A Middleware Layer Monitoring Structure for the Real-time Middleware

Yoon-Seok Jeong¹, Tae-Wan Kim²*, and Chun-Hyon Chang³
Department of Computer Science and Engineering, Konkuk University, Seoul, Korea
(Tel : +82-2-450-3539; E-mail: {ysjeong¹, twkim², chchang³@konkuk.ac.kr)

Abstract: This paper proposes MMS (TMO model-based Middleware Monitoring Structure), a structure for monitoring the middleware. MMS can be an infrastructure to check the performance and stability of the middleware, and to guarantee the reliable operation of the TMO applications by managing the operational status of the middleware threads. Moreover, this paper implements a middleware monitor as a supporting tool for developers to monitor the MMS-instrumented middleware. Thus, developer can manage the real-time system more stably, using the MMS-instrumented middleware and the middleware monitor proposed in this paper.

Keywords: TMO model-based Middleware Monitoring Structure, Middleware Monitor, Real-time Middleware

1. INTRODUCTION

Systems used for industrial purposes should feature real-time properties to respond rapidly to changes in the operational status of related devices. To secure high responsiveness, real-time systems are configured by mounting real-time operating systems. However, real-time operating systems have drawbacks in that they are expensive and have low compatibility with the manifold hardware environments. Thus, mounting real-time middleware on general operating systems has been considered.

The Time-Triggered Message-Triggered Object (TMO) model-based middleware can be an alternative in a choice of the real-time middleware to be used for industrial purposes. The TMO middleware can guarantee that systems in distributed environments operate in a reliable manner through elements such as real-time processes, a structure for data sharing and real-time logical multicast channels [3, 4, 5].

However, the TMO middleware should be instrumented with supporting functions such as real-time monitoring to be utilized in industrial areas. Thus, this paper proposes MMS (TMO model-based Middleware Monitoring Structure), a monitoring infrastructure configured within the middleware.

2. TMO MODEL

The proposed scheme in this paper is modeled in such a way as to utilize the proven functions provided by the TMO model. The TMO structuring scheme was established in the early 1990’s with a concrete syntactic structure and execution semantics for economical and reliable design along with an implementation of real-time systems [3, 4, 5]. TMO is a syntactically minor and semantically powerful extension of the conventional object(s). As depicted in Fig. 1, the basic TMO structure consists of four parts.

Fig. 1 The basic structure of the TMO model

- Spontaneous Method (SpM): A new type of method. A SpM is triggered when the real-time clock reaches specific values determined at design time. The SpM has an AAC (Autonomous Activation Condition), which is a specification of the time-windows for execution of the SpM.
- Service Method (SvM): A conventional service method. An SvM is triggered by service request messages from clients.
- Object Data Store (ODS): The basic units of storage which can be exclusively accessed by a certain TMO method at any given time or shared among TMO methods (SpMs or SvMs).
- Environment Access Capability (EAC): The list of entry points to remote object methods, logical communication channels, and I/O device interfaces.

3. TMO MODEL FOR MONITORING

Fig. 2 shows the TMO model for monitoring that supports monitoring functions [6]. Several elements are added to this model to enable the monitoring concept to

This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the ITA (Institute of Information Technology Assessment)

*Corresponding Author
be applied to the TMO model [3, 4, 5]. Basically, corresponding elements are designed so that they are automatically created and activated by the middleware.

- **Monitoring ODS (MODS):** Refers to the storage for sharing the results of monitoring TMO objects. It is assigned to each TMO object by a unit of MODS Segment (MODSS).
- **Monitoring Dedicated Method (MDM):** Refers to the dedicated monitoring SpM method that is activated periodically in the same manner as general SpMs and transfers data in the data store at periodic intervals.
- **Monitoring Dedicated Channel (MDC):** Refers to the dedicated channel used for the transfer of monitoring data among TMO systems.
- **Control Dedicated Channel (CDC):** Refers to the dedicated channel used for receiving control input data from external systems or external applications.

**Fig. 2 The structure of the TMO model that supports monitoring functions**

We use this model as the basic TMO platform for designing and implementing the monitoring structure for the middleware.

### 4. TMO-BASED MIDDLEWARE MONITORING STRUCTURE

#### 4.1 Monitoring Structure

MMS is a structure for monitoring the middleware. MMS, which is configured as a monitoring infrastructure within the middleware, runs functions to observe the middleware. It is made up by two components as depicted in Fig 3: thread sensors, which are inserted in the middleware threads and activated in a periodic manner, for monitoring threads and storing the operational data from them in the data store; a data store outside the middleware, which is designed to permit general applications to access it whereas the internal one, MCB(Method Control Block) does not provide a way to be accessed. The data store is called JMA (Joint Memory Area).

**Fig. 3 The interlocking structure between elements for the middleware monitoring in the middleware**

Thread sensors gather the status data of the middleware threads. The Linux-based middleware was made to have three threads (this paper does not treat CST(Clock Synchronization Task)) [2]: WMTM(Watchdog & TMO Management Thread), which is activated in a period manner and calls TMO methods, namely, SpMs or SvMs, based on the defined timing conditions; IMMTH(Incoming Message Management Thread) and OMMTH(Outgoing Message Management Thread), which manage messages sent and received by TMO methods. These two threads are included in MCS(Message Communication Systems) as shown in Fig. 4;

**Fig. 4 The hierarchical structure of LTMOS**

To evaluate the performance or the robustness of the middleware in a certain environment, the threads, which are the essential elements of the middleware, should be checked and analyzed. In short, the thread sensors in the proposed MMS monitor the middleware threads, and gather data such as the activation time of the threads. Fig. 5 depicts the basic structure of JMA. The reason why we introduce this data store, JMA, is to detach a non real-time task with high load from the TMO applications and to allot the task to general applications. This needs a data store which is accessible whenever the general applications try to get the monitoring data. Thus, it is designed to make both the TMO applications and
the general ones access to it, using the shared memory.

JMA consists of four sub areas: a feedback data area, which keeps the control signals sent by general applications; an overall data area for TMO applications, which holds data related to not methods but a whole TMO object; a system data area, which stores data collected by a system monitor, and which is filled with the system performance data; and lastly, a middleware area, which has the middleware-related data gathered by the thread sensors. In this paper, we focus on just the middleware area.

![Feedback Data Area, TMO Overall Data Area, System Data Area, Middleware Area](image)

Fig. 5 The basic structure of JMA

4.2 Mechanisms

The elements for the middleware monitoring stated in the section 4.1 run based on the following mechanisms proposed in this paper.

A Monitoring Mechanism

The thread sensors are activated every 1 second while the middleware threads do every 10ms. Therefore, monitoring is conducted based on not tracing but sampling as shown in Fig. 6 [1]. This result from the fact that monitoring functions may disturb the middleware in the operation whenever the monitoring-related works, such as sensing and data storing, take longer than the operation of threads does.

![Monitoring Time Slot, SAMPLING, TRACING](image)

Fig. 6 The concept of sampling and tracing

A Performance Quantifying Mechanism

The monitoring data kept in JMA is used for the analysis of the middleware performance. This paper uses the performance level technique to quantify the middleware performance [7]. The performance areas are determined in Table 1, considering the difference between the activation interval of the middleware threads, which is 10ms, and the actual activation: For example, the green area that has the difference less than 1ms. Then, the weighted value, namely, the performance level is given to each area. Finally, at the monitoring phase, the gathered data are quantified according to the performance level.

<table>
<thead>
<tr>
<th>Level</th>
<th>Condition</th>
<th>PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green(Normal)</td>
<td>activation interval &lt; 1ms</td>
<td>2</td>
</tr>
<tr>
<td>Yellow(Alert)</td>
<td>1ms &lt; activation interval &lt; 10ms</td>
<td>1</td>
</tr>
<tr>
<td>Red(Critical)</td>
<td>10ms &lt; activation interval</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1 Performance Level

A Performance Evaluation Mechanism

This paper proposes the MP (Middleware Performance), a time-accumulated metric to represent the middleware performance during the specific period as only one value. The MP metric has the value between 0 and 1. It is possible to state that the closer the value is to 1, the more normal the middleware performance is. Let \( n, m, v \) and \( PL_{ij} \) be the number of threads, the number of sampling during the period, the maximum weighted value, and the performance level of the thread \( i \) at the \( j \)-th sampling time respectively. The equation to calculate the middleware performance is denoted as follows.

\[
MP = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{1}{m} \sum_{j=1}^{m} PL_{ij} \right)
\]

5. IMPLEMENTATION

We implemented the MMS and the proposed mechanisms in LTMOS (Linux TMO System), a TMO model-based middleware which runs on Linux [2]. First, we made the thread sensors in the middleware. And we implemented the JMA and management methods for it. JMA is constructed physically based on the shared memory. The shared memory provides a way to access data positioned in the specific memory address, without any additional processing, and makes data shared with other applications. Table 2 shows the methods to manage JMA built on the shared memory.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMA_Open()</td>
<td>It creates the shared memory.</td>
</tr>
<tr>
<td>JMA_Close()</td>
<td>It deletes the shared memory.</td>
</tr>
<tr>
<td>JMA_Write(int NodeId, void*Value)</td>
<td>It writes a value at the specific address in the shared memory.</td>
</tr>
<tr>
<td>JMA_GetItem(int NodeId)</td>
<td>It returns the value written in the shared memory.</td>
</tr>
</tbody>
</table>

Moreover, we implemented a middleware monitor as
a supporting tool for developers to observe the MMS-instrumented middleware as depicted in Fig. 7. In fact, the middleware monitor is a kind of general applications illustrated in Fig. 3. The monitor accesses JMA directly, and gets the monitoring data of the middleware in it.

In Fig. 7, there are three tabs for the threads on the left panel. Each tab draws variations of each thread, and displays the intervals between actual activations in the middle panel. A panel at the bottom of center shows the performance level of the specific middleware thread. In Fig. 7, the performance level is in the green area. This means the thread runs in a normal manner. Lastly, a value on the panel at the left side states the robustness of the whole TMO middleware. This value comes from calculation based on MP proposed in this paper. The value of the robustness in Fig. 7 is 1 which means that the middleware has been running concretely and robustly from the startup of the middleware to this point. Therefore, we can assure that the operation of the middleware is stable and reliable, with the value of the time-accumulated metric.

**Fig. 7 The snapshot of the middleware monitor based on MMS**

**6. CONCLUSION**

In conclusion, this paper proposed MMS, a monitoring structure for the real-time middleware. MMS can be an infrastructure to keep a check on the performance and stability of the middleware, and moreover, to guarantee the reliable operation of the TMO applications indirectly by observing and managing the operational status of the essential middleware threads. Thus, developer can manage the real-time system more stably, using the MMS-instrumented middleware and the middleware monitor proposed in this paper.

In the future, we will study to combine an analysis module which analyzes the status of the middleware dynamically and controls it, based on the analyzed data, into the middleware monitoring infrastructure, MMS.

**REFERENCES**


