MODE HANDLING IN THE TIME-TRIGGERED ARCHITECTURE

H. Kopetz, R. Nossal, R. Hexel, A. Krüger, *
D. Millinger, R. Pallierer, C. Temple *,[1]

* *Institut für Technische Informatik, Technische Universität Wien, Austria*

M. Krug **

** *Daimler Benz AG, Stuttgart, Germany*

Abstract:
Large real-time applications often impose stringent requirements on the system architecture concerning the number of different operating conditions under which they must operate. Modes and mode changes are a powerful concept to represent these operational phases within the system architecture.
This paper focuses on the modes and the mode change strategy available in the time-triggered architecture. The time-triggered architecture is outlined and the principle of modes and mode changes is introduced. The modes introduced in the TTA meet the requirements imposed by different operating conditions of hard real-time systems. The requirements for mode changes are identified and the conflict between consistency and speed is resolved by introducing two distinct types of mode changes.
Keywords: Distributed Real-Time Systems, Static Scheduling, Modes

1. INTRODUCTION

Safety critical hard real-time application typically exhibit a number of mutual exclusive operational phases. Each of these phases requires different control activities. A computer system on board an aircraft, for instance, performs different actions during taxiing on the ground and during level flight with the autopilot engaged. Computer systems controlling hard real-time applications must offer real-time and fault-tolerance capabilities. The Time-Triggered Architecture (TTA) based on the Time-Triggered Protocol (TTP) (Kopetz and Grünsteidl, 1994) meets both demands.

The TTA, like any time-triggered system, relies on static scheduling of tasks and messages. These schedules are created pre-runtime, thus the system cannot adapt to changing conditions during runtime. This problem is overcome by introducing the concept of modes. A *mode* reflects a certain operational phase of the real-time application. Characteristics of a mode are the task set executed in this mode and the set of messages transmitted in this mode.

Mode changes are used to change between statically defined modes. They are the only means to change the temporal behaviour during operation. If the operating conditions or the operational phase change, the controlling computer system has to perform a mode change. Changing to another mode means to proceed according to a different temporal control pattern, i.e., a different task and message schedule. Work on

[1] This work has been supported in part by the ESPRIT project "Time-Triggered Architecture (TTA)" and the BRITE EURAM project "Safety Related Fault-Tolerant Systems in Vehicles (X-by-Wire)".

mode change extensions for a priority inheritance based scheduler is presented in (Sha *et al.*, 1989). Tindell et al. (Tindell *et al.*, 1992) deal with mode changes in a deadline monotonic scheduling algorithm. In the context of time-triggered architectures Fohler (Fohler, 1993) has developed a static scheduler capable of various modes and mode changes.

This paper shows how modes are integrated into the TTA and the protocol. The communication system of the TTA provides the mechanisms for mode handling to the application program. Hence the application has a powerful means for adapting to different conditions that is yet easy to use.

The rest of the paper is organized as follows. Section 2 gives an overview of the Time-Triggered Architecture focussing on time-triggered system activation and the data structures containing the temporal control pattern. An introduction to modes and mode changes is given in Section 3. Based on this general concept Section 4 goes into detail on the mode handling in the Time-Triggered Architecture. The paper is concluded in Section 5.

2. OVERVIEW OF THE ARCHITECTURE

In the Time-Triggered Architecture the controlling computer system consists of at least one *computational cluster* (figure 1). A computational cluster comprises a set of self-contained *node computers*. The nodes communicate via a (replicated) broadcast bus using the Time-Triggered Protocol. Therefore, a single cluster is also characterized as being a *broadcast domain*.

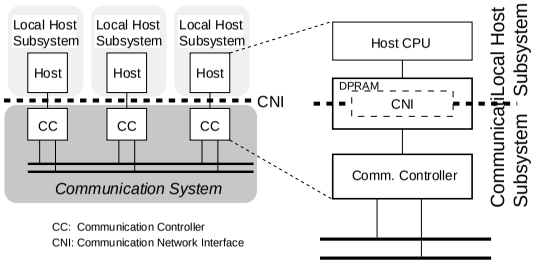

Fig. 1. Architecture of a Cluster and a Node.

Messages mutilated in the communication system are assumed to be detected and discarded. Further, nodes are assumed to be *fail-silent*, i.e., only crash failures and omission failures can occur. Self-checking mechanisms causing a high error detection coverage are employed to enforce this property. Extensive fault-injection experiments with the MARS prototype system (Kopetz *et al.*, 1989) have shown the feasibility of this assumption (Karlsson *et al.*, 1995).

Within a cluster, a global time base — of sufficiently small granularity with respect to a specific application — is established by synchronizing the clocks located within the nodes (Kopetz *et al.*, 1995).

2.1 *Structure of a Node Computer*

Two main subsystems can be identified within a node computer: (1) the *communication subsystem*, which autonomously and reliably transmits messages between the nodes of the distributed system, and (2) the *host subsystem*, which executes the distributed real-time application. The operation of both subsystems is controlled by static configuration data structures. The *MEssage Descriptor List (MEDL)* resides in the communication subsystem (the TTP controller), and contains the attributes of the messages sent and received by the protocol. According to this list the TTP controller periodically and autonomously reads the messages to be sent from the CNI and writes received messages to the CNI. Similarly, the attributes of the tasks executed by the host processor under the control of a time-triggered operating system are stored in the *TAsk Descriptor List (TADL)*.

The interface between communication and host subsystem is called the *Communication Network Interface (CNI)* (Krüger and Kopetz, 1995). It is located in a dual-ported memory and enables the local host processors to send or receive messages by writing to or reading from this memory area. The main part of the CNI facilitating this message exchange is designed as a *data sharing interface* without any control information exchange. This reduces the probability of control error propagation and temporally encapsulates the subsystems. Consequently, the CNI can be seen as a temporal firewall between communication subsystem and host subsystem, each subsystem representing an error-containment region.

2.2 *Mechanisms of the Time-Triggered Protocol*

TTP is an integrated time-triggered protocol that provides prompt transmission of messages with high data efficiency, a responsive membership service, a fault-tolerant clock synchronization service, mode change support, error detection with short latency, and distributed redundancy management. TTP makes best use of the a priori information available in a TTA to reduce the number and size of messages, e.g., by retrieving the message identification from the a priori known time of message reception.

In TTP all nodes are forced to agree on their *controller states (C-states)*. The controller state consists of three fields: the MEDL position, the global time and the membership. The MEDL position field is a pointer to the current entry in

the MEDL, i.e., it identifies the current mode and TDMA slot. The time field contains the global time at the beginning of the current TDMA slot. The membership field indicates which nodes have been active and inactive at their last membership point. To enforce C-state agreement between a sender and a receiver the CRC of a normal message is calculated over the message contents concatenated to the local C-state. A receiver can only interpret the frame if sender and receiver agree about the controller state at the time of sending and receiving. In case the C-state of the sender is different from the C-state of the receiver, any message sent by the sender will be discarded by the receiver, because the CRC check will fail.

Access to the transmission medium is controlled by a static TDMA scheme. Each node is allowed to send messages only during a predetermined time span, called its *TDMA slot*. The sequence of the periodic TDMA slots is called a *TDMA round*. With regard to the duration of the TDMA slots and to the sending sequence of the nodes, all TDMA rounds are equal. However, the length and contents (the application data) of the frames may differ. The set of periodically recurring TDMA rounds (with possibly different message length and contents) is called a *cluster cycle*.

To implement the fail-silence behavior assumed above, each node is equipped with two *bus guardians (BGs)*. These devices autonomously check and regulate the access times of a node to the bus, i.e., they grant bus access to the node only within its TDMA slots independent of the remaining protocol execution.

3. MODES AND MODE CHANGES

The time-triggered system paradigm as introduced in the previous section stipulates a constant behavior of the system, i.e., the temporal control pattern does not change over time. This is a rather inflexible approach that restricts the applicability of time-triggered real-time systems. Hence a mechanism to change the temporal control during operation has to be devised. In this section an appropriate mechanism, *modes* and *mode changes* will be introduced.

3.1 *Modes*

A possibility of classifying real-time applications is the dynamics with respect to their behavior. On the one hand there are applications whose behavior does not change during their operation. They continuously perform the same function [2]. An example for such a system is a hydro power plant. On the other hand there are applications that consist of a number of distinct phases. In each phase the system performs a certain function. Consider the example of an aircraft. A flight from point A to point B consists of the phases taxiing on the airport, takeoff, climbing, level flight, descending, landing, and taxiing again. In each of these phases some functions are important for the operation of the plane, whereas others are of minor priority or need not be carried out at all.

A mode of a real-time computer system reflects a certain operating condition or operational phase of the controlled object. Each mode of operation is characterized by a task set that is executed in this mode and a set of messages transmitted during this mode. Thus each mode has its unique temporal control structure, i.e., its own MEDL and its own TADL. Different modes have different task and message sets and a different temporal control pattern. However, the task and message sets need not be disjunct. For instance — revisiting the example given above — a task handling the gyro compass, a part of a plane's navigation system is needed during taxiing on the airport as well as during the plane's flight phase.

3.2 *Mode Changes*

Mode changes are the only means to change the temporal control structure during operation. In a mode the assigned, predetermined MEDL and TADL is executed in a periodic manner. This scheme does not allow for changes of the function of the system. If the operating conditions or the operational phase change, the controlling computer system has to perform a mode change. Changing to another mode means to proceed according to a different temporal control pattern.

Jahanian (Jahanian *et al.*, 1988) presented three types of mode changes according to the handling of the currently executing tasks.

(1) The easiest possibility is to abort all tasks and start a complete new task set in the destination mode. In this case the mode change can be performed immediately.
(2) The counterpart to this possibility is to complete all currently executing tasks and switch to the new mode then.
(3) Only some of the current activities are finished before the mode change takes place.

Fohler (Fohler, 1993; Fohler, 1994) advanced this concept and introduced so-called *Transition Modes*. The schedule of a transition mode is designed to appropriately handle the termination of activities.

[2] In this characterization maintenance or installation are not considered to be normal operational phases.

3.2.1. *Flexibility vs. Safety*

So far modes have been identified to be a powerful concept for changing a system's function during operation. Yet this flexibility introduces a number of potential dangers into the system.

Changes at Wrong Point in Time The most obvious danger is a mode change at a "wrong" point in time. A mode change occurs at a wrong point in time, if the state of the controlled object does not require a mode change. Considering the above example of an aircraft a mode change to taxiing mode would be wrongly timed if it occurred during level flight.

Inconsistencies Mode changes can cause inconsistencies between the states of the nodes of a cluster. If only a subset of the nodes perform a mode change two cliques of nodes are formed. Each clique assumes the communication pattern of the mode it is in, the cliques cannot communicate with each other. Thus, a correct operation of the system cannot be guaranteed any more.

3.2.2. *Requirements*

To avoid the problems arising a number of requirements for mode changes can be identified.

- The points in time when mode changes are allowed must be determined before runtime like the temporal control patterns of the modes. During runtime mechanisms must ensure that mode changes can only be performed at these preplanned points in time.
- It must be assured that mode changes are executed only when this is required by the state of the controlled object. Note that this requirement is different from the first. The first requirement determines the points in time when changes can be taken. As the system operates in a periodic manner, these points in time recur periodically. This requirement determines if a change is to be performed at one of these points.
- Concepts for guaranteeing the consistent change from a mode to another must be found. In this context consistent means that all nodes of a cluster change the mode at the same point in time. In other words all nodes of a cluster must agree on the current mode at all points in time.

4. MODE HANDLING IN TTA

The previous section presented the basic concepts of modes and mode changes. In this section the mode handling strategy of TTP is explained. It will be shown how mode changes are initiated and which restrictions apply. Two types of mode changes supported by TTP will be introduced, immediate and deferred mode changes.

4.1 *The Mode Change Process*

In the section "Modes and Mode Changes" it was stated that modes reflect certain operating conditions of the controlled object. Interaction with the environment is only performed by the host CPU. Thus only the host CPU knows about the operating conditions of the controlled object and only the host CPU is able to determine whether a mode change is required. Consequently mode changes in TTP are initiated by the host subsystem.

If the operating conditions of the controlled object change, the host CPU monitoring these conditions issues a mode change request to its local TTP controller. This implies that only certain nodes are able to initiate mode changes to certain modes.

4.1.1. *Performing the Mode Change*

In the following a brief overview of the mode change procedure is given. Explaining all mechanisms, especially the handling of failures, is beyond the scope of this paper.

Deferred mode changes	
Deferred change 1	010
Deferred change 2	100
Deferred change 3	110
Immediate mode changes	
Immediate change 1	011
Immediate change 2	101
Immediate change 3	111
Special meanings	
Stay in current mode	000
Clear pending deferred change request	001

Table 1. The meaning of mode change bits in the frame header.

After the host CPU has passed a mode change request to its local TTP controller (see above subsection), the latter checks whether the node is allowed to request this change. The information on allowed mode change requests is stored in the MEDL of the respective mode and is determined during the off-line planning phase of the system. If the requested change is allowed the TTP controller sets the mode change bits in the header of the next frame it sends to the value corresponding to the requested successor mode (see table 1). As there are three mode change bits in the frame header and two bit patterns are required to encode special actions ("stay in current mode" and "clear pending deferred mode change") six successor modes can be addressed.

All other nodes receiving the frame containing the mode change request in the header check

in the current MEDL whether the sending node is allowed to request this mode change. If the requested change is allowed all nodes perform the mode change at the predetermined point in time (see following subsection).

It is important to note that the bus access pattern is identical in all modes, i.e., there is a fixed TDMA cycle pattern for all modes. This restriction is imposed by the Bus Guardian, in particular by its demand for utmost independence from the TTP controller. As mentioned in section "Overview of the Architecture" the Bus Guardian is not aware of the current operational mode of the computer system. Moreover, it is neither informed about mode changes nor is it able to notice them by himself. Thus different access patterns in different modes would interfere with the Bus Guardian's open-close mechanism.

4.1.2. *The Conflict between Consistency and Speed*

TTP allows up to six successor modes for each mode. The corresponding six mode changes are divided into two groups,

- three *deferred* mode changes and
- three *immediate* changes.

The two categories of changes reflect two different principles, consistency on the one hand and speed on the other hand. Whereas deferred mode changes are agreed upon by all nodes of the cluster and can be performed one TDMA cycle after the request at the earliest, immediate changes are executed immediately after the request. Both mode change concepts have specific advantages, neither of them is well suitable for all situations. Therefore TTP supports both principles, which are explained in more detail in the following subsections.

4.2 *Deferred Mode Changes*

Deferred mode changes can be used to reflect changes in normal operational conditions of the controlled object, e.g., a change from take-off mode to climb mode. As stated above deferred mode changes reflect the principle of consistency, i.e., maintaining an equal state among all nodes of a cluster and between the state of the protocol on the one hand and the state of the application on the other hand.

Deferred mode changes can be requested at any time during a cluster cycle. However, the mode change is performed at the beginning of the next cluster cycle. At this defined point in time the application should be in the *ground state* (Ahuja *et al.*, 1990). Application ground state means that no application task on any node is currently executing (either running or preempted). For this reason no task must be aborted or finished in the succeeding mode.

The possibility to request a deferred mode change at any time during the cluster cycle implies that the mode change is pending till the start of the next cluster cycle. When this point in time arrives each node changes to the requested successor mode autonomously, i.e., without further notice. This automatism is applicable to all correctly operating nodes that have received the frame containing the mode change request. Nodes that reintegrate into the ensemble after this frame has been sent are not aware of the pending mode change. Hence they cannot change at the predetermined point in time. To overcome this problem pending mode changes become part of the C-State. An integrating node receiving an I-frame also receives information on the pending mode change and can perform the change consistently with the rest of the ensemble.

The deferred mode change mechanism also allows a form of agreement on mode changes. Using a special bit pattern in the frame header (see table 1) pending mode change requests can be cleared again.

The mechanism presented so far does not restrict requests in any way. This means that more than one request can be issued during a cluster cycle. In this case the deferred mode change requests must be prioritized. If more than one change is requested the highest priority change is executed. Prioritizing can be done either by assigning static priorities to the numbers of deferred mode change requests or by dynamically making the last request the highest priority one. In the first possibility deferred change request 1 could be assigned the highest priority and deferred request 3 the lowest.

Deferred mode changes are addressed relatively, i.e., the number of the requested mode change is used to look up in a table that contains the absolute mode number of the destination mode. As opposed to direct addressing, where three bits to encode a mode change request would result in only seven possible modes of the system, the indirect addressing scheme facilitates an unbounded number of modes.

4.3 *Immediate Mode Changes*

An immediate mode change is executed immediately at the end of the FTU slot where the mode change was requested. The change itself is not agreed upon and thus reflects the principle of speed.

The immediate execution of this type of mode change requests avoids two problems arising with

deferred changes. First of all there is no need to prioritize changes. Since changes are carried out immediately after they have been requested there is at most one pending request at any time. For the same reason maintaining information about pending changes is not required.

Immediate mode changes are addressed absolutely, i.e., an immediate mode change leads to the same mode regardless of the mode the change was initiated from. Hence multiple immediate mode changes are idempotent. On the other hand the encoding scheme of the mode change bits in the frame header (see Table 1) restricts the number of immediate modes to three if direct addressing is used.

Though immediate changes can be handled easily in the communication subsystem they impose a number of serious problems on the application. Deferred changes are executed while the system is in ground state; immediate changes on the other hand cannot wait for such a ground state. Hence there are a number of application tasks executing when the mode change is invoked. These tasks must be aborted, which should not leave the application in an undefined state. In the destination mode the task set must be started amid a cycle unless there is no phase relation between the communication and the task schedule.

5. CONCLUSION

In this paper the concept of modes and mode changes as a means for changing the temporal properties of a time-triggered system has been presented. First, an overview of the time-triggered architecture was given. Then the basic concepts of modes and mode changes were outlined, followed by an explanation of the mode handling strategy used in the time-triggered protocol. Special emphasis was put on the description of the discrepancy between consistency and speed, which was addressed by the deferred and immediate mode change strategies employed in TTP.

The mode handling features presented in this paper are a service provided by the communication subsystem. A mode change strategy, making use of these mechanisms, must be implemented at the host level. Issues addressed by a mode change strategy encompass questions like how to reach agreement on the necessity of a mode change. In the context of the time-triggered operating system being developed by our group an appropriate mode change strategy will be defined and evaluated.

6. REFERENCES

Ahuja, M., A. D. Kshemkalyani and T. Carlson (1990). A basic unit of computation in distributed systems. In: *10th Int. Conf. on Distributed Computing Systems*. Paris, France. pp. 12–19.

Fohler, G. (1993). Realizing changes of operational modes with pre run-time scheduled hard real-time systems. In: *Responsive Computer Systems* (H. Kopetz and Y. Kakuda, Eds.). Vol. 7 of *Dependable Computing and Fault-Tolerant Systems*. pp. 287–300. Springer-Verlag.

Fohler, G. (1994). Flexibility in Statically Scheduled
Real-Time Systems. PhD thesis. Technisch-Naturwissenschaftliche Fakultät, Technische Universität Wien. Wien, Österreich.

Jahanian, F., R. Lee and A. Mok (1988). Semantics of modechart in real time logic. In: *Proc. of the 21st Hawaii International Conference on Systems Sciences*. pp. 479–489.

Karlsson, J., P. Folkesson, Jean Arlat, Yves Crouzet and Günther Leber (1995). Integration and Comparison of Three Physical Fault Injection Techniques. In: *Predictably Dependable Computing Systems*. Chap. V: Fault Injection, pp. 309 – 329. Springer Verlag.

Kopetz, H., A. Damm, Ch. Koza, M. Mulazzani, W. Schwabl, Ch. Senft and R. Zainlinger (1989). Distributed Fault-Tolerant Real-Time Systems: The MARS Approach. *IEEE Micro* **9**(1), 25–40.

Kopetz, H., A. Krüger, D. Millinger and A. Schedl (1995). A Synchronization Strategy for a Time-Triggered Multicluster Real-Time System. In: *Proc. 14th Symposium on Reliable Distributed Systems*. Bad Neuenahr, Germany.

Kopetz, H. and G. Grünsteidl (1994). TTP — A Protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer* pp. 14–23.

Krüger, A. and H. Kopetz (1995). A Network Controller Interface for a Time-Triggered Protocol. In: *SAE Symposium on Future Transportation Electronics: Multiplexing and In-Vehicle Networking*. Society of Automotive Engineers. SAE Paper No. 952576.

Sha, L., R. Rajkumar, J. Lehoczky and K. Ramamritham (1989). Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems* **1**(3), 243–265.

Tindell, K.W., A. Burns and A.J. Welling (1992). Mode changes in priority pre-emptively scheduled systems. In: *Proc. 13th Real-Time Systems Symposium*. pp. 100–109.