The Modeling and Simulation of an Automotive Braking System Using the TTP/C Protocol

Robert France and Howard Curtis
(EE382C – Embedded Software Systems)
****

Abstract: TTP/C, which represents one variant of the Time-Triggered Protocol (TTP), is designed to address safety-critical real-time control systems in the automotive sector. Of high interest in analyzing TTP/C, given its emphasis on "x-by-wire" environments wherein electronic control systems do not have hydraulic or mechanical back-up components, are the aspects of the TTP/C protocol and architecture which concern themselves with reliability and fault tolerance. In this paper, the authors briefly discuss the key safety-related constructs of TTP/C. An experimental project in which a TTP/C-based braking system was simulated in software, with particular emphasis on the behavior of the "bus guardian," is then described and analyzed.

The sections of this report cover the following topics: (1) description of the requirements of the SAE Class C specification for safety-critical systems; (2) overview of the reliability related aspects of the TTP/C protocol and architecture, including the bus guardian; (3) summary of modeling and simulation work in TTP/C reported in the literature; (4) description of the objectives and approaches of the modeling and simulation work conducted in this project; (5) opportunities for future investigation, and (6) summary and conclusions.

Key Papers:

For the Literature Review portion of our project, the following represent three key papers:

>> Ross Bannatyne, "Time Triggered Protocol: TTP/C," *Embedded Systems Programming*, March 1999, pps. 76-86.
>> Hermann Kopetz, "Should Responsive Systems be Event-Triggered or Time-Triggered?", *IEICE Transactions in Information & Systems*, Vol. E76-D, No. 11 (November 1993), pps. 1325-1332.
>> Hermann Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, Norwell (MA), 1997. (Monograph)

For the Simulation and Modeling portion of our project, the following represent three key papers:

>> Elmar Dilger, Thomas Fuhrer, and Bernd Muller, "The X-By-Wire Concept: Time-Triggered Information Exchange and Fail Silence Support by New System Services," *Advances in Safety Technology*, Society of Automotive Engineers, 1998, pps. 141-149.
>> B. Hedenetz and R. Belschner, "Brake-by-Wire Without Mechanical Backup by Using a TTP-Communication Network," *SAE International Congress and Exposition* (Detroit, Michigan), 1998, pps. 1-9.
>> Hermann Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, Norwell (MA), 1997. (Monograph)

1. Introduction and Overview

TTP/C, which represents one variant of the Time-Triggered Protocol (TTP), is designed to address safety-critical real-time control systems in the automotive sector. Given the TTP/C emphasis on "x-by-wire" environments, in which there exist no mechanical or hydraulic back-ups for a system such as brakes or steering in a car, the aspects of TTP/C which provide reliability and fault-tolerance are of high interest. In our project work, we have sought to implement a first-order model of the behavior of a TTP/C system for automotive braking, with emphasis on an element of the TTP/C controller node called the "bus guardian."

The bus guardian should be considered a representative choice of modeling target, rather than the exclusive purpose of this study. Our objective is to experiment with approaches to broad-based modeling of the behavior of TTP/C systems, both when functioning normally and in the presence of injected faults. The bus guardian provides an excellent target for this investigation, given its well-defined behavioral characteristics, and the fact that it must interact closely and continuously with other elements of the TTP/C node.

2. Description of the Requirements of the SAE Class C Specification

The general requirements for safety-critical protocols and systems in the automotive domain are defined by the SAE (Society for Automotive Engineers) Class C specification. The key requirements for Class C automotive communication systems include the following [1]:

- A communication system which supports composability, meaning that subsystems which are developed independently, tested, and certified compliant with TTP/C requirements can then be integrated with high assurance that they will work together.
- Support for the connection of replicas of processing units, and for the distribution of these replicas, so as to avoid failures of the functions provided by processing units.
- The provision of an independent device to guard against failures induced by babbling idiots (babbling idiots are processing nodes which emit a constant stream of unnecessary messages, thus monopolizing the communications bus).
- Provision of a mechanism that permits a distributed application to know the status of all connected system components (in TTP/C, this facility is called the membership service).

Communic ation systems which are in wide use in automobiles today, such as CAN, A -BUS, VAN, J1850-DLC, and J1850-HBCC, cannot meet this rigid set of requirements; most of them lack synchronization, fault -tolerant characteristics, and deterministic behavior [2].

3. Aspects of TTP/C that Assure Reliability and Fault-Tolerance

In work at the Technical University of Vienna, Dr. Hermann Kopetz and his colleagues have designed TTP/C from the ground up to address stringent reliability requirements, such as those reflected in SAE Class 3. The following contribute to meeting the challenge posed by Class C.

(1) Dual-Bus Architecture: Typically, the physical implementation that supports TTP/C features a bus with two separate channels, and the TTP/C protocol supports this dual -bus model. Even if one of the bus circuits fails or is cut, signals will continue to travel.

(2) Fail-Silence: TTP/C processing nodes (also called Fail Silent Units – FSUs) are designed to meet the criterion that each individual node "must deliver either re sults which are correct in both the value and the time domain or no results at all." [3] Nodes are assigned heavy responsibility for guaranteeing this so -called fail -silent property through their internal behavior, and thus typically include both hardware and software mechanisms to identify faulty operation. [4]

(3) Replication: In addition to replication of the communication bus, TTP/C supports replication of system processing nodes at the level of both the Fail Silent Unit and that of the Fault Tolerant u nit (a Fault Tolerant Unit, or FTU, consists of two or three FSUs, providing hardware redundancy). Under the TTP/C protocol, when a node withdraws from the system through internal identification of faulty behavior, or because it has been eliminated under TTP/C's membership resolution capability, the functioning back -up node takes over in providing the target service.

(4) Membership: The concept of Membership in TTP/C is the community corollary of Fail Silence. Whereas the individual TTP/C node is assigned pr imary responsibility for diagnosing and responding to faults in its internal behavior, information about whether a node is functioning properly or is faulty must be rapidly shared with all other nodes in a TTP/C. This is the function of the Membership Ser vice. TTP/C's Membership Service is based on the fact that each node in a TTP/C system has a designated TDMA time slot

within which it sends its messages, and all nodes in the system have a priori knowledge of this schedule. Thus, if a node fails (or thr ough self-diagnosis of a fault, chooses not) to broadcast a message in its assigned time slot, other nodes are aware of the failure. [5]

(5) Bus Guardian: The bus guardian is a module of a TTP/C node (for additional detail on the architecture of the TTP/C node, see our earlier Literature Review). While physically located within the TTP/C Controller, the bus guardian functions independently to insure that the node it belongs to places messages on the communication bus only within its assigned time slot (there are actually two bus guardians per node in most TTP/C implementations – one for each of the redundant buses). In the event that the node seeks to broadcast a message out of turn, the bus guardian intercepts and cuts off this behavior. In this fashion, th e bus guardian protects the communications bus, and the entire TTP/C system, against the occurrence of "babbling idiots." [6]

As noted, the primary objective of this project has been to design and implement an initial simulation of a TTP/C-based system for vehicular steering control, with emphasis on the behavior of the bus guardian.

4. The Modeling and Simulation of TTP: Prior Work

As the overall TTP family protocol and architecture matures, discussions of modeling and simulation have begun to appear in the literature. For example, Bernd Hedenetz has reported on work conducted at Daimler Benz Research (DBR), which has accumulated experience in designing TTA - based systems (TTA is the TTP variant for non -safety critical applications such as body electroni cs in automobiles). [7] DBR employs a seven-step system development process for embedded systems which includes: Requirements Specification; Architectural Design; Functional Design; Functional Simulation; Fault Modeling; Realization & Implementation; an d Test & Fault Injection. DBR uses Statemate from I-Logix and MatLab/Simulink from MathWorks to support the Functional Design and Functional Simulation steps. Fault modeling can be achieved by injecting faults into the same software models. In the Realization & Integration step, DBR uses code generation tools. [8]

5. Modeling and Simulation Approach in This Project

Although some modeling and simulation work has begun to appear related to the design of time triggered systems, this is still a newly emerge nt field. As such, much work remains to be done, especially regarding how best to exploit the properties of the underlying technologies such as the TTP/C protocol at higher levels of the system. Membership, protection against babbling idiot nodes, and globally synchronized time are all powerful properties, but require special effort to properly utilize them in the application software.

In fact, in addition to the properties mentioned above, adding temporal predictability and composability to functional com posability makes the TTP/C particularly well suited to complex, distributed systems including safety -critical systems. Given these properties and the nature of the systems suitable for development using TTP/C, a natural question is how to best go about de signing, specifying, and implementing these systems. In particular, it seemed to the authors that formal modeling techniques and the associated body of knowledge and tools would be a very useful framework in which to design and simulate TTP/C systems with an eye to formal verification and validation in the future. Additionally, at least some pieces of the working system could then be used to synthesize actual production code.

Although a complete treatment is beyond the scope of this work, our intent is to create a high level, yet accurate, model of a physical system based on TTP/C. Furthermore, we intend to show that such a model can effectively be used to determine the behavior of the modeled system in fault conditions that may be difficult to duplicate i n a physical instantiation of the system.

5. 1 The Physical System: Brake-by-Wire

The physical system we chose to model was a simple five node brake -by-wire system. It contains a brake pedal node which outputs a single value ranging from 1 to 255, linearl y correlated to the position of the pedal. This value is propagated to four brake nodes, which then calculate the required force and apply this to the wheel under their control. The brake nodes also output the current value of the brake force being applied to the wheel. A value of 0 is used by the bus guardian to indicate a failure mode within the brake node, and that no brake force is being applied to the wheel. Figure 1 depicts the system we are simulating. The semantics used are as the usual, objects re presented by blocks, arrows depict methods, and so on.

5.2 The Model

The model of our brake system was designed to accurately model the operating characteristics of the physical system. Figure 2 depicts the order of the nodes' respective message slots as implemented in our system. The time model used in our simulation is discrete time, where each unit of advancement is a single message slot in the cluster cycle. Because the time slot management is done only in the schedule, each of the nodes runs once duri ng each slot and is not allowed to advance time. However, only the node assigned to a given time slot is allowed to send a value, a characteristic enforced by each node's bus guardian. Additionally the sending node is scheduled first, as the model is faith ful to the concept of state variables used in the TTP/C communication model.

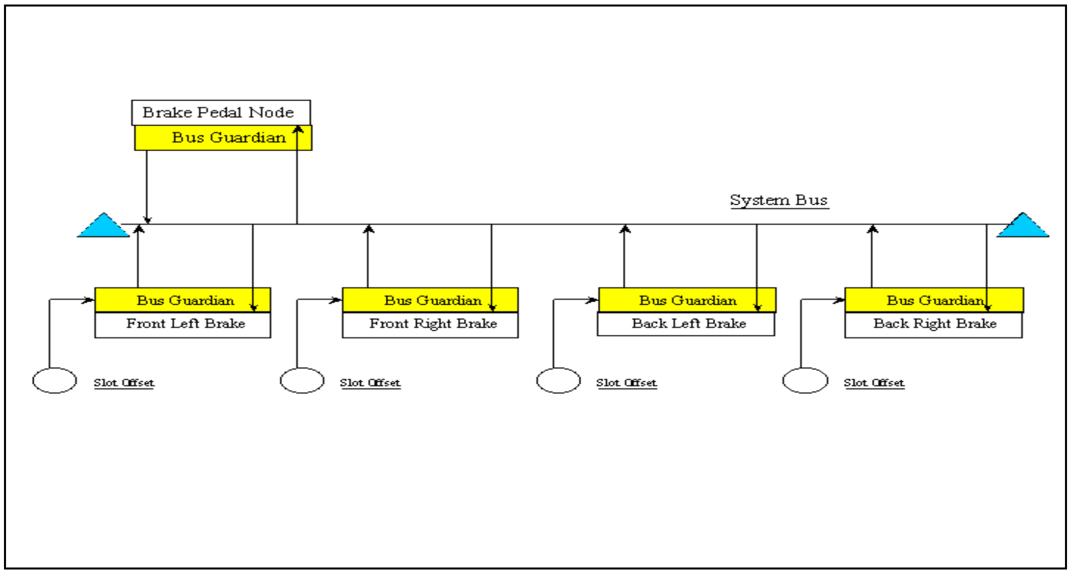

Figure 1: Schematic diagram depicting TTP/C -based steering system simulated in this study.

The only major departure from an TTP/C actual system we have identified is that our m odel is single threaded. In a real system each node would be running simultaneously. This results in variance in how some faults would manifest themselves if the TTP/C hardware failed. In particular, a babbling idiot

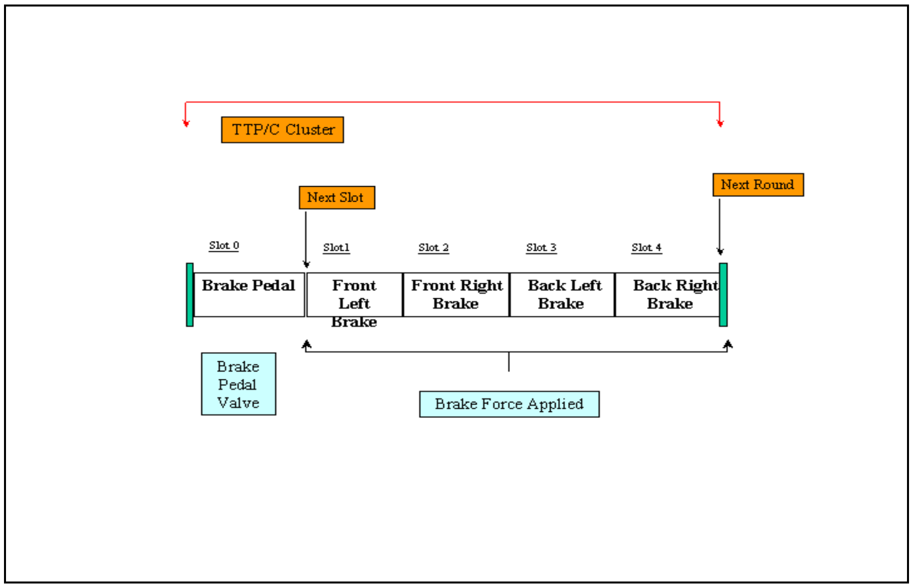

Figure 2: Order of TTP message slots in the simulation of a TTP/C -based braking system.

node would dead-lock a single threaded system if the bus guardian logic failed. So one limitation of our approach is in modeling certain faults in the bus guardian logic. We chose to create an incorrec t time slot transmission as an application flaw, perhaps by a discrepancy between the MEDL values and the nodes' perceived current time slot value. This fault is modeled correctly at the bus level, as is shown in the results section.

5.3 The Implementation

The function `main()`contains the object instantiations and the schedule. It was implemented in C++ and both the MEDL and schedule were created by hand. A sample can be seen in Figure 3 illustrating the schedule and object creation..

Each node type is implemented as a class containing all needed data stores and methods. The program was developed in Microsoft Visual Studio 5, and run under Windows NT.

```
// Create our bus and all the nodes
TTP_C_BUS      bus(NUMBER_NODES);
TTP_C_BUS&     busRef = bus;
BrakePedal     pedal(PEDAL_ID,
BRAKE_FULL_ON);
BrakeNode      flBrake(FL_BRAKE_ID);
BrakeNode      frBrake(FR_BRAKE_ID);
BrakeNode      blBrake(BL_BRAKE_ID);
BrakeNode      brBrake(BR_BRAKE_ID);

pedal.pushPedal(busRef, iterate %255);


flBrake.applyBrake(busRef);
frBrake.applyBrake(busRef);
blBrake.applyBrake(busRef);
brBrake.applyBrake(busRef);
bus.nextSlot();
```

Figure 3: Example code

The code is strictly ANSI compliant, uses no operating system calls, and sho uld be easily portable to any system. It was also compiled by the Cygnus g++ compiler for NT.

5.4 Results of Simulation Experiment

The result of our experiment is that the model performed as expected in both the normal and error cases. The error case result is shown in Figure 4 as it is the more interesting of the two. Observe that when the timeslot comparison is corrupted by introducing a static offset, the bus guardian in the disturbed node does not allow transmission outside of its timeslot. This err or was introduced in the back right wheel during timeslots 125 -149, and in the front left wheel during timeslots 230 -242. See Figure 5 for the mechanism used to introduce the error. This would allow the system to compensate by not applying brake force to the diagonal wheel, keeping the car traveling in a straight path. The failure mode here is silence on the bus and applying no braking force to the wheel. This allows the system the chance to adapt and take corrective action. Strengths of the model are that the schedule is fully static, and changes can be made to each node's application code with no effect on the overall systems. Only adding or deleting nodes requires a change to the schedule. We were also able to demonstrate that the behavior at the bus leve l can be correctly modeled at a high level, without loss of accuracy, but only loss of detail. This is important as additional detail needed for each system can be added to individual nodes as desired with the assurance that the system will still function correctly. As noted earlier, the one exception is the current single-threaded execution model that would allow one node to block system execution.

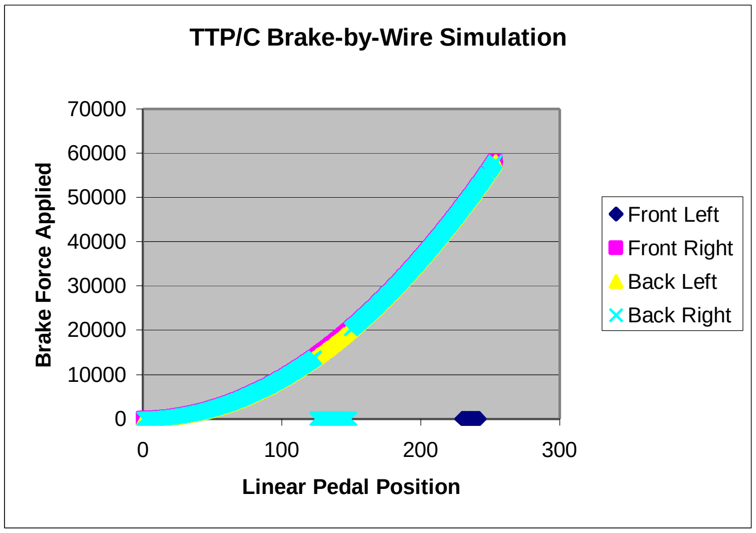

Figure 4: Results of brake fault injection for back right and front right wheels.

```
#define ERROR_ON
#if defined ERROR_ON
if (iterate == 230) {
    flBrake.setSlotOffset(1);
}

if (iterate == 243) {
    flBrake.setSlotOffset(0);
}

if (iterate == 125) {
    brBrake.setSlotOffset(1);
}

if (iterate == 150) {
    brBrake.setSlotOffset(0);
}

#endif
```

Figure 5: Fault injection mechanism.

6. Future Study

Several areas identified during our project warrant further development and exploration. The first would be the addition of a membership function. This would be quite easy to do in the exis ting framework. However, to facilitate easier development of future systems, it would be worthwhile to develop the TTP/C functionality as a standard base class and then create derived classes for the system nodes that inherit the desired TTP/C functionalit y. This would of course need to be done while maintaining suitability for inclusion in higher level systems such as Ptolemy. Additionally, a multi - threaded execution model would be needed to support the simulation of several failure modes. It may also be required if there are any observable system side effects, other than simply monitoring the TTP/C bus. Finally, several alternative modeling options are feasible, such as SR (Synchronous Reactive) and several other dataflow variants. It would be useful to see a more complete evaluation of which combination would be most suitable. In fact, both of the initial approaches we considered suffered from at least one violation of the domain semantics. For example, the current message slot value is represented as a st ate variable. Whichever node is assigned to that slot is required to produce a value, but all nodes in the system are given a chance to react to the new value in the same instant of discrete time. This is the mechanism that allows us to model the multi -threaded real system in a single thread of execution. Unfortunately, it also means that the number of values produced by a node is sometimes zero, and sometimes one. It depends on whether the current timeslot is the one assigned to that node. This rules out SDF. This question of how best to model the TTP/C within the strict domain semantics supplied by Ptolemy is important. If a suitable approach was found it would allow the modeling and simulation to be done in the existing, full -featured Ptolemy environment, rather than requiring the model to be hand coded. There would also be significant validation benefits to mapping the TTP/C behavior into a well-studied set of domains, whose properties have already been explored and proven.

7. Summary and Conclusion

In summary, we have examined the motivation for, development of, and central properties of the TTP/C. We were able to apply formal modeling techniques to the design of a simple brake -by-wire system. This model was executable and allowed us to demonstrate the be havior of a key TTP/C system property, fail-silence, as provided by the bus guardian. Finally, we examined our results and highlighted several areas for future study. Hopefully by successfully demonstrating that this is a useful and viable approach to the design and simulation of TTP/C based systems, we will encourage future studies in how these concepts can be applied to bring these systems to market in a timely and efficient manner.

8. References

[1] B. Hedenetz and R. Belschner, "Brake-by-Wire Without Mechanical Backup by Using a TTP - Communication Network," *SAE International Congress and Exposit*ion (Detroit, Michigan), Feb. 1998, p. 2.
[2] B. Hedenetz and R. Belschner, "Brake-by-Wire Without Mechanical Backup by Using a TTP - Communication Network," p. 2.
[3] B. Hedenetz and R. Belschner, "Brake-by-Wire Without Mechanical Backup by Using a TTP - Communication Network," p. 3.
[4] 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," p. 145.
[5] Hermann Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications,* pps. 133 and 179.
[6] Hermann Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications,* p. 173.
[7] Bernd Hedenetz, "A Development Framework for Ultra-Dependable Automotive Systems Based on a Time-Triggered Architecture", the 19th IEEE Real-Time Systems Symposium, Dec. 1998, pps. 358-367.
[8] Bernd Hedenetz, "A Development Framework for Ultra-Dependable Automotive Systems Based on a Time-Triggered Architecture", the 19th IEEE Real-Time Systems Symposium, Dec. 1998, pps. 360.