Chapter 14
The Time-Triggered Architecture

Overview This final chapter puts a closing bracket around the contents of the book. It is shown by a concrete example *that it is possible* to integrate the different concepts that have been explained in the previous 13 chapters of this book into a *coherent framework*. This coherent framework, the *time-triggered architecture (TTA)*, is the result of more than 25 years of research at the Technische Universität Wien where numerous master and PhD students have contributed their part to the investigations. We must also gratefully mention the many inputs from colleagues from all over the world, particularly form the IFIP Working Group 10.4, that provided critical feedback and constructive suggestions. At first, the research was driven by curiosity to get a deep understanding of the notions of *real-time*, *simultaneity*, and *determinism*. In the later phases, the active participation by industry brought in the technical and economic constraints of the real world of industry and helped to adapt the concepts. What now has the appearance of a consistent whole is the result of many iterations and an intense interaction between theoretical insights and practical necessities.

The chapter starts with a short description of the TTA, showing some examples of the industrial uptake and the impact of the time-triggered technology. The following section portrays the architectural style of the TTA. The architectural style explains the key principles that drive the design of an architecture. In the TTA, these principles relate to *complexity management*, a *recursive component concept*, *coherent communication* by a single mechanism, and concern for *dependability and robustness*. The architectural style is concretized in the architectural services that form the contents of Sect. 14.3. Space does not permit to describe these services in detail. The reader is referred to other documents, starting with the book on GENESYS [Obm09] to get more in-depth information. The final section of this closing chapter presents the results of a recent research project that puts the time-triggered architecture on a system-on-chip, where a time-triggered network on chip connects the IP-cores, the components of the TTA. We hope that in the future some of these innovative ideas will be taken up by industry to provide a cost-effective execution environment that supports the design of large understandable and dependable real-time systems.

H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 325
Real-Time Systems Series, DOI 10.1007/978-1-4419-8237-7_14,

14.1 History of the TTA

This short view back on the origins and the history of the time-triggered architecture illustrates the close interdependence among basic research, technology research, and industrial innovation. The TTA experience has shown that an industrial deployment of a radical innovation takes a long time – much more than 10 years. This is a reminder to funding agencies that a short-range project view will not produce radical innovations, but only marginal improvements of existing technology.

14.1.1 The MARS Project

Around 1980, the industrial development of real-time systems proceeded in an ad hoc manner without a strong conceptual foundation. Systems were built according to the intuitive feeling of skilled practitioners. During an extensive commissioning phase, experienced programmers finely tuned the diverse parameters of real-time systems such as task priorities and communication timeouts. Even minor changes and additions to the software were problematic, since they required an expensive readjustment and retest of many system parameters. Since system design was not based on solid theoretical foundations, there was always a risk that the system will fail to meet the performance requirements in infrequently occurring *rare event scenarios*. As discussed in Chap. 1, the predictable performance of a real-time system in a rare-event scenario is of paramount importance in many applications (see also the example in Sect. 1.2.1 on a *power blackout*).

In 1979, the MARS (MAintainable Real-Time System) project started at the Technical University of Berlin with the objective to develop a strong conceptual basis, constructive methods and a new architectural framework for the systematic design and maintenance of distributed real-time systems [Kop85]. During the evaluation of the first MARS prototype around 1982, it became clear that more fundamental research, both at the conceptual and experimental level, was needed to gain a better understanding of the following topics:

- Development of a proper model of time and construction of a fault-tolerant global time base in a distributed real-time system. Such a distributed real-time base, without reliance on a single central clock, is at the core of dependable real-time systems.
- The notion of state, determinism and simultaneity and its role in the development of fault-tolerant systems.
- Real-time communication protocols that provide timeliness and error-detection coverage with minimal jitter.
- The effectiveness of mechanisms that establish the fail-silence of architectural units, such that cost-effective structures for fault masking can be built.

In 1983, the MARS project moved from the TU Berlin to the Vienna University of Technology, where the Austrian Ministry of Science and Technology and the European Commission generously funded the continuation of the research. In the following years the focus of the research shifted to investigate fundamental problems in the field of fault-tolerant clock synchronization and the design of real-time protocols. A prototype VLSI chip, the CSU (clock synchronization unit), was developed to support the fault-tolerant clock synchronization in distributed systems [Kop87]. The CSU was used in a second academic prototype of MARS that was subjected to extensive fault-injections experiments [Kar95,Arl03]. This academic prototype received some attention from the research and funding communities after publishing a video of a control application with this prototype, *The Rolling Ball on MARS* – the video can be downloaded from the web [Mar91].

14.1.2 The Industrial TTA Prototype

In the following years, the success of the academic prototype and the strong industrial interest in building real-time systems constructively led to a number of technology-oriented research projects, funded mainly by the European Commission. In these projects, the first industrial-strength prototype of the time-triggered architecture, using the TTP protocol, was developed with strong participation of the automotive industry. This industrial prototype was used in a car to implement a fault-tolerant brake-by-wire system. The precise interface specifications of the components, enabled by the availability of a global time base, were decisive for reducing the planned commissioning time of the prototype brake-by-wire system by an order of magnitude. The success of this prototype gave industrial credibility to the concepts of the time-triggered architecture. In 1998, TU-Vienna launched a spinoff company, TTTech (Time-Triggered TECHnology) to further develop and market the time-triggered technology. In the meantime, TTTech has been successful to deploy the time-triggered technology in a number of reputable industrial projects, among them the Airbus A 380, the Boeing 787 (the *Dreamliner*), the AUDI A 8 premium car, and the NASA Orion program [McC09].

14.1.3 The GENESYS Project

Recognizing the strategic importance of embedded computing, the European Commission formed, together with industry, academia, and national governments, the European technology platform ARTEMIS (Advanced Research and Technology for EMbedded Intelligence and Systems) in 2004. It is a goal of ARTEMIS to develop a cross-domain embedded system architecture, supported by design methods and tools, to significantly improve the functionality, dependability, and cost-effectiveness of embedded systems. In a first phase an expert working group consisting of

industrial and academic partners captured the requirements and constraints of such a cross-domain architecture [Art06]. The following GENESYS (GENeric Embedded SYStems) project, submitted by a consortium of 20 industrial and academic partners coming from the different embedded system domains, was funded by the Framework Program of the European Commission to develop a blueprint for such an architecture that should consider the captured requirements and should be applicable in the industrial domain as well as in the multimedia domain. The architecture blueprint, developed during the GENESYS project, has been strongly influenced by the concepts and experience with the time-triggered architecture. The blueprint is published in a book freely available on the web [Obm09]. At the time of writing, there are ongoing ARTEMIS projects (INDEXYS and ACROSS) that implement the GENESYS architecture.

14.2 Architectural Style

The *architectural style* describes the principles and structuring rules that characterize an architecture (see Sect. 4.5.1). A *principle* is an accepted statement about some fundamental insight in a domain of discourse. Principles establish the basis of the formulation of operational rules, i.e., the *services* of an architecture.

14.2.1 Complexity Management

As outlined in Chap. 2, the management of the ever-increasing *cognitive complexity* – the antonym of *simplicity* – of embedded systems is a subject of steadily increasing concern. The architectural style of the TTA is shaped to a significant degree by this quest to control the growth of complexity of large embedded systems and to facilitate the building of understandable systems.

In Sect. 2.5, seven design principles that lead to understandable systems have been introduced. Each one of these seven principles is part of the architectural style of the TTA. Of distinct importance for the TTA is principle *seven*, the *principle of consistent time*. Embedded computer systems must interact with the physical environment that is ruled by the progression of physical time. The progression of physical time is thus a first-order citizen and not an add-on of the cyber-model that is the basis of the computer control of the physical environment. The availability of a fault-tolerant sparse global time base in every node of a large embedded system is at the foundation of the TTA. This global time base helps to simplify a design. In the TTA, this global time base is used:

- To establish a *consistent temporal order* of all relevant events in cyber space and to solve the problem of simultaneity in a distributed computer system. The consistent temporal order is a prerequisite for introducing the notion of a consistent state of a distributed system. A consistent notion of state is needed when a new (repaired) component must be reintegrated into a running system.

- To build systems with *deterministic behavior* that are easy to understand, avoid the occurrence of *Heisenbugs*, and support the straightforward implementation of fault-masking by redundancy.
- To monitor the *temporal accuracy of real-time* data and to ensure that a control action at the interface to the physical environment is based on data that is temporally accurate.
- To synchronize multimedia data streams that originate from different sources.
- To *perform state estimation* in order to extend the temporal accuracy of real-time data to the *instant of use* in case the dynamics of the physical process outpace the capabilities of the computer system.
- To precisely specify the temporal properties of interfaces such that a component-based design-style can be followed. The *reuse of components* is critically dependent on a precise interface specification that must include the temporal behavior.
- To establish *conflict-free time-controlled communication channels* for the transport of time-triggered (TT) messages. TT-messages have a short delay and minimal jitter and thus help to reduce the dead-time in distributed phase-aligned control loops. This is of particular importance in smart grid automation.
- To *detect the loss of a message* within one time granule. A short error- detection latency is fundamental for any action taken to increase the availability of a system.
- To avoid the *replay of messages* and to strengthen the services of security protocols.

Considering the above listed services that can best be accomplished by using a fault-tolerant sparse global time, it is our opinion that the availability of a consistent global time real-time base is an element of the solution space and not of the problem space in the design of a real-time system.

14.2.2 Component Orientation

The notion of a component introduced in Chap. 4 is the *primitive structure element* of the time-triggered architecture. A TTA component is a self-contained hardware/software unit that interacts with its environment solely by the exchange of messages. A component is a unit of design and a unit of fault-containment (FCU). The message-based linking interfaces (LIF) of a component are precisely specified in the domains of time and value and are agnostic about the concrete implementation technology of a component. A component can be used on the basis of its interface specification without knowing the internals of the component's implementation. The time-triggered integration framework of the TTA ensures that real-time transactions that span over more than one component have defined end-to-end temporal properties. In a time-critical RT-transaction, the computations of the components and the message transport by the time-triggered communication system can be *phase-aligned*.

It is a principle of the TTA that a component can be expanded to a new cluster without changing the specification of the LIF of the original component that becomes the *external LIF* of the new cluster. After such an expansion, the external LIF is provided by a gateway component that supports on the other side a second LIF to the new (expanded) cluster (the *cluster LIF*).

Example: Viewed from the in-car cluster, the gateway component of the in-car cluster in the right lower corner of Fig. 4.1 has two interfaces, the *cluster LIF* at its upper side and the *external LIF* at its lower side. At the external LIF only those information items from the *in-car cluster* are made available to other cars that are relevant for the safe coordination of the traffic.

We thus have a *recursive component concept* in the TTA. Depending on the point of view taken, a set of components can be viewed as a cluster (focus on the *cluster LIF* of the gateway component) or as a single component (focus on the *external LIF* of the gateway component). This recursive component concept makes it possible to build well-structured systems of arbitrary size within the TTA.

Components can be integrated to form hierarchical structures or network structures. In a hierarchical structure, a designated gateway component links different levels of the hierarchy. We now take the view from the lower level of the hierarchy. The designated gateway component has two LIFs, one to the lower level of the hierarchy (the *cluster LIF*) and another one to the higher level of the hierarchy (the *external LIF*). Since the different hierarchical levels can obey different architectural styles, the designated gateway component must resolve the ensuing property mismatches. The *external LIF* of the gateway component, viewed from the lower level, is a *local unspecified interface* of the cluster. Vice versa, the *cluster LIF* of the gateway component becomes a *local unspecified interface* when viewed from above.

Example: In a TTMPSoC (Time-Triggered MultiProcessor on a Chip) the primitive element is an IP-core (Intellectual Property core). An IP-core implements a self-contained component. The set of IP-cores on the chip forms a cluster of components that are connected by a time-triggered network on chip (TTNoC). A gateway IP-core has (in addition to the *cluster LIF*) an *external LIF* that connects the chip to the outside world. From the point of view of the outside world, the external LIF of the gateway IP-core can be considered an interface to the *chip component,* i.e., the whole TTMPSoC. At a higher level, a cluster of *chip components* forms a *device component,* and *so on.*

If a cluster is linked with *two* (or more) gateway components to *two* (or more) other clusters, flat network structures can be created. In such a flat network structure, a gateway component can filter the information that is available within a particular cluster and will present only those information items at its external LIF that are relevant for the services of the respective connected cluster.

14.2.3 Coherent Communication

The only communication mechanism of the TTA is the unidirectional BMTS (basic message transport service) that follows, whenever possible, the fate-sharing model

of the Internet. The fate-sharing model was formulated by David Clark, an architect of the DARPA net, as follows*: The fate-sharing model suggests that it is acceptable to lose the state information associated with an entity if, at the same time, the entity itself is lost* [Cla88, p. 3]. The fate-sharing principle demands that all state information that is associated with a message transfer must be stored in the endpoints of the communication. Even in the design of the time-triggered network-on-chip (TTNoC), described in Sect. 14.4, the fate-sharing principle was considered. The fate sharing principle can be applied in a safety-critical configuration if the end systems are guaranteed to be fail-silent. Otherwise, information about the intended temporal behavior of the end-systems has to be also stored in an independent FCU, a *guardian* or the *network*, to contain the faulty behavior of a babbling node (see also Sect. 6.1.2).

As long as different subsystems of the TTA are connected by time-triggered communication systems, such as the TTNoC, TTP, or TTEthernet, the BMTS is characterized by a *constant* transport delay and a minimal jitter of one granule of the global time. If a message is transported via an event-triggered protocol (e.g., in the Internet), no such temporal guarantee can be given.

This single coherent communication mechanism makes it possible to move a component (which can be an IP-core of a system-on-chip) to another physical location without changing the basic communication mechanism among the components.

14.2.4 Dependability

The architectural style of the TTA is strongly influenced by dependability concerns such that secure, robust, and maintainable embedded systems can be built with reasonable effort. The following principles of the TTA support the construction of dependable systems:

- A component is a fault-containment unit (FCU). Temporal failures of components are contained at the component boundaries by the time-triggered communication system.
- The BMTS is *multicast*. An independent diagnostic component can observe the behavior at the component LIF without probe effects.
- The BMTS and the basic system component services avoid NDDC (non-deterministic design constructs) so it is possible to build systems with *deterministic* behavior in the TTA.
- Fault tolerant units (FTU) of replicated deterministic components can be formed to mask an arbitrary fault in any one of the components.
- Components publish their ground state periodically, in order that a diagnostic component can monitor the plausibility of the ground state, can detect anomalies, and initiate a reset and restart of a component in case of a ground-state corruption caused by a transient fault. This principle helps to improve the system robustness.

- The available global time can be used to strengthen the security protocols.
- The recursive component concept improves the *evolution*, since new functions can be implemented by expanding a single component into a new cluster of components, without changing the properties of the *external LIF* of the new cluster.
- Every TTA system that is connected to the Internet should support a *secure download service* so a new software version of any component can be downloaded automatically.

14.2.5 Time Aware Architecture

The time-triggered architecture provides the framework for the design of a dependable monolithic real-time system. If we link TTA systems, designed by different organizations, to form a *system of systems (SoS)*, then the design rules must be relaxed in order to match the realities of widely distributed systems that interact by event-messages across the Internet (see also Sect. 4.7.3). If each one of the autonomous constituent systems of an SoS has access to a synchronized global time of known precision, then we call such an SoS a *time-aware architecture* (TAA). With today's technology it is relatively easy to implement a TAA: at every site, a GPS receiver captures the worldwide time-signal and synchronizes the clocks at the different sites of the TAA. Although a TAA is not time-triggered, it can still accrue many of the advantages of a global time listed in Sect. 14.2.1, provided all messages contain the time-stamp of the sender in their data field.

14.3 Services of the TTA

14.3.1 Component-Based Services

The TTA provides the platform and platform services for the integration of components. We distinguish between three categories of services: (1) the *core system services* that are needed in any instantiation of the architecture, (2) the *optional system services* that provide, in addition to the core services, functionality that is needed in many, but not all, instantiations of the architecture, and (3) the *application specific services* that are specific to a given application or application domain. The core system services and the optional system services are provided either by standalone *system components* or are part of the generic middleware (GM) of the component software (see also Sect. 9.1.4).

We consider it a major achievement that in the TTA there is no *need for a large monolithic operating system*. It is difficult to estimate the execution time of a real-time computation if dynamic operating system mechanisms and a hypervisor stand

between the *application code* and the *execution of the code* by the hardware. Furthermore, the certification of a large monolithic operating system is challenging. In the TTA, many of the conventional operating system functions can be implemented by self-contained *system components*. Whenever a system component is mature and stable, its implementation can be moved from a software-on-a-CPU implementation to an ASIC, thus realizing a very significant reduction of energy requirements (see Fig. 6.3) and silicon real estate.

Every software-on-a-CPU component has a local lightweight operating system and generic middleware (GM) that implements standardized high-level protocols and interprets the control messages that arrive via the TII interface (see Sect. 4.4).

14.3.2 Core System Services

The core system services of the TTA are *minimal* in the sense that only those services that are absolutely essential to build higher-level services or to maintain the desired properties of the architecture are included in the set of core system services. The core services must be free of NDDCs (non-deterministic design constructs, see Sect. 5.6.3) in order that *deterministic computations* can be implemented. In many cases, the implementation of a powerful dynamic system service is partitioned into a small core system service and a more intricate optional system service, since in a static safety-critical system only the core system services are needed and therefore the subject of certification.

> **Example:** A dynamic message scheduler that must be part of any dynamic resource management is not included in the core system services. However, a much simpler *checker* that checks the properties of a schedule and ascertains that the constraints of a static safety critical schedule have not been violated by the dynamic scheduler is part of the core system services.

The following paragraphs give a high-level overview of the TTA services. A more detailed description of the services can be found in [Obm09].

Basic Configuration Service. This service loads a *job*, i.e., the core image of the software that has been generated by a development system, onto the specified hardware unit and thereby generates a *TTA component*. The core image of the software includes the application software, the GM (generic middleware, see Sect. 9.1.4), and the local operating system. The basic configuration service is also needed to reconfigure the system after a hardware unit has failed permanently. The basic configuration service includes: (1) a secure hardware identification service to uniquely identify the hardware and (2) a basic boot service that accesses the boot access point of the hardware unit via the TII interface. The basic boot service establishes a connection to a development system that holds the *job* (the core image of the component software, including the application software, the GM, and the local operating system of the component) for the identified hardware unit.

Inter-Component Channel Configuration Service. This service configures the cluster-local inter-component communication system by establishing, naming, connecting, and disconnecting the ports and communication channels of the component-LIFs within a cluster. This service observes the *fate sharing principle*.

Basic Execution Control Service. This service is used to control the execution of a component. Execution control is realized by sending an appropriate trusted control message to the TII port of the respective component. It is assumed that in every component there is a local resource manager (LRM) that accepts and executes these messages. The LRM is part of the GM (generic middleware) of a component. This service can be used to reset (by a hardware reset) and restart a component with a *g-state* that is contained in the restart message.

Basic Time Service. This service establishes the global time of specified precision of a component. The global time is provided by the platform. The time-format of the TTA is a binary time format based on the physical second. The basic time service includes a timer interrupt service.

Basic Communication Service. This service enables the application software of a component to send and receive time-triggered, rate-controlled, and event-triggered messages. This service is implemented by the communication system of the platform, supported by the GM of the component.

14.3.3 Optional System Services

An optional system service encapsulates a well-defined supportive functionality into a self-contained *system component* that interacts with the GM of the application components by the exchange of messages. Alternatively, an optional service can be implemented directly in the GM of an application component. The optional services are useful across many application domains and may be needed on many different occasions. They simplify the system development process by providing ready building blocks, i.e., *new concepts* that can be reused on the basis of their specification. The optional system services form an open set that can be extended if need arises.

Diagnostic Services. These services include the periodic g-state externalization of a component, a membership service that informs all components of a cluster consistently about the health state of the other components of a cluster, and a g-state monitoring, analysis, and recovery service such that a failed component can be reintegrated into a running cluster.

External Memory Management Service. In many applications, the local scratchpad memory of a component must be augmented by an external memory that can hold large amounts of data. The external memory management service, implemented in a standalone memory component, manages the storage of and access to long-lived data and implements needed security and integrity mechanisms.

Security Services. A basic security service – the provision of a tamper-resistant unique identification of any component is part of the core system services. Using this core system service, a dedicated optional security component can be provided that encrypts and decrypts all messages that leave or arrive in the defined security domain. For example, such a security domain can be a system-on-chip. Depending on the application requirements, symmetric or asymmetric ciphers can be supported. This service is used to build a secure boot service. A secure boot service should be part of any device that is connected to the Internet.

Resource Management Services. At every level of the TTA a resource management service is provided. At the lowest level, the component level, a *local resource manager,* the LRM that is part of the GM, controls the resources local to a component. The LRM of the component can be parameterized by a *cluster resource manager,* the CRM, that takes a holistic view of the functions of the whole cluster and may request a component to shut down (power gating) in order to save energy. The CRM, implemented in a self-contained system component, may contain a dynamic scheduler that integrates the scheduling of the real-time task with voltage and frequency scaling in order to optimize the energy consumption while still meeting all deadlines.

Gateway Services. Gateway components are needed to interface a cluster to its external environment, i.e., other clusters, the physical process, the human operator, or the Internet. A gateway component must resolve the property mismatches that exist between the inner world of a cluster and the external world. In particular a gateway component has to provide one or more of the following services [Obm09, p. 76]:

- *Control of the physical interface* (mechanical and electrical) to the physical plant.
- *Protocol translation.* The protocol at the external interface has to conform to the given LIF standards of the environment, while the *cluster LIF* determines the protocol at the inner interface.
- *Address mapping.* While the address space inside the cluster is constrained, the name space of the environment, e.g., the Internet, is wide open. The gateway component has to map internal addresses to outer addresses.
- *Name translation.* The name-spaces within a cluster and the outside world are in many cases *incoherent*. The gateway component must resolve this incoherency.
- *External clock synchronization*. The outer interface of a gateway component may have access to an external time reference (e.g., GPS time) that must be brought into the cluster.
- *Firewall erection*. The gateway component must protect the cluster from malicious outside intruders.
- *Wireless connection*. A gateway component may provide a wireless connection to the outside world and perform the connection management.

14.4 The Time-Triggered MPSoC

The shift of the computer industry from single processor systems to *multiprocessor systems on chips* (MPSoC) is driven primarily by power and energy concerns, as discussed in Sect. 6.3.2. This shift presents a tremendous opportunity for the embedded systems industry, since a hardware architecture that consists of many self-contained IP-cores that can operate concurrently without any non-functional dependencies and that are connected by an appropriate network-on-chip (NoC) provides a much better match to the needs of many embedded applications than a powerful single sequential processor.

Viewed from the point of the TTA, an IP-core is considered to be a component and the whole MPSoC implements a cluster. Within the GENESYS project, funded by the European Commission, we developed a first academic prototype of a TTMPSoC (time-triggered multiprocessor system on chip) to understand the constraints and opportunities of this new technology. The project was completed in 2009 and a prototype TTMPSoC that supports an automotive application was implemented on an FPGA [Obm09].

Figure 14.1 depicts the overall architecture of the prototype TTMPSoC with eight IP-cores. There are two types of structural units in Fig. 14.1, the *trusted structural units* (denoted by the bold boxes) and the *non-trusted structural units* (normal boxes). The *trusted structural units*, i.e., the *Trusted Network Authority* (TNA), the eight *trusted interface subsystems* (TISS), and the *time-triggered network on chip* (TTNoC) form a *trusted subsystem* that is vital for the operation of the chip and is assumed to be free of design faults. In high-reliability applications, the trusted subsystems can be hardened (e.g., by using error correcting codes) to tolerate transient hardware faults. An arbitrary failure (caused by a transient hardware fault or a software error, such as a *Heisenbug*) of a *non-trusted structural unit* will not impact the operation of other, independent units of the chip.

At the center of Fig. 14.1 is the time-triggered network on chip (TTNoC) that connects the *IP-cores* via the TISSes. Only the TNA has the authority to write a new network configuration that determines the time-triggered sending slots for each

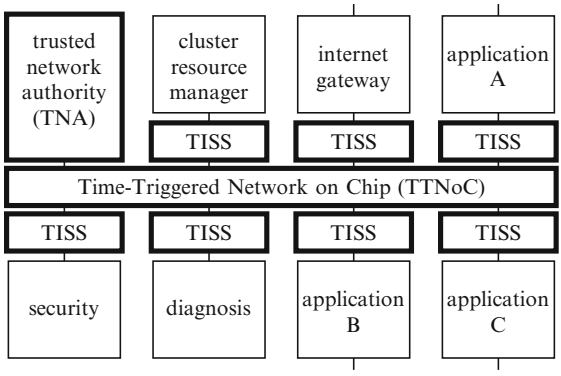

Fig. 14.1 Architecture of the prototype TTMPSoC

non-trusted IP-core into a TISS. If a non-trusted component violates its temporal specification, the TISS will detect and contain the failure. The cluster resource manager, a non-trusted system component, can calculate dynamically a new communication schedule at the request of an application component (in Fig. 14.1 component A, B, or C). The Cluster Resource Manager sends the new schedule to the TNA. The TNA verifies the schedule and checks whether any safety constraint is violated before writing the new schedule into the respective TISSes. Only the TNA has the authority to control the execution of a component via the TII interface of an application component. Otherwise, a single non-trusted component with a software error could send erroneous control messages, requesting the components to terminate, to all components and ruin the whole chip.

The architecture of Fig. 14.1 assures that any temporal fault of a non-trusted component is contained by the TISS and will not affect the communication among the correct components. It is thus possible to build TMR structures of three IP-cores to mask an arbitrary fault in any one of the components. In safety-critical applications, TMR structures can be built where each IP-core of a triad resides on a different chip in order to avoid spatial proximity faults and to tolerate a complete chip failure. The diagnostic component monitors the behavior of the components and checks the g-states of the components for plausibility. It is the task of the security component to decode and encode all messages that enter or leave the TTMPSoC. A more detailed description of the TTMPSoC, including the prototype application, can be found in [Obm09]. The implementation of the TTNoC is described in [Pau08].

Points to Remember

- The *architectural style* describes the principles and structuring rules that characterize an architecture. In the TTA, these principles relate to complexity management, a recursive component concept, coherent communication by a single mechanism, and concern for dependability and robustness.
- The availability of a fault-tolerant sparse global time base in every node of a large embedded system is part of the foundation of the TTA. This global time base helps to simplify a design.
- The time-triggered integration framework of the TTA ensures that real-time transactions that span over more than one component have defined end-to-end temporal properties.
- It is a principle of the TTA that a component can be expanded to a new cluster without changing the specification of the LIF of the original component that becomes the *external LIF* of the new cluster. After such an expansion, the external LIF is provided by a gateway component that supports on the other side a second LIF to the new expanded cluster (the *cluster LIF*).
- Depending on the point of view taken, a set of components can be viewed as a cluster (focus on the *cluster LIF* of the gateway component) or as a single component (focus on the *external LIF* of the gateway component).

- Components can be integrated to form hierarchical structures or network structures.
- It is an important design principle of the TTA that there is only a *single communication mechanism* among the components, no matter whether the components are close together, as in a system-on-chip, or far away at another place in the world, connected by the Internet.
- The core system services of the TTA are *minimal* in the sense that only those services that are absolutely indispensable to build higher-level services or to maintain the desired properties of the architecture are included in the set of core system services.
- The core services must be free of NDDCs (non-deterministic design constructs), such that *deterministic computations* can be implemented.
- A *TTA job* is the core image of the component software that includes the application software, the GM, and the local operating system of the component.
- At every level of the TTA, a resource management service is provided. At the lowest level, the component level, a *local resource manager,* the LRM that is part of the GM, controls the resources local to a component.
- The LRM of the component can be parameterized by a *cluster resource manager* that takes a holistic view of the functions of the whole cluster.

Bibliographic Notes

A first description of the MARS project can be found in [Kop85]. The VLSI chip for clock synchronization is presented in [Kop87]. This chip was used in the academic prototype of the MARS architecture [Kop89]. The time-triggered protocol TTP for the communication among the nodes of MARS was published in [Kop93]. The first overview of the time-triggered architecture appeared in the PDCS book [Kop95] which is presented in a more mature form in [Kop03a]. A formal analysis of time-triggerd algorithms has been carried out by John Rushby [Rus99]. The rational of the TTEthernet protocol is described in [Kop08]. The TTMPSoC is presented in [Pau08]. An overview of the GENESYS is given in the book [Obm09].

Review Questions and Problems

14.1 List the problems where the availability of a global time base contributes to finding a solution in a distributed real-time system!
14.2 What is the *fate-sharing principle*?
14.3 List the design principles of the TTA that help to build dependable systems!
14.4 Why is *deterministic behavior* a desired property of a real-time transaction?
14.5 Why should components publish their ground state periodically?
14.6 How can the availability of a global time strengthen a security protocol?

14.7 Why is a large monolithic real-time operating system problematic in real-time systems?
14.8 How are conventional operating system functions implemented in the TTA?
14.9 What are the differences between *core system services* and *optional system services* in the TTA?
14.10 What are the functions of the generic middleware?
14.11 Why is it necessary to split some system functions?
14.12 What is included in the concept of a TTA job?
14.13 List the core system services of the TTA!
14.14 List some of the optional system services of the TTA!
14.15 What are the functions of a gateway component?