Should Responsive Systems be Event-Triggered or Time-Triggered?

Hermann Kopetz

Research Report Nr. 16/93
November 1993

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

Phone: (43) 1 58801 8180
Fax: (43) 1 58 69 149
E-Mail: hk@vmars.tuwien.ac.at

IEICE TRANS. INF. & SYST., VOL. E76-D, NO. 11 NOVEMBER 1993

INVITED PAPER *Special Issue on Responsive Computer Systems*

Should Responsive Systems be Event-Triggered or Time-Triggered ?

Hermann KOPETZ†, *Nonmember*

SUMMARY In this paper the two different paradigms for the design of responsive, i.e., distributed fault-tolerant real-time systems, the event-triggered (ET) approach and the time-triggered (TT) approach, are analyzed and compared. The comparison focuses on the temporal properties and considers the issues of predictability, testability, resource utilization, extensibility, and assumption coverage.

key words: *real-time, distributed systems, fault tolerance*

1. Introduction

A growing explicit demand for dependable real-time computer systems can be noticed in the last few years. This demand is caused by the increasing cost advantage of microlectronics solutions over the classical control system technologies. Already today it is cost effective to replace a mechanical control system in a mass produced product, such as an automobile, by a highly integrated microcontroller. These new computer based control systems do not only provide the functionality of their mechanical predecessors, they also have the potential for new improved services.

From a user's point of view it makes little difference whether a specified service is implemented by a fault-tolerant distributed system or by a highly reliable central system. Whereas the timeliness requirement is a service attribute at the user/system interface, distribution and fault-tolerance are implementation characteristics. They are, however, the most promising implementation technologies for the design of cost effective dependable real-time systems. One way to satisfy the intensifying demands on computer system dependability and flexibility is thus by the integration of the disciplines of real-time computing, fault-tolerant computing, and distributed computing into a single new field—the field of responsive computer systems.

A real-time system has to meet the deadlines dictated by its environment. If a real-time system misses a deadline, it has failed. Since temporal properties are system properties, they depend on the entire distributed 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 a real-time architectures are cultivated. In an event-triggered (ET) architecture the system activities, such as sending a message or starting the execution of a task, are triggered by the occurrences of events in the environment or in the computer system. In a time triggered (TT) architecture the activities are triggered by the progression of the global time. Whereas TT-designs are prevailing in safety critical real-time applications, the ET-designs are common in non-safety critical applications.

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

2. 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 make the following assumptions:

(1) 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.
(2) 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 [4].
(3) The nodes are fail-silent and messages which are mutilated by the communication system can be detected and discarded.
(4) Some nodes, the interface nodes, support a connection to the intelligent instrumentation, i.e., the

Manuscript received February 19, 1993.
† The author is with Technical University of Vienna, Vienna, Austria.

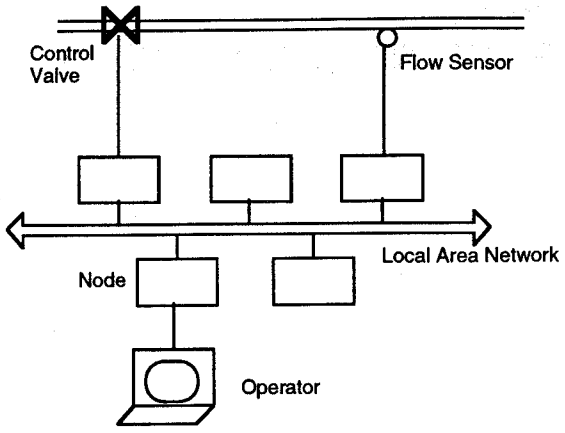

Fig. 1 Real-time application.

sensors and actuators, which are at the interface to the controlled object.

We call a *physical sensor* with an associated microcontroller an *intelligent sensor*. It is purpose of an intelligent sensor to encapsulate and hide the detailed technical interface of the physical sensor from the client computer and to provide a preprocessed abstract high level sensor reading to the client computer.

2. 1 RT Entities and Observations

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. An *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). An 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).

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.

$$O = \langle \text{entity name, value, point of observation} \rangle$$

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.

In typical real-time applications, the computer system has to perform a multitude of different functions concurrently, 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. 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.

2. 2 Downcalls and Upcalls

We distinguish between two different types of observations, downcalls and upcalls. The first is normally related to the notion of sampling, the second to the notions of interrupt in ET systems and polling in TT systems.

In the downcall mode (sampling) the intelligent sensor delivers the most recent observation about the state of the RT-entity either as a consequence of receiving a request (a downcall) from its client or at predetermined points in time (the sampling points). We call the data that is related to a downcall *sampled data* or s-data. The periodic observation of an RT-entity, e.g., a temperature, is a typical example for the generation of s-data. s-data is always 'fresh" since it has been collected immediately before its delivery.

In the upcall mode the intelligent sensor reports to the client about a significant state change that has occurred in the RT entity in the recent history. We call the data that is related to an upcall *occurrence data* or o-data. Whereas in an ET system o-data is reported immediately by the interrupt mechanism, the delivery of the o-data in a TT system can be delayed by up to one observation granule, the polling period of the TT system. We distinguish between two different types of upcalls, the predictable events (the P-events) and the chance events (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-even 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 impor-

tant 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.

2.3 Temporal and Value Accuracy

Observations of the controlled object taken at a particular point in time are transmitted to, manipulated by, and stored within RT-objects [6] 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 an 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 an 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 an 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.

3. Event Triggered Systems

In a purely event triggered system all system activities are initiated by the occurrence of significant events in an RT-entity or an RT-object. A *significant event* is a change of state in an 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.

An intelligent sensor is event triggered (ET) if it reports a new observation to its client when it realizes a significant change in the value of the associated RT entity. This event triggered interruption of the concurrent operation of the client computer is normally realized with the help of the interrupt mechanism.

3.1 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.

A difficult problem in the design of an ET system relates to the problem of flow-control of upcalls. Since a computer system has only a limited processing capability, it is important to limit the flow of o-data, i.e., the interrupt requests, from the sensor to the client computer, both during normal operation and in the case the sensor is faulty (e.g., an intermittent contact failure leading to an interrupt shower). This is normally done by specifying a minimal interval d_{min} between two interrupt requests. This interval d_{min} has to be enforced by the intelligent ET-sensor. If the interval between the occurrence of o-data is smaller than d_{min}, then all following occurrences have to be stored and delayed until the end of d_{min}. The immediate response of an intelligent ET-sensor can be guaranteed only for the first occurrence of o-data.

It is evident that an (unintelligent) physical ET sensor that is directly connected to an interrupt line of the client computer cannot realize this flow control and thus can upset the operation of the client computer in case it requests interrupt service too frequently.

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.

3.2 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 [1] 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 [6].

3.3 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 [11]. Finding a feasible schedule, provided it exists, is another difficult problem. The known analytical solutions to the dynamic scheduling problem [10], [16] assume stringent constraints on the task set which are difficult to maintain. 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 [15].

3.4 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 suppored).

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 [14]. 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.

4. Time-Triggered Systems

TT systems are designed according to the principle of resource adequacy [9] i.e., it is assumed that sufficient computing resources are available to handle the specified peak load scenario, even if all faults specified in the fault hypothesis occur simultaneously.

Whereas in an ET-system the information about the occurrences of events has to be diffused 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 lattice [12]. The distance between the lattice points of this observation lattice is called the observation granularity g_0 [8]. Independent of the current activity in an RT-entity, the state of an 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 lattice points of the observation lattice in a TT-architecture. Because of this *enforced regularity*, TT-architectures are inherently more predictable than ET-architectures.

4.1 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. There is no explicit acknowledgement. In a TT-system, an intelligent sensor is allowed to report its findings only at the lattice points of a globally synchronized observation lattice.

This implicit flow control in TT-systems will only function properly if the instrumentation supports the *state view*, i.e., the information produced by an intelligent sensor must express the best estimate of the current state of the RT-entity and not its change of state. If necessary, the intelligent sensor has to transform the initial event information generated by a physical 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 observation granularity g_0.

In a TT system upcalls have to be stored in the intelligent sensor until they are polled at the next latticepoint of the observation lattice. This distributed storage of upcalls in the intelligent sensors solves the problem of the correlated event showers in TT systems. A TT sensor can delay the notification about a significant state change of the RT-entity, i.e., the o-data, by g_0. This observation granularity g_0 limits the temporal precision of polled events. If this precision is insufficient for the given purpose, then it is possible that the intelligent sensor reports about the point in time, when the significant occurrence happened within the last observation granule with the resolution of the perception granularity of the sensor. We call such an observation that contains, in addition to the data value, some timing information relative to the last latttice point, a *timed observation*. Timed observations solve the problem of the recording precision, but not the problem of the extra delay caused by the observation granularity in a TT system. Timed observations provide the time information in the data domain, but not as a control element.

If the RT-entities change faster than specified, then the intelligent sensor has to determine the state value that is most relevant for the next point on the observation lattice. For this purpose it applies an appropriate abstraction function over the physical sensor readings. In such a situation some short lived less important 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.

4.2 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 [3]. 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 lattice points of the observation lattice, known a priori to all communicating partners.

The communication protocol for the dissemination of state messages in a TT-system is a diffusion based 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-protocols 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.

4.3 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 lattice points of the *action lattice* [12]. The difference between two adjacent lattice points of the action lattice determines the *basic cycle time* of the real-time executive. The basic cycle time is a lower bound for the responsiveness of a TT-tystem.

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 [5], which is a TT-architecture, the latticepoints of the observation lattice, the action lattice 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 [17].

4.4 Replica Determinism

In a TT-system all task switches, mode switches, and communication activities are synchronized globally by the action lattice. 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 [5].

5. Comparative Evaluation

5. 1 Predictability

TT-systems require careful planning during the design phase. First it is necessary to establish the observation lattice 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. Depending on the sequence and timing of the events which are presented to the system in a specific application scenario, different schedules unfold dynamically.

5. 2 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 encapsulation* exists, 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 [2].

In a TT-system, the results of the performance test of every task can be compared with the established detailed plans. 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 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 lattice is not relevant [18].

5. 3 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 scheduling algorithms, the buffer management, and the explicit synchronization of the tasks.

5. 4 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. There are no control signals crossing the interface between two TT subsystems. Every subsystems excercises autonomous control derived from the progression of the synchronized time that is available locally. 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.

5.5 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 [13]. 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 an 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. 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 compromised. The critical internal assumption in ET system concerns the services of the dynamic scheduler. Will the scheduler find a feasible solution for the evolving task scenario within the given time constraints ?

6. 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 behavior of a TT-design is predictable. If an ET-design is chosen, this detailed planning phase is not necessary. On the other side, the verification 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 encapsulation, 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 a TT-design.

The investigation, whether the advantageous properties of these two contrary architectures can be combined into a single coherent architecture is an interesting research issue. 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 encapsulation 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 PDCS. The author would like

to thank D. Seaton from DELTA 4 and the MARS group, in particular W. Schütz, for useful comments on an earlier version of this paper. This is a revised version of a presentation on *Time-Triggered versus Event-Triggered Real-Time Systems* [7], given at the Dagstuhl Seminar, Operating Systems of the Nineties and Beyond, in July 1990.

References

[1] Cristian, F., Aghili, H., Strong, R. and Dolev, D., "Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement," *Proc. ETCS15*, Ann Arbor, Mich., pp. 200-206, Jun. 1985.

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

[3] Kopetz, H. and Merker, W., "The Architecture of MARS," *Proceedings FTCS 15*, Ann Arbor Mich, USA, IEEE Press, Jun. 1985.

[4] Kopetz, H. and Ochsenreiter, W., "Clock Synchronization in Distributed Realtime Systems," *IEEE Trans. Comput.*, pp. 933-940, Aug. 1987.

[5] Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W., Senft, C. and Zainlinger, R., "Distributed Fault-Tolerant Realtime Systems: The MARS Approach," *IEEE Micro*, vol. 9, no. 1, pp. 25-40, Feb. 1989.

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

[7] Kopetz, H., "Time-Triggered versus Event-Triggered Real-Time Systems," *Proc. of the Workshop: Operating Systems of the 90ties and Beyond*, Springer Lecture Notes on Computer Science, 1991.

[8] Kopetz, H., "Sparse Time versus Dense Time in Distributed Real-Time Systems," *Proc. of the 14th Distributed Computing System Conference*, Yokohama, Japan, IEEE Press, Jun. 1992.

[9] Cy-Clone., *An Approach to the Engineering of Resource Adequate Cyclic Real-Time Systems*, Journal of Real-Time System, vol. 4, pp. 55-83, 1992.

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

[11] Mok, A. K., *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*, Ph. D. Dissertation, M. I. T., 1983.

[12] *Specification and Design for Dependability*, Esprit Project no. 3092 (PDCS: Predictably Dependable Computing Systems), 1st Year Report, LAAS, Toulouse, 1990.

[13] Powell, D., *Fault Assumptions and Assumption Coverage*, PDCS report RB4 (2nd year deliverable 1991) and Report LAAS, Toulouse no. 90.074, Dec. 1990.

[14] ed, Powell D., *DELTA-4: A Generic Architecture for Dependable Distributed Computing*, Esprit Research Reports Project 818/2252, Springer Verlag, Heidelberg, 1991.

[15] Puschner, P. and 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, Jan. 1991.

[16] Sha, L., Rajkumar, R. and Lehoczky, J. P., "Priority Inheritence Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. Comput.*, vol. 39, no. 9, pp. 1175-1185, Sep. 1990.

[17] Schneider F. B., "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial," *ACM Computing Surveys*, vol. 22, no. 4, pp. 299-320, Dec. 1990.

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

Hermann Kopetz received his PhD degree in physics "sub auspiciis praesidentis" from the University of Vienna, Austria in 1968. He was a manager of a computer process control department at Voest Alpine in Linz, Austria, before joining the Technical University of Berlin as a professor for Computer Process Control in 1978. Since 1982 he is professor for Real-Time Systems at the Technical University of Vienna, Austria. His research interests focus at the intersection of real-time systems, fault-tolerant systems, and distributed systems. Dr. Kopetz is the chief architect of the MARS project on distributed fault-tolerant real-time systems at the Technical University of Vienna. He has published more than 70 papers in these fields and has been active in the organisation of many international conferences. From 1990 to 1992 he was the chairman of the IEEE TC on Fault-Tolerant Computing.