Introduction to TTP/C and TTP/A

Wilfried Elmenreich
Institut für Technische Informatik
Vienna University of Technology
Vienna, Austria
wil@vmars.tuwien.ac.at

Richard Ipp
TTTech Computertechnik
Schönbrunnerstraße 7
Vienna, Austria
ipp@tttech.com

TTP/C and TTP/A are the real-time protocols of the Time-Triggered Architecture (TTA). Both protocols use a Time-Division Multiple Access (TDMA) scheme to enable collision-free bus allocation.

TTP/C focuses on the interconnection of components in order to form a highly dependable real-time system that is sufficient for critical applications such as X-by-wire in the automotive and avionics domains. TTP/C implements a replicated bus system and a guardian that prevents babbling idiot failures.

TTP/A aims at an easy and economically integration of sensors and actuators into a network. TTP/A can be implemented on low-cost microcontrollers, which suggests each transducer having a TTP/A interface. The interface concept of TTP/A supports a modular design and an easy integration and management of transducers.

1 Introduction

Distributed electronic control systems promise an improvement of cost, safety, weight reduction, and maintainability in several domains. The decreasing cost of electronic microcontrollers has led to a shift from traditional electro-mechanical solutions to distributed electronic control systems in the automotive domain. Avionic systems depend on reliable communication systems with a second focus on weight. The requirements from these domains call for a dependable real-time communication system that interconnects the control nodes.

Complemental to such a communication system will be a transducer bus that provides an interface to the various sensors and actuators of such a system. Besides real-time capabilities such a transducer bus must also support non-real-time access for configuration and maintenance.

It is the objective of this paper to describe two time-triggered protocols that fulfill the above described requirements. The TTP/C protocol provides a highly dependable real-time communication service with a fault-tolerant clock synchronization and membership service. TTP/C is suitable for X-by-wire systems in the automotive and avionics domain.

The time-triggered fieldbus TTP/A is intended for the integration of smart transducers in all types of distributed real-time control systems. Although the first target are automotive applications, TTP/A has been designed to meet the requirements of process control systems as well. TTP/A provides a well-defined smart transducer interface to the sensors and actuators of a network. TTP/A has no fault tolerance mechanisms by default, but supports deterministic real-time communication with low latency and small jitter. TTP/A supports low cost implementations on wide set of available component-off-the-shelf microcontrollers.

The following sections of this paper are organized as follows: Section 2 explains the principles of the Time-Triggered Architecture that are common to both protocols. Section 3 introduces the principles of operation of TTP/C. Section 4 ex-

plains the principles of operation of the TTP/A protocol. Section 5 compares both approaches and depicts an architecture that integrates both protocols. The paper is concluded in Section 6.

2 The Time-Triggered Architecture

The Time-Triggered Architecture (TTA) [1] establishes a framework for the implementation of dependable distributed real-time embedded applications.

The basic building block of the TTA is a node, which comprises a network interface, a time-triggered communication controller, and a host computer. Nodes within a TTA cluster exchange data using the TTP/C communication protocol, which uses a TDMA scheme for bus arbitration. Each node forms a fault containment region. In order to tolerate the loss of a communication channel, two replicated channels connect the nodes to build a cluster.

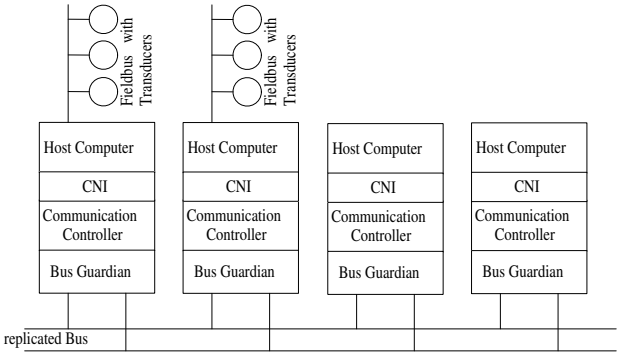

Figure 1: Bus network topology

Currently, the TTA supports two network topologies, the *bus* and the *star* topology. In the bus topology (see Figure 1) each node is equipped with a bus guardian component that controls access of the node's communication controller to the communication medium. The bus guardian is implemented as a separate component so that in case of failure of a component the node is prevented from sending outside its assigned time slot, i. e., preventing a babbling idiot failure of a node.

The star topology implements two star couplers that act as central bus guardians (see Figure 2). The star approach has several advantages over the bus approach:

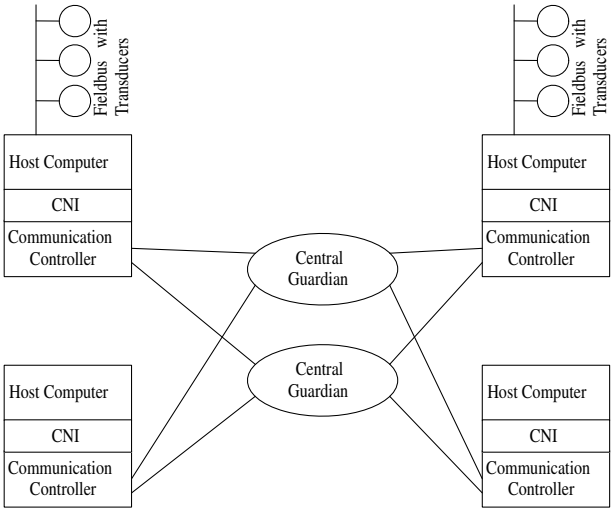

Figure 2: Star network topology

- Since the star coupler is spatially apart from the nodes the star topology is resistant against spatial proximity faults.
- The central guardians isolate arbitrary node failures. Slightly-off-specification faults, which are not caught in the bus topology are handled by performing a signal reshaping of the sender's message.
- Since only one bus guardian per channel is necessary, the star approach is economically attractive [2].

3 The TTP/C Protocol

TTP/C is a fault-tolerant time-triggered protocol that provides:

- An autonomous fault-tolerant message transport at known times and with minimal jitter between the CNIs of the nodes of a cluster by employing a TDMA strategy on replicated communication channels.
- A fault-tolerant clock synchronization that establishes the global time base without relying on a central time server.
- A Membership service to inform every correct node about the consistency of data transmission. This service can be viewed as a

distributed acknowledgment service that informs the application promptly if an error in the communication system has occurred.

- Clique avoidance to detect faults outside the fault hypothesis, which cannot be tolerated at the protocol level.

In TTP/C communication is organized into TDMA rounds as depicted in Figure 3. A TDMA round is divided into slots. Each node in the communication system has its sending slot and must send frames in every round. The frame size allocated to a node can vary from 2 to 240 bytes in length, each frame usually carrying several messages. The cluster cycle is a recurring sequence of TDMA rounds; in different rounds different messages can be transmitted in the frames, but in each cluster cycle the complete set of state messages is repeated. The data is protected by a 24 bit CRC (Cyclic Redundancy Check). The schedule is stored in the message descriptor list (MEDL) within the communication controller.

The clock synchronization is necessary to provide all nodes with an equivalent time concept. In doing so it makes use of the common knowledge of the send schedule. Each node measures the difference between the a priori known expected and the observed arrival time of a correct message to learn about the difference between the senders clock and the receivers clock. A fault-tolerant average algorithm needs this information to periodically calculate a correction term for the local clock so that the clock is kept in synchrony with all other clocks of the cluster. The membership service uses a distributed agreement algorithm to determine whether, in case of a failure, the outgoing link of the sender or the incoming link of the receiver has failed.

The core algorithms of TTP/C have been formally verified to prove their correctness. Real tests with multi-million fault injections and heavy ion radiation experiments and experiments with electromagnetic interferences have been carried out successfully.

3.1 Fault Hypothesis and Fault Handling

Provided that the components of a properly configured TTP/C based system are in different fault containment regions, each can fail in an arbitrary way. Under this assumption the probability of two concurrent independent component failures is small enough to be considered a rare event that can be handled by an appropriate never-give-up (NGU) strategy. However, it should be noted that a very prompt error detection mechanism is needed to ensure that two consecutive single faults are not becoming concurrent.

As for hardware faults, TTP/C is designed to isolate and tolerate single node faults. By introducing a bus guardian it is guaranteed that a faulty node cannot prevent correct nodes from exchanging data. The bus guardian ensures that a node can only send once in a TDMA round, thereby eliminating the problem of babbling idiots that monopolize the communication medium.

Moreover, TTP/C implements a never-give-up strategy (NGU) for multiple fault scenarios: if a node detects faults that are not covered by the fault hypothesis, it notifies the application. The application may now decide either to shutdown in fail-safe environments or to restart in fail-operational environments with an agreed consistent state among all nodes of the distributed system.

3.2 Fault Tolerance

The mechanisms described above ensure fault tolerance at the communication subsystem level in TTP. These mechanisms of the communication subsystem guarantee that faulty nodes cannot prevent correct nodes from communicating and serve as communication platform for the application. At the application level fault tolerance needs to be implemented by a fault tolerance layer and an appropriate application design. Fault tolerance can be realized by replicating a software subsystem on two failsilent nodes. Tolerance of a single arbitrary node failure can be ensured by TMR (Triple Modular Redundancy) voting.

Both mechanisms will tolerate single component faults with the respective failure semantics and are thus fit to handle both transient and permanent hardware faults. To re-establish tolerance to single component faults despite the presence of a permanent hardware fault, TTP/C supports the implementation of transparent hot-stand-by spares.

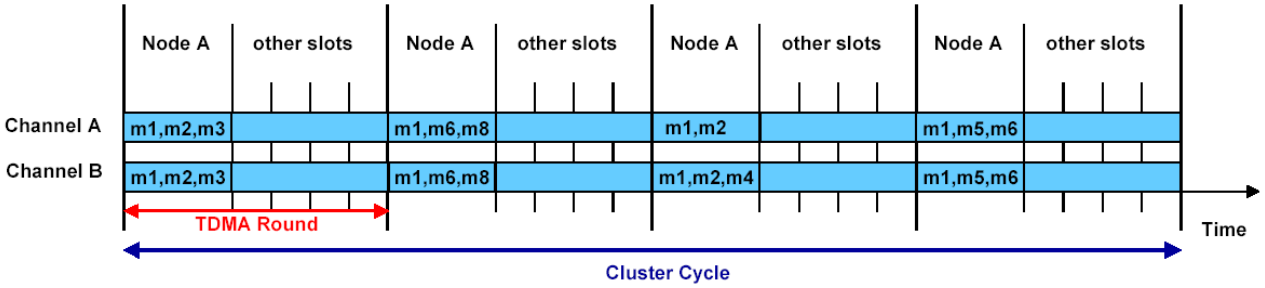

Figure 3: Frames, messages, slots, TDMA round, and cluster cycle in TTP/C communication

Fault tolerance is realized by a redundant unit in the network taking over the function of a defective unit, without being noticed at the function level. This is why data consistency is necessary.

3.3 Support for Consistency

Data consistency can facilitate the design and development of complex distributed systems considerably. In single node systems consistency can be taken for granted because data written to memory is available to all software subsystems at the same time and all subsystems read the same value, provided that the node is correct. In distributed systems it is no longer justified to assume this kind of consistency. There are two reasons for this: Firstly, message transmission delays that have an effect on the current state must taken into consideration; it is not guaranteed that a message arrives at all receivers at the same point in time. Secondly, individual nodes may fail or messages may get lost.

Design and programming in a distributed system become laborious and difficult, especially for complex applications, if state consistency is not supported at the communication platform level independent from the application. Application-independent support of state consistency is not only capable of relieving the application CPU of executing consistency protocols but, more important, of verifying the correctness of complex consistency algorithm for all applications. This makes it necessary to implement data consistency at the communication controller side of the CNI.

The huge number of possible node failures, communication failures, or differences in message arrival timing and sequence would make the logic of the application subsystem large and complex. Therefore state consistency should be supported as a basic service of the communication subsystem that satisfies the following properties. Provided that the fault hypothesis holds, a system is termed consistent if

- All correct nodes agree on the same data,
- All nodes agree on the data sent by a correct sender
- All correct subsystems deliver the received value at the same point in time.

It is assumed that a node sends data to a series of receiving nodes. TTA is designed to support communication consistency in the hardware directly on the protocol level. The mechanisms that support consistency are described in the following.

3.4 Membership and Acknowledgment

A major philosophy in the design of TTP/C is that the protocol should transmit data consistently to all correct nodes of the distributed system and that, in case of a failure, the communication system should decide on its own which node is faulty. These properties are achieved by the membership protocol and an acknowledgment mechanism.

Each node of a TTP/C based cluster maintains a membership list with all nodes that are considered to be correct. This information is updated locally in accordance with successful (or unsuccessful) data transmissions and thus reflects the local view of the receiving node on all other nodes. With each transmission, each receiver sees and checks the senders membership that is included

in the senders transmission or hidden in the CRC calculation.

Acknowledgment: After transmission, node A seeks acknowledgment from the other nodes in order to determine whether the transmission was accepted at the receiver (on the communication level). This is achieved by checking the membership list of the first (and possibly second) successive sender. If these nodes show node A in their membership list, they state that As transmission was successfully received. Otherwise, A is informed that the transmission was unsuccessful. Due to the time-triggered principle, re-transmission of the state message is done in the next cycle.

Membership Consistency Check: Due to the strict round-robin scheme of the TDMA round, each node sees and checks the membership lists of all other nodes in one TDMA round. Each sender with a different membership list is assumed as incorrect. This ensures a consistent view of all nodes, which accept each other in the membership, i.e., they communicate successfully with one another.

Clique Avoidance: In order to detect multiple component faults and inconsistencies and to support the never-give-up strategy, the clique avoidance mechanism is active. Before each send operation of a node the algorithm checks if the node is a member of the majority clique. In case the node is in a minority clique, this means that a very rare fault scenario outside the fault-hypothesis has occurred which led to an inconsistency. This condition is signaled to the application software, which can decide whether to initiate fail-stop or fail-operational activities.

The combination of these algorithms together with the common time base established by the clock synchronization provides communication consistency. This guarantees that all correct nodes receive the same information at the same point in time. TTA thus provides the application software with a very powerful programming model that allows efficient handling complex distributed software systems.

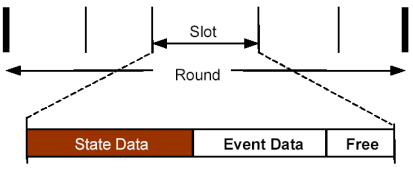

Figure 4: Real-time and on-demand frame partitioning

3.5 Consistency and Events

In addition to the mechanisms for guaranteed, timely, and consistent transmission of real-time data, TTA also provides mechanisms for the exchange of event data. The need for event data usually arises from on-demand diagnosis, parameter calibration, and debugging.

The 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. Part of the frame can be used for event messages. The event-triggered messages are piggy-backed on TTP/C frames.

This partitioning of each slot in state data, event data, and spare bandwidth for future extensions is depicted in Figure 4.

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

Even though the use of a node-local bandwidth is less efficient with respect to the overall bandwidth than the use of a system-wide bandwidth, extensive studies with large car manufacturers have shown that the described event transport mechanism satisfies their requirements completely. A standard diagnostic protocol, for example, can be implemented with as little as 0.8% net bandwidth of a 10 Mbit/s TTP/C system.

In order to support the migration of other software systems, an implementation of the widely used TCP/IP protocol and of the CORBA IIOP protocol on top of the basic TTP/C communication service is in progress.

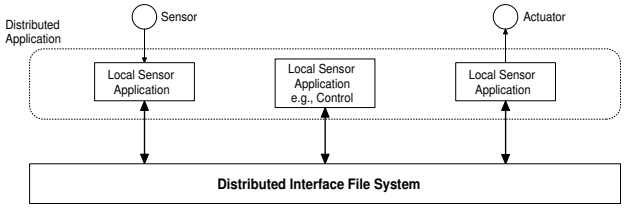

Figure 5: Logical Network Structure

4 The TTP/A Protocol

TTP/A is a low-cost fieldbus protocol for the interconnection of smart transducers. The protocol has been standardized in 2002 by the Object Management Group (OMG) as *smart transducer interface* standard[3]. The standard comprises TTP/A as the time-triggered transport service within a distributed smart transducer subsystem and defines a well-defined interface between the transducers and a CORBA network.

The smart sensor technology offers a number of advantages from the points of view of technology, cost, and complexity management [4]:

- Electrically weak non-linear sensor signals can be conditioned, calibrated and transformed into digital form on a single silicon die without any noise pickup from long external signal transmission lines [5].
- The smart sensor contains a well-specified digital communication interface to a sensor bus, offering "plug-and-play" capability if the sensor contains a reference to its documentation in form of an electronic data sheet as it is proposed in the IEEE 1451.2 Standard [6].
- It is possible to monitor the local operation of the sensing element via the network and thus simplify the diagnosis at the system level.

The internal complexity of the smart-sensor hardware and software and internal failure modes can be hidden from the user by well-designed fully specified smart sensor interfaces that provide just those services that the user is interested in.

4.1 Interface File System

The information transfer between a smart transducer and its client is achieved by sharing information that is contained in an internal interface file system (IFS), which is encapsulated in each smart transducer.

The IFS provides a unique address scheme for transducer data, configuration data, self-describing information and internal state reports of a smart transducer [7]. It establishes a stable intermediate structure that is a solid base for smart transducer services. The implementation of the IFS is economically feasible to assign local intelligence even to low-cost I/O devices like 8-Bit microcontrollers with 4 Kbytes Flash ROM and less than 64 bytes of RAM memory.

The IFS is the source and sink for all communication activities and acts as a temporal firewall [8] that decouples the local transducer application from the communication activities. A time-triggered sensor bus will perform a periodical time-triggered communication to copy data from the IFS to the fieldbus and write received data into the IFS. The ROund Descriptor List (RODL), a predefined communication schedule defines time, origin, and destination of each protocol communication. The instants of updates are specified a priori and known by the communicants. Thus, the IFS acts as a *temporally specified interface* that decouples the local transducer application from the communication task.

Each transducer can contain up to 64 files in its IFS. An IFS file is an index sequential array of up to 256 records. A record has a fixed length of four bytes (32 bits). An IFS record is the smallest addressable unit within a smart transducer system. Every record of an IFS file has a unique hierarchical address (which also serves as the global name of the record) consisting of the concatenation of the cluster name, the logical name, the file name and the record name.

Besides access via the master node, the local applications in the smart transducer nodes are also able to execute a clusterwide application by communicating directly with each other. Figure 5

depicts the logical network view for such a cluster-wide application.

4.2 The Three Interfaces of a Smart Transducer

In order to support complexity management and composability, it is useful to specify distinct interfaces for functional different services [4]. As depicted in Figure 6, a smart transducer node can be accessed via three different interfaces.

Real-Time Service (RS) Interface: This interface provides the timely real-time services to the component during the operation of the system.

Diagnostic and Maintenance (DM) Interface: This interface opens a communication channel to the internals of a component. It is used to set parameters and to retrieve information about the internals of a component, e. g., for the purpose of fault diagnosis. The DM interface is available during system operation without disturbing the real-time service. Usually, the DM interface is not time-critical.

Configuration and Planning (CP) Interface: This interface is necessary to access configuration properties of a node. During the integration phase this interface is used to generate the "glue" between the nearly autonomous components. The CP interface is not time-critical.

4.3 Principles of Operation

TTP/A is a time-triggered protocol used for the communication of one active master with or among smart transducer nodes within a cluster. This cluster is controlled by the master, which establishes a common time base among the nodes. In case of a master failure, a shadow master can take over control. Every node in this cluster has a unique alias, an 8 bit (1 byte) integer, which can be assigned to the node a priori or set via the configuration interface.

The bus allocation is done by a Time-Division Multiple-Access (TDMA) scheme. Communication is organized into rounds consisting of several

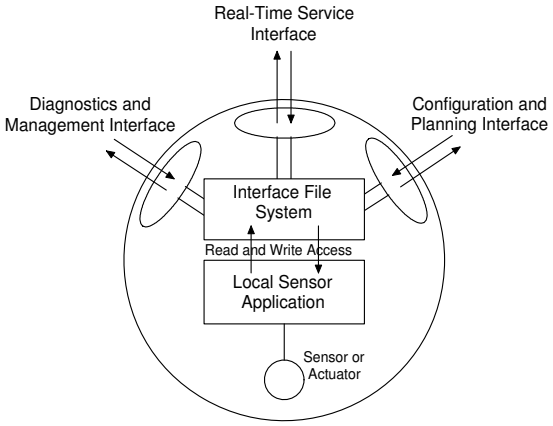

Figure 6: Three Interfaces to a Smart Transducer Node

TDMA slots. A slot is the unit for transmission of one byte of data. Data bytes are transmitted in a standard UART format. Each communication round is started by the master with a so-called fireworks byte. The fireworks byte defines the type of the round and is a reference signal for clock synchronization. The communication pattern for each round is predefined via the RODL, which is distributively stored in all nodes. The protocol supports eight different firework bytes encoded in a message of one byte using a redundant bit codesupporting error detection. One fireworks byte is a regular bit pattern, which is also used by slave nodes with an imprecise on-chip oscillator for startup synchronization (see figure 7).

Generally, there are two types of rounds:

Multipartner round: This round consists of a configuration dependent number of slots and an assigned sender node for each slot. The configuration of a round is defined in a datastructure called "RODL" (ROund Descriptor List). The RODL defines which node transmits in a certain slot, the operation in each

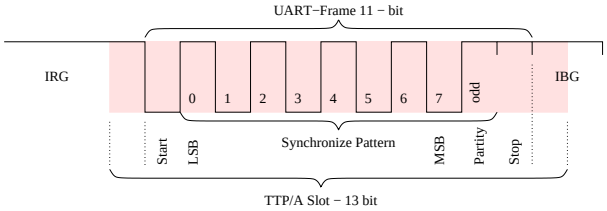

Figure 7: Synchronization Pattern

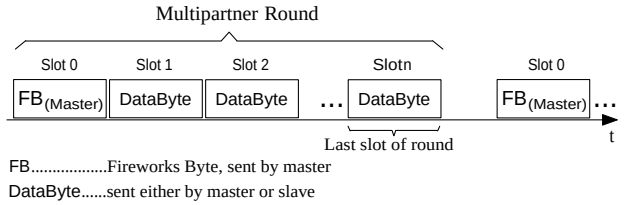

Figure 8: Example for a TTP/A multipartner round

individual slot, and the receiving nodes of a slot. RODLs must be configured in the slave nodes prior to the execution of the corresponding multipartner round. An example for a multipartner round is depicted in Figure 8.

Master/slave round: A master/slave round is a special round with a fixed layout that establishes a connection between the master and a particular slave for accessing data of the node's IFS, e. g. the RODL information. In a master/slave round the master addresses a data record using a hierarchical IFS address and specifies an action like reading of, writing on, or executing that record.

The master/slave rounds establish the DM and the CP interface to the transducer nodes. The RS interface is provided by periodical multipartner rounds. Master/slave rounds are scheduled periodically between multipartner rounds as depicted in Figure 9 in order to enable maintenance and monitoring activities during system operation without a probe effect.

5 Integration of TTP/C and TTP/A

By integrating both protocols into the time-triggered architecture, TTP/A and TTP/C well complement each other. Since both protocols are

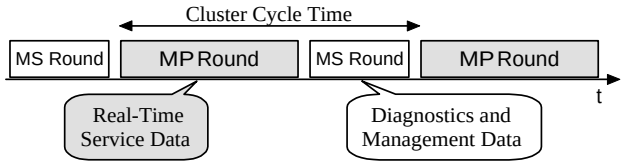

Figure 9: Recommended TTP/A Schedule

obeying a time-triggered schedule, the TTP/A and TTP/C system can be synchronized in order to support synchronous interaction of all nodes in the network.

Figure 10 depicts a fault-tolerant architecture with two TTP/C nodes and two TTP/A networks containing a set of transducers. Each TTP/C node controls vice versa one master node of one TTP/A network and a shadow node to the other TTP/A network. The dotted ovals around the transducers indicate that these transducers are redundantly measuring the same real-time entity. The given example tolerates an arbitrary node failure of any node in the network. Since measurements from different sensors of the same real-time entity are not replica deterministic, it is necessary to run an agreement protocol between the TTP/C application in order to get a consistent system state.

Besides stand-alone implementations of TTP/A [9, 10, 11], there exist also some prototype implementations of the TTP/A master protocol for TTP/C nodes. The most modular solution comes in the form of a PCMCIA card [12] which can be used as I/O card in a TTP/C node (or any other computer system with a PCMCIA interface). Figure 11 depicts the size of the PCMCIA gateway card (without cover). The card hosts a microcontroller and provides interfaces to a TTP/A and CAN bus. The microcontroller can be configured via the PCMCIA interface and runs the TTP/A master protocol.

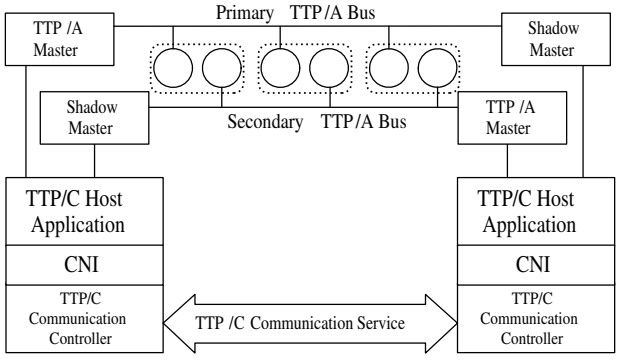

Figure 10: Integrated architecture with two TTP/C nodes and TTP/A networks

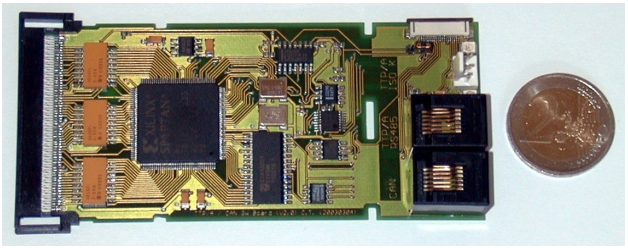

Figure 11: PCMCIA Gateway Card in comparison to size of 2 Euro coin

6 Conclusion

Although TTP/C and TTP/A are both time-triggered real-time communication protocols, they are not in a competing situation.

TTP/C focuses on the interconnection of components in order to form a highly dependable real-time system that is sufficient for critical applications such as X-by-wire in the automotive and avionics domains.

On the other hand TTP/A supports an easy integration of sensors and actuators into a network. The cost of a TTP/A transducer node are typically below the cost of the transducer, which suggests each transducer having a TTP/A interface. The interface file system concept provides a common name space for the data items that are exchanged among the transducer nodes and a master node. It establishes a stable intermediate structure that is a solid base for many new services.

The common principles of TTP/C and TTP/A support an integration of both systems in order to create dependable economically feasible control systems with distributed sensing.

Acknowledgements

This paper was supported by the European IST project NEXT TTA under contract No. IST-2001-32111.

References

[1] C. Scheidler, G. Heiner, R. Sasse, E. Fuchs, H. Kopetz, and C. Temple. Time-Triggered Architecture (TTA). *Advances in Information Technologies: The Business Challenge, IOS Press*, 1997.

[2] TTTech Computertechnik, Schönbrunnerstraße 7, Vienna, Austria. *The Time-Triggered Architecture – A Platform for Safety-Critical Applications in the Automotive Industry*, 2002. Available at http://www.tttech.com.

[3] Object Management Group (OMG). *Smart Transducers Interface V1.0*, January 2003. Specification available at http://doc.omg.org/formal/2003-01-01 as document ptc/2002-10-02.

[4] H. Kopetz. Software engineering for real-time: A roadmap. In *Proceedings of the IEEE Software Engineering Conference*, Limmerick, Ireland, 2000.

[5] P. Dierauer and B. Woolever. Understanding smart devices. *Industrial Computing*, pages 47–50, 1998.

[6] L. H. Eccles. A brief description of IEEE P1451.2. *Sensors Expo*, May 1998.

[7] H. Kopetz, M. Holzmann, and W. Elmenreich. A universal smart transducer interface: TTP/A. *International Journal of Computer System Science & Engineering*, 16(2):71–77, March 2001.

[8] H. Kopetz and R. Nossal. Temporal firewalls in large distributed real-time systems. *Proceedings of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '97)*, pages 310–315, 1997.

[9] P. Peti and L. Schneider. Implementation of the TTP/A slave protocol on the Atmel ATmega103 MCU. Technical Report 28/2000, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, August 2000.

[10] R. Obermaisser and A. Kanitsar. Application of TTP/A for the Otto Bock Axon bus. Technical Report 27/2000, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, July 2000.

[11] R. Kapeller. Design and implementation of a TTP/A master and gateway controller on a 32-bit microcontroller. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2001.

[12] M. Borovicka. Design of a gateway for the interconnection of real-time communication hierarchies. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2003.