Time-Triggered Architecture: A Consistent Computing Platform

THE TIME-TRIGGERED ARCHITECTURE PROVIDES A CONSISTENT COMPUTING PLATFORM FOR LARGE COMPLEX APPLICATIONS AND SAFETY-RELEVANT SYSTEMS. TTA IS ALREADY IN USE IN RAILWAY SYSTEMS, AND THE AEROSPACE AND AUTOMOTIVE INDUSTRIES ARE BEGINNING TO ADOPT IT, WITH FIRST PRODUCTS MAKING THEIR WAY INTO THE FIELD.

Reinhard Maier
TTTech Computertechnik

Günther Bauer
Vienna University of Technology

Georg Stöger
Stefan Poledna
TTTech Computertechnik

The number of electronic control units in automobiles has increased rapidly in recent years. Analysts estimate that electronics will drive or enable 90 percent of all automotive innovations in the next decade. New functions such as lane departure warning, automatic emergency braking, traffic sign recognition, and heading control are pushing overall electronics complexity. Future by-wire systems such as brake by wire, steer by wire, and electronic vehicle dynamics control demand high dependability levels. Consequently, the electronics architecture is crucial to the development road map of car manufacturers and their suppliers.

The state of the art in automotive electronics is loosely coupled electronic control units. Each ECU has its own internal system structure, with little commonality in this structure among different ECUs. Today's high-end cars come equipped with up to 70 ECUs, each acting autonomously with only limited dependence on the other ECUs in the network. Information exchange among ECUs occurs through event-triggered communication systems with relatively low communication speeds, typically a controller area network (CAN) with up to 500 kilobits per second. Most ECUs can continue to operate even if communication fails.

Next-generation functions require a paradigm shift from loosely coupled ECUs to an integrated electronics architecture—a distributed safety-critical real-time computing platform. Four major forces drive this shift:

- *Cost.* An integrated architecture allows moving, distributing, and adding functions across the network. This lets designers reduce the number of ECUs in the car, substantially decreasing overall system cost.
- *Stability and reliability.* Design faults (hardware and software) are a major source of instability and low reliability in complex systems. An integrated architecture supporting state consistency can abstract from and hide many difficulties of distributed real-time systems, thus leading to simpler application software and reduced development time.
- *Safety.* New functions such as by-wire systems and collision avoidance are safety relevant. An integrated electronics archi-

0272-1732/02/$17.00 © 2002 IEEE

tecture must support partitioning, which lets safety-relevant and non-safety-relevant functions coexist without needing to validate all functions to the highest safety level.
- *Sensors.* Subsystems can share sensors. Besides allowing detection of individual sensor failures, this feature reduces the total number of sensors, thus decreasing the cost of the distributed control system.

In automotive applications, an electronics architecture must satisfy demanding requirements in safety, availability, and fault tolerance. This integrated architecture must support ease of system integration; component upgrades and migration of existing systems; and a large variety of car models, platforms, and equipment options. Finally, the architecture must meet the automotive market's stringent cost constraints. A central property of such an architecture, state consistency can substantially help satisfy the challenging requirements of an integrated automotive-electronics platform.

The time-triggered architecture (TTA), along with its fault-tolerant time-triggered communication protocol (TTP/C), meets these requirements and provides state consistency. The Vienna University of Technology, TTTech and its industry partners, and several leading research institutions have joined forces over the past 20 years to develop TTA. While meeting automotive requirements and cost targets, the available TTA components also satisfy the highest level of aviation safety certification (DO-178B Level A and DO-254 Level A). TTA is already used in railway systems and special vehicles, and several applications in the aerospace and automotive domain are underway—for example, a safety-critical application in the new Airbus A380. TTA is an open, cross-industry standard supported by TTA-Group (http://www.ttagroup.org).

Time-Triggered Architecture

A computer architecture establishes a blueprint and a framework for designing a class of computing systems that share common characteristics. TTA generates such a framework for the domain of distributed, embedded, real-time systems in high-dependability environments. The architecture sets up the computing infrastructure for implementing applications. It provides mechanisms and guidelines to partition a large application into nearly autonomous subsystems along small and well-defined interfaces. Doing so controls the evolving artifact's complexity. Architecture design is thus interface design. By defining an architectural style that all component interfaces observe, the architecture avoids property mismatches at the interfaces and eliminates the need for unproductive glue code.

A central characteristic of TTA is its treatment of time as a first-order quantity. TTA decomposes a large embedded application into clusters and nodes and, at every node, provides a fault-tolerant global time base of known precision. TTA takes advantage of this global time to precisely specify the interfaces among the nodes, simplify communication, establish state consistency, promptly perform error detection, and support the timeliness of real-time applications.

TTA structure

TTA's basic building block is a node. This self-contained unit includes a processor with memory, an input-output subsystem, time-triggered communication controller, operating system, and the relevant application software. These components can be on a single silicon die.

Two replicated communication channels connect the nodes to build a cluster. The physical interconnection structure and communication controllers of all the cluster's nodes form the communication subsystem. In TTA, this subsystem is autonomous and executes a previously specified, periodic time division multiple access (TDMA) schedule. The communication subsystem reads a data frame containing state information from the communication network interface (CNI) at the sending node at the previously specified fetch instant. The subsystem then delivers this data frame to the CNIs of all of the cluster's receiving nodes at the known delivery instant, overwriting the previous version of this frame. The periodic fetch and delivery instances are contained in the message-descriptor list (MEDL), a scheduling table within each communication controller that is consistently known to all communication controllers in a cluster.

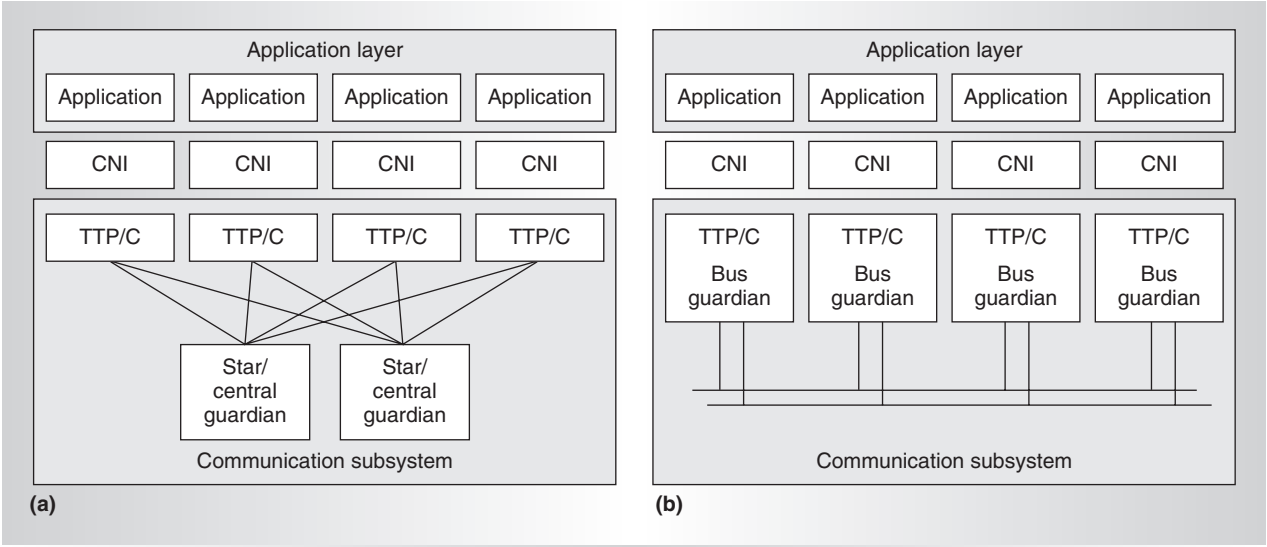

Figure 1. TTA star with central guardian (a), and TTA bus with local-bus guardians (b).

The current TTA implementation is based on two different network topologies, as Figure 1 shows: a star topology (TTA star) and a bus topology (TTA bus). TTA star implements smart central guardians that all of a cluster's nodes share. These smart guardians isolate arbitrary node failures to support safety-critical applications. From a cost point of view, TTA star is attractive because it requires only one bus guardian per channel. Nodes of TTA bus implementation have a node-local bus guardian that prevents babbling-idiot faults. (A babbling idiot is a node that sends more often than specified or even all the time and thus monopolizes the communication medium.)

Design principles

Several principles drove the initial design of TTA.

Consistent distributed-computing platform. TTA's main purpose is to provide a consistent distributed-computing base to all correct nodes so that designers can build reliable, distributed applications with manageable effort. If a node cannot assume that every other correct node works on the same state, then distributed algorithm design becomes cumbersome because designers must solve the intricate agreement problem at the application level.[1] Therefore, TTP/C provides consistency support at the communication subsystem level.

Unification of interfaces: Temporal firewalls. A good architecture must rest on a few orthogonal concepts that are reusable in many different situations, thus reducing the effort required to understand complex systems. Driven by its time-triggered schedule, TTP/C autonomously carries data frames from the sender's CNI to the receiver's CNI. The sender can deposit the information in its local CNI memory, according to the information push paradigm, whereas the receiver must pull the information out of its local CNI memory. Because no control signals cross the CNI, TTA interface eliminates control-error propagation by design. (The communication system derives control signals for the fetch and delivery instances from the progress of global time and its local schedule table MEDL.) We call such an interface a *temporal firewall.*

Composability. In a distributed real-time system, the nodes interact via the communication system to provide the emerging real-time services. These services depend on the timely provisioning of real-time information at the node interfaces. We define an architecture to be *composable* in the temporal domain if it guarantees the following four principles:[2]

- independent development of nodes,
- stability of services before integration of a new function,
- constructive integration of nodes to gen-

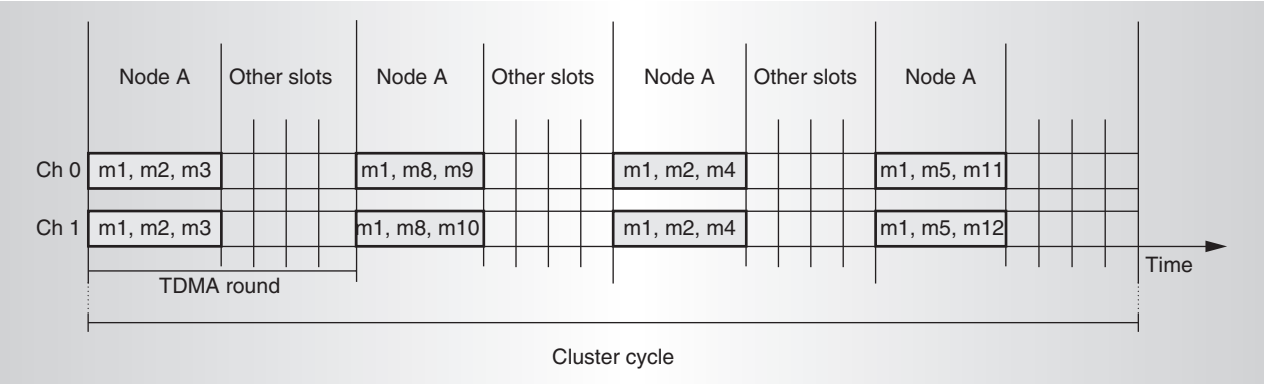

Figure 2. Cluster cycle in TTP/C communication, including TDMA rounds, frames (around shaded areas), slots, and messages.

erate emerging services, and
- replica determinism.

Scalability. TTA targets the design of complex, distributed, real-time applications. You can construct a complex system supporting many different functions most efficiently if the effort required to understand a particular system function is independent of system size. Horizontal layering (abstraction) and vertical layering (partitioning) are the means to combat the complexity of large systems. In TTA, CNIs encapsulate a function and provide it only with environmental properties that are relevant to that function's operation.

TTP/C protocol

You can find the detailed TTP/C specification at http://www.ttagroup.org. This fault-tolerant time-triggered protocol provides the following services:

- A TDMA medium access strategy on replicated communication channels enables autonomous fault-tolerant message transport with known delay and minimal jitter among the CNIs of a cluster's nodes.
- Fault-tolerant clock synchronization establishes the global time base without relying on a central time server.
- A membership service informs every correct node about the consistency of data transmission. This distributed acknowledgment service promptly informs the application of an error in the communication system. If state consistency is lost, this service will quickly notify the application.
- Clique avoidance detects faults outside the fault hypothesis that are intolerable at the protocol level.

TTP/C organizes communication into TDMA rounds, as Figure 2 shows. TTP/C divides a TDMA round into time slots and assigns each node in the communication system to one slot, its sending slot. Each node must send frames in every round. The frame size allocated to a node can vary between 2 bytes and 240 bytes. Each frame usually carries several messages. The cluster cycle is a recurring sequence of TDMA rounds; in different rounds, different messages can be transmitted in the frames, but the complete set of state messages repeats in each cluster cycle. A 24-bit cyclic redundancy check (CRC) protects the data from error. Each node stores the schedule in MEDL within the communication controller.

TTP/C's clock synchronization exploits the common knowledge of the send schedule. Every node measures the difference between a correct message's expected and observed arrival time to learn about the difference between the sender's and receiver's clock. A fault-tolerant average algorithm uses this information to periodically calculate a correction term for the local clock to synchronize it with all the cluster's other clocks.

In the case of a failure, the membership service employs a distributed agreement algorithm to determine whether the sender's outgoing link or the receiver's incoming link has failed. Several research institutions have formally verified the TTP/C's core algorithms to prove their correctness. Moreover, several

research institutions have done multimillion dollars worth of fault injections to validate protocol functionality in the presence of faults.

Fault hypothesis and fault handling

In a properly configured TTP/C-based system, any single component can fail in an arbitrary failure mode because the components are in different fault containment regions. Under this assumption, the likelihood of two concurrent, independent, component failures is remote enough for us to consider it a rare event, which an appropriate never-give-up (NGU) strategy can handle. However, a prompt error detection mechanism is necessary to ensure that two consecutive, single faults do not become concurrent.

For hardware faults, TTP/C isolates and tolerates single-node faults. A bus guardian ensures that a faulty node cannot prevent correct nodes from exchanging data. This guardian guarantees that a node can only send once in a TDMA round, thus eliminating the problem of babbling idiots that monopolize the communication medium.

Finally, TTP/C implements an NGU strategy for multiple fault scenarios: If a node detects faults that the fault hypothesis does not cover, it informs the application. The application can shut down in fail-safe environments. In environments that must remain operational despite failures, applications can restart with an agreed-upon consistent state among all the distributed system's nodes.

Fault tolerance

At the communication subsystem level, the TTP/C ensures fault tolerance via the bus guardian and the NGU strategy. These fault-handling mechanisms ensure that faulty nodes cannot prevent correct nodes from communicating. At the application level, fault tolerance requires a fault tolerance layer and an appropriate application design. Application designers can implement fault tolerance by replicating a software subsystem on two fail-silent nodes. Triple modular redundancy (TMR) voting can ensure tolerance of a single arbitrary node failure.

Both dual and triple redundancy tolerate single-component faults with the respective failure semantics; both mechanisms can thus handle transient and permanent hardware faults. To reestablish tolerance to single-component faults despite a permanent hardware fault, the TTP/C supports the implementation of transparent, hot, stand-by spares via shadow nodes. (A shadow node is a redundant node in the system that receives the same data from the bus as its primary node. It also executes the same application tasks, but does not send any data. If the primary node fails, the shadow node takes over the sending slot of the primary node and starts sending data.)

Providing support consistency

State consistency can considerably ease the design and development of complex distributed systems. In single-node systems, you can take consistency for granted: Data written to memory becomes available to all software subsystems simultaneously, and if the node is correct, all subsystems read the same value. In distributed systems, however, you cannot assume this level of consistency. First, you must consider message transmission delays that affect the current state; it's not guaranteed that a message will simultaneously arrive at all receivers. Second, individual nodes might fail, or messages might get lost.

Abstract system theory introduces the notion of *state* to separate the past from the future:

> 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.[3]

From this view, it follows that the notions of state and time are inseparable. If an event that updates the state does not coincide with a well-defined tick of a global clock on a sparse time base, then the notion of a systemwide state becomes ambiguous: An application cannot determine whether the system state at a given clock tick includes this event. TTA's sparse time base makes it possible to define a systemwide notion of time,[2] a prerequisite for

an indisputable border between the past and future. In turn, these features make it possible to define a consistent systemwide state.

To ease the development of complex applications for a distributed system, it's important for a communication platform to support state consistency in a way that is independent of the application. Application-independent support of state consistency not only relieves the CPU from executing consistency protocols but, more important, makes it possible to verify a complex consistency algorithm's correctness once for all applications. These requirements make it necessary to implement data consistency at the CNI's communication controller side.

The many possible failure scenarios—whether at the node, in the communication subsystem, or because of differences in message arrival timing or sequence—would be difficult to account for in the application logic. This also argues for making state consistency a basic service of the communication subsystem. Assuming a node sends data to a set of receiver nodes and that the fault hypothesis holds, the system is *consistent* if

- all correct nodes agree on the same data;
- all nodes agree on the sent data, provided the sender is correct; and
- all correct subsystems deliver the received value at the same point in time.

TTA implements communication consistency support in hardware and at the protocol level. One of the main philosophies behind TTP/C design is that the protocol should transmit data consistently to all correct nodes. Moreover, in case of a failure, the communication system should independently decide which node is erroneous, provided the stated fault hypothesis holds.

The membership, acknowledgment, and clique avoidance mechanisms let us achieve these properties.

This combination of algorithms, along with the common time base established by the clock synchronization, provides communication consistency. TTP/C guarantees that all correct nodes simultaneously receive the same information. TTA thus provides the application software with a very powerful programming model to efficiently handle complex distributed software systems.

Membership mechanism

Each node of a TTP/C-based cluster maintains a membership list containing all the nodes considered correct. Each receiver updates this information locally according to successful (or unsuccessful) data transmissions; the information thus reflects the local view of the receiving node about all other nodes. With every transmission, each receiver sees and checks the sender's membership, which is explicitly included or hidden in the CRC calculation of the sender's transmission.

Because of the TDMA round's strict round-robin scheme, each node sees and checks all other nodes' membership lists during just one TDMA round. Every sender with a different membership list is assumed to be incorrect. This feature ensures a consistent view of all nodes that mutually accept (communicate successfully with) one another in the membership.

Acknowledgment mechanism

After a data transmission, node A seeks acknowledgment from the other nodes to determine whether the receiver (at the communication level) accepted the transmission. TTP/C accomplishes this by checking the membership list of the first (and possibly second) successive sender: If these nodes show node A in their membership, they state that A's transmission was successfully received. Otherwise, the nodes inform A that the transmission was unsuccessful. Because of the time-triggered principle, the node retransmits the state message in the next cycle.

Clique avoidance mechanism

This mechanism detects multiple component faults and inconsistencies, and supports the NGU strategy. Before every send operation of a node, the algorithm checks whether the node is a member of the majority clique. If the node is in a minority clique, a rare fault scenario outside the fault hypothesis has occurred, leading to an inconsistency. TTP/C conveys the presence of this condition to application software, which can decide whether to initiate fail-stop or fail-operational activities.

Exchanging event data

TTA also provides mechanisms for the exchange of event data. The need for event data typically arises from on-demand diagno-

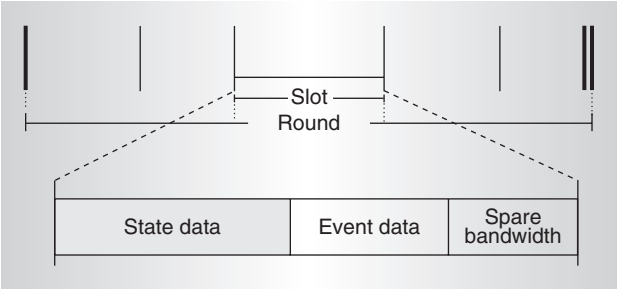

Figure 3. Real-time and on-demand frame partitioning, showing sections for state data, event data, and spare bandwidth.

sis, parameter calibration, and debugging.

TTA event transport mechanism is based on the allocation of dedicated bandwidth for event data. A TTP/C frame can carry up to 240 bytes, and part of the frame can handle event messages. Event-triggered messages piggyback on TTP/C frames. Figure 3 shows the slot partitioned into sections for state data, event data, and spare bandwidth for future extensions.

Event data atop TTP/C

An important property is that this scheme arbitrates the bandwidth for on-demand transmission only among the event messages at each node, not among the event messages for all nodes in the system. This node-local arbitration ensures full temporal composability, which is not possible with global-bandwidth contention. Furthermore, all fault isolation capabilities of TTA also apply to these dynamically arbitrated channels. Thus, a faulty node cannot possibly occupy the channels of some correct node. Moreover, all consistency mechanisms described earlier are also valid for the event messages. This enables TTA to extend full consistency from time- to event-triggered messages.

Although node-local bandwidth arbitration is less efficient than systemwide bandwidth arbitration with respect to overall bandwidth use, extensive studies by major automobile manufacturers show that our event transport mechanism completely satisfies these manufacturers' requirements. For example, the implementation for a standard diagnostic protocol can consume as little as 0.8 percent of a 10-Mbit/s TTP/C system's net bandwidth.

To support the migration of other legacy software systems, an EU funded research project is working on an implementation that incorporates

- the transmission-control protocol/Internet protocol (TCP/IP) and
- the common object request broker architecture Internet InterOrb protocol (Corba IIOP)

atop the basic TTP/C communication service.

Migration strategy for CAN-based software

The generic approach for event data transmission makes it possible to layer the event-triggered CAN protocol atop TTP/C. This provides a clean migration strategy for the large body of CAN-based software without requiring substantial software rewrites. One application emulated a high-speed CAN controller running at 500 Kbps with 5 percent net bandwidth atop a 10-Mbps TTP/C system. Based on the global, sparse time base, we can establish a meaningful order based on the carrier sense, multiple access with collision avoidance (CSMA/CA) protocol, but the timing differs. We have evaluated a hardware-based CAN emulation, which is register compatible with widely used CAN modules.

Other time-triggered automotive protocols

Here we compare TTP/C with two other time-triggered protocols for automotive electronic systems: time-triggered CAN[4] and FlexRay.[5] In comparing these protocols, we emphasize consistency support and the approach to combining event- and time-triggered communication. We begin with a brief overview of TTCAN and FlexRay based on public information (http://www.ttagroup.org).

TTCAN

Time-triggered CAN is an extension of the well established event-triggered CAN protocol. Communication involves periodic transmissions of a reference message by a time master. This reference message introduces a systemwide reference time. Alternatively, an external event can trigger the reference message. Based on this reference, TTCAN defines several so-called exclusive windows, as Figure 4 shows. These windows are equivalent to the time slots in a TDMA system. TTCAN assigns

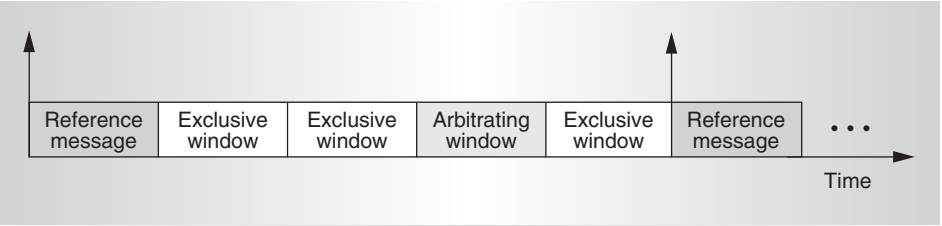

Figure 4. TTCAN time windows.

each exclusive window to a specific node, which can send a data frame. In addition, the protocol defines arbitrating windows. Within these windows, all network nodes can transmit frames according to the event-triggered CSMA/CA access scheme used by CAN.

Because CAN preserves the original CSMA/CA channel access protocol for event messages, it is inherently limited to a 1 Mbit/s data transmission rate. Because CAN provides only one communication channel and a master-slave algorithm handles clock synchronization, TTCAN cannot tolerate arbitrary, single-component failures.

An interesting feature of CAN is its acknowledgment and retransmission mechanism, which uses the CSMA/CA principle. The sender transmits an acknowledgment bit at the end of the frame, which is set to the logical *true* condition, and a recessive state on the channel represents this condition. If any of the receiver nodes has experienced a reception error, that node can immediately change the state to a dominant channel level, indicating the logical *false* condition. This mechanism can ensure consistent message delivery for most cases. (Under specific fault scenarios, the acknowledgment bit itself might be received inconsistently.)

FlexRay

FlexRay is a combination of two different protocols: a time-triggered TDMA scheme and a minislotting protocol for event-triggered transmission. FlexRay also provides a mode that makes it compatible with Byteflight—a data bus protocol for automotive applications (http://www.byteflight.com/homepage.htm). FlexRay supports different modes of operation for clock synchronization:

- a distributed fault-tolerant midpoint algorithm for the TDMA mode, and
- a master-slave algorithm for the Byteflight mode.

The master-slave algorithm in turn can establish a reference based on either time or external events. The distributed midpoint algorithm serves as a reference for a set of TDMA slots with equal length. Following this set is a dynamic segment for events. During the dynamic segment, the slot counter for the minislotting protocol is incremented. If a node wants to send any event messages, it must wait until the slot counter has reached the unique ID assigned to the message. Event messages can have different lengths. The advantage of minislotting over CSMA/CA is that minislotting has no restriction in communication speed.

Similar to the TTP/C, FlexRay supports two redundant communication channels for fault tolerance, as Figure 5 (next page) shows. Because of the lack of published fault hypothesis information, we do not know which types and frequencies of faults the protocol intends to tolerate or how FlexRay tolerates all types of single-component failures.[5]

A consortium is developing FlexRay, and it has not yet published a specification.

Comparison with TTP/C

TTP/C and TTCAN offer different levels of support for communication data consistency. FlexRay, on the other hand, does not appear to support message acknowledgment, consistent message delivery, or detection of inconsistencies in the communication system.[5]

Table 1 (next page) compares TTP/C, TTCAN, and FlexRay. Kopetz[6] and Rushby[7] give more detailed comparisons of time-triggered automotive and aerospace protocols, including FlexRay, SAFEbus, Spider, and TTP/C.

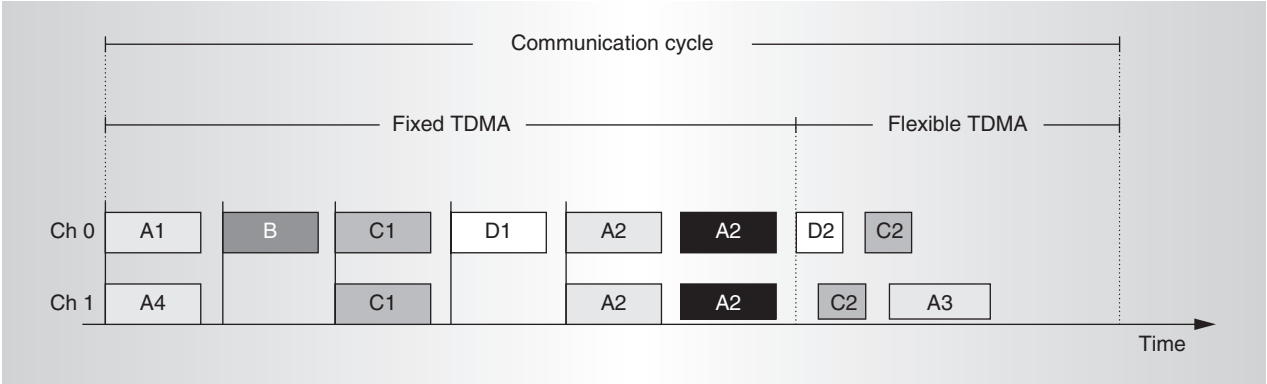

Figure 5. FlexRay communication cycle with fixed and flexible parts.

Table 1. Comparison of the TTP/C, TTCAN, and FlexRay communication protocols.

Feature	TTP/C	TTCAN	FlexRay
Transmission speed	25 Mbps Unlimited, higher speeds possible	1 Mbps Limited by CSMA/CA access	10 Mbps Unlimited, higher speeds possible
Maximum frame length for application data (bytes)	240 (variable for each node)	8	246 (all slots must be the same size for TDMA parts; individual message lengths are possible for the minislotting part) 12 (Byteflight mode)
Consistency support	Acknowledgment, membership, and clique avoidance	Acknowledgment	None apparent[5]
Event-handling strategy	Event channel atop TDMA (CAN emulation) Event messages are temporal composable Lower bandwidth efficiency	TDMA plus arbitrating windows for CSMA/CA Event messages are not temporal composable Higher bandwidth efficiency	TDMA plus dynamic segment for minislotting Event messages are not temporal composable Higher bandwidth efficiency

TTP is on its way to becoming a COTS component. The cost advantages of such components make them preferable to specialized designs. However, the industries (PC, consumer, telecommunications, entertainment, and so on) driving and defining COTS components have no requirements for ultra-high dependability; consequently, such components are not yet available.

The TTA-Group (http://www.ttagroup.org), an open industry consortium of major players in the aerospace and automotive industries, aims to establish TTP as the first COTS component for ultra-high dependability. The aerospace industry has extensive experience in building ultra-highly dependable and safety-critical systems, and the automotive industry has the necessary volume to establish a COTS component. TTP is already accepted in the aerospace industry for the highest safety-critical level; beginning in 2004, it will be included in planes such as the Airbus A380. TTP also meets automotive requirements, and its cost is equivalent to that of a dual-channel CAN controller. Automotive applications based on TTP will become available in 2005. MICRO

Acknowledgment

This work is supported in part by the European Information Society Technologies (IST) project, Next TTA, under project number IST-2001-32111.

References

1. L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Programming Languages and Systems*, vol. 4, no. 3, July 1982, pp. 382-401.
2. H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic, Norwell, Mass., 1997.
3. M.D. Mesarovic and Y. Takahara, *Abstract Systems Theory*, Springer-Verlag, New York, 1987.
4. *Road vehicles—Controller Area Network (CAN)—Part 4: Time Triggered Communication*, Working Draft ISO 11898-4, Int'l Standards Organization, Geneva; http://www.iso.ch/iso/en/stdsdevelopment/techprog/workprog/TechnicalProgrammeSCDetailPage.TechnicalProgrammeSCDetail?COMMID=808.
5. *Proc. FlexRay Int'l Workshop*, FlexRay Consortium, 2002; http://www.flexray.com/htm/infws402.htm.
6. H. Kopetz, "A Comparison of TTP/C and FlexRay," Inst. for Computer Eng., Vienna, 2001; http://www.vmars.tuwien.ac.at/php/pserver/extern/listpaper.php?viewmode=paper&year=2001.
7. J. Rushby, *A Comparison of Bus Architectures for Safety-Critical Embedded Systems*, tech. report, Computer Science Laboratory, SRI Int'l, Menlo Park, Calif., 2001; http://www.csl.sri.com/papers/buscompare/.

Reinhard Maier is a protocol expert for real-time systems at TTTech Computertechnik, Vienna. His research interests include real-time buses and fault-tolerant real-time systems. Maier has an MSc in computer science from the Vienna University of Technology and an MBA from Vienna University.

Günther Bauer is a technical coordinator for the IST research project in the Real-Time Systems Group at the Vienna University of Technology. His research interests include the design and implementation of a central guardian for TTA. Bauer has a diploma in electrical engineering and a PhD in computer science from the Vienna University of Technology.

Georg Stöger is director of training and support at TTTech Computertechnik. His research interests include training and support for projects in the area of time-triggered architectures, especially by-wire architectures. Stöger has a diplomingenieur (master's) of computer science certificate from the Vienna University of Technology.

Stefan Poledna is cofounder and chief executive officer of TTTech Computertechnik, and he lectures on fault-tolerant computer systems at the Vienna University of Technology. His research interests are in the area of automotive electronics and fault-tolerant systems. Poledna has an MSc in computer science and a PhD in computer science from the Vienna University of Technology.

Direct questions and comments about this article to Reinhard Maier, TTTech Computertechnik AG, Schoenbrunnerstrasse 7, A-1040 Vienna, Austria; maier@tttech.com.