The Timed Asynchronous Model and its Application in Time-Triggered Protocols

Ruiqi Hu

Abstract

The time-triggered protocols (TTPs) are designed for developing fault-tolerant distributed hard real-time systems based on reliable communication networks. They do not apply in systems with only asynchronous communication networks. The timed asynchronous (TA) model is an accurate description of the asynchronous distributed systems with a communication network which has a non-negligible failure frequency. In this survey, we introduce the basic ideas of both TTPs and the TA model in detail and propose a new approach to model distributed real-time systems in such a way that TTP based local clusters with reliable intra-cluster communication networks communicate on unreliable inter-cluster communication networks via TA based protocols.

1. Introduction

The Time-Triggered Protocols (TTPs) are designed for the implementation of time-triggered hard real-time systems [Kop94]. TTP/C is the first protocol that achieves low cost and high dependability at the same time. This makes it suitable for high-speed, single-failure operational safety-critical applications such as automobile manufacture industry. TTP assumes the underlying communication network to be highly reliable.

The timed asynchronous (TA) system model [Cri99] can be used to describe existing distributed systems built from network workstations connected by unreliable communication networks. It allows many needed services such as clock synchronization, membership, consensus etc. to be implemented.

More and more distributed systems are using unreliable communication networks, e.g. systems based on mobile communication are asynchronous by their nature. It is possible to model such systems with an integration of the TA model and TTPs to maintain the wide system coverage of the TA model and the cost-effective properties of TTPs at the same time.

With the rapid evolution in both software and hardware industry, many commercial operating systems that address real-time concerns are emerging. As an example, Windows CE™ is designed to be a new, portable, real-time, modular operating system that features popular Microsoft programming interfaces and that is supported by tools that enable rapid development of embedded and dedicated systems [Mur98]. It is nevertheless beneficial for system designers to take advantage of what these operating systems provide to build their real-time systems.

The purpose of our research is to investigate ways to implement the prototyped timed asynchronous and time-triggered services on Windows CE powered devices and propose a modeling method that integrates TA services and TTP services together in the design of a distributed real-time system. In this project, we expect to build a demonstration that shows

the capability of this method.

2. Time-Triggered Protocols

Event-triggered architectures and time-triggered architectures are two fundamentally different paradigms for the design of real-time systems. In an event-triggered architecture, all activities, such as task activation and communication, are initiated as consequences of events and all scheduling and communication decisions are made dynamically. It is more flexible and more responsive during the low-load scenarios. On the other hand, time-triggered system are driven by the progression of the global time. All tasks and communication actions are initiated when the real-time clock reaches predetermined values. A distributed time-triggered system must synchronize the local clocks to represent a globally synchronized time base of specified precision [Kop87]. Time-triggered systems are more robust and testable, and perform well even in overload scenarios [Kop94].

2.1 TTP in detail

A typical TTP system are a cluster of Fault-Tolerant Units (FTUs), each of which consists of one or more nodes, interconnected by a communication network with dual communication buses. A node consists of two subsystems: the host and the communication controller. The Communication-Network Interface (CNI) is the internal interface between them. The communication controller has a local memory to hold the MEssage Descriptor list (MEDL) that determines at what point in time a node is allowed to send a message, and when it can expect to receive a message from another node. It also contains independent hardware devices, the Bus Guardians (BSs), that monitor the temporal access pattern of the controller to the replicated buses, and terminate the controller operation in case a timing violation in the regular access pattern is detected [Kop97].

TTP is a time-division-multiple-access (TDMA) protocol where every node can only send its message on the shared communication buses in a specific assigned time slot in a

predetermined TDMA schedule which is shared by all the nodes. The time-triggered nature of TTP brings it following properties:

Use of A Priori Knowledge: TTP has a sparse time base [Kop97] in which events only happen at predetermined time points in the globally synchronized time base. Since it is TDMA based, TTP allows every message to be broadcasted and uses a simplified acknowledgement scheme [Kop94]. The synchronous schedule is generated at the initialization phase of the system.

Clock Synchronization: TTP maintains a Byzantine-resilient clock synchronization with a precision about 10^{-6} second by performing the Fault-Tolerant Average (FTA) algorithm periodically, with the support of hardware in most cases [Kop97].

Composability: the CNI between the TTP controller and the host computer is fully specified in the value and temporal domain. This allows the host to be built and tested independently. Since the CNI will remain the same during a system integration, the properties of the subsystems will not be invalidated after the system integration. This enables the adoption of a cost-effective bottom-up design method in developing time-triggered real-time systems.

Fault Tolerance: The independent bus guardians ensure that a node either delivers a message at the correct moment or not at all, i.e. the nodes support the fail-silent abstraction in the temporal domain. The host software is responsible to achieve fail silence in the value domain by ensuring space and/or time redundancy that all the internal failures of a host are detected before a non-detectable erroneous output message is transmitted [Kop97]. TTP also provides replica determinism: a message arrives at all recipients at exactly the same time with exactly the same contents or it does not arrive at all [Kro98].

Membership: The membership service in TTP can inform all system nodes of the failure of a node with minimum delay. Each node broadcasts its membership vector together with any message it sends. Each node updates its membership vector after receiving a message from another node by checking the status of the corresponding field in the mes-

sage.

Two variants of the TTP are available now, the full version TTP/C and the scaled-down version TTP/A, which share a compatible CNI. The TTP/C protocol is the full version of the TTP that provides all services needed for the implementation of a fault-tolerant distributed hard real-time systems [Kop97].

3. The Timed Asynchronous Model

Distributed systems can be classified as synchronous or asynchronous depending on whether their underlying communication and process management services can provide “certain communication” or not [Cri96]. A system is synchronous if it guarantees certain communication and is asynchronous otherwise[Cri99]. A certain communication has the property that at any time there is a minimum number of correct processes, and any message sent by a correct process to a correct destination process is received and processed at the destination within a known amount of time. This means the probability the message is not received and processed in time is “negligible”. This property can be achieved upon the assumption that the frequency of failures that can occur in a system is bounded [Cri91]. Unfortunately, in many distributed systems this assumption can not be fully satisfied.

The TA model is proposed by Flaviu Cristian and Christof Fetzer to describe asynchronous distributed systems. It makes following basic assumptions on the underlying system [Cri99]: 1) All services are timed: their specification is prescribed not only in the value domain, but also in the temporal domain; 2) Interprocess communication is via an unreliable datagram service with omission/performance failure semantics: the only failures that messages can suffer are omission (message is dropped) and performance failures (message is delivered late); 3) Processes have crash/performance failure semantics: the only failures a process can suffer are crash and performance failures; 4) Processes have access to hardware clocks that run within a linear envelope of real-time; and 5) No bound exists on the rate of communication and process failures that can occur in a system. These assumptions

are practical because: 1) Nearly all the workstations now are using high-precision quartz clocks; and 2) For those services without any response time promises, a high level abstraction that depends on them (it could be a human user at the highest level) can define a time-out to decide if they are failed.

This model adequately describes existing distributed systems build from network workstations since many practically needed services such as clock synchronization, membership, consensus, election, and atomic broadcast have been shown to be implementable [Cri99] [Fet99]. In [Fet96] the notion of fail-awareness is introduced as a systematic means of transforming synchronous service specifications into fail-aware specifications that become implementable in timed asynchronous systems.

3.1 TA in detail

A timed asynchronous distributed system consists of a finite set of processes which communicate via a datagram service. Processes run on the computer nodes of a network. Lower level software in the nodes and the network implements the datagram service. Each process has access to a local hardware clock. The process management service that runs in each node uses this clock to manage alarm clocks that allow the local processes to request to be awakened whenever desired [Cri99].

The hardware clock consists of an oscillator and a counting register that is increased by the ticks of the oscillator. It can drift apart from real-time because of the imprecision of the oscillator, temperature changes, and aging. The model assumes there exists a constant maximum drift rate that bounds the absolute value of the drift rate of a correct clock. Hence in this model, all the correct clocks are within a narrow linear envelope of real-time. The drift rate can be further divided into systematic drift error due to the imprecision of the oscillator and drift errors due to other reasons such as aging or changes in the environment. A hardware clock can be calibrated by multiplying a constant factor to reduce the systematic drift error and it could reduce the drift rate by a magnitude of two in prac-

tise. Clocks can be externally synchronized if at any instant the deviation between any correct clock and real-time is bounded by a known constant, or internally synchronized if the deviation between any two correct clocks is bound by a known constant. Either of these synchronization needs to be performed periodically to account for the ongoing drift of all clocks [Cri99].

The datagram service provides primitives for transmitting unicast and broadcast messages. It must satisfy following requirements: 1) Validity: if the datagram service delivers a message *m* to a process *p* at time *t* and identifies process *q* as *m*'s sender, then *q* must sent *m* at some earlier time $s < t$, 2) No-duplication: each message has a unique sender and is delivered at a destination process at most once, and 3) Min-Delay: any message sent between two remote processes has a transmission delay that is at least a constant lower bound transmission delay. The datagram service by itself does not ensure the existence of an upper bound for the transmission delay of messages. However, since all services in the model are timed, a one way time-out could be defined so that within which the actual messages sent or broadcasted are likely to be delivered. It is possible for a message to have a transmission delay greater than the time-out value, in which case the message suffers a performance failure. If a message is never delivered, it suffers an omission failure. The choice of this time-out may vary with the size of the message, network load, and also depends on the protocol [Cri99].

A process can be in one of the following three modes: 1) up: it is executing its 'standard' program code, 2) crashed: a process stopped executing its code, and 3) it is executing its state 'initialization' code either after its creation or when it restarts after a crash. A process can set an alarm clock to be awakened at some specified future clock time. It will not take a step after it sets its alarm clock unless it is awakened for that alarm time or it receives a message before it is awakened for that alarm time. When a process crashes, the process management service forgets any active alarm time it has set before crashing. The scheduling delay is defined as the time interval between the earliest real-time at which the

hardware clock shows a value at least the preset alarm clock and the actual time the process is awaken. The process management service does not ensure the existence of an upper bound on this scheduling delay. Again, a timeout delay can be defined so that actual scheduling delays are likely to be smaller than it. A non-crashed process suffers a performance failure when it is not awakened within the timeout interval of the last alarm time it has specified [Cri99].

4 Plans for Implementation of the Project

As described in the previous two sections, TTP assumes its underlying communication network to provide certain communication. It is a reasonable assumption since TTP is designed for systems with dedicated communication network that can provide highly reliable transmission with a message failure rate about 10^{-9} per hour. However, if we want to build a distributed system based on unreliable communication networks like mobile networks, the assumptions TTP made are no longer satisfied. However, the TA model can be used to describe these systems accurately. It is thus possible to model the whole system into several clusters which communicate via an asynchronous communication network. Each cluster consists of one or more FTUs communicating via synchronous communication network. TTPs are fully functional to implement intra-cluster services and TA based protocols can be used to implement inter-cluster services. A special FTU derived from TTP gateway [Kop97] acts as the interface between them.

The utmost objective of this research is to demonstrate the capability of the approach above. In order to make TTPs and the TA model cooperate effectively, many issues are needed to be investigated, such as integrating the global probabilistic clock synchronization [Cri89] and the cluster-wise FTA algorithm [Kop97] to achieve system-wide clock synchronization, using the failure transformation mechanism proposed in to achieve synchronous properties from timed asynchronous model with hardware watchdogs, designing parameterized clock synchronization and atomic broadcast algorithms to ensure quality of

services, and dynamically generating TDMA schedules for TTP to improve system adaptability. All of these mean a huge amount of work and it is too ambitious to finish all of them within such a limited time of one semester.

Hence, I plan to focus on the modeling and simulating part of this project first. It might be a good starting point to design a distributed service (a system clock synchronization protocol is under consideration now) based on this distributed system model and simulate it on a prototyped system implementation. The hardware environment will be based on Windows CE powered PCs (CEPCs).

5 KEY PAPERS

[Cri89], [Cri99], and [Kop94].

REFERENCES

[Cri89] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, vol. 3, no. 3, pp. 146-158, Mar. 1989. An earlier version IBM Research Report, San Jose, RJ 6432, 1988.

[Cri96] F. Cristian. "Synchronous and asynchronous group communication," *Communications of the ACM*, vol. 39, no. 4, pp. 88-97, Apr. 1996.

[Cri99] F. Cristian and C. Fetzer, "The timed asynchronous distributed system model," *IEEE Trans. on Parallel and Distributed Systems*, vol. 10, no. 6, June, 1999.

[Fet96] C. Fetzer and F. Cristian, "Fail-awareness in timed asynchronous systems," In *Proc. of ACM Symp. on Principles of Distributed Computing*, Philadelphia, May 1996, pp. 314-321a. http://www.cs.ucsd.edu/~cfetzer/FA.

[Fet99] C. Fetzer and F. Cristian, "A highly available local leader service," *IEEE Trans. on Software Engineering*, vol. 25, no. 5, Sep. 1999. http://www.cs.ucsd.edu/~cfetzer/HALL.

[Kop87] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," *IEEE Trans. Computers*, vol. 36, no. 8, pp. 933-940, Aug. 1987.

[Kop94] H. Kopetz and G. Grunsteidl, "TTP - A protocol for fault-tolerant real-time systems," *IEEE Computer*, pp. 14-23, Jan. 1994.

[Kop97] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, Norwell (MA), ISBN 0-7923-9894-7, 1997.

[Kro98] G. Kross, "The time-triggered communication protocol TTP/C," *Real-Time Magazine*, No. 4, pp. 100-101, 1998.

[Mur98] J. Murray, *Inside Microsoft Windows CE*, Microsoft Press, ISBN 1-5723-1854-6, 1998.