, if the duration between the time-of-observation and the instant t is less than the accuracy interval $d_{acc}$, which is an application specific parameter associated with the dynamics of the given RT entity. An RT image is thus *valid* at a given instant if it is an accurate representation of the corresponding RT entity, both in the value and the time domains (Kopetz and Kim, 1990). While an *observation* records a fact that remains valid forever (a statement about an RT entity that has been observed at an instant), the validity of an RT image is *time-dependent* and is invalidated by the progression of real-time.

### *3.3 State-Information versus Event-Information*

The information that is exchanged across an interface is either *state information* or *event information*, as explained in the following paragraphs. Any property of a *real-time (RT) entity* (i.e., a *relevant state variable*) that is an observed by a node of the distributed real-time system at a particular instant, e.g, the temperature of a vessel, is called a *state attribute* and the corresponding information *state information*. A *state observation* records the state of a state variable at a particular instant, the *point of observation*. A *state observation* can be expressed by the atomic triple

*<Name of the observed state variable, observed value, time of observation>*

For example, the following is a state observation: "*The position of control valve A was at 75 degrees at 10:42 a.m.*" State information is *idempotent* and requires an *at-least once semantics* when transmitted to a client. At the sender, state information is *not consumed on sending* and at the receiver, state information requires an *update-in-place* and a *non-consumable read*. State information is transmitted in *state messages*.

A sudden change of state of an RT entity that occurs at an instant is an *event*. Information that describes an event is called *event information*. Event information contains the *difference* between the state *before* the event and the state *after* the event. An *event observation* can be expressed by the atomic triple

*<Name of the observed state variable, value difference, time of event>*

For example, the following is an event observation: "*The position of control valve A changed by 5 degrees at 10:42 a.m.*" Event observations requires *exactly-once semantics* when transmitted to a consumer. Events must be *queued on sending* and *consumed on reading*. Event information is transmitted in *event messages*.

Periodic state observations or sporadic event observations are two *alternative* approaches for the observation of a dynamic environment in order to reconstruct the *states and events* of the environment at the observer (Tisato and DePaoli, 1995). Periodic state observations produce a sequence of equidistant "snapshots" of the environment that can be used by the observer to reconstruct those events that occur within a minimum temporal distance that is longer than the duration of the sampling period. Starting from an initial state, a complete sequence of (sporadic) event observations can be used by the observer to reconstruct the complete sequence of states of the RT entity that occurred in the environment. However, if there is no minimum duration between events assumed, the observer and the communication system must be infinitely fast.

### *3.4 Linking Interface (LIF)*

In a distributed real-time system, the components realize *emerging services* by the timely exchange of messages across linking interfaces (LIF) (Jones *et al.*, 2001). The emergent services refer to those properties of the system of components that are not part of the components themselves but come into existence by the interactions among the components across the LIFs (Leveson, 2000).

The specification of a LIF should be *complete* and of *minimal cognitive complexity*. Completeness implies that all information that a user of the services needs to deploy a component must be contained in the LIF specification. The LIF specification must thus comprise the following parts:

(1) The *syntactic specification* of the messages, i.e., the specification of the data elements that cross the interface. Out of the sequence of bits in a message the syntactic specification forms larger (*information*) *chunks* (such as a number, a string, a method call, or a structure consisting of a combination thereof) and assigns a *name* to each chunk. Although from the view of mechanical processing any name would suffice, a *descriptive name* that establishes a link between the chunk and its meaning helps human understanding. The syntactic specification bridges the gap between the logical level and the informational level (Avizienis, 1982). It can be formalized by expressing it in a well-defined interface definition language (IDL).

(2) The *temporal specification* of the message send and receive instants, e.g., at what instants the messages are sent and arrive, how the messages are ordered, and the rate of message arrival. This information can be formalized if an appropriate model of real-time

is available. In non-safety critical applications the temporal specification can be expressed in probabilistic terms.

(3) The *conceptual interface model specification*. In many cases the *concept* associated with the *name of a chunk* (the result of the syntactic specification) is not sufficiently precise to cover all aspects of the semantics of the interface data. In these cases it is necessary to specify a *conceptual interface model* that relates the names of the *chunks* to the user's conceptual world and thus assigns a deeper meaning to the *chunks* generated by the syntactic specification. It follows that the conceptual interface model must be expressed in concepts that are familiar to the user of the interface services. The conceptual model bridges the gap between the *informational level* and the *user's level* (Avizienis, 1982).

Of course, there are interdependencies between these three different parts of the specification. For example, the concept of an *observation*, introduced before, contains an element of each one of the three parts. Nevertheless, we feel that the introduced partitioning brings structure into a LIF specification and thus contributes to solving the specification problem.

We subsume under the term *operational specification* of an interface the *syntactic specification* and the *temporal specification*. Consistency of the operational specification of interacting components is a necessary (but not sufficient) prerequisite for the proper operation of the system of components. Consistency of the specifications of the conceptual interface models of the communicating partners, called the meta-level specifications, assures that the meaning of the information chunks in all involved components is in agreement with the user's intent.

Whereas the operational specification must be rigorous, it is often difficult to develop a formal system for the meta-level specification if the information chunks exchanged across an interface relate to concepts of the real world (as they normally do). For example, how can we formalize the concept of *temperature* or the concept of *drive at a safe speed*? This qualitative difference in the nature of the operational specification and meta-level specification justifies the clear distinction between these two kinds of specifications.

The demand for minimal cognitive complexity of a LIF has the following consequences:

(1) An interface should only serve a single purpose. If there is a need to interact with a component for different purposes, different interfaces should be provided. For example, a component may support, in addition to a LIF, a *diagnostic interface* for diagnosis and a *configuration interface* for configuring the component into a new environment (Kopetz, 2001). Since these three interfaces are directed to different user groups and deal with different views of a component, interfacing would become unnecessarily complicated for each one of these users if all three interfaces were integrated into a single one (Ran and Xu, 1997).

(2) If multiple users of a sequential LIF can access a component concurrently (e.g., a server of an e-commerce application), then it should be tried to hide this concurrency from the LIF user. The concurrency inside the component should not be visible from the vantage-point of the LIF, since concurrency increases the cognitive complexity of an interface significantly.

(3) The LIF conceptual model should be structured along a stratified *means-end hierarchy* (Vicente and Rasmussen, 1992) in order to limit the amount of information that must be dealt with at a selected level of abstraction. This requirement is derived from the characteristic of human cognitive information processing (Reason, 1990). The proper structure and representation of the conceptual interface model is crucial for controlling the conceptual effort needed to understand the component and for reusing the component in a new context.

As mentioned before, communication across an interface is only successful if the operational specification and the meta-level specification of the interfaces of all communication partners are consistent with each other. If there are syntactic property mismatches among the interface specifications of the communicating partners, it is possible to resolve these syntactic property mismatches by connection systems inserted between the component interfaces (Jones *et al.*, 2001). These connection systems are not needed, if all components are designed according to the same architectural style. Property mismatches that have their cause in conflicts of the conceptual interface models cannot be resolved by connection systems but require human intervention.

## 4. PRINCIPLES OF OPERATION

In this Section we describe the principles of operation of a time-triggered real-time computing system.

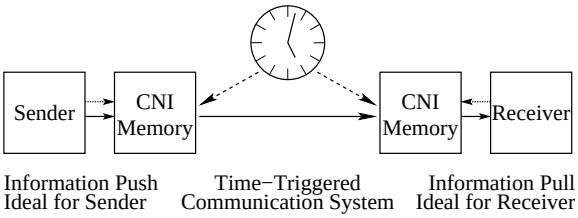


Fig. 4. Data Flow (full line) and Control Flow (dashed line) at a Temporal Firewall Interface

### *4.1 Temporal Firewall Interfaces*

Many hard real-time systems in control applications have a regular behavior. The sensors observe the state of the controlled object, the control algorithms are calculated, and the setpoints are delivered to the actuators. The components of the systems, the sensors, control modules, and actuators, exchange *state information* periodically. In order to support these applications effectively, a special interface, the *temporal firewall*, has been designed as the fundamental interface of a time-triggered architecture. A temporal firewall is an operationally fully specified digital interface for the unidirectional exchange of *state information* between a sender/receiver over a time-triggered communication system. The basic data and control transfer of a temporal firewall interface is depicted in Figure 4, showing the data and control flow between a sender and a receiver.

The CNI memory at the sender forms the output firewall of the sender and the CNI memory of the receiver forms the input firewall of the receiver. The sender deposits its output information into its temporal firewall (update in place) according to the information *push* paradigm, while the receiver must *pull* the input information out of its CNI (non-consumable read). The transport of the information is realized by a time-triggered communication system that derives its control signals autonomously from the progression of time. It is *common knowledge* to the sender and the receiver at what instants (on a sparse time base) the typed data structure is fetched from the sending CNI and at what instants this data structure is delivered at the receiving CNI by the communication system. On input, these precise operational interface specifications (in the temporal and value domain) are the *pre-conditions* for the correct operation of the application software. On output, the precise interface specifications are the *post-conditions* that must be satisfied by the application software, provided the preconditions have been satisfied by the environment.

Since no control signals cross such a temporal firewall interface, control-error propagation across this interface is eliminated by design. A temporal firewall also eliminates (low-level) concurrency from the interface. The sparseness of the global time establishes a system wide *action lattice*, the lattice points of which are precisely synchronized with the global time. The behavior of a system can be explained by the sequential stepwise progression through this action lattice. The elimination of concurrency from the interface simplifies the understanding of the interface, since the human mind is ill-equipped to handle concurrent processes (Reason, 1990).

### *4.2 Observation Lattice*

We call the periodic sequence of time points of the *action lattice* at which an RT-entity observes its environment the *observation lattice*. The distance between the lattice points of this observation lattice is called the observation granularity $g_0$ (which often will be significantly larger than the granularity of the action lattice (Kopetz, 1992)). Independent of the current activity in a RT-entity, the state of a RT-entity is sampled at a constant rate at the *a priori* specified instants of the observation lattice. Although it is impossible to influence the time of occurrence of a (rare) event in a RT entity, the time of recognition of the consequences of this event — the new state — is restricted to the lattice points of the observation lattice in a TT-architecture. Because of this *enforced regularity*, TT-architectures are inherently more predictable than ET-architectures.

At the communication network interfaces (CNI) within a node of a time-triggered architecture, the pictures of the RT entities are periodically updated by the real-time communication system to establish temporally accurate RT-*images* of the RT-entities. The computational tasks within the host of a node take these temporally accurate RT-images as inputs to calculate the required outputs within an *a priori* known worst-case execution time (WCET). The outputs of the host are stored in the CNI and transported by the time-triggered communication system to the CNIs of other nodes at *a priori* determined instants. The *interface nodes* transform the received data to/from the representation required by the controlled object or the human operator and activate control actions in the physical world (Kopetz, 1998).

### *4.3 Guaranteed Resource Availability*

TT systems must be designed according to the principle of resource adequacy (Lawson, 1992). It is assumed that sufficient computing resources are available to handle the specified peak load scenario of all hard real-time tasks, even if all faults specified in the fault hypothesis occur simultaneously. In the past, there have been a number

of real-time applications where resource adequacy was ruled out on the basis of economic arguments. It was accepted that the system does not provide the desired service in case of the occurrence of a rare event of system overload. However, with the decreasing cost of computing resources the economic arguments for not providing resource adequacy are losing ground.

### *4.4 Prompt Error Detection in the Temporal Domain*

The *a priori* knowledge of the *receive instant* of a message in a time-triggered system can be used to implement prompt error detection at the receiver of the message. As soon as the instant of arrival of a time-triggered message has passed without the message arriving, an error handling process can be initiated at the receiver. This form of *time-based error detection* at the receiver can also be deployed in systems that provide unidirectional information and unidirectional control flow only. In an event-triggered system, where it is not known *a priori* when a message ought to arrive at a receiver, a bi-directional control flow is required for error detection. This bi-directional control makes the interface more complex, in particular when a multi-cast communication topology must be supported (Kopetz, 1999).

## 5. ADDITIONAL BENEFITS OF A GLOBAL TIME

As mentioned in the introduction, a time-triggered computing system requires the existence of a globally synchronized time in order to generate the temporal triggers for the communication and significant processing tasks in the individual nodes. If the existence of a dependable global time base can be assumed, the solution to many other difficult problems in the design of a distributed real-time system can be simplified. Thus time moves from the problem space to the solution space.

### *5.1 Temporal Validity of Data*

The concept of an observation, introduced in Section 3.2, contains the *instant of observation* as an integral part. Only if a global time is available is it possible for any node to learn precisely about the age of an observation. Knowledge about the age of an observation determines whether the observation may be used in a particular real-time control context. If no global time base is available, often implicit assumptions about the performance of the communication system are used to establish the age of an observation. These (implicit) assumptions about the temporal delay may become invalid when the system is deployed in a dynamically changing environment, giving rise to sporadic errors.

### *5.2 Consistent Order of Events*

The timestamps of events on a sparse global time-base can be used to establish a consistent order of events in distributed system. A consistent order is needed by many distributed algorithms. If no global time-stamps of events are available, special distributed algorithms for the establishment of a consistent order must be executed. These algorithms require additional computational and communication resources, which are not needed in a TT system that is based on a sparse global time.

### *5.3 Mutual Exclusion*

The availability of a sparse global time base can be used to deterministically solve the mutual exclusion problem when accessing shared resources in a distributed system. For example, the time-axes can be partitioned into an (infinite) set of successive periodic windows. Each window consists of a sequence of durations, where each one of these durations is assigned to a process that intends to access the common resource. A process is only allowed to access the resource during the duration assigned to it *a priori*. Such a time-based solution of the distributed mutual exclusion problems is free of race conditions and maintains replica determinism in the distributed system.

### *5.4 Synchronization of Distributed Actions*

The global time can also be used to precisely coordinate a distributed action that is composed of a set of local actions performed at different nodes of a distributed system. The timestamps of the instants when the distributed action must take place are transported to the nodes participating in the distributed action ahead of the distributed action. As soon as the globally synchronized time reaches the prescribed value, every node executes its local part of the distributed action, thus giving rise to a tightly synchronized global distributed action, even if the communication system is not free of jitter.

### *5.5 Membership Service*

In order to establish a consistent distributed computing base it is necessary to provide a uniform view about the *health state* of each node

— whether it is operating correctly or has failed — to all correct nodes of the ensemble within a short latency. The service that provides this view is called the *membership service*. It is known that it is impossible to implement a timely membership service in a purely asynchronous system (Fischer *et al.*, 1985). Every membership service is based on some type of *failure detector* that is based on *a priori* knowledge about the behavior of a node in the temporal domain (Fetzer, 2001). In a time-triggered system the periodic transmission of messages by the nodes at instants that are *common knowledge* to all members of the ensemble can be interpreted as *life signs* of the nodes and be used for implementing the membership service (Kopetz and Grünsteidl, 1993).

### *5.6 Coordination of Timeaouts*

Most communication protocols use timeouts for error detection. If there is no global time base available, these time-outs can only refer to the clock of the local node but not to a system-wide notion of time. The existence of a sparse global time makes it possible to coordinate these time-outs on a system-wide basis and thus eliminate many tricky problems, e.g., the *orphan problem* (Rao *et al.*, 2000).

## 6. MEETING THE REQUIREMENTS

In this section we argue that within a properly designed time-triggered architecture (see, e.g., (Kopetz and Bauer, 2001)) it is possible to satisfy all requirements listed in Section 2.

### *6.1 Temporal Predictability*

TT-systems require careful planning during the design phase. First it is necessary to establish the interaction pattern among the components and the temporal and value parameters of the CNIs. Because of this early planning effort, a detailed specification of the temporal properties of the LIF of each component is available after the architecture design phase. The dynamics of a RT-entity determines the granularity of the observation lattice and thus the required minimum duration of states that can be guaranteed to be observed. (If these assumptions are violated, short lived states may evade the observation.) Since the temporal control structure of a TT system is derived from the progression of the global time, TT systems are *as predictable as time*.

In an ET-system it is not necessary to develop such a set of detailed plans during the design phase, since the execution schedules evolve dynamically as a consequence of the actual demand. Depending on the sequence and timing of the events, which are presented to the system in a specific application scenario, different schedules unfold dynamically. In an ET-system, the critical external assumption is the conjectured distribution of the event occurrences, both in normal and peak load situations. This distribution forms the basis for the system test and the predictions about the adequacy of the system to meet the specified deadlines. If this assumed distribution is not representative of the distributions that evolve in the real world, then the basis for predicting the timeliness of the system is compromised. The temporal behavior of ET systems is thus less predictable than that of TT systems.

### *6.2 Consistent Distributed Computing Base*

In a distributed TT system it is *a priori common knowledge* at which instant a message of a correct node must arrive at all other nodes. This common knowledge can be used to design a consistent distributed computing base, such as the one realized in the time-triggered architecture with the TTP protocol (Kopetz and Grünsteidl, 1993), p181. TTP is based on a time-division-multiple-access (TDMA) strategy to replicated communication channels. The TTP protocol provides, in addition to a fault-tolerant clock synchronization, a distributed membership service and a clique avoidance service. The membership service of TTP informs consistently all correct nodes about the *health state* of all nodes within two TDMA rounds. If a fault outside the fault hypothesis causes the formation of cliques, the clique avoidance mechanism of TTP will force the minority clique into a restart in order that a consistent distributed computing base remains available at all times. The correctness of the membership protocol of TTP has been investigated by formal methods (Rushby, 2000).

It is impossible to maintain a consistent distributed computing base in an ET system that has to cope with faults (Fischer *et al.*, 1985).

### *6.3 Composability*

In a distributed real-time computer system the nodes interact via the communication system to provide the emerging real-time services. These emerging services depend on the timely arrival of the real-time information at the linking interfaces of the nodes. For an architecture to be composable, it must adhere to the following four principles:

(1) Independent node development.
(2) Stability of prior services.
(3) Communication system performance.
(4) Replica determinism.

**Independent Node Development:** A composable architecture must distinguish distinctly between architecture design and node design. Principle one of a composable architecture is concerned with design at the architecture level. Nodes can only be designed independently of each other, if the architecture supports the exact specification of all node services provided at the LIFs during architecture design. The interface data structures musts be precisely specified in the value domain and in the temporal domain and a proper *conceptual interface model* of the node service, as *viewed* by a user of the node, must be available (See also Section 3.2. Only then is the node designer in the position to know exactly *what* can be *when* expected from the environment and *what* must be *when* delivered to the environment by the node across its LIF. This knowledge is a prerequisite for the independent development of the node software, for reusing a node in a new application context, and for reasoning about the composition of nodes into a system-of-systems. The temporal firewall interface of a time-triggered architecture, developed during the architecture design phase, supports this requirement.

**Stability of Prior Services:** Principle two of a composable architecture is concerned with the design at the node level. A node is a nearly autonomous subsystem that comprises the hardware, the operating system and the application software. The node must provide the specified services at its LIF and to its environment. The design of the node can take advantage of any established software engineering methodology, such as object based design methods, and can use any scheduling technique as long as the instants defined in the temporal firewall specification are met. Since an input firewall is accessed by the receiver according to the information pull paradigm, the temporal behavior of a component is not compromised by the integration. The *stability-of-prior-service* principle ensures that the validated service of a node — both in the value domain and in the time domain — is not refuted by the integration of the node into an encompassing system-of-systems.

**Communication System Performace:** Principle three of a composable architecture is concerned with the performance of the communication system. Normally, the integration of the nodes into the system follows a step-by-step procedure. The performability-of-the-communication-system principle ensures that if n nodes are already integrated, the integration of the $n+1$ node will not disturb the correct operation of the $n$ already integrated nodes. This principle guarantees that the integration activity is linear and not circular. It has stringent implications for the management of the network resources. If the network resources are managed dynamically, then it must be ascertained that even at the *critical instant*, i.e., when all nodes request the network resources at the same instant, the specified timeliness of all communication requests can be satisfied. Otherwise failures will occur sporadically with a failure rate that is increasing with the number of integrated nodes. A time-triggered communication system satisfies this requirement. **Example:** If a real-time service requires that the network delay must always remain below a critical upper limit (because otherwise a local time-out within the node may signal a communication failure) then the dynamic extension of the network delay by adding new nodes may be a cause of concern. In a dynamic network the message delay at the critical instant (when all nodes request service at the same instant) increases with the number of nodes. The system of Figure 5 will work correctly with up to four nodes. The addition of the fifth node may lead to sporadic failures.

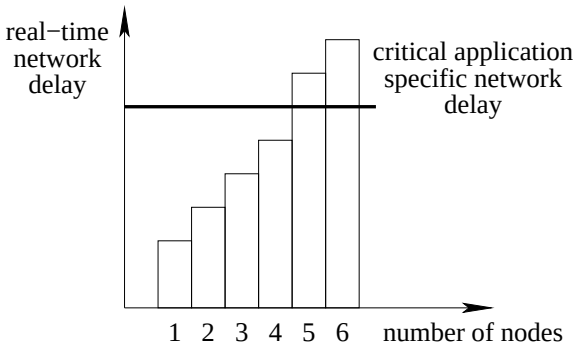


Fig. 5. Maximum network delay at critical instant as a function of the number of nodes.

**Replica Determinism:** If fault-tolerance is implemented by the replication of nodes, then the architecture and the nodes must support replica determinism. A set of replicated nodes is *replica determinate* (Poledna, 1994) if all the members of this set have the same externally visible state, and produce the same output messages at points in time that are at most an interval of $d$ time units apart (as seen by an omniscient outside observer). In a fault-tolerant system, the time interval $d$ determines the time it takes to replace a missing message or an erroneous message from a node by a correct message from redundant replicas. The implementation of replica determinism is simplified if all nodes have access to a globally synchronized sparse time base and use the time to the mutual exclusion problem (see Section 5.3).

Replica determinism also decreases the testing and debugging effort significantly.

In a time-triggered system all four of these principles can be satisfied. A properly designed TT system thus supports composability. Since the temporal specification of the CNI of an ET system is necessarily incomplete (there is no notion of global time in a pure ET system), temporal composability is difficult to achieve in an ET systems.

### *6.4 Validation*

In a TT-system, the results of the test of every node can be compared with the specification of the CNI. Since the time-base is discrete and determined by the granularity of the action lattice, every input case can be observed and reproduced in the domains of time and value. Therefore testing of a properly designed TT-system is deterministic and constructive. To achieve the same test coverage, the effort to test an ET-system is much greater than that required for the testing of the corresponding TT-system. The difference is caused by the smaller number of possible execution scenarios that have to be considered in a TT-system, since in such a system the order of state changes within a granule of the observation lattice is not relevant (Schütz, 1991).

### *6.5 Extensibility*

From a temporal point of view, every node of a TT-system is encapsulated. There are no control signals crossing the CNI between two TT subsystems. Every subsystems exercises autonomous control derived from the progression of the synchronized time that is available locally. As long as a modified node does not require a change of the CNI, the change has no temporal effect on the rest of the system. The effort required to add a new node depends on the information flow to/from this new node. If the node is passive, i.e., it does not send any information into the system (e.g., a display), then no modification of the existing system is required since the communication protocols are unidirectional and of the broadcast type. If the new node is active, i.e., information is sent from the new node into the existing system, then a preplanned spare communication slot must be taken or a new schedule must be generated for the complete cluster.

### *6.6 Fault Tolerance*

In a TT-system all communication activities are synchronized globally by the global time. Non-deterministic decisions can be avoided and replica determinism can be maintained without additional inter-replica coordination. In a replica-deterministic TT system fault-tolerance can be implemented by standard techniques (e.g., by triple modular redundancy, TMR) within an autonomous fault-tolerance layer that provides the same CNI, both in time and value, then the non-fault-tolerant system. An example of such a transparent implementation of fault-tolerance is described in (Bauer and Kopetz, 2000).

## 7. CONCLUSION

Distributed real-time computing architectures are moving into the mainstream for the design and implementation real-time applications in the aerospace and automotive markets. It is now widely accepted (Rushby, 2001), that in these safety critical applications time-triggered architectures are the preferred alternatives. In this paper we have discussed the requirements of distributed real-time systems and shown how it is possible to satisfy these requirements within a time-triggered architecture. We consider the precise specification of the temporal properties of the linking interfaces as the main contribution of the time-triggered paradigm. These precise interface specifications form the basis for the composable integration and the reuse of components.

## ACKNOWLEDGEMENTS

This work has been supported in part by the IST Project “DSoS” and the IST Project “Next TTA”.

## REFERENCES

Avizienis, A. (1982). The four-universe information system model for the study of fault tolerance. In: *IS Fault-Tolerant Computing (12$^{\text{th}}$FTCS)*. IEEE Press.

Bauer, G. and H. Kopetz (2000). Transparent redundancy in the time-triggered architecture. In: *IC Dependable Systems and Networks (DSN2000)*. IEEE Press. New York. pp. 5–13.

Boyer, R. S. and J. S. Moore (1981). *The Correctness Problem in Computer Science*. Academic Press. London. ISBN: 0-12-122920-3.

Constantinescu, C. (2002). Impact of deep submicron technology on the dependability of VLSI circuits. In: *IC Dependable System and Networks (DSN 2002)*. IEEE Press.

Fetzer, C. (2001). Enforcing perfect failure detection. In: *IC Distributed Computing Systems*. Meza, Arizona, USA.

Fischer, M., N. Lynch and M. Paterson (1985). Impossibility of distributed consensus with one faulty process. *ACM* **32**(2), 374–382.

Jones, C., H. Kopetz, M. Paulitsch, M.-O. Killijian, E. Marsden, N. Moffat, D. Powell, B. Randell, A. Romanovsky and R. Stroud (2001). Revised version of dsos conceptual model. Project Deliverable for DSoS (Dependable Systems of Systems), Research Report 35/2001. Technische Universität Wien, Institut für Technische Informatik. Treitlstr. 1-3/182-1, 1040 Vienna, Austria.

Kopetz, H. (1992). Sparse time versus dense time in distributed real-time systems. In: *IC Distributed Computing Systems (DCS 1992)*. pp. 460–467.

Kopetz, H. (1997). *Real-Time Systems, Design Principles for Distributed Embedded Applications*. 1999 3rd ed.. Kluwer Academic Publisheres. Boston. ISBN: 0-7923-9894-7.

Kopetz, H. (1998). The time-triggered model of computation. In: *Real-Time Systems Symposium (RTSS98)*. IEEE Press. Madrid, Spain. pp. 168–177.

Kopetz, H. (1999). Elementary versus composite interfaces in distributed real-time systems. In: *IS Autonomous Decentralized Systems (ISADS99)*. IEEE Press. Tokyo, Japan. pp. 26–33.

Kopetz, H. (2001). The temporal specification of interfaces in distributed real-time systems. In: *EMSOFT*. Vol. 2211 of *Lecture Notes in Computer Science*. Springer. pp. 223–236. ISBN: 3-540-42673-6.

Kopetz, H. and G. Bauer (2001). The time-triggered architecture. Technical Report 22/2001. Technische Universität Wien, Institut für Technische Informatik. Treitlstr. 1-3/182-1, 1040 Vienna, Austria. submitted to Proceedings of the IEEE special issue: Modeling and Design of Embedded Software.

Kopetz, H. and G. Grünsteidl (1993). Ttp — a time-triggered protocol for fault-tolerant real-time systems. In: *IS Fault-Tolerant Computing (23th FTCS)*. IEEE Press. Toulouse, France.

Kopetz, H. and K. Kim (1990). Temporal uncertainties in interactions among real-time objects. In: *9th Symposium on Reliable Distributed Systems*. IEEE Computer Society Press. Huntsville, AL, USA.

Lamport, L., R. Shostak and M. Pease (1982). The byzantine generals problem. *ACM Transactions on Programming Languages and Systems* **4**(3), 382–401.

Lawson, H.W. (1992). Cyclone - an approach ot the engineering of resource adequate cyclic real-time systems. *Real-Time Systems* **4**(1), 55–84.

Leveson, N.G. (2000). Intent specifications: An approach to building human-centered specifications. *Software Engineering* **26**(1), 15–35.

Poledna, S. (1994). Replica Determinism in Fault-Tolerant Real-Time Systems. PhD thesis. Technical University of Vienna.

Ran, A. and J. Xu (1997). Architecting software with interface objects. In: *8th Israeli Conference on Computer Systems and Software Engineering*. IEEE Press.

Rao, S., L. Alvisi and H.Vin (2000). The cost of recovery in message logging protocols. *IEEE Transactions on Knowledge and Data Engineering* **12**(2), 160–173.

Reason, J. (1990). *Human Error*. Cambridge University Press.

Rushby, J. (2000). Formal verification of group membership for the time-triggered architecture. Technical report. SRI International. Menlo Park California.

Rushby, J. (2001). A comparison of bus architectures for safety critical embedded systems. Technical report. SRI International. Menlo Park California.

Schütz, W. (1991). On the testability of distributed real-time systems. In: *IS Relieable Distributed Systems*. IEEE Press. Pisa, Italy. pp. 52–61.

Tisato, F. and F. DePaoli (1995). On the duality between event-driven and time-driven models. In: *13th IFAC DCCS 1995*. Toulouse, France.

Vicente, K.J. and J. Rasmussen (1992). Ecological interface design: Theoretical foundations. *IEEE Transactions on Systems, Man, and Cybernetics* **22**(4), 589–606.