Design and Implementation of a Run-time TMO Monitor on LTMOS

Yoon Seok Jeong, Tae Wan Kim, Chun Hyon Chang
Department of Computer Science and Engineering, Konkuk University, 1 Hwayangdong, Kwangin-gu, Seoul, 143-701, Korea
ysjeongkku@cse.konkuk.ac.kr, {twkim, chchang}@konkuk.ac.kr

Abstract

A real-time system should provide services on time for users. For this, it is needed to have a monitoring function. But monitoring itself causes some problems to influence real-time services. To solve these problem, we present the architecture for a run-time TMO monitor using DS(Data Store) and an implementation of a run-time TMO monitor which checks timing behavior of TMO(Time-triggered Message-triggered Object) programs running on LTMOS(Linux TMO system). The architecture for a run-time TMO monitor using DS(Data Store). DS gives some benefit as follows. First, DS separates a run-time TMO monitor from TMO programs providing real-time services. An independent monitor can minimize the influence to real-time TMO programs. Second, DS allows for any other applications not supporting TMO to access TMO method’s information. We present Data Store structure and DS API in LTMOS to support this new architecture and develop a run-time TMO monitor and a TMO program based on this architecture. The architecture presented in this paper can be used in all the areas which need real-time monitoring.

1. Introduction

Nowadays, real-time systems are used in various areas and needed to provide functions to process real-time data and control it. The main issue related to real-time systems is to provide a real-time monitoring function which observes if real-time processes work normally or not. For this, it is needed not post monitoring which depends on log data but run-time monitoring which can know the state of process at run-time. In short, real-time processes should be observed continuously by a run-time monitor.

In this paper, we present a run-time TMO monitor. A run-time TMO monitor which observes the operation of real-time processes at run-time is implemented on the LTMOS(Linux TMO System) supporting the TMO(Time-triggered Message-triggered Object) model on Linux. We also present the architecture with DS(Data Store) for a run-time TMO monitor and the structure of DS. This gives flexibility and efficiency to a monitor. In addition, we present DS API to support DS and the new architecture.

The rest of the paper is organized as follows. Chapter 2 summarizes the concept of TMO model and monitoring. In chapter 3 and 4, the design and implementation of a run-time TMO monitor on LTMOS and the architecture with DS for a run-time TMO monitor is described. In the last chapter, the conclusion of this paper and the future works are described.

2. Previous Works

2.1 TMO Model

The time-triggered message-triggered object(TMO) structuring scheme was established in early 1990’s with a concrete syntactic structure and execution semantics for economical reliable design and implementation of RT systems[1][2]. The basic TMO structure is depicted in figure 1. TMO is a syntactically minor and semantically powerful extension of the conventional object(s).

TMO is the extension of conventional real-time model. This model consists of four parts.

• EAC(Environment Access Capability) : List of gates to objects to providing efficient call-paths to remote object methods, logical communication channels, and I/O device interfaces. The EAC is responsible for the interface call of the communication channels and the I/O devices.
• ODS(Object Data Store) : Storage of properties and states of the TMO object. An ODS is a common object data store being accessed by the SpM and the SvM, and could not be accessed simultaneously.
• SpM(Spontaneous Method) : A time triggered method which runs in real-time a periodic manner.
• SvM(Service Method) : A message triggered method which responds to external service requests. The SvM has deadline itself.

2.2 LTMOS

LTMOS(Linux TMO System) is developed to support fundamental functions to run TMO objects on Linux. Figure 2 depicts the structure of LTMOS[3][4]. LTMOS operates on Linux kernel and controls the execution of TMO application based on TMO real-time object model. LTMOS can be divided into five layers.

• Layer 0 (Linux System Layer) : This layer provides Hardware control, real-time scheduling, multi threading and synchronous tool.
• Layer 1 (Broadcasting Platform and High Resolution Timer Layer) : This layer provides a underlying network communication platform to support broadcast between nodes. Soft timer interrupt handler with the 10msec-precision is supported in this layer.
• Layer 2 (System Task Layer) : This layer is the time-critical thread group activated by high resolution timer handler periodically and include like below. CST(Clock Synchronizat) synchronizes clocks of distributed nodes. WTMT(watch& TMO Management Thread) tasks charge of activating SpM’s in time by controlling the priority of each method. IMMT(Incoming Message Management Thread) and OMMT(Outgoing Message Management Thread) are for the message delivery among methods.
• Layer 3 (LSI(LTMOS Service Interface) & Object/Macro library) : This layer contains interface functions for distributed IPC primitives and management of SpM’s and SvM’s called by applications.
• Layer 4 : TMO application runs on this layer.
3. Design

3.1 Overall Architecture

Figure 4 depicts the overall system architecture for a run-time TMO monitor. There are four layers in this architecture. Shaded parts are developed for a run-time TMO monitor in this paper.

- **Application Layer**: this layer supports two types of applications. One is general applications which run on Linux kernel. The other is TMO applications running on LTMOS which supports real-time TMO programming.
- **Middleware Layer**: this layer provides fundamental functions to run TMO objects on Linux. There are some kinds of middleware which support TMO such as WTMOS and LTMOS. This paper uses LTMOS to implement a run-time monitoring which run on Linux.
- **OS Layer**: this Level supports fundamental function of OS.
- **Hardware Layer**: this paper just focuses a memory in all hardware. A memory is accessed to be read and written by a run-time TMO monitor and LTMOS.

3.2 Run-time TMO monitor

A basic function of monitor is to periodically monitor the operation of specific programs. This function consists of some sub functions. The run-time TMO monitor has four functions as depicted in figure 5. The shaded ovals in figure 5 represent functions which the run-time TMO monitor provides and solids arrows represent information flows during the execution of programs.

3.2.1 Accessing Data

This function is to gather the TMO method’s state data from Data Store. The process of accessing data in Data Store is depicted in figure 6. Data Store area is created when a run-time TMO monitor is started and it is deleted when a monitor is terminated. A run-time TMO monitor can access data after Data Store is created. Reading and writing data processes in dotted box in figure 6 are repeatedly operated whereas Creating/Opening and Closing/Deleting Data Store processes are executed just one time.

To access Data Store and get data, a run-time TMO monitor provides two modules, DS Connection Module and Data Reading Module. These modules are shaded parts in figure 7.
DS Connect Module provides a function to create and delete Data Store area. This paper makes two functions for managing Data Store. TMO methods’ information consists of method id, method type, execution time, left execution time, execution period and deadline. These data are stored in Data Store which is made for reading and writing TMO method’s state data.

The structure of Data Store is designed as depicted in Figure 8. It consists of Reference-Record, Write-Record and Meta data-Record. Reference-Record and Write-Record have a same schema. The reason why duplicate data is used is for data consistency. For example, the consistency will be lost if display application reads Write-Record when TMO application is writing on Write-Record. To prevent this problem, TMO application is only permitted to write on Write-Record and copy data to Reference-Record by whole record after TMO application completes writing. Other applications can access just Reference-Record and get data from it. This way can solve data inconsistency.

3.2.2 Analyzing Data

This function is to analyze data from Data Store. This includes correctness checking and performance evaluation as a data analysis function. Performance evaluation relates to check how fast the execution time of method is. Figure 9 depicts the mechanism of time checking for TMO method.

On the contrary, correctness checking relates to checking if a TMO method keeps the deadline. In real-time systems, the most important thing is to serve the request on time. So a run-time TMO monitor should focus correctness checking with the deadline. Figure 10 depicts the mechanism of correctness checking for TMO method.

3.2.3 Displaying Data

This function is to display TMO method’s data. This paper has display part separated from data generating part. If these two parts are integrated into one application, the display part influence the operation of data generating part. This is a crucial issue because TMO should serve real-time services for users on time. Figure 11 depicts this separation model. This paper makes a run-time TMO monitor as a display part and TMO application as a data generating part. This paper implements Data Store to deliver data from data generation part to display part.

Figure 7. Structure of the Run-time TMO monitor

Figure 8. Structure of Data Store

Figure 9. Execution time checking Mechanism for performance evaluation

Figure 10. Comparing execution time to deadline for Correctness Checking

Figure 11. Separation Display part from Data Generation part
drawing the graph of execution time of TMO methods, the other is a panel displaying texts about method information.

![DS Connection Module] Data Display Module
| Dynamic Generation Module |
| Data Reading Module |

**Figure 12. Structure of Run-time TMO Monitor focusing Displaying Data function**

Dynamic Generation Module is for dynamically creating components on the panel for each method. All the components relating to displaying should be created dynamically because the number TMO objects and methods is not known.

### 3.2.4 Reporting Violation

This function is to inform users or system manager that a specific event occurs. This function relates to Event Interpretation Specification and Action Specification. In this paper, Event Interpretation Specification defines the state of method running on the system. If the state of method matches the specification, the run-time TMO monitor executes an Action to manage the abnormal state of method. In this paper, the Event Interpretation specification has just one rule which is to check if the method violates deadline of method or not. An Action depends on the Action specification defined by users. This also relates to Correctness Checking.

- **Violation Event**: The method violates the deadline of TMO method.

In this paper, Action specification has one rule which is to send an alarm to a user when the method violates the Event Interpretation specification.

- **Violation Action**: A TMO monitor sends an alarm to a user

Figure 13 depicts the process of Reporting Violation.

![Event Occurring]

**Figure 13. Process of Reporting Violation**

### 3.3 TMO Application for simulation

Figure 14 depicts the structure of TMO application. It operates on a single node and it has a SpM and a SvM method. SpM method gives a function to periodically increase the global variable and call SvM method. SvM method has a function to print out the global variable. The global variable, counter, is included in Counter Segment of ODS. This is an integer variable and has 0 as the initial value. This is increased by SpM method.

![TMO Application]

**Figure 14. Structure of TMO application for Simulation**

### 3.4 LTMOS with DS API

In figure 15, some APIs are added to the layer 3, LSI & Object/Macro library, to implement a run-time TMO Monitor on LTMOS architecture as depicted in figure 2.

![TMO Application]

**Figure 15. Architecture of LTMOS with DS API**

The example to use DS API is like below. DS APIs is created for providing functions which create, delete
Data Store and write data in it. OpenSM() are inserted into LTMOS kernel’s creator and make Data Store area. CloseSM() are inserted into TMOExit() to close and delete Data Store area. The function for Method registering uses WriteSM() to write method’s information on Data Store.

```
kern::kernel() {
    ... 
    LSI.OpenSM();
    ...
}
void CLTMOSServiceInterface::SpM_Register (int peri, int dead, int start, int stop) {
    ... 
    WriteSM(pMCB->nMethodId,
            METHOD_SpM, deadline, 0, period, 0);
    ... 
    void TMOExit () {
        LSI.CloseSM();
    }
```

4. Implementation

The architecture and run-time monitor explained in the previous section have been implemented as shown in Figure 16, 17. Figure 16 displays the monitoring result of SvM method at current. A right-side box shows method’s information. The red line on the white panel shows the execution time of method and the blue line displays deadline. Memo box at the bottom displays the alarm message for a user when a method violates the Event Interpretation Specification and Action Specification. In this paper, Event Interpretation Specification is defined to violate the method’s deadline and Action Specification is to inform a user of an alarm. If the red line is above the blue line, it causes a specific event and action to manage this event. A bar in the middle of run-time TMO Monitor shows the percentage about execution time comparing to the deadline.

Figure 17 displays the monitored result of SpM method at current. A right-side box also shows method’s information.

Figure 18 shows a run-time TMO monitor with full options. It can be divided into three parts. The part at the top is for TMO monitor and the middle part is for a system monitor. Moreover, the violation report function was added to a run-time monitor. The Memo box at the bottom is made for Action Execution.

In figure 19, alarm information is written on the memo box. It occurs when the method’s execution time is over the deadline. In the specific area, a red line is above a blue one as shown in the run-time TMO monitor’s panel. This means that a violation event occurs. At the same time, the state of CPU usage is increasing extraordinarily. This means that system’s state influences the action of the TMO program.

```
Figure 16. Screen for SvM method
```

```
Figure 17. Screen for SpM method
```

```
Fig. 18 Screen for integrated Monitor
```
Figure 19. Execution of TMO Application

Figure 19 shows the snapshot of TMO program. TMO program consists of a TMO object which contains a SvM and a SpM method. SpM is called every 1 second to increase the value of counter and call SvM method. When a SvM method is called, it prints the value of counter variable.

4. Conclusion and Future Work

We have designed and implemented the new architecture with Data Store to support a run-time TMO monitor. The architecture was implemented on LTMOS which operates on Linux kernel. This new architecture with Data Store gives flexibility to a monitor by providing Data Store scheme. Data Store which we made plays a role of connecting TMO application to a monitor application. Therefore, any other applications which want to monitor TMO method can access method’s information through Data Store. The architecture also gives efficiency to a monitor by separating a monitor from the target TMO program. The separation reduces the influence caused by displaying data to a TMO program. We also presented a run-time TMO monitor and TMO application. And we provide DS API to support run-time TMO monitoring and programmers who want to make a TMO program.

The future work is to add the technique of system monitoring to a run-time monitor. A TMO program is influenced according to the state of resource of system. Without considering system’s state, it is hard to provide the exact monitoring. By adding this system monitoring to a run-time TMO monitor, we can provide more exact monitored results.

References


