Event–Triggered versus Time–Triggered Real–Time Systems

H. Kopetz

Institut für Technische Informatik
Technische Universität Wien
Austria

hk@vmars.tuwien.ac.at

Research Report Nr. 8/91

Proc. Int. Workshop on Operating Systems of the 90s and Beyond
Lecture Notes in Computer Science
Vol. 563, pages 87–101, Berlin, Germany, 1991, Springer Verlag

Institut für Technische Informatik
Technische Universität Wien
Treitlstrasse 3/182.1
A–1040 Wien
Austria

Phone: (43) 222 58801 8180
Fax: (43) 222 56 91 49

Event-Triggered versus Time-Triggered Real-Time Systems

H. Kopetz
Technical University of Vienna
Vienna, Austria

Abstract

This paper compares the temporal properties of event-triggered and time-triggered distributed real-time systems. In an event triggered system a processing activity is initiated as a consequence of the occurrence of a significant event. In a time-triggered system, the activities are initiated periodically at predetermined points in real-time. In the first part of this paper, a model of a distributed real-time system is presented and the characteristic attributes of TT-systems and ET-systems are described. The comparison focuses on the issues of predictability, testability, resource utilization, extensibility, and assumption coverage.

INTRODUCTION

A real-time system has to meet the deadlines dictated by its environment. If a real-time system misses a deadline, it has failed. Designing real-time systems is challenging: the functional specifications must be met within the specified time constraints.

Temporal properties are system properties. They depend on the entire system architecture, i.e., the structure of the application software, the scheduling decisions within the operating system, the delays of the communication protocols, and most important, on the performance characteristics of the underlying hardware.

At present, two fundamentally different paradigms for the design of real-time systems are cultivated. An event-based design focuses on the occurrences of events in the controlled object to initiate the appropriate system activities. A state-based design is driven by the

periodic observation of the state of the controlled object and starts actions as a consequence of the analysis of these states. Whereas the state based design is prevailing in safety critical real-time applications, the event based design is common in the non-safety critical community [LeL89].

In this paper the advantages and disadvantages of these two design paradigms are analyzed and compared. The emphasis of this comparison is on the temporal properties of the two approaches. After the presentation of a model of a distributed real-time system the two architectures are described in detail. In the following section we compare the architectures from the point of view of predictability, testability, resource utilization, extensibility, and assumption coverage.

REAL-TIME SYSTEM MODEL

A real-time application can be decomposed into a controlled object (e.g., the machine that is to be controlled) and the controlling computer system. The computer system has to react to stimuli from the controlled object within an interval of real-time that is dictated by the dynamics of the controlled object.

We assume that the computer system is distributed as shown in Fig. 1. It consists of a set of self-contained computers, the *nodes* which communicate via a Local Area Network by the exchange of messages only. We also assume that the nodes are fail-silent and that messages which are mutilated by the communication system can be detected and discarded. Some nodes, the *interface nodes*, support a connection to the intelligent instrumentation, i.e., the sensors and actuators, which are at the interface to the controlled object. We further assume that all clocks in the nodes are synchronized such that an *approximate global time base* of sufficiently small granularity is available to all clients in the different nodes of the distributed computer system [Kop87].

From the point of view of the control system designer, the behavior of a real-time application can be modelled by a set of computational processes operating on representations of RT-entities. A *RT-entity* is an element of interest for the given purpose, either in the controlled object (an *external RT-entity*) or in the computer system (an *internal RT-entity*). A RT-entity has a time-dependent internal state. Examples of RT-entities are the temperature of a particular vessel (external RT-entity), the speed of a vehicle (external RT-entity), or the intended position of a specific control valve (internal RT-entity).

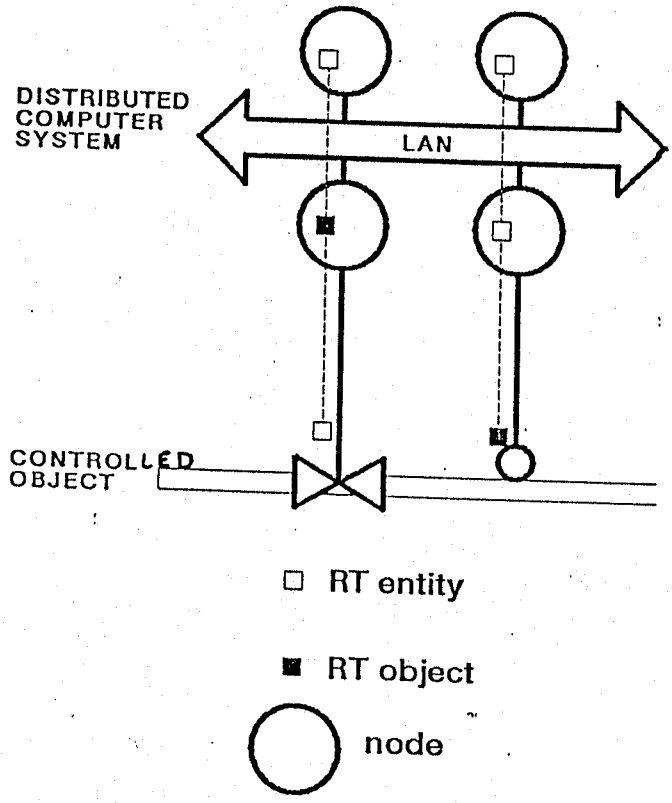

Fig. 1: Real-Time Application

Every RT-entity is in the *sphere of control* of a subsystem (e.g., in the controlled object or in a particular node of a distributed computer system) that establishes the value of this RT-entity at any point in real-time. From outside its sphere of control, the state of an RT-entity can only be observed, but not modified. We call the atomic tuple

O = <entity name,value,point of observation>

an *observation* O(e,v,t) of an RT-entity e with value v at time t. The value v of an observation represents an estimation of the state of the RT-entity at time t. Observations of the controlled object taken at a particular point in time are transmitted to, manipulated by, and stored within *RT-objects* [Kop90] in the nodes of the computer system. This transportation and manipulation of the observations take a certain span of real-time, determined by the delay characteristics of the communication protocols and the processing speed within the nodes. Thus an observation that is available to the client of a RT-object in a remote node at a particular point in time t-use is already aged by this span of real-time. We call the length of this time-span, i.e., the difference between the point of observation and the

point of use of an observation, the *temporal accuracy*, and the difference between the actual value of the RT-entity at time t and the value of the corresponding observation the *value accuracy* of the observation. If the value of a RT-entity is continuous, then the value accuracy and the temporal accuracy are related by the gradient of the value in relation to time. An observation that is inaccurate, either in the domain of value or in the domain of time, can cause a failure in the service of the real-time computer system.

The patterns of change of a RT-entity, i.e., the dynamics of the RT-entity, determine the response time requirements on the computer system. If the response time of the computer system is much longer than the time between significant changes in the state of the RT-entity, then it is difficult to perform any real-time control of the controlled object. Only if the speed of the computer system matches the dynamics of the controlled object is real-time control feasible.

In typical real-time applications, the computer system has to perform a multitude of different functions in parallel, e.g., the display of observations to the operator, the monitoring of observations (both their value and rate of change) to detect alarm conditions, the processing of observations by process models in order to find new setpoints, etc.. In distributed computer systems, these different functions are normally allocated to different nodes. In order to guarantee a consistent behavior of the distributed computer system as a whole, it must be assured that all nodes operate on the same versions of the observations at about the same time.

Depending on the triggering mechanisms for the start of the communication and processing activities in each node of the computer system, two distinctly different approaches to the design of real-time computer applications can be distinguished. In the *event triggered* (ET) approach all communication and processing activities are initiated whenever a significant change of state, i.e., an event, in a RT-entity or the corresponding RT-object is noticed. In the *time triggered (TT)* approach all communication and processing activities are initiated periodically at predetermined points in time. In the following sections we will analyze the differences between these two competing design philosophies.

EVENT TRIGGERED SYSTEMS

In a purely event triggered system all system activities are initiated by the occurrence of significant events in a RT-entity or a RT-object. A *significant event* is a change of state in a RT-entity or RT-object which has to be handled by the computer system. Other state changes

of RT-entities are considered *insignificant* and are neglected. In many implementations of ET-systems the signalling of significant events is realized by the well known interrupt mechanism, which brings the occurrence of a significant event to the attention of the CPU. In following discussion we distinguish between two different types of significant events, the *predictable events* (the *P-events*) and the *chance e vents* (the *C-events*).

The future occurrence of P-events can be determined by applying known laws, e.g., of physics, to the present state of the system. For example the increase of an electric current through a wire will result in a later change of the temperature of the wire. P-events can be anticipated and the resources required to transmit and process them in the future can be reserved ahead of time.

The occurrence of a particular C-event, on the other side, is caused by a *chance process* and cannot be predicted deterministically. A chance process is an inherently random process, e.g., the radioactive decay of a particular atomic nucleus. The occurrence of such a particular C-event can only be predicted probabilistically. Yet, the behavior of large aggregates of C-events can be described by statistical measures. A typical example for a C-event is the occurrence of a fault, either in the controlled object or in the computer system. Consider the case of a pipe rupture in a reactor. Such a C-event will result in correlated important changes of state (i.e., significant events) in many of the RT-entities in the controlled object, causing many coincident events signalled to the computer system. It is impossible to service all these coincident events instantaneously, since every computer system has only a finite processing capacity. Therefore mechanisms for the intermediate storage of these "*event showers*" and for the restriction of the flow of information from the controlled object to the computer must be put into place.

Flow Control

Flow control is concerned with the synchronization of the speed of the sender of information with the speed of the receiver, such that the receiver can follow the sender. Since the controlled object is not in the sphere of control of the computer system, there is no possibility to limit the occurrence of C-events in the controlled object in case the computer system cannot follow. Therefore provisions must be made that event showers can be buffered at the interface between the controlled object and the computer system. Sometimes it may even be decided to discard events which occur too frequently. Several engineering solutions are applied to restrict the flow of events at this interface. These include hardware implemented low pass filters, intermediate buffering of events in hardware and/or software, etc..

Within an ET-system, *explicit flow control* mechanisms with buffering have to be implemented between a sending and a receiving node. The time span which an event message has to wait in a buffer before it can be processed reduces the temporal accuracy of the observation and must thus be limited. The provision of the proper buffer size is a delicate problem in the design of ET-systems, since the estimation of a particular peak demand can only be performed on a probabilistic basis.

Communication Protocols

The communication protocols in ET-systems are event triggered. Since only the sender has knowledge about the point in time when a message has to be transmitted, the error detection is based on a timeout at the sender waiting for an acknowledgement message from the receiver (*Positive Acknowledgement or Retransmission* Protocol). In these protocols k-fault-tolerance is achieved by retransmitting a lost message k times.

To avoid a congestion of the communication system in case of coincident events, explicit congestion control mechanisms have to be implemented in the network layer of the communication system.

To maintain a consistent order of the events at all receivers, atomic broadcast [Cri85] protocols have been proposed as basic communication mechanisms in distributed ET-systems. The *temporal uncertainty* of these asynchronous communication protocols, i.e, the difference between the maximum protocol latency and the minimum protocol latency, can be significant and has an adverse effect on the temporal accuracy of the information [Kop90].

Scheduling Strategy

Operating systems for ET-systems are demand driven and require a dynamic scheduling strategy. In general, the problem of deciding whether a set of real-time tasks whose execution is constrained by some dependency relation (either mutual exclusion or precedence), is schedulable is NP-hard [Mok83]. Finding a feasible schedule, provided it exists, is another NP-hard problem. The known analytical solutions to the dynamic scheduling problem [Liu73,Sha90] assume stringent constraints on the task set which are unrealistic for practical reasons. In practice most ET-systems resort to static priority scheduling. During the commissioning of the system the static priorities are tuned to handle the observed load patterns. No analytical guarantees about the peak load performance can be given.

The processing delays caused by the execution of the interrupt handlers and the buffer managers in ET operating systems are often neglected, although they can be significant. Experiments have shown that the temporal variability of the execution time of a hard real-

time task depends more on these unpredictable operating system activities then on the data-dependency of the task in question [Pus91].

Replica Determinism

Many real-time systems have to be fault-tolerant, i.e., they have to provide the specified timely service despite the occurrence of faults specified in the *fault-hypothesis*. In many real-time applications the time needed to perform checkpointing and backward recovery after a fault has occurred is not available. Therefore fault-tolerance in distributed real-time systems has to be based on active redundancy. Active redundancy requires *replica determinism*, i.e., the active replicas must take the same decisions at about the same time in order to maintain state synchronism. If replica determinism is maintained, fault-tolerance can be implemented by duplex fail-silent selfchecking nodes (or by Triple Modular Redundancy with voting if the fail-silent assumption is not supported).

State synchronism is difficult to achieve in asynchronous ET-systems based on dynamic preemptive scheduling strategies. Therefore alternate models have been developed for the implementation of fault-tolerance, e.g., the *leader-follower model* of DELTA 4 [Pow88]. In this model the leader takes all nondeterministic decisions and forces the follower to take the same decisions. This requires some additional communication activity between the leader and the follower.

The time constrained reintegration of repaired replicas is still an open research issue in ET architectures .

TIME-TRIGGERED SYSTEMS

Whereas an ET-system the information about the occurrences of events has to be spread promptly within the distributed computing system, the rapid dissemination of the current states of the RT-entities to all nodes of the distributed computing system is the fundamental concern in a TT-system. This dissemination of the states is performed periodically in periods which match the dynamics of the relevant RT-entities. We call the periodic sequence of time points at which an RT-entity is observed its *observation grid* [PDC90]. Independent of the current activity in a RT-entity, the state of a RT-entity is polled at a constant rate. Although it is impossible to influence the time of occurrence of a chance event in the controlled object, the time of recognition of such an event is restricted to the grid points of the observation grid

in a TT-architecture. Because of this *enforced regularity*, TT-architectures are inherently more predictable than ET-architectures.

Flow Control

The flow control in a TT-system is *implicit*. During system design appropriate observation, message, and task activation rates are fixed for the different RT-entities, based on their specified dynamics. It has to be assured at design that all receiver processes can handle these rates. If the RT-entities change faster than specified, then some short lived intermediate states of the RT-entities will not be reflected in the observations and will be lost. Yet, even in a peak load situation, the number of messages per unit time, i.e., the message rate, remains constant.

This implicit flow control will only function properly if the instrumentation of a TT-system supports the *state view*, i.e., the information produced by a sensor must express the current state of the RT-entity and not its change of state. If necessary, a local microcontroller has to transform the initial event information generated by a sensor into its state equivalent. Consider the example of a push button, which is a typical *event sensor*. The local logic in this sensor must assure that the state "push button pressed" is true for an interval that is longer than the grid granularity of the observation grid of this RT-entity.

Communication Protocols

The most common message handling primitives in distributed systems are biased toward event information, i.e., a new message is queued at the receiver and consumed on reading. In TT-systems a more appropriate message model for handling state information is required. We call such a message model a *state message* model and the corresponding message a state message [Kop85]. The semantics of a state message is related to the semantics of a variable in programming languages. A new version of a state message overwrites the previous version. A state message is not consumed on reading. State messages are produced periodically at predetermined points in real-time, the grid points of the observation grid, known a priori to all communicating partners.

The communication protocol for the dissemination of state messages in a TT-system is a reliable broadcast protocol. By *reliable broadcast* we mean a protocol which makes it possible for the receiver to detect the loss of any broadcast message. In a TT-system this error detection is performed by the receiver of the information based on the global knowledge about the expected arrival time of each message. Fault-tolerance can be achieved by massive redundancy, i.e., sending a message k+1 times if k transient failures are to be tolerated.

The information flow in TT-protcols is unidirectional. It is the responsibility of the receiver to verify that all required messages are available at the proper time. In a TT-system it is thus possible to add additional receivers without changing the message rate or the communication protocol.

Scheduling Strategy

Operating systems for TT-systems are based on a set of static predetermined schedules, one for each operational mode of the system. These schedules have to consider all necessary task dependencies and provide an implicit synchronization of the tasks at run time. They are constructed at compile time. At run time a simple table lookup is performed by the operating system to determine which task has to be executed at particular points in real time, the grid points of the *action grid* [PDC90]. The difference between two adjacent grid points of the action grid determines the *basic cycle time* of the real-time executive. The basic cycle time is a lower bound for the responsiveness of a TT-system.

In TT-systems all input/output activities are preplanned and realized by polling the appropriate sensors and actuators at the specified times. Access to the LAN is also predetermined, e.g., by a synchronous time division multiple access protocol.

In the MARS system [Kop89], which is a TT-architecture, the gridpoints of the observation grid, the action grid and the access to the LAN are all synchronized with the global time. Since the temporal uncertainty of the communication protocols is smaller than the basic cycle time, the whole system can be viewed as a distributed state machine [Sch90].

Replica Determinism

In a TT-system all task switches, mode switches, and communication activities are synchronized globally by the action grid. Nondeterministic decisions can be avoided and replica determinism can be maintained without additional interreplica communication.

The basic cycle time of a TT-system introduces a discrete time base of specified granularity. Since a TT-system operates quasi-synchronously, TMR structures as well as selfchecking duplex nodes can be supported for the implementation of active redundancy without any difficulty.

The reintegration of repaired components is well understood in TT-architectures [Kop89].

COMPARATIVE EVALUATION

Predictability

TT-systems require careful planning during the design phase. First it is necessary to establish the observation grid for the RT-entities and then the maximum execution time of time-critical tasks must be determined. After the allocation of the tasks to the nodes and the allocation of the communication slots on the LAN, appropriate execution schedules have to be constructed offline. Because of this accurate planning effort, detailed plans for the temporal behavior of each task are available and the behavior of the system in the domain of time can be predicted precisely.

In an ET-system it is not necessary to develop such a set of detailed plans during the design phase, since the execution schedules are created dynamically as a consequence of the actual demand. Analytical schedulability tests do not exist for distributed systems with mutual exclusion and precedence relations between tasks. Depending on the sequence and timing of the events which are presented to the system in a specific application scenario, different schedules unfold dynamically which may or may not meet the timeliness requirements.

Testability

The confidence in the timeliness of an ET-system can only be established by extensive system tests on simulated loads. Testing on real loads is not sufficient, because the *rare C-events*, which the system has to handle (e.g., the occurrence of a serious fault in the controlled object), will not occur frequently enough in an operational environment to gain confidence in the peak load performance of the system. The predictable behavior of the system in rare-event situations is of paramount utility in many real-time applications

Since no detailed plans for the intended temporal behavior of the tasks of an ET-system exist, it is not possible to perform "constructive" performance testing at the task level. In a system where all scheduling decisions concerning the task execution and the access to the communication system are dynamic, no *temporal firewalls* exist, i.e., a variation in the timing of any task can have consequences on the timing of many other tasks in different nodes. The critical issue during the evaluation of an ET-system is thus reduced to the question, whether the simulated load patterns used in the system test are representative of the load patterns that will develop in the real application context. This question is very difficult to answer with confidence. Field maintenance data indicate that some application scenarios cannot be adequately reproduced in the simulated system test [Geb88].

In a TT-system, the results of the performance test of every system task can be compared with the established detailed plans. Since the time-base is discrete and determined by the granularity of the action grid, every input case can be observed and reproduced in the domains of time and value. Therefore testing of TT-systems is more systematic 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 also 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 grid is not relevant [Sc90a].

Resource Utilization

In a TT-system all schedules are fixed and planned for the peak load demand in the specified operating mode. The time-slot assigned to a specific task must be at least as long as the maximum execution time of this task. If the difference between the average and the maximum execution time of a task is large, then in most cases only a small fraction of the allocated time slot will be needed by this task. If many different operating modes have to be considered, then there is the potential problem of combinatorial explosion of the number of static schedules.

In an ET-system only those tasks that have been activated under the actual circumstances have to be scheduled. Since the scheduling decisions are made dynamically, the CPU will be available again after the actual (and not the maximum) task execution time. On the other hand, run-time resources are required for the execution of the dynamic scheduling algorithm and the synchronization and buffer management.

If load conditions are low or average, then the resource utilization of an ET-system will be much better than that of a comparable TT-system. In peak load scenarios the situation can reverse, since the time available for the execution of the application tasks is reduced by the increasing processing time required for executing the interrupt handling, buffer management, synchronization, and scheduling algorithms.

Extensibility

Every successful system must 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 task and the verification of the temporal properties of the modified system.

In ET-systems it is easy to modify an operative task or to add a new task to an existing node, since all scheduling and synchronization decisions are deferred until the activation of these tasks at run time. Adding a new node requires the modification of some protocol parameters. Yet, a local change in the execution time of one task can impact the temporal properties of another task in a different node, e.g., by delaying its access to the LAN. Since the temporal effects of a change are not encapsulated to a task slot or a node, a local change can ripple through the entire system. Because no binding temporal parameters for the individual tasks are specified, a retesting of the temporal properties of the total system is required to assess the temporal consequences of a change in an individual task. Considering that both the probability of change and the system test-time are proportional to the number of tasks, the cost of assessing the consequences of a local change increases more than linearly with the number of tasks, i.e., with system size. Such an architecture does not scale well to large applications.

From a temporal point of view, every task of a TT-system is encapsulated. As long as a modified task does not exceed the specified maximum execution time of the original task, the change has no temporal effect on the rest of the system. If the maximum execution time is extended, or a new task is added, then the static schedules have to be recalculated. 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 communication slot must be generated by recalculating the communication schedules.

If the number of tasks is required to change dynamically during system operation, then it is not possible to calculate static schedules. This is the case where the number of RT-entities cannot be fixed. In such applications, ET-systems are the only alternative.

To summarize, extending an ET-system requires retesting the whole system, extending a TT-system may require a recalculation of the static schedules.

Assumption Coverage

Every control system is built on a set of assumptions concerning the behavior of the controlled object and of the computer. The probability that these assumptions are violated limits the dependability of the whole application [Pow90]. We denote those assumptions that refer to the behavior of the controlled object the *external assumptions* and those that refer to the behavior of the computer system the *internal assumptions*.

In a TT-system the critical external assumptions refer to the rate of change of the values of the RT-entities. The dynamics of a RT-entity determines the granularity of the observation grid 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. The dominant internal assumption in a TT-system refers to the maximum execution time of tasks, which must be evaluated during the design.

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 lost.

CONCLUSION

The implementation of a given external specification by a TT-design requires a detailed design phase. In this design phase the maximum execution time of all time critical programs must be established and the execution schedules for all operational modes must be calculated. As a result of this detailed planning effort, the temporal behaviour of a TT-design is predictable. If an ET-design is chosen, this detailed planning phase is not necessary. On the other side, the verfication of an ET-design demands much more extensive system tests than the verification of a TT-design. Since an ET-design does not support any temporal firewalls, it does not scale as well as a TT-design. From the point of view of resource utilization, an ET-design will, in many cases, be superior to that of an TT-design.

The investigation, whether the advantageous properties of these two contrary architectures can be combined into a single cohererent architecture is an interesting research isssue. One possible solution is the application of an ET-design within a node, but a TT-design for the communication between the nodes of a distributed system. Such an architecture would support temporal firewalls between the nodes and thus provide excellent testability at the architectural level, while increasing the flexibility of scheduling and the resource utilization within the nodes.

ACKNOWLEDGEMENT

This work has been supported, in part, by ESPRIT Basic Research Action Nr. 3092, Project PDCS. The author would like to thank the MARS group, in particular W. Schütz, for helpful comments on an earlier version of this paper.

REFERENCES

[Cri85]
Cristian, F., Aghili, H., Strong R. Dolev,D., Atomic Broadcast: From Simple message diffusion to Byzantine Agreement, Proc. FTCS 15, Ann Arbor, Mich., June 1985, pp.200-206

[Geb88]
Gebman, J., Mciver, D., Shulman, H., Maintenance data on the fire control radar, Proceedings of the 8th AIAA Avionics Conference, San Jose, cal. Oct. 1988

[Kop85]
Kopetz, H., Merker, W., The Architecture of MARS, Proceedings FTCS 15, Ann Arbor Mich, USA, IEEE Press, June 1985

[Kop87]
Kopetz, H., Ochsenreiter, W., Clock Synchronization in Distributed Raltime Systems, IEEE Transactions on Computers, August 1987, pp.933-940

[Kop89]
Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W., Senft, C., Zainlinger, R., Distributed Fault-Tolerant Realtime Systems: The MARS Approach, IEEE Micro, Vol. 9, Nr. 1, pp., 25-40, Febr. 1989

[Kop90]
Kopetz, H., Kim, K., Temporal Uncertainties in Interactions among Real-Time Objects, Proc. of the 9th IEEE Symp. on Reliable Distributed Systems, Huntsville, Al, Oct. 1990

[LeL89]
G.LeLann, Critical Issues in Real-Time Computing, In: Proc. of Workshop on Communication Networks and Distributed Operating Systems within the Space Environment, ESTEC, October 1989

[Liu73]
Liu, C.L., Layland, J.W., Scheduling Algorithms for Multiprogramming in a Hard Realtime Environment, Journal of the ACM, Februay 1973, pp.46-61

[Mok83]
Mok, A.K., Fundamental design problems of distributed systems for the hard real-time environment, Ph.D. dissertation, M.I.T., 1983

[PDC90]
Specification and Design for Dependability, Esprit Project Nr. 3092 (PDCS: Predictably Dependable Computing Systems), 1st Year Report, LAAS, Toulouse, 1990

[Pow88]
Powell, D., Bonn, G., Seaton, S., Verissimo, P., Waeselnyk, F., The Delta-4 Approach to Dependability in Open Distributed Computing Systems, Proc. of the FTCS-18, IEEE Press, pp.246-251

[Pow90]
Powell, D., Fault Assumptions and Assumption Coverage, PDCS report RB4 (2nd year deliverable 1991) and Report LAAS, Toulouse Nr. 90.074, Dec. 1990

[Pus91]
Puschner, P., Vrchoticky, A., On the Feasibility of Response Time Predictions--An Experimental Evaluation, Research Report No.2/91, Technical University of Vienna, Institut für Technische Informatik, January 1991

[Sha90]
Sha, L., Rajkumar, R., Lehoczky, J.P., Priority Inheritence Protocols: An Approach to Real-Time Synchronization, IEEE Transactions on Computers, Vol. 39, No. 9, Sept. 1990, pp. 1175-1185

[Sch90]
F.B. Schneider, Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial, ACM Computing Surveys, Vol 22, Nr. 4, December 1990, pp. 299-320

[Sc90a]
Schütz, W, A Test Strategy for the Distributed Real-Time System MARS, IEEE Compeuro 90 "Computer Systems and Software Engineering", Tel Aviv, Israel, May 1990, pp. 20 - 27.