The Time-Triggered Architecture

HERMANN KOPETZ, FELLOW, IEEE AND GÜNTHER BAUER

Invited Paper

The time-triggered architecture (TTA) provides a computing infrastructure for the design and implementation of dependable distributed embedded systems. A large real-time application is decomposed into nearly autonomous clusters and nodes, and a fault-tolerant global time base of known precision is generated at every node. In the TTA, this global time is used to precisely specify the interfaces among the nodes, to simplify the communication and agreement protocols, to perform prompt error detection, and to guarantee the timeliness of real-time applications. The TTA supports a two-phased design methodology, architecture design, and component design. During the architecture design phase, the interactions among the distributed components and the interfaces of the components are fully specified in the value domain and in the temporal domain. In the succeeding component implementation phase, the components are built, taking these interface specifications as constraints. This two-phased design methodology is a prerequisite for the composability of applications implemented in the TTA and for the reuse of prevalidated components within the TTA. This paper presents the architecture model of the TTA, explains the design rationale, discusses the time-triggered communication protocols TTP/C and TTP/A, and illustrates how transparent fault tolerance can be implemented in the TTA.

***Keywords*—Distributed systems, embedded systems, real-time systems, safety-critical systems, time-triggered architecture (TTA), TTP/C.**

I. INTRODUCTION

Computer architectures establish a blueprint and a framework for the design of a class of computing systems that share a common set of characteristics. The time-triggered architecture (TTA) generates such a framework for the domain of large distributed embedded real-time systems in high-dependability environments. It sets up the computing infrastructure for the implementation of applications and provides mechanisms and guidelines to partition a large application into nearly autonomous subsystems along small and well-defined interfaces in order to control the complexity of the evolving artifact [1]. Architecture design is thus interface design. By defining an architectural style that is observed at all component interfaces, the architecture avoids property mismatches at the interfaces and eliminates the need for unproductive "glue" code.

Characteristic for the TTA is the treatment of (physical) real time as a first-order quantity. The TTA decomposes a large embedded application into clusters and nodes and provides a fault-tolerant global time base of known precision at every node. The TTA takes advantage of the availability of this global time to precisely specify the interfaces among the nodes, to simplify the communication and agreement protocols, to perform prompt error detection, and to guarantee the timeliness of real-time applications.

Research work in the field of distributed dependable real-time computer architectures for safety-critical applications started more than 30 years ago with the design of the STAR computer [2] and the Software Implemented Fault Tolerance [3] and Fault Tolerant Multiprocessor [4] projects. These projects were carefully evaluated and gave rise to new designs about ten years later: Fault-Tolerant Parallel Processors [5], Multicomputer Architecture for Fault Tolerance [6], and the architectural concepts of the Airbus flight control system [7]. In 1992 the first paper on SAFEbus [8], the architecture that was later deployed in the Boeing 777 aircraft for flight control, became available. In excellent publications by Lala [9], Avizienis [10], Rechtin [11] and Laprie [12], the fundamental concepts and architectural principles for the design of dependable systems are clarified at about that time. For example, Lala states that field experience with approximate voting was not at all satisfying. At about the same time, a heated debate started concerning the cost efficiency of design diversity for the tolerance of design faults [13]–[15]. The important ARINC 178B standard [16], published in 1992, that deals with software development for safety-critical avionics systems contains no clear statement about the use of software design

Manuscript received December 20, 2001; revised August 31, 2002. This work was supported in part by the European Information Society Technologies projects NEXT TTA, Fault Injection for Time-Triggered Architectures, Systems Engineering for Time-Triggered Architectures, and Dependable Systems of Systems; in part by the Time-Triggered Sensor Bus project of the government of Austria; and in part by the Defense Advanced Research Projects Agency projects Model-Based Integration of Embedded Software and Networked Embedded Software Technology.

The authors are with the Vienna University of Technology, A-1040 Vienna, Austria (e-mail:hk@vmars.tuwien.ac.at; gue@vmars.tuwien.ac.at).

Digital Object Identifier 10.1109/JPROC.2002.805821

diversity. This issue has not been resolved until today. In Europe, DELTA 4 [17], a research project funded by the European Strategic Programme for Research in Information Technology (ESPIRIT), investigated fundamental issues in the design of distributed dependable architectures at the beginning of the 1990s and uncovered a number of fundamental concepts concerning state recovery in distributed systems. Although the research community at that time was in agreement that a conscientious architectural design phase that establishes the architectural style is of utmost importance for the development of large dependable distributed real-time systems, industrial praxis took a different view. The General Accounting Office's report [18] about the experiences with the air-traffic control project, presumably the largest distributed real-time system project of its time, paints a vivid picture of the practice of system development in that period.

Amid all these research activities, the work on the TTA started in 1979 at the Technical University of Berlin with the Maintainable Architecture for Real-Time Systems (MARS) project. A first report on the MARS project [19] appeared in 1982 and was later published at the IEEE's 15th International Symposium on Fault-Tolerant Computing in 1985 [20]. After 1982, different versions of the MARS architecture have been implemented at the Vienna University of Technology [21], [22], and it became clear that a hardware-supported fault-tolerant clock synchronization is a fundamental building block of a TTA. At about that time, the important concept of temporal accuracy of real-time information was introduced by Kopetz and Kim [23], [24]. The TTP/C protocol, which includes a clock synchronization service and a membership service, was first published in 1993 [25]. A prototype version of the TTA, including a new clock synchronization chip [26] was built in the context of the European Predictably Dependable Computing Systems project. This new prototype implementation has been subject to extensive fault injection experiments [27], [28]. From these experiments, it became evident that an independent guardian must be implemented in order to avoid "babbling idiot" failures in a distributed safety-critical system based on shared communication channels. In 1995, a research cooperation with DaimlerChrysler resulted in an industrial "proof of concept" by demonstrating a time-triggered protocol–equipped "brake-by-wire" car by 1997 [29]. In 1998, the first TTP/C communication controller chip, developed with the support of the European ESPRIT project TTA, was finished. Also in 1998, a high-tech spinoff company of the Vienna University of Technology was founded with the mission to further develop and market the time-triggered (TT) technology [30]. In 1999, Alcatel investigated the TTA and decided to use it in safety-critical train control applications. In 2000, Honeywell selected the TTA for flight control, and Audi decided to use the TTA in future "drive-by-wire" applications. In the last few years, the TT technology received increasing attention for the design of safety-critical real-time applications. A number of new time-triggered protocols, in addition to SAFEbus and TTP, have recently been published: TTCAN [31], FlexRay [32], and Spider [33]. An excellent recent survey by Rushby [34], [35] contains a comparison of some of these communication architectures.

This paper on the TTA is organized as follows. Section II deals with the architectural model of the TTA, presents the model of a sparse time base, elaborates on the important concept of temporal accuracy of real-time information, and discusses the fundamental differences between the event-triggered and TT view of reality. Section III presents the principles that guided the design of the TTA: the provision of a consistent distributed computing base, the unification of interfaces and the temporal firewall concept, composability in the domains of value and time, scalability, and openness to the integration of legacy systems and the information infrastructure, and the transparent implementation of fault tolerance in order to control the application software complexity in fault-tolerant real-time systems. Section IV deals with the communication infrastructure of the TTA: the TTP/C protocol, the TTP/A protocol, and the implementation of event channels on top of the basic TT communication service. Section V is devoted to the issue of transparent implementation of fault tolerance. Finally, Section III presents the two-phase design methodology of the TTA and discusses the architecture from the point of view of validation. The paper finishes with a conclusion in Section VII.

II. Architecture Model

The computational model that guides the design of the TTA is the TT model of computation [36].

A. Model of Time

The model of time of the TTA is based on Newtonian physics. Real time progresses along a dense timeline, consisting of an infinite set of instants, from the past to the future. A duration (or interval) is a section of the timeline, delimited by two instants. A happening that occurs at an instant (i.e., a cut of the timeline) is called an event. An observation of the state of the world is thus an event. The time stamp of an event is established by assigning the state of the node-local global time to the event immediately after the event occurrence. A fault-tolerant internal clock synchronization algorithm establishes the global time in the TTA. Owing to the impossibility of synchronizing clocks perfectly and the denseness property of real time, there is always the possibility of the following sequence of events: clock in node j ticks, event e occurs, clock in node k ticks. In such a situation, the single event e is time-stamped by the two clocks j and k with a difference of one tick. In a distributed system, the finite precision of the global time base and the digitalization of time make it—in general—impossible to consistently order events on the basis of their global time stamps. The TTA solves this problem by the introduction of a sparse time base [37, p. 55]. In the sparse-time model, the continuum of time is partitioned into an infinite sequence of alternating durations of activity and silence, as shown in Fig. 1. The duration of the activity interval, i.e., a granule of the global time, must be larger than the precision of the clock synchronization.

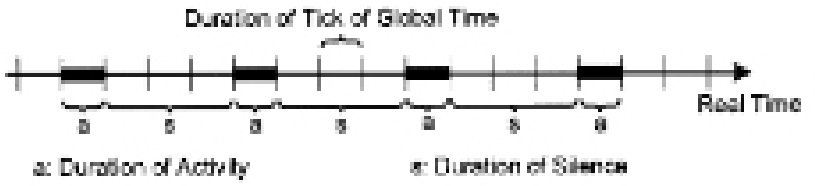

Fig. 1 Sparse time base.

From the point of view of temporal ordering, all events that occur within an interval of activity are considered to happen at the same time. Events that happen in the distributed system at different nodes at the same global clock tick are thus considered simultaneous. Events that happen during different durations of activity and that are separated by the required interval of silence can be consistently temporally ordered on the basis of their global time stamps. The architecture must make sure that significant events, such as the sending of a message, occur only during an interval of activity. The time stamps of events that are outside the control of the distributed computer system (and therefore happen on a dense timeline) must be assigned to an agreed duration of activity by an agreement protocol.

In the TTA, there exists a uniform external representation of time that is modeled according to the global positioning system (GPS) time representation. The time stamp of an instant is represented in an eight-byte integer, i.e., two words of a 32-bit architecture. The three lower bytes contain the binary fractions of the second, giving a granularity of about 60 ns. This is the accuracy that can be achieved with a precise GPS receiver. The five upper bytes count the full seconds. The external TTA epoch assigns the value 2^{38} to the start of the GPS epoch, i.e., 00:00:00 Coordinated Universal Time on January 6, 1980. This offset has been chosen in order that also instants before January 6, 1980 can be represented by positive integers in the TTA. Thus, events that occurred between 8710 years before January 1980 and 26 131 years after January 1980 can be time-stamped with an accuracy of 60 ns. There are different internal time representations in the TTA that match the time format to the capabilities of the hardware (8-, 16-, or 32-bit architectures) and the requirements of the application. Since not all time stamps are based on a global time with a precision of 60 ns, an attribute field is introduced in the external representation indicating the precision of a time stamp [38].

B. Time and State

In abstract system theory, the notion of state is introduced in order to separate the past from the future [39, p. 45]: "The state enables the determination of a future output solely on the basis of the future input and the state the system is in. In other words, the state enables a "decoupling" of the past from the present and future. The state embodies all past history of a system. Knowing the state "supplants" knowledge of the past. Apparently, for this role to be meaningful, the notion of past and future must be relevant for the system considered."

Taking this view, it follows that the notions of state and time are inseparable. If an event that updates the state cannot be said to coincide with a well-defined tick of a global clock on a sparse time base, then the notion of a systemwide state becomes diffuse. It is not known whether the state of the system at a given clock tick includes this event or not. The sparse time base of the TTA, explained previously, makes it possible to define a systemwide notion of time, which is a prerequisite for an indisputable borderline between the past and the future, and thus the definition of a systemwide distributed state. The "interval of silence" on the sparse time base forms a systemwide consistent dividing line between the past and the future and the interval when the state of the distributed system is defined. Such a consistent view of time and state is very important if fault tolerance is implemented by replication, where faults are masked by voting on replicated copies of the state residing in different fault containment regions. If there is no global sparse time base available, one often recourses to a model of an abstract time that is based on the order of messages sent and received across the interfaces of a node. If the relationship between the physical time and the abstract time remains unspecified, then this model is imprecise whenever this relationship is relevant. For example, it may be difficult in such a model to determine the precise state of a system at an instant of physical time at which voting on replicated copies of the distributed state must be performed.

C. Real-Time Entities and Real-Time Images

In the TT model, a distributed real-time computer system is modeled by a set of nodes that are interconnected by a real-time communication system as shown in Fig. 2. All nodes have access to the global time. A node consists of a communication controller (CC) and a host computer. The common boundary between the CC and the host computer within a node is called the communication network interface (CNI) (the thick black line in Fig. 2), the most important interface of the TTA.

The dynamics of a real-time application are modeled by a set of relevant state variables, the real-time entities (RT entities) that change their state as time progresses. Examples of RT entities are the flow of a liquid in a pipe, the setpoint of a control loop, or the intended position of a control valve. An RT entity has static attributes that do not change during the lifetime of the RT entity, and has dynamic attributes that change with time. Examples of static attributes are the name, the type, the value domain, and the maximum rate of change. The value set at a particular instant is the most important dynamic attribute. Another example of a dynamic attribute is the rate of change at a chosen instant.

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

$$\text{Observation} = (\text{Name}, \text{Value}, t_{\text{obs}})$$

consisting of the name of the RT entity, the instant when the observation was made (t_{obs}), and the observed value of the RT entity. A continuous RT entity can be observed at any instant while a discrete RT entity can only be observed when the state of this RT is not changing.

A real-time image (RT image) is a temporally accurate picture of an RT entity at instant t_i if the duration between the