Time Triggered Protocol (TTP/C): A Safety-Critical System Protocol

Howard Curtis and Robert France
EE382C Literature Survey (October 24, 1999)
****

Abstract

This paper examines the Time Triggered Protocol (TTP), for the support of distributed real-time systems which has recently emerged from research into the commercial world, and TTP/C, a variant of TTP for safety-critical systems that is coming into use in the automotive industry. The culmination of more than 20 years of effort, TTP has been the focus of more than 100 masters level theses and 25 doctoral dissertations. [1] In the sections that follow, we begin by discussing several requirements of embedded safety critical systems. Next we will describe the TTP/C and many of its key features and compare these features to those of the Controller Area Network (CAN) protocol, which is currently used in automotive systems. This comparison will illustrate why TTP/C, the first instantiation of a TTP primarily for use in automotive systems, is the first protocol to qualify as a SAE (Society of Automotive Engineers) Class C protocol. [2] Finally, we will examine the current state of simulation and modeling of TTP based systems. As part of this examination the authors will outline a proposed approach to high-level modeling of TTP systems.

Introduction

The trend to replace expensive, inefficient mechanical systems with more cost effective, highly featured electronic systems has been increasing in the automotive sector since the first replacement of electro-mechanical engine management more than 20 years ago. However, the use of all-electronic systems in safety-critical applications such as braking or steering requires new technologies and more sophisticated engineering practices.

TTP/C is a protocol based on the TTP (Time-Triggered Protocol) and implemented so that it meets the SAE requirements for a class C automotive protocol. Class C protocols are suitable for high-speed, single-failure operational safety-critical applications. The protocols currently used by automotive engineers, such as the J1850 family and CAN, are suitable only for Class A and Class B systems, which are subject to less rigorous requirements. TTP/C is the first protocol to meet the additional safety critical requirements of Class C. The first class of applications in development using TTP/C are the 'X-by-wire' applications, such as brake-by-wire, in which all-hydraulic and mechanical systems are replaced by electronics. [3]

The MARS Project and Its Follow-ons

The origins of the Time-Triggered Protocol (TTP), and its derivative TTP/C for use in safety critical systems, are found in the European Commission funded MARS (Maintainable Real-time System) Project, which began in 1979. The motivation for the project was the insight that within 20 years it would become possible to implement a node of a distributed real-time system in a single chip, and that this chip would be inexpensive. The original MARS architecture includes node clusters, fault-tolerant

units, nodes, tasks, and a fault-tolerant global time base, all of which is consistent with present-day TTP. In addition, the notion of using fail-silent behavior mechanisms when faults are detected has been a key concept from the beginning of MARS. The researcher who has led the development effort on TTP and TTP/C is Dr. Hermann Kopetz of the Technical University of Vienna. [4]

In 1989, the follow-on project Predictably Dependable Computer Systems (PDCS) was initiated under the European Esprit Program. PDCS involved creating a prototype of PDCS and then testing this through fault-injection experiments conducted at three European universities. (HC2) More recently still, a third project called "Safety Related Fault Tolerant Systems in Vehicles" (referred to as the "X-by-Wire" Project for short) was funded under the EC program Brite-EuRam III. The X-by-Wire Project has specifically explored the application of TTP/C-based systems in the automotive sector. Participants are Daimler-Benz Research, Fiat Research Center, Ford Europe, Volvo, Bosch, Magneti Marelli, Mecel, the University of Chalmers, and the Vienna University of Technology. [5]

Embedded Safety-Critical Systems and TTP/C

The Time-Triggered Domain versus the Event Triggered Domain: The two most common domains employed in the development of transportation systems are event triggered and time triggered. Until TTP/C, time-triggered domain engineering has been primarily used in the aerospace industry, where propulsion and navigation systems must be highly reliable and fault tolerant. However, aerospace systems such as those based on the ARINC standards are too complex and costly for use in automobiles. Event-triggered methods been the primary tool of engineers in automotive control electronics.

At the highest level, event-triggered systems advance in response to a sequence of events. Because the occurrence of these events is not time determinant, the system's behavior is not predictable in time. In fact, functional predictability is also lessened, as event-triggered systems are not temporally composable. By contrast, a time-triggered system is driven by a globally synchronized clock. Thus the behavior of the nodes can be specified in time as well as by functionality.

Classes of Safety-Critical Protocols: The SAE multiplex specifications define three classes of multiplex protocols. Class A protocols have the lowest requirements both in terms of performance and features. They are primarily intended for use in automotive body electronic applications where the performance and feature requirements are the lowest. Cost is the driving factor in Class A applications. Class B is suitable for high-speed applications such as engine management, which demand up to 1 megabit/sec. However, determinism and other safety related requirements are still not imposed. Class C is the most demanding protocol family, and includes several key safety-related features, such as protection against babbling idiots, deterministic behavior in all cases, low and bounded latency, and distributed clock synchronization.[6,7] These concepts will be described in more detail in the discussion of TTP characteristics and comparison to CAN below.

A principal reason that TTP/C is the first protocol to qualify as Class C, is that the previous protocols are all event triggered. Event-triggered systems are susceptible to several serious failure modes. For example, CAN uses a bit-wise priority arbitration scheme to control bus access. So if a node begins to send a high-priority message repeatedly, all lower priority messages are blocked from transmission. This is known as a "babbling idiot" failure, and must be guarded against to qualify for use in a safety critical system. Additionally, as the load on the bus increases, the worst case transmission time of messages is

not predictable, and the worst case may not be bounded. As control systems ultimately have temporal deadlines, message delivery must be bounded and held within its deadline to ensure continued correct operation. For safety-critical systems even this is not enough; the variance in delivery time, jitter, must be small and known to allow completely time determinant operation in all modes. . There are several good references for a complete discussion of these issues, including [8] and [9].

Key Characteristics of TTP/C with Comparison to CAN

TTP/C is a type of TDMA (Time Division Multiple Access) protocol, in which periodic time slots are assigned to individual processing nodes in a system at design time. Thus, for instance, in an automotive system, the braking, steering, suspension, and power-train subsystems may all have their own TTP/C nodes, each of which is replicated in a Fault-Tolerant Unit (FTU) to improve reliability. Each of the four subsystems is assigned its own time slot, which it uses to periodically broadcast state messages that enjoy guaranteed transmission time. [10] Eight key characteristics of TTP/C, and a comparison of each to CAN, are presented in the following section.

Node Architecture:

TTP/C: System nodes in a TTP/C system consist of a Host, a Controller Network Interface (CNI), and a TTP/C communications controller. The Host runs the application software for the relevant system function (for instance, the control software for the braking system in an automobile). The CNI stands between the Host and the TTP/C controller, effectively de-coupling the applications-level software from the network. Within the CNI is a Message Descriptor List, which contains information controlling bus access, and a data sharing interface which is typically implemented with dual port RAM (allowing the Host and the TTP/C controller to access the shared memory independently). The TTP/C communications controller provides the actual connection between the TTP/C node and the shared network. The controller supports the protocol with several essential services: "…the TTP/C controller provides guaranteed transmission times with minimal latency, jitter, fault-tolerant clock synchronization, and fast error detection." [10]

CAN: There is no fixed architecture for the CAN nodes or the corresponding interface to the host node, only the specification of the protocol itself. So each suppliers' CAN product has a different interface and feature set in terms of buffering, interrupts, etc. [11, 12]

Scheduling:

TTP/C: Due to the TDMA roots of TTP/C, system scheduling is static: the determination of when a node can place a message on the system bus is made at design time. The points in time when the various nodes in a system are authorized to transmit form the lattice points of a TTP/C "action lattice." The time difference between two adjacent points in the action lattice represent the "basic cycle time" of the system, and set a lower bound on the response time of the system. At run time, the OS in a TTP/C system uses table lookups from a table created at compile time to determine which tasks must be executed. [7]

CAN: CAN messages are scheduled dynamically. Each message is assigned a priority and the bus is given to the highest priority message requesting it at the time. This arbitration scheme means that CAN is exposed to system failure in the presence of a babbling idiot. Additionally, as the bus becomes increasingly busy, the latency and jitter associated with message delivery also increase, and the system becomes more and more difficult to predict. [11, 12]

Use of State Messages:

TTP/C: Under TTP/C, nodes generate a state message in each TDMA round (of course, when some event has occurred in the intervening time since the last message, the state message will change). State messages are posted to the CNI (Controller Network Interface) by the TTP/C Host for transmission over the system network. Messages are not queued; they are broadcast in each round, and then written over with the next message. [10]

CAN: In CAN, as with most event-driven systems, messages generated by system nodes reflect the occurrence of events in the domain of control. These events must all be queued and processed in order. [11, 12]

Clock Synchronization:

TTP/C: The access of the communication controllers in TTP/C system nodes to the bus, as noted, is regulated via the a priori allocation of time slots. A commonly maintained keeper of global time is thus critical to the proper operation of the protocol. The TTP/C controller as currently implemented in an IC provides a synchronized clock with a tick duration of 1 microsecond. Within a cluster of TTP/C nodes, all nodes are aware of which node has access to the bus during a specified time slot, given the a priori scheduling allocation. By noting the time when messages are received from other nodes (TTP/C is a broadcast protocol, so all nodes hear all messages) with the known schedule, a node can calculate the difference between the clock of the sending node and its own clock. [13]

CAN: CAN provides no notion of global clock synchronization. [11, 12]

Implicit Flow Control:

TTP/C: Flow control under TTP/C is implicit: that is to say, nodes receiving messages do not respond with a receipt acknowledgement, as they would under an explicit flow control regime. As nodes and intelligent sensors in a TTP/C system can only broadcast a message at fixed intervals, or lattice points, they may have to hold information on a development in a controlled object until the next transmit opportunity. For some kinds of events, such as the pushing of a button by an operator, the sensor must be capable of guaranteeing that the state of the button being pushed endures, or is at least preserved, until the next lattice point. In some cases, "… some short lived less important intermediate states of the RT-entities will not be reflected in the observations and will be lost. Yet, even in a peak load situation, the number of messages per unit time, i.e. the message rate, remains constant." [13]

CAN: CAN relies on explicit flow control and message addressing. [11, 12]

Composability:

TTP/C: TTP/C supports a robust level of "composability" – the capability to carry a thoroughly tested subsystem into a larger system, and to be able to depend on the subsystem retaining the same characteristics that it demonstrated in isolation. Composability is good under TTP/C because of the strict segregation of subsystems in the time domain, with each subsystem node being allocated its own time slot. In the auto industry, this feature potentially allows the rapid integration of components provided by multiple suppliers into a larger framework, without the need to perform extensive system integration tuning and testing. [14])

CAN: CAN systems are functionally composible. They are also somewhat temporally composible as discussed by Tindell: "A generally perceived problem with CAN is that it is unable to guarantee the timing performance of lower priority messages. Recent work has developed analysis to bound message latencies under arbitrary error rate assumptions." [15] However, there is no protection against babbling

idiots, nor is a CAN controller guaranteed to be fail silent (see below). This prohibits CAN from qualifying as a true Class C protocol, and precludes it from use in safety-critical systems.

Membership:

TTP/C: In order that a real-time system operate reliably, malfunctioning system components must be identified rapidly and either terminated or isolated from the remainder of the system. The role of the membership service component of a real-time protocol is to inform all system nodes of the failure of a node with minimal delay. Under TTP/C, a node membership field, maintained as a status register in the CNI, contains a so-called "node membership vector." This node membership vector contains one bit for every node in a TTP/C cluster, with the bit set to True if the node in question is operating correctly and to False if the node is not operating or is flawed. The node membership vector is updated by checking that expected messages from other nodes in the cluster are received, and by analyzing the cyclic-redundancy check (CRC) fields in the messages received. [16]

CAN: No provision for membership. [11, 12]

Reliability and Fault-Tolerance:

As TTP/C targets embedded systems with safety-critical requirements, the need for highly reliable operation is clear. Several aspects of the TTP/C protocol, and the way it is implemented in real-world systems, serve to provide reliability and fault-tolerant behavior. One of these, the TTP/C membership service, is described above. Other elements include:

>> Fail-Silence: TTP/C nodes are designed in such a way as to have heavy responsibility for detecting faults in their own operation. The principle of operation is that, "each and every node must deliver results which are correct in both the value and the time domain, or no results at all." [17] If a node detects an abnormality in its operation, it switches itself off. At a software level, TTP/C supports, for example, a variety of techniques, including double execution, double execution with reference checks, validity checks, assertion checking, and signature checks.

CAN: No guarantee for fail-silence. [11, 12]

>> Bus Guardian: The Bus Guardian (BG) is a hardware element of the TTP/C Controller which serves as a portal to the system bus. The key role of the Bus Guardian is to enable the bus driver only during the transmission slot for its node, and otherwise to guarantee that the bus driver is in a disabled state. This serves to prevent "babbling idiot" failures.

CAN: No bus guardian or similar construct. [11, 12]

>> Replication of System Components: The TTP/C protocol supports replication of elements of a real-time system in order to provide fault tolerance. First the hardware elements of a node – Host, CNI, and TTP/C controller – can be replicated to provide redundancy in the case a component fails. Bundled together, the redundant set of components then form a Fault-Tolerant Unit (FTU). [18] In addition, TTP/C supports dual system buses, which carry a duplicate set of signals.

CAN: Does not guarantee replica determinism. Does have a two-wire physical interface that supports "limp home" operation in most fault scenarios. [11, 12]

One key aspect of TTP/C in error detection is that the receiving node is responsible for identifying missing or errant transmissions. This is in contrast to event-driven protocols, where typically the sending node retains this responsibility through some sort of acknowledgement mechanism.

Modeling and Simulation

Existing Research and Tools: As TTP/C is only recently becoming commercially viable, there are not yet many modeling and synthesis tools available. Prof. Hermann Kopetz has started a commercial venture to support TTP/C in the marketplace. This company, TTTech, has introduced a tool set for the commercial development of TTP/C-based systems. However, it currently supports only the development and implementation of the communication system itself; the tools do not support modeling of an overall system. Likewise, there is no simulation capability currently present. [19] As TTP/C contains all the needed services to build time-deterministic systems, suitable for use in safety critical applications, such an addition would be valuable.

Studies are beginning to appear that support the idea of application-level design being approached using graph theory. [20] In this paper the authors use conditional process graphs combined with several scheduling algorithms to demonstrate that it is possible to algorithmically create static schedules for systems designed as conditional process graphs. Furthermore they show that the schedules can be bounded, and that a MEDL description allowing the schedule to run on a TTP/C system can be derived. Of secondary interest is the notion that the ordering of the slots in the TDMA round can be used to optimize the overall schedule length.

Given that the number of such studies is increasing, we see a need for an environment that would allow the researcher to quickly create and evaluate how different scheduling algorithms work for a variety of system architectures.

Requirements for Application-Level Modeling: Although the detailed requirements of a complete commercial quality environment are many, the high-level requirements are fairly straightforward:

I. The scheduling algorithm should be separate from the application modeling environment, but easy to connect. It should also be easy to implement the algorithm using standard tools.
II. The application modeling domain should have equivalent properties to the final target system. This ensures the modeled execution is consistent with the final execution on the target system.
III. The technologies used should be based on already known and easy to work with tools and languages. The basis for the system is to provide researchers a powerful, but easy to use environment, allowing them to focus on their studies rather than the creation or learning of a complex set of methods and tools.

Proposal for Implementation

We propose a partial implementation of a system such as that described above. This system would utilize common tools and languages, namely Ptolemy, C++, and C. We would create an example

application system. Having done this, we would devise and implement a scheduling algorithm to generate the MEDLs for a TTP/C-based system. In the third activity, we would create new stars within Ptolemy SDF, that support the modeling and simulation of the devised application system. These stars would be created at a high level of abstraction, but with enough detail to allow the introduction of various faults, such as a broken wire, or babbling idiot node. The goal would be to demonstrate that such an environment is achievable and allows the modeling of a system in SDF. Although beyond the scope of this study, the next step would be to actually create a TTP/C system that allows the devised applications to execute on real hardware and provides the final validation of the approach.

Conclusion

In the course of this paper, we have discussed several key features of automotive safety-critical systems. We examined a newly available protocol, the TTP/C, and discussed its suitability for implementing such systems. Additionally, we provided a brief comparison to a leading and widely deployed event triggered protocol, CAN, to illustrate the strengths of a time-triggered protocol, and how TTP/C address the weakness in the existing event-driven protocols. Finally, we described an environment that would allow researchers in this field to more easily evaluate several aspects of creating TTP/C based systems by providing a modeling and simulation front-end to supplement existing commercial implementation tools. As the use of modeling and simulation becomes increasingly prevalent in automotive systems engineering, and as pressure on design cycle-times increases, this capability will be useful. Additionally the possibility to further extend the environment to synthesize the final application directly from the model is an important development. This would provide not only a productivity enhancement, but also move the design of safety critical systems for cars one step closer to being a mathematically provable activity, rather than today's less rigorous approach.

References

[1] Motorola WWW site (author unknown), TTTech Newsletter, http://www.mot-sps.com/automotive/pdf/TTPNews.pdf, p. 1.

[2] SAE, "Class C Application Requirement Considerations," *SAE Recommended Practice*, J2056/1, SAE, June 1993.

[3] Elmar Dilger, Thomas Fuhrer, Bernd Muller, and Stefan Poledna, "The X-by-Wire Concept: Time-Triggered Information Exchange and Fail Silence Support by New System Services," *SAE 1998 Conference*, pps. 141-42.

[4] Hermann Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, pps. 286-87.

[5] Elmar Dilger, Thomas Fuhrer, Bernd Muller, and Stefan Poledna, "The X-by-Wire Concept: Time-Triggered Information Exchange and Fail Silence Support by New System Services", SAE 1998 Conference, pps. 141-42.

[6] Ross Bannatyne, "Time Triggered Protocol – Fault Tolerant Serial Communications for Real-Time Embedded Systems," *Wescon 1998*, p. 86.

[7] SAE, "Glossary of Vehicle Networks for Multiplexing and Data Communications," *SAE Recommended Practice*, J1213/1, SAE, Sept. 1997.

[8] Hermann Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*.

[9] Hermann Kopetz, "Should Responsive Systems be Event-Triggered or Time-Triggered?," *IEICE Transactions of Information and Systems*, Vol. E76-D, November 1993, p. 1329.

[10] Ross Bannatyne, "Time Triggered Protocol – Fault Tolerant Serial Communications for Real-Time Embedded Systems," *Wescon 1998*, p. 88.

[11] CIA, "CAN Specification 2.0, Part A," http://www.can-cia.de/.

[12] CIA, "CAN Specification 2.0, Part B," http://www.can-cia.de/.

[13] Georg Kroiss, "The Time Triggered Communication Protocol TTP/C," *Real Time Magazine*, No. 4, 1998, p. 101.

[14] Stefan Poledna, "The Time-Triggered Communication Protocol TTP/C," *Real Time Magazine*, No. 4, 1998, p. 100-101.

[15] Ken Tindell, "Guaranteeing Message Latencies on Control Area Network (CAN),' *International CAN Conference* paper, September 1994.

[16] Hermann Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, pps. 133 and 179.

[17] Elmar Dilger, Thomas Fuhrer, Bernd Muller, and Stefan Poledna, "The X-by-Wire Concept: Time-Triggered Information Exchange and Fail Silence Support by New System Services," *SAE 1998 Conference*, p. 145.

[18] Hermann Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, p. 177.

[19] TTTech website, http://www.tttech.com.

[20] Paul Pop, Petru Eles, Zebo Peng, "Scheduling with Optimized Communication for Time-Triggered Embedded Systems," *Communications of the ACM, 1999*, pps. 178-182.

H. Curtis & R. France (10/23/99: Version 3)