

A GUI Based Aid for Generation of Code-Frameworks of TMOs

S. J. Kang and K. H. (Kane) Kim
DREAM Lab
UCI
{seokjook, khkim}@uci.edu

Position Statement

1. Introduction

In recent years, there have been rapid growths in interests of industry in engineering of distributed real-time embedded-computing (DRE) systems. While improved methods for engineering highly reliable DRE systems appear regularly these days, the state of the art remains inadequate for dealing with large-scale challenging DRE systems, which are under increasing demands from societies. Main issues are the ease of programming such applications and the reliability of such application systems.

The first co-author has been advocating pursuit of a new design paradigm called the *general-form timeliness-guaranteed (GT) design* paradigm. The essence of the GT design paradigm is to realize distributed real-time (RT) computing in a general form not alienating the main stream computing industry and yet allowing system engineers to confidently produce certifiable distributed RT systems.

A concrete approach formulated by the first co-author together with his research collaborators for facilitating the GT design paradigm is called the TMO (*Time-triggered Message-triggered Object*) structuring scheme. On one hand, TMO is meant to be the basic building-block for distributed RT programs. On the other hand, TMO structuring supports various phases of system engineering, from abstract designs to detailed implementations. Moreover, TMO is an effective mechanism not only for variable-degree abstraction of DRE systems under design but also for variable-accuracy simulation of the application environments.

We have been enabling TMO programming without creating any new language or compiler. Instead, an API that wraps around the execution support services of a middleware named the *TMO Support Middleware (TMOSM)* was provided. Also, we built some macros and have been working toward expanding the set of macros to simply the jobs of TMO programmers. However, we recently realized that a different approach would be more effective. That is, the approach of providing a GUI tool for interactive design of TMO-network structures and automated generation of code-frameworks, looked much more promising. Indeed, our research and development in this direction during the past year has revealed that this GUI-based code-framework generation approach is much more cost-effective than the macro-based approach.

The GUI tool that is evolving in the authors' lab has been named the Visual Studio for TMO (ViSTMO). It is a graphic-design-oriented tool that helps TMO designers and programmers to build TMO network applications efficiently.

2. TMO Basic Structure

The TMO scheme was established in early 1990's with a concrete syntactic structure and execution semantics for economical reliable design and implementation of RT systems.

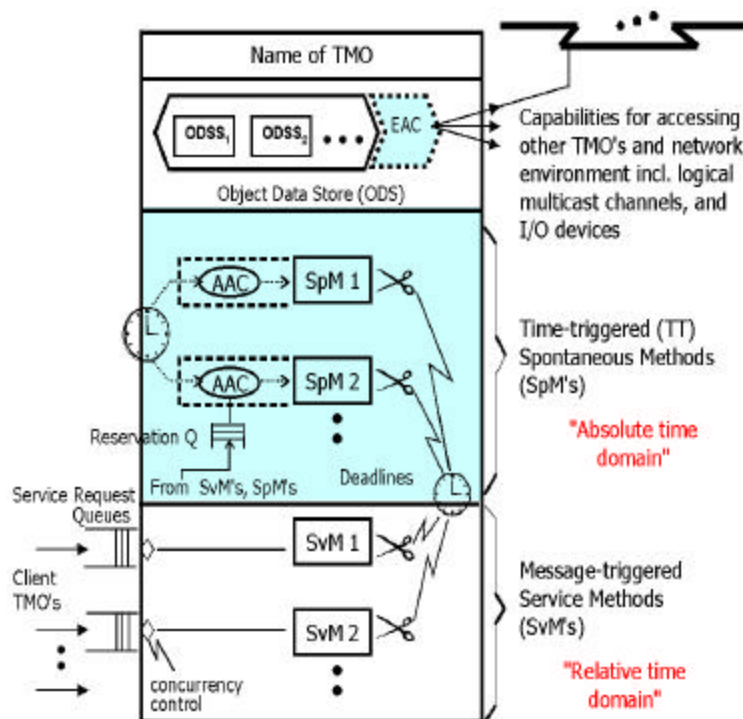


Figure 1. *The Basic Structure of TMO (Time-triggered Message-triggered Object)*

TMO is a natural and syntactically minor but semantically powerful extension of the conventional object(s). As depicted in Figure 1, the basic TMO structure consists of four parts:

- ODS-sec** = object-data-store section: list of *objectdata-store segments* (ODSS's);
- EAC-sec** = *environment access-capability* section: list of *gate* objects providing efficient call-paths to remote object methods, logical communication channels, and I/O device interfaces;
- SpM-sec** = *spontaneous-method* section: list of *spontaneous methods*;
- SvM-sec** = *service-method* section.

Major features are summarized below.

- (a) *Distributed computing component*: The TMO is a distributed computing component and thus TMOs distributed over multiple nodes may interact via *remote method calls*. To maximize the concurrency in execution of client methods in one node and server methods in the same node or different nodes, client methods are allowed to make *non-blocking* types of service requests to server methods.
- (b) *Clear separation between two types of methods*: The TMO may contain two types of methods, *timetriggered (TT-) methods* (also called the *spontaneous methods* or *SpMs*), which are clearly separated from the conventional *service methods (SvMs)*. The SpM executions are triggered upon reaching of the RT clock at specific values determined at the design time whereas the SvM executions are triggered by calls from clients which are transmitted by the execution engine in the form of service request messages calls. Moreover, actions to be taken at real times, which can be determined at the design time, can appear only in SpMs.
- (c) *Basic concurrency constraint (BCC)*: This rule prevents potential conflicts between SpMs and SvMs and reduces the designer's efforts in guaranteeing timely service capabilities of TMOs. Basically, *activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place*. An SvM is allowed to execute only when an execution time-window big enough for the SvM that does not overlap with the execution time-window of any SpM that accesses the same ODSSs to be accessed by the SvM, opens up. However, the BCC does not stand in the way of either concurrent SpM executions or concurrent SvM executions.
- (d) *Guaranteed completion time and deadline*: The TMO incorporates deadlines in the most general form. Basically, for output actions and method completions of a TMO, the designer guarantees and advertises execution time-windows bounded by start times and completion times.

Triggering times for SpMs must be fully specified as constants during the design time. Those real-time constants appear in the first clause of an SpM specification called the *autonomous activation condition (AAC)* section. An example of an AAC is "for t = from 10am to 10:50am every 30min start-during (t, t+5min) finish-by t+10min" which has the same effect as {"start-during (10am, 10:05am) finish-by 10:10am", "start-during (10:30am, 10:35am) finish-by 10:40am" }

3. Illustrations of ViSTMO

ViSTMO provides a *graphics-based design editor* that supports interactive design of application TMO networks and a *code-framework generator* that produces a framework of each application TMO designed with the aid of the graphics-based design editor. The major components and functionality of ViSTMO are shown in the Figure 2.

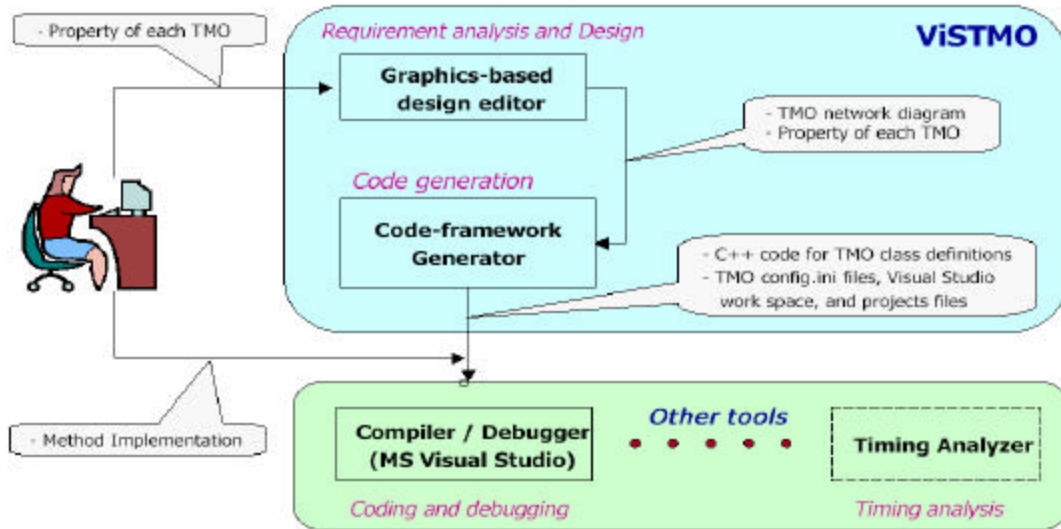


Figure 2. Major components and functionality of ViSTMO

The graphics-based design editor of ViSTMO provides user-friendly GUIs for TMO network application programmers. Only thing the programmers should do is just to fill out the empty fields on dialog boxes. From the input by a programmer, the ViSTMO can check possible errors and report them to the programmer.

The creation of an SpM is a good example that demonstrates how much ViSTMO can help the programmers. Figure 3. shows a dialog box for adding a SpM to a TMO class.

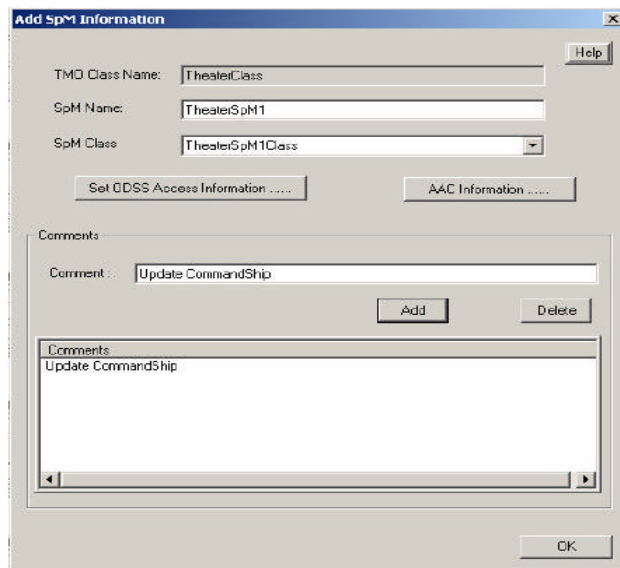


Figure 3. A Dialog box for adding a SpM to a TMO

In order to add a SpM to a TMO class, a programmer should provide the name of SpM, the name of SpM class, ODSS access information, AAC information, and comments that describes the behavior of the SpM.

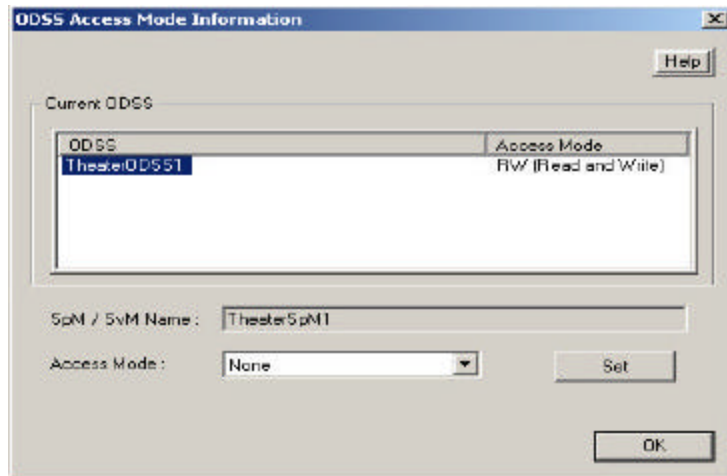


Figure 4. A Dialog box for setting ODSS access mode

Figure 4 shows how the programmer can set the ODSS access mode (e.g., Read only or Read and Write) of the SpM.

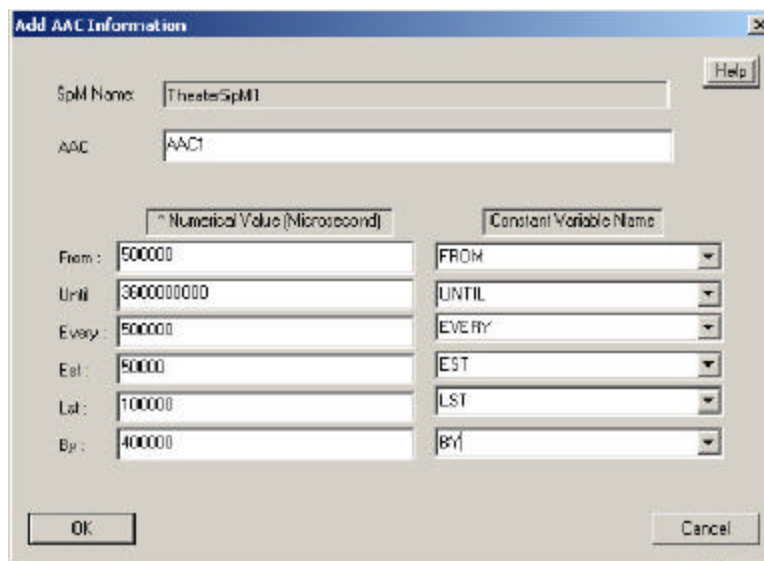


Figure 5. A Dialog box for setting AAC

And Figure 5 shows a dialog box for specifying the timing parameters, AAC. The programmer should give numerical values in microsecond unit for the timing specification of an SpM. Also, the programmer can assign an optional symbolic constant for each numerical value. The timing specification in Figure 5 can be translated to the following,

“for t = from SYSTEM_START_TIME + 500,000 to SYSTEM_START_TIME + 3,600,000,000 every 500,000 start-during (t+50,000, t+100,000) finish-by t+400,000”
 ※ Time Unit : Micro second.

Using the information provided through the dialog boxes in Figure 3, 4, and 5, the code-framework generator will generate the class definition and the constructor of the SpM.

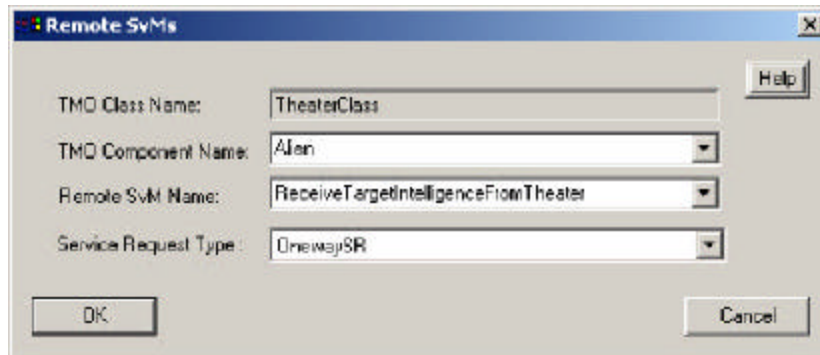


Figure 6. A Dialog box for a remote SvM call

The code-framework generator can generate a connection code for a remote SvM call after a programmer inputs the server TMO name, the SvM name, and the type of the service request (SR) using the dialog box depicted in Figure 6. Currently TMOSM supports 8 types of service requests. They include BlockingSR (type 1 and 2), NonBlockingSR, OnewaySR, NonBlockingGetResultOfNonBlockingSR, BlockingGetResultOfNonBlockingSR (type 1 and 2), BlockingGetResultOfNonBlockingSR, and ClientTransferSR.

Currently, the research on ViSTMO is in the user interface design phase. DREAM Lab. is working on the first prototype that amply demonstrates the feasibility and potential benefits of both the graphics-based design editor and the code-framework generator. During this phase, the major functionalities will be implemented and evaluated.

What we learned and the directions of our current and future work in this area will be discussed.