Schedulability Analysis for TMO-based Distributed Real-Time Embedded System

Yong Hoon Choi, Hyung Taek Lim and Chaedeok Lim
Electronics and Telecommunications Research Institute
{yonghoon, htlim, cdlim}@etri.re.kr

Abstract

The time-triggered message-triggered object (TMO) scheme is a general-style component programming scheme and supports design and implementation of real-time systems. In this paper, we present an approach to analyze schedulability for distributed real-time embedded systems based on TMOSM (Time-triggered Message-triggered Object Support Middleware) that enables the TMO scheme in general purpose operating systems.

1. Introduction

In distributed real-time software component design approaches, object oriented computing paradigm has been rapidly more and more prevalent with RT-CORBA, RT-UML, TMO, and so on. The TMO scheme [3], developed by the DREAM (Distributed Real-time Ever Available Microcomputing) laboratory at University of California, Irvine, is a powerful distributed real-time extension of the conventional object-oriented scheme. With help of its specialized middleware [6], the TMO structuring scheme can be implemented on various software/hardware platforms.

Recently many embedded systems have strict timing requirements in highly critical application domains such as nuclear plant control, air traffic control, and medical control. Also, they are often distributed with several processor and communication resources. Therefore, analyzing timing behavior of such systems is becoming increasingly important. Schedulability analysis for distributed real-time systems has been intensively studied. Most of the researches are, however, based on the traditional task model so that the application of schedulability analysis to real-world system models requires additional efforts.

In this paper, we introduce practical strategies for statically analyzing timing behaviors of TMO-structured distributed real-time embedded systems. The core parts of the timing analysis are to transform the TMO model to the traditional task model and to provide the worst-case response time for TMO elements, which have their deadline. The effects of the TMO scheme and its execution engine on the response time of such time-critical TMO elements are also considered. The response times are analyzed under the compound policy of a fixed priority preemptive scheduling and the RR scheduling.

This paper is organized as follows. Section 2 gives an overview of the TMO scheme. Section 3 shows the strategies for mapping from the TMO models into analyzable task models. In Section 4, the timing analysis scheme for the task model is presented. Section 5 offers our conclusions and future works.

2. Overview of the TMO Scheme

2.1. TMO Model

The Time-triggered Message-triggered Object (TMO) structuring scheme is a natural and syntactically small but semantically powerful extension of the conventional object-oriented design/implementation technique [3]. Figure 1 shows the structure of a TMO object in the TMO model. A TMO contains the following components.

- *ODS (Object Data Store)*: Storage of properties and states of the TMO object,
- *SpM (Spontaneous Method)*: A time triggered method which runs in real-time in a periodic manner,
- *SvM (Service Method)*: A message triggered method which responds to external service requests,
- *AAC (Autonomous Activation Condition)*: Activation condition for a SpM,
- *EAC (Environment Access Capabilities)*: List of gates to remote object methods.

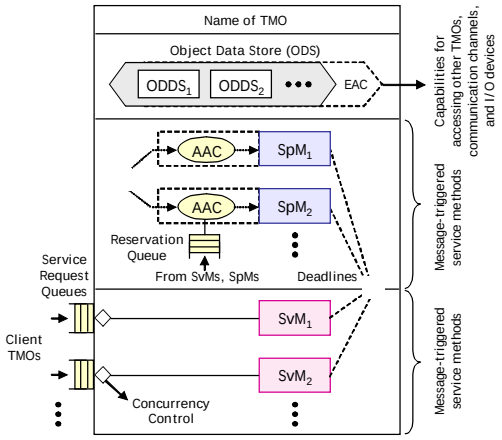

Figure 1. TMO model

2.2. TMO Execution Engine

Although a variety of execution engines in forms of kernel and middleware have been developed for supporting the TMO scheme, their basic structures are very similar [2] [4] [6]. In this paper, we used a TMO support middleware, named KelixRT, developed by the RTSE Lab at Konkuk University, since not only its source code but the source code of Linux where it runs are available. The availability of source codes makes it easier to analyze the worst case execution times (WCET) of the middleware and OS components that are essential data for schedulability analysis.

Figure 2 shows the structure of a KelixRT TMO system. KelixRT controls application threads with the help of Linux and consists of five middleware threads acting as follows. The timer handler thread is awakened periodically by Linux and wakes up the other middleware threads. WTMT1 (Watchdog Timer Management Thread) detects SpM threads to be executed in the next time slice and moves them to the ready queue in KelixRT. ICT (Incoming Communication Thread) delivers incoming messages to the corresponding SvM threads and moves the SvM threads to the ready queue. OCT (Outgoing Communication Thread) handles the messages to be sent to other TMOs. WTMT2 wakes up all threads in the ready queue. When all the middleware threads finish their job, they sleep until the next activation of timer handler.

2.3. System Model

The target system is a real-time embedded system distributed with Controller Area Network (CAN), extensively used in small scale distributed systems such as automobile and medical applications. The CAN contains an embedded networking bus that arbitrates between messages on the bus by using priorities. Each message in the CAN is tagged with a unique priority which serves to identify the message.

On each node, the TMO-structured system is deployed as shown in Figure 2. The combination of a fixed priority preemptive scheduling and the RR scheduling is used for middleware/application task scheduling. The priority ceiling protocol (PCP) is used for preventing priority inversions in task synchronization within the TMO.

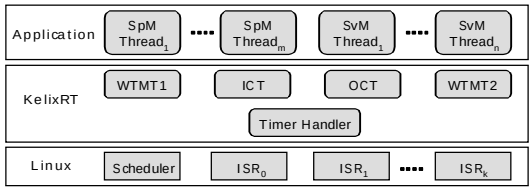

Figure 2. Structure of a KelixRT TMO system

3. Mapping from TMO Model to Task Model

3.1. TMO Support Middleware Tasks

All five middleware threads are periodic tasks. Each middleware thread is given a different static priority to ensure its execution order. The timer handler is given the highest priority, and WTMT1, ICT, OCT, and WTMT2 are given the next highest priorities in descending order.

Due to the fact that the priorities of the middleware threads are higher than one of any application thread, once the timer handler is activated, all the application threads are suspended until WTMT2 wakes up ready application threads and sleeps by itself. For simplicity of response time analysis, we consider the five middleware threads as a single task. The WCET of the KelixRT middleware is the sum of the WCETs of the five middleware threads and those of the four executions of the Linux scheduler.

3.2. TMO Application Tasks

In a first look, a SpM can be regarded as a single periodic task since it is triggered with a static inter-arrival time, it can be regarded as a single periodic task. Also, since a SvM is triggered by messages with a fixed minimum inter-arrival time, it can be regarded as a single sporadic task. Any SvM cannot have higher priority than the ones of SpMs in a TMO due to the basic concurrency constraint of the TMO model.

Allowing real-time objects to interact with each

other via service method invocation, however, may cause some significant change in such task models. Figure 3 gives the behavior of a service method invocation in TMO. A client task τ_c is divided into three subtasks: before request τ_{c1}, after request and before service return τ_{c2}, and after service return τ_{c3}.

There are two types of method invocations: blocking and non-blocking. In blocking call, the client waits until a result message is returned from the service method after calling a service method. If the client method τ_c does a non-blocking invocation, it has no the second sub-task τ_{c2}, that is, there is no parallel executions of the client method and the service method. In non-blocking call, the client waits until a result message is returned from the service method after calling a service method.

To enable the task precedence as in Figure 3, the priorities of the first two sub-tasks of the client task are given the same priority of the client task while the sub-task τc3 after service time is given a lower of priority over that of the service method. The response time of a client method τ_c that invokes a service method is the response time of its last sub-task τ_{c3}.

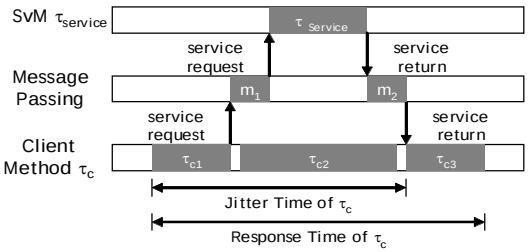

Figure 3. Service method invocation

4. Timing Analysis

Previous researches [1] [5] presented schedulability analysis for arbitrary deadlines on a single processor to one for a distributed hard real-time system where tasks with arbitrary deadlines communicating using various network protocol. This section presents an extension of those approaches to TMO-based systems.

4.1. Basic Processor Response Time Analysis

Under a fixed priority preemptive scheduling, for a task set τ of periodic and sporadic fixed priority tasks, the response time of a task τi can be calculated as follows:

$$hp_i = \{\tau_k \mid \tau_k \in \tau \wedge P_k < P_i\} \quad (1)$$

$$R_i = C_i + B_i + \sum_{\tau_k \in hp_i} \left(\left\lceil \frac{R_i}{T_k} \right\rceil C_k \right) \quad (2)$$

where R_i is the response time of τ_i, C_i is the worst case execution time of τ_i, hp_i is the set of higher priority tasks over τ_i, P_i is the priority of τ_i, B_i is the blocking time of τ_i, and T_i is the period of τ_i.

4.2. Task Synchronization & Activation Jitter

ODSS (Object Data Store Segment) is a set of shared resources accessed by SpMs and SvMs within the same TMO. If Priority Ceiling Protocol (PCP) is used for preventing priority inversion, we can receive the bounded blocking time for a task as follows.

$$lp_i = \{\tau_k \mid \tau_k \in \tau \wedge P_k > P_i\} \quad (3)$$

$$B_i = \max_{\substack{k \in lp_i \\ S \in locks(k,i)}} (C_{k,S}) \quad (4)$$

where B_i is the blocking time for task *i*, *locks(k,i)* is the set of Object Data Stores asccessed by task *k* with ceiling priorities higher than or equal to the task *i*'s priority, and lp_i is the set of lower priority tasks over τ_i. Let $C_{k,S}$ be the worst-case locking time of task *k* on Object Data Store *S*.

The LST (Latest Start Time) field in Autonomous Activation Condition represents the maximum release jitter time for SpM activation. The effect of the maximum delay J_i due to jitter is added to the response time of SpMs in the following manner.

$$R_i = C_i' + B_i + \sum_{\tau_k \in hp_i} \left(\left\lceil \frac{J_i + R_i}{T_k} \right\rceil C_k \right) \quad (5)$$

4.3. Round Robin Scheduling

Tasks with the same priority are scheduled by the RR scheduling policy. First, we need to calculate the response time of RR-scheduled tasks within the time window allocated to the same-priority tasks.

The response time B_i for task *i* only in a set of same priority tasks can be computed as follows:

$$\Phi_i = \{\tau_k \mid \tau_k \in \tau \wedge C_k \le C_i \wedge P_k = P_i\} \quad (6)$$

$$\Pi_i = \{\tau_k \mid \tau_k \in \tau \wedge C_k > C_i \wedge P_k = P_i\} \quad (7)$$

$$R_i = |\Pi_i| \times q_i \times \lceil C_i / q_i \rceil + \sum_{\tau_k \in \Phi_i} C_k \quad (8)$$

where q_i is the time quantum given by the RR scheduling policy, Φ_i are the equal or shorter execution tasks, and $\prod_i$ are the longer execution tasks.

Now we consider the response time analysis in the combination of a fixed priority preemptive scheduling and the RR scheduling. The same priority tasks, scheduled by the RR, acts like a single task with the priority in a fixed priority scheduling. The response time R_i for task *i* is computed as follows:

$$C_i' = |\Pi_i| \times q_i \times \lceil C_i / q_i \rceil + \sum_{\tau_k \in \Phi_i} C_k \quad (9)$$

$$R_i = C_i' + B_i + \sum_{\tau_k \in hp_i} \left(\left\lceil \frac{J_i + R_i}{T_k} \right\rceil C_k \right) \quad (10)$$

For the general case where the worst-case response time and deadline of a task may be more than its period, the response time equation is changed as follows:

$$w_i(q) = (q+1)C_i' + B_i + \sum_{\tau_k \in hp_i} \left(\left\lceil \frac{J_i + w_i(q)}{T_k} \right\rceil C_k \right) \quad (11)$$

$$R_i = \max_{q=0,1,2\ldots} \left(J_i + w_i(q) - qT_i \right) \quad (12)$$

4.5. Communication Schedulability Analysis

A service method invocation consists of three steps: sending a service request, processing the service, and returning the service result. To get the response time of the invocation, we need to compute the WCET of service request/return messages. The request/return messages contain object/method names of client/server methods, input/output parameters, deadline, and time stamp. The size of each message over the system can be computed from this information. The period and priority for messages are the same as those of the tasks that invoke them. The worst-case response time R_m of a given message m in a CAN field bus is analyzed as follows: [1]

$$R_m = \omega_m + C_m \quad (13)$$

$$\omega_m = B_m + \sum_{\forall j \in hp_m} \left\lceil \frac{\omega_m + J_j + \tau_{bit}}{T_j} \right\rceil C_j \quad (14)$$

$$B_m = \max_{\forall k \in lp_m} C_k \quad (15)$$

$$C_m = \left(\left\lfloor \frac{53 + 8s_m}{5} \right\rfloor + 66 + 8s_m \right) \tau_{bit} \quad (16)$$

where ω_m is the queuing delay, hp_m is the set of messages of higher priority over the message m, lp_m is the set of messages of lower priority over the message m, B_m is the longest blocking time of the message m by lp_m, T_j is the period of a message j, J_j is the queuing jitter of the message j, C_m is the upper bound of frame transmission time with maximum insertion of stuffed bits, s_m is the size of message m in bytes, and τ_{bit} is the time to transmit a single bit. 53 bits beside the data part are exposed to the bit stuffing mechanism. The maximum overhead of a message is 66 bits.

4.6. Holistic Schedulability Analysis

To compute the response time of TMO methods invoking a service method remotely in a distributed environment as in Figure 3, we need to integrate the processor and communications schedulability analysis. Based on the holistic schedulability analysis developed by Tindell and Clark [5], the response time of the client method τ_c is same as the response time of the last sub-task τ_{c1} of the client method with the following jitter:

$$J_{\tau_{c3}} = R_{\tau_{c1}} + \max(R_{m_1} + R_{\tau_{service}} + R_{m_2}, R_{\tau_{c2}}) \quad (17)$$

In case of non-blocking method invocation where the client method τ_c has the sub-task τ_{c2}, if the service result arrives before the completion of τ_{c2}, the response time of the client method consist of only the execution times of local sub-tasks: τ_{c1}, τ_{c2}, and τ_{c3}.

5. Conclusion and Future Works

In this paper, we presented a practical way to analyze timing behaviors of TMO-based distributed real-time embedded systems. The goal of the timing analysis is to allow real-time software engineers to easily ensure that a real-time system meets its timing requirements in an earlier development phase such as design and implementation. To achieve this goal, we are currently developing a timing analyzer plug-in for ESTO, an Eclipse-based integrated development environment for embedded real-time applications. We believe that the tool, coupled with a WCET analyzer, can provide a cost-effective way to verify timing behaviors of distributed real-time systems.

6. References

[1] Joao Rodrigues, etc., *"Schedulability Analysis of an Event-Based Real-Time Protocol Framework,"* Proc. of Int'l Workshop on Object-oritented Real-Time Dependable Systems, 1999.

[2] K.H. Kim, etc., *"A Timeliness-Guaranteed Kernel Model and Implementation Techniques,"* Proc of Int'l Workshop on Real-Time Computing Systems & Applications, Oct. 1995, pp.80-87.

[3] K.H. Kim, C. Subbaraman, *"Principles of Constructing a Timeliness-Guaranteed Kernel and Time-triggered Message-triggered Object Support Mechanisms,"* Proc. ISORC, Apr. 1998, pp.80-89.

[4] H.J. Kim, etc., *"TMO-Linux: a Linux-based Real-time Operating System Supporting Execution of TMOs,"* Proc. ISORC, May 2002, pp.288-294.

[5] K. Tindell, and J. Clark, *"Holistic Schedulability Analysis for Distributed Hard Real-Time Systems"*, Microprocessing & Microprogramming, Vol. 40, Issue 2-3, April 1994, pp.117-134.

[6] Kim, K.H., etc., *"An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation,"* Proc. ISORC, May 1999, pp.54-63.