Design and Implementation of Time Management Service for IEEE 1516 HLA/RTI

Jeong Hee Hong*, Jae Hyun Kim and Tag Gon Kim*****
School of EECS
KAIST
373-1 Guseong-dong, Yuseong-gu
Daejeon, Korea 305-701
Tel: +82-42-869-3454
Fax: +82-42-869-8054
jhhong@smslab.kaist.ac.kr*, jhkim@smslab.kaist.ac.kr, tkim@ee.kaist.ac.kr*****

Keywords: IEEE 1516, HLA, RTI, Time Management, Modular design, GALT calculation algorithm

Abstract

With the development of time management service of Run-Time Infrastructure (RTI), it is necessary to consider an efficient design approach and an algorithm of Greatest Available Logical Time (GALT) computation. However, many time management services of existing RTIs have difficult in modification and extension. Although some RTIs avoid this difficulty through modular design, they comply with not IEEE 1516 HLA/RTI but HLA 1.3. In addition, a lot of RTIs made use of well-known Mattern's algorithm for GALT computation. However, Mattern's algorithm has a few limitations for applying to HLA/RTI.

This paper proposes a modular design and an implementation of time management service for IEEE 1516 HLA/RTI. We divided the time management service module into two sub-modules: a TIME module and a GALT module and used Mattern's algorithm improved for IEEE 1516 HLA/RTI. The paper also contains several experimental results in order to evaluate our time management service module.

1 INTRODUCTION

The HLA is the IEEE 1516 standard for interoperation between heterogeneous simulators which are developed with different languages and platforms. The Run-time Infrastructure (RTI) is a software implementing the IEEE 1516.1 Federate Interface Specification [2].

In the RTI, time management service is an important part for simulation time synchronization among simulators. The Time management service determines federate policies and manages logical time advancement of each federate. There are two important policies, time regulating and constrained. The time regulating federates can generate Time Stamp Order (TSO) messages. In turns, time constrained federates receive TSO messages, and process them in a non-decreasing time order. To maintain time causality, the time constrained federates are not allowed to advance their local logical time lager than the current Greatest Available Logical Time (GALT). GALT is defined minimum of federate's time plus lookahead. In other words, GALT is the maximum logical time that a federate may advance securely.

The HLA does not provide a specific implementation, and mandate the use of any particular programming languages. Currently many commercial RTIs [4] have been released but their design is not fully exposed. In addition, the time management services of open-RTIs are difficult to modify and to extend their RTIs since they are embedded in their RTIs. In order to avoid these drawbacks, a modular design approach is suitable for developing an RTI. Modularity enhances modifiability, maintainability and reusability. To our best knowledge, no such time management services with a modular architecture exist. Therefore, we have developed an RTI called *SMSRTI* [7] that is based on the IEEE 1516 standard for research on HLA/RTI.

This paper introduces design and implementation of a time management service module. When implementing the time management service, there are two important issues: an algorithm of GALT computation and an efficient design approach.

First of all, the GALT calculation algorithm has considerable influence on the performance of the time management service. We utilized well-known Mattern's algorithm used by many other RTIs. We also improved Matten's algorithm which has some limitations for IEEE 1516 HLA/RTI.

Second, our time management service of SMSRTI with modular architecture takes also a modular design framework for modeling RTI efficiently. We divided the time management service module into two sub-modules: a TIME

module and a GALT module. These two sub-modules encourage reusability, modifiability and maintainability within SMSRTI through modular design with explicit interfaces of their own. When applying and comparing with various GALT computation algorithms, our time management module minimizes modification by modifying only GALT module instead of whole time management service module. Since two sub-modules of the time management service module perform independently, it promotes reuse of the TIME module.

The paper is organized as follows: Section 2 briefly reviews High Level Architecture. The design of time management service is described in Section 3. The GALT calculation algorithm for time management service is addressed in Section 4. Section 5 shows experimental results. Finally Section 6 concludes the paper.

2 HIGH LEVEL ARCHITECTURE: BRIEF REVIEW

2.1 High Level Architecture

The High Level Architecture (HLA) is a specification for interoperation among distributed heterogeneous simulations. The HLA specification is approved as IEEE 1516 standard in 2000. The standard consists of two pars: Federate Interface Specification and Object Model Template (OMT).

Federate Interface Specification defines essential services of 6 categories. These services are also listed as a form of Application Programming Interfaces (APIs) written in various programming languages. The Run-Time Infrastructure (RTI) is an implementation of HLA Federate Interface Specification. The HLA only defines APIs and lefts detailed algorithms to RTI software developers.

OMT is a documentation format for shared objects and message types by which simulators exchange events. OMT is used to describe Federation Object Model (FOM) and Simulation Object Model (SOM). The FOM is a collection of all shared objects and interactions defined inside whole distributed simulation, i.e. federation. The SOM is a list of shared objects and interactions that a simulator, i.e. federate, sends to or receives from other simulators.

Three major functionalities are necessary to accomplish simulation interoperation; time synchronization, data exchange, and simulation control/management. Federate Interface Specification refines these functionalities into 6 management groups.

The HLA/RTI provides explicit time management services, while traditional parallel and distributed time synchronization algorithms manage the simulation time of a local process with only input events form other processes. The HLA/RTI has a list of time services that enable a federate to facilitate conservative or optimistic time synchronization methodology. Time management services are explained in detail in the following sections.

A federate is able to use two communication methodologies for exchanging data with other federates: shared objects and interactions. There exist, however, no physically shared objects inside a federation because each federate is a modular and distributed across the network. In a modular system, a component should notify the modification of its interval variable to be shared with other components whenever the variable is changed. To implement non-modular components in a modular system, environment, the shared objects need to be declared, first, in the FOM and to be managed by lists of notification functions in the HLA.

Finally, the HLA supports control functions to organize a federation, miscellaneous functions for synchronization points, save and restoration, monitoring and management functionality by Management Object Model (MOM).

2.2 Time Management Service

The HLA time management services are highly parameterized to support the wide variety of Time Stamp Order (TSO) synchronization requirements of federates. The federate, first, declare their policies and then, they utilize the time management services to control simulation time advances and TSO message delivery.

A federate can declare its policy by setting its time regulating and time constrained flag. A federate whose time re regulating is turned on can send TSO messages and hence can prevent other federates from advancing their simulation time. A federate whose time constrained flag is turned on can receive TSO messages and hence can be constrained by other time regulating federates. Each TSO message is tagged with a timestamp by the sending federate, time regulating federate, and is delivered to the receiving federate, time constrained federate, in the order of non-decreasing timestamps.

The RTI guarantees a federate will not receive any TSO messages with time stamp less than its logical time. A federate should explicitly request for the advancement of its logical time. The RTI grants the time advancement of the federate by GALT value. GALT is explained in detail in Section 3. There are three different services to request for advancing of logical time: *Time Advance Request* (TAR)/*Time Advance Request Available* (TARA), *Next Message Request* (NMR)/*Next Message Request Available* (NMRA) and *Flush Queue Request* (FQR). Some federates that internally employ a time-stepped mechanism use TAR. NMR is suited for the event-driven mechanism. FQR is preferred for the optimistically synchronized federates.

3 GALT CALCULATION ALGORITHM

GALT is the maximum logical time that a federate may advance securely. GALT is defined as follows:

GALT = min{ federate i's time + lookahead }

GALT is also called Lower Bound Time Stamp (LBTS) in HLA 1.3 [8]. The computation of GALT is essential for implementing time management service.

A GALT calculation algorithm should solve a key problem called the transient message problem. Here, the

transient message problem means there is any message which has been sent but has not yet received at its destinations [9]. Thus, the GALT calculation algorithm must account for all messages sent and received.

There are a lot of algorithms for resolving transient message problem. Samadi's algorithm considers these messages using acknowledgement messages [10]. Messages are accounted for by counting messages, as in Mattern's well-known algorithm using a coloring scheme and a vector counter method [11]. Other approaches divide messages into different groups, such as the bounded-lag approach that distinguishes between "near-future" and "far-future" events [12], and the event horizon approach [13].

On computing GALT, we introduced an algorithm which is based upon Mattern's GVT algorithm but does not appear limitations such as his.

3.1 Mattern's GVT algorithm [11]

Mattern's algorithm is a popular one for solving the transient message problem. Mattern introduced a cut-based GVT algorithm to capture messages in transit during the GVT computation. A simple scheme that guarantees no messages in transit is provided by the use of a coloring scheme and a vector counter method. Here, the cut is implemented by a control message traversing the nodes involved in the computation. He showed using two cuts C and C' is enough to compute the new GVT.

The coloring scheme is used for detecting the messages in transit across the cut. When each process is visited by a control message from previous process at the first cut C, the process changes its color from white to red, assuming all of the processes are initially colored white. A white process only sends white messages and a red process only sends red messages. For ease of illustration, he presented that the processes are formed a ring topology.

The concept of the vector counter method is to count the transient messages across the cut. Let assume that all three processes are initially colored white. Every process P_i counts the number of white messages, which are sent or received, and records in its local vector V_i. When a process P_i receives a white message from other processes, a local vector $V_i[i]$ decreases. Otherwise, it increases $V_i[j]$ every time it sends a white message to other process P_j ($i \neq j$). Arriving to a process P_i, a control message with a control vector C circulating along the ring colors the process red, accumulates its local vectors of the process and resets it to zero. If C[i]=0, the process P_i sends the control message to next process P_j. Otherwise, the process P_i waits until C[i]=0. When the control message returns to the initiator, the initiator checks whether C=0 or not, where C[i] for some i indicates the number of white messages, i.e. transient messages. If C=0, then a new GVT value can be computed. Otherwise, it means a second cut C' is necessary. In the second cut C', the control message waits at each process P_i until all in-transit messages have been received. Since any process P_i does not send a new white message after the first cut C, two cuts C and C' are enough to compute the new GVT.

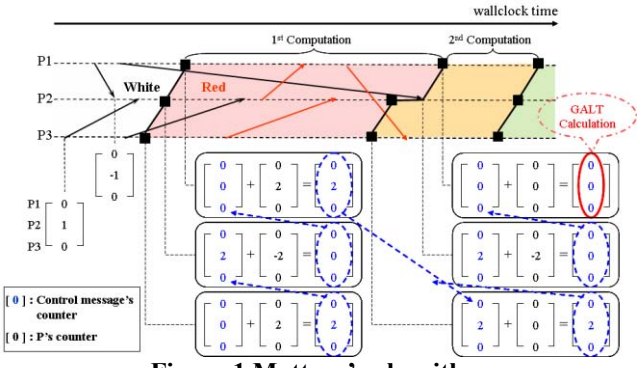

Figure 1 Mattern's algorithm

Figure 1 illustrates an example using the coloring scheme and the vector counter method.

3.2 Adapted Mattern's GVT algorithm

Even though Mattern's algorithm is outstanding, there are several restrictions for applying to HLA/RTI. We utilized an adapted Mattern's algorithm to overcome some restrictions of Mattern's algorithm. Our algorithm have some superficial resemblance to Mattern's algorithm, although they differ significantly in that unreliable communication is supported in our algorithm and our algorithm is implemented through a centralized approach.

When some messages are transmitted via best-effort channels, Mattern's algorithm can lead to limitations. All messages can be sent through either reliable or best-effort channels. Both reliable and best-effort messages are able to be time-stamped. The difference between the two types is in their potential to be lost and to violate a time causality constraint. Reliable messages are never lost and always arrive to the destination in non-decreasing time order. Best-effort messages, however, can be lost and can not be delivered in a timely manner in relation to time-stamped order.

Figure 2 shows restrictions of Mattern's algorithm for HLA/RTI: CASE 1 and CASE 2.

CASE 1: There is any message to be lost.

CASE 2: There is any message with time stamp less than its destination's logical time.

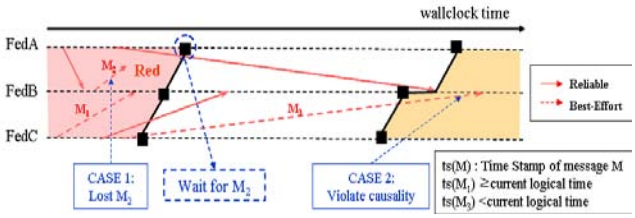

Figure 2 Restrictions of Mattern's algorithm

The developers of the time management service of the RTI have to deliberate the CASE 1. In the GALT calculation, transient messages are red messages which are

transmitted through reliable or best-effort channels. If a message is reliable such as a message M_1, Mattern's algorithm does not matter. Otherwise, a GALT computation can not be terminated in CASE 1 such that a best-effort message like M_2 was a lost one. After receiving a control message from previous federate B, the federate A could not send the control message to next federate C. Because the federate A has to wait for the lost message M_2 according to the vector counter method of Mattern's algorithm, the sum of the control message's counter vector C[A] and the federate A's local counter vector $V_A[A]$ is never less than zero. As a result, it causes a deadlock in the GALT computation.

In order to avoid above case, we proposed that only reliable messages are considered in the vector counter method. There is no deadlock in our GALT calculation algorithm; on the other hand, another problem such as CASE 2 may appear. In this case, a time causality violation happens because a message, like M_3, with time stamp less than federate's current logical time arrives after completing the GALT computation.

We suggested deleting a late-arriving message such as M_3 in a TSO message queue whenever the message leads to a time causality violation. In other words, we considered as if the message M_3 was lost like the message M_1. It enables to ensure the correctness of simulation because of a characteristic of the best-effort transportation. If the message was critical to simulation, the message would have to be transported by reliable channels. Therefore, our GALT computation algorithm has no longer restrictions such as Mattern's algorithm.

4 Architecture of Time Management Service

The time management service module is a part of SMSRTI. The next Section addresses the overall architecture of SMSRTI briefly. The subsequent Section presents the proposed time management service module of SMSRTI in detail.

4.1 Overall Architecture of SMSRTI

Figure 3 shows the overall architecture of SMSRTI with distributed architecture same as DMSO RTI 1.3NG [4], pRTI [5], MÄK RTI [6] and DRTI [14]. It is constructed with an RTI execution, federation executions and federates. The RTI execution manages creation/destruction of each federation execution and communicates federation executions. The federation execution manages a join/resign request as well as data of each federate joined in the federation execution. Each federation execution operates with others independently. In other words, they do not communicate each other. A federate consists of a user code and a Local RTI component (LRC) that constitutes an RTI ambassador, a federate ambassador, and modules. The modules of LRC communicate message-based between internal components of SMSRTI. Whenever the user code calls a HLA service, the RTI ambassador sends a message which is defined for processing API to either the LRC or joining federation execution. If there are APIs which have callbacks, the LRC sends messages to the federate ambassador. All internal components of the SMSRTI can exchange messages which are transmitted by either reliable (TCP/IP) or best-effort (UDP/IP) channels specified the IEEE 1516 standard.

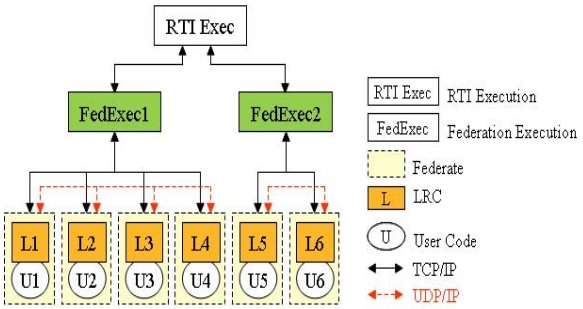

Figure 3 Overall architecture of SMSRTI

The proposed design of time management service module is explained in detail in the following section.

4.2 Design Principles

Figure 4 represents the framework of time management service module. The bold dashed line box expresses time management service part which exists in both the LRC and the federation execution. The proposed time management service module consists of two sub-modules: a TIME module and a GALT module.

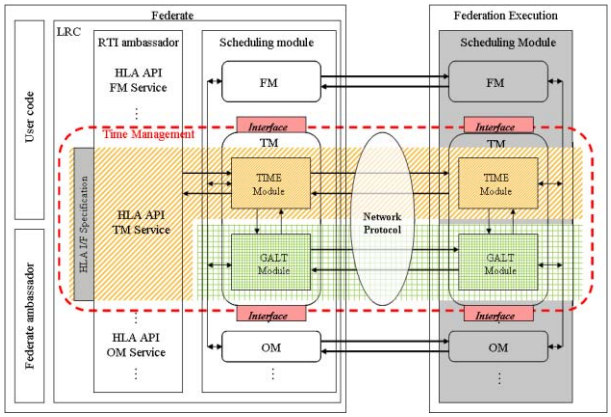

Figure 4 Framework of time management

First, the TIME module supports API services for setting federate's policy and advancing federate's logical time. According to IEEE 1516 standard, a federate can call Enable Time Regulation, Disable Time Regulation, Enable Time Constrained and Disable Time Constrained to determine federate's policy. Also a federate can use Time Advance Request, Time Advance Request Available, Next message Request, Next Message Request Available and Flush Queue Request for advancing its logical time. Furthermore, this module provides callback functions.

Second, the GALT module manages a GALT computation. As mentioned previously, GALT calculation

algorithm is based upon Mattern's GVT algorithm. For implementing the GALT calculation algorithm, we take a centralized approach. The centralized approach is suitable to join/resign federates during a simulation and can be easy to consider a best-effort transportation channel because the federation execution manages all joined federates. The GALT module of the time management service module of the federation execution controls a start/end state of the GALT computation periodically, while that of each federates' LRC counts transient messages.

We defined the interfaces with sub-modules of time management service module so that they interact with other components of SMSRTI. We also specified a format of message exchanged between each federate's LRC and its federation execution, which allows them to communicate through network using TCP/IP.

Our modular design for the time management service offers some advantages. Two sub-modules enhance reusability, modifiability and maintainability of the time management service module within an RTI software through modular design with explicit interfaces of their own. The time management service module of SMSRTI for research is easy to compare with various GALT calculation algorithms. It encourages reuse of the TIME module and minimizes modification by modifying only GALT module instead of whole TM service model.

4.3 TIME Module

The TIME module as an event scheduling module provides basic time management services such federate policy-setting and federate's logical time advancement. We designed the TIME module using a sequence diagram of Unified Modeling Language (UML) 2.0 and a state transition diagram of Discrete Event Systems Specification (DEVS) modeling. After display each API service of time management using the sequence diagram, and describe the state transition diagram for representing all API services. As mentioned earlier, messages of the sequence diagram were defined for interacting with other components of SMSRTI.

We implemented the state transition diagram for the TIME module based on sequence diagrams. Figure 5 represents the state diagram for TIME module. For example, *Time Advance Request* (TAR) service, one of time advancement services, is as follows.

(1) An user sends TAR to the RTI ambassador.
(2) After checking exceptions, the RTI ambassador sets a TimeAdvancePending flag.
(3) The RTI ambassador sends a TIME_ADVANCE message to TIME module of LRC and TSO queue.
(3) Time module transitions into IDLE to TAR_WAIT state.
(4) If there are TSO messages with time stamp less than GALT, TSO queue will delivery them.
(5) If request time is strictly less than GALT, TSO queue will send DONE to TIME module. Otherwise, it will wait until GALT is exactly greater than request time.
(6) When receiving DONE, TIME module will report TIME_ADVANCE_GRANT callback to federate ambassador.
(7) Federate ambassador send *Time Advance Grant* to the user.

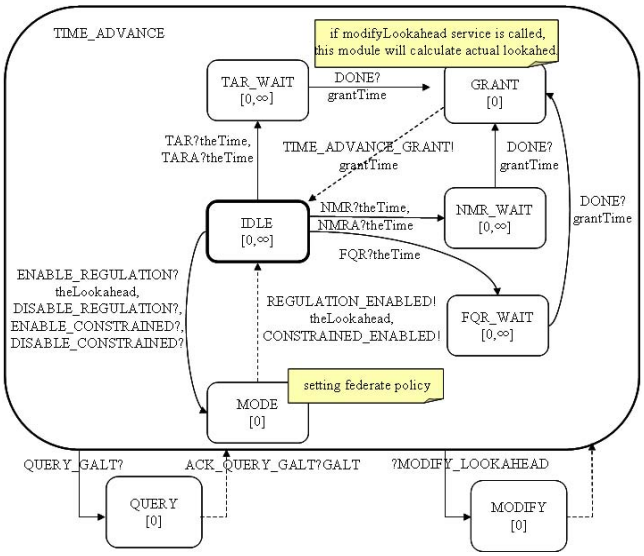

Figure 5 State diagram of LRC's TIME module

4.4 GALT Module

The GALT module for the GALT computation operates with the TIME module independently. We adopted a centralized approach for implementing the GALT calculation algorithm instead of fully distributed one. The federation execution, which is an initiator of a new GALT calculation as a central controller, has a vector counter C such as a control vector counter of Mattern's algorithm. Each federate i has its two local vector counters V_{i1} and V_{i2}. The V_{i1} indicates the number of messages colored by a previous cut, and V_{i2} counts messages sent or received between previous cut and current cut. The procedure of the GALT computation is as follows. A sequence diagram for this procedure is described in Figure 6.

(1) A federation execution periodically sends a start message, namely REQ_CTRL_MSG.
(2) Receiving REQ_CTRL_MSG with a vector counter C, local time list of joined federates and a GALT flag, each joined federate sets a cut-point and counts in-transit messages across cuts.
(3) If a GALT flag of REQ_CTRL_MSG is turned on, each federate obtain a new GALT which is the minimum among logical time of other federates.
(4) If a federate i's local vector counter $V_i[i]$ is less than or equal to the vector counter C[i] of REQ_CTRL_MSG, the federate i reports ACK_CTRL_MSG to the federation execution. Otherwise, the federate i waits until $V_i[i] \leq C[i]$.

(5) The federate i swaps $V_{i1}$for V_{i2} and then, resets V_{i2} to zero.

(6) The federation execution unifies data of ACK_CTRL_MSG provided that ACK_CTRL_MSGs are reported from all joined federates.

(7) If the sum of the vector counter C and $\sum V_{i1}$ is zero, a GALT flag is turned on.

(8) The federation execution updates REQ_CTRL_MSG and delivers it to joined federates in the next GALT computation.

In above (4), ACK_CTRL_MSG contains each federate's logical time and local vector counters V_{i1} and V_{i2}. The time regulating federate returns its current logical time plus lookahead, and non-regulating federate returns infinite time. The difference from Mattern's algorithm is that a control message visits federates not sequentially but concurrently.

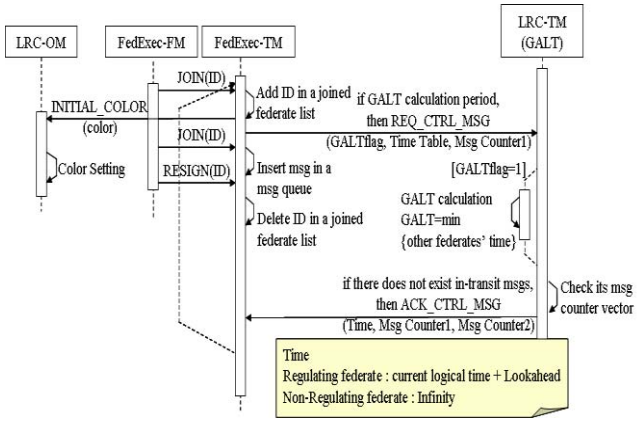

Figure 6 Sequence diagram of GALT computation

We can obtain two advantages from such centralized approach. It first affords great simplicity of design as well as ease of managing data of joined federates. Second, it can easily handle some federates that join in or resign from the federation execution during simulation. As discussed earlier, every GALT calculation is initiated by a start signal of the federation execution. Since the federation execution sends a start message to all joined federates, joining federates during simulation can participate in the next GALT calculation until waiting completion of previous GALT calculation. Because these late-joining federates have to color same as other existing federates, color data of existing federates is informed them when joining in the federation execution.

In the case resigning from the federation execution during simulation, the federation execution deletes the resigning federate's data before colleting federates' message counter and logical time and then, the federate resign from its federation execution.

5 EXPERIMENTS

Figure 7 describes the experimental environment equipped with seven computers. One executes an RTI execution and a federation execution, and others execute federate program. Each federate is both regulating and constrained federate. There exists great number of experimental configurations that have different parameter values. Such parameters include a GALT calculation period, lookahead, the number of joined federates and the number of messages.

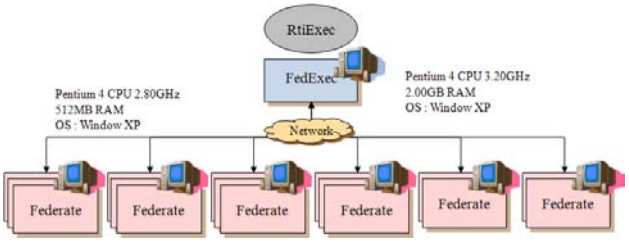

Figure 7 Experimental environment

In the federation, each federate is modeled to use the *Time Advance Request* service to advance logical time, then call *Send Interaction* service to send a reliable TSO message subscribed by all other federates. The logical time interval for each step is 2 and lookahead of each federate is set to 1.9 which is default value. The time stamp of each TSO message is the logical time of next step. The service time recorded in various configurations is an interval between sending a *Time Advance Request* service and receiving a *Time Advance Grant* callback.

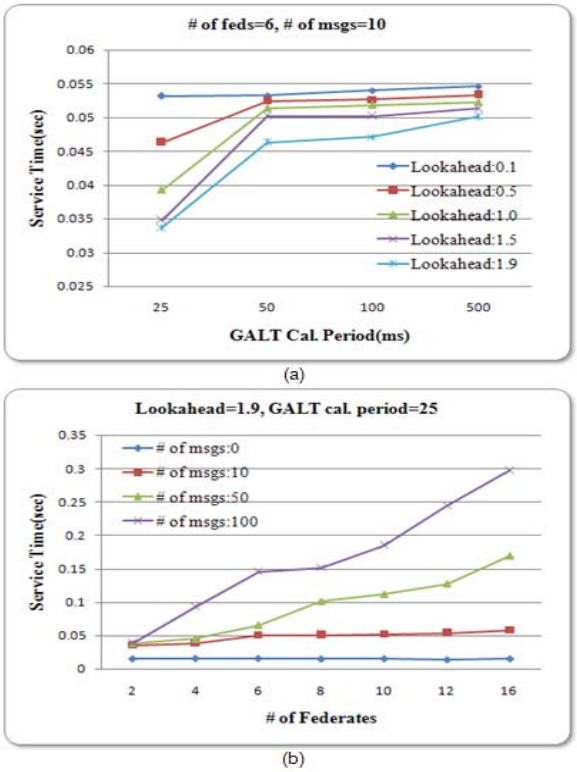

Figure 8 Experimental results

The performance of time management service module can be affected considerably by the period of GALT computation. Frequent update of the GALT computation decreases pending time for time advancement of federates. The result is depicted in Figure 8 (a). This shows that the service time is inversely proportional to lookahead. It is clear that the large lookahead makes the low frequency of GALT calculation. We note that the service time increases with the number of used federates. The major reasons for this increase may be interpreted as follows: (1) When the number of federates is increasing, the number of control messages is also increasing during GALT calculation. (2) In the case of large number of federates, a single round control message is not sufficient to catch the transient messages. We also compared the number of messages sent by each federate. The more messages, which are received before *Time Advance Grant* callback, cause much service time. The results are shown in Figure 8 (b).

6 CONCLUSION

The paper proposed a modular design approach and implementation of the time management service for IEEE 1516 HLA/RTI. The modular design framework supports RTI developers to be free to replace modules. On computing GALT, we introduced an algorithm which is based upon Mattern's GVT algorithm but does not appear limitations such as his. For implementing time management service module with two sub-modules, we adopted a centralized approach. Through experimental results, we confirmed our SMSRTI including proposed time management service module and existing verified RTIs show a similar trend. Further work would improve performance of time management service module by using optimized GALT calculation algorithm

7 ACKNOWLEDGEMENTS

This work was supported by the Korea Science and Engineering Foundation under grant R01-2006-000-11118-0(2006).

8 REFERENCES

[1] IEEE, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules, Std 1516, 2000.

[2] IEEE, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface Specification, Std 1516.1, 2000.

[3] IEEE, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template (OMT), Std 1516.2, 2000.

[4] Defense Modeling and Simulation Office (DMSO), High Level Architecture Run-Time Infrastructure RTI 1.3NG Programmer's Guide, Version 5, 1999.

[5] Pitch Technologies, 2006, http://www.pitch.se/prti.

[6] MÄ K technologies, 2006, http://www.mak.com/rti.htm.

[7] J. H. Kim, "Proposal of High Level Architecture Extension and Run-Time Infrastructure Implementation," Ph.D. Thesis, KAIST, Daejeon, Korea, 2006.

[8] Defense Modeling and Simulation Office (DMSO), High Level Architecture Interface Specification, Version 1.3NG, Washington D.C., 1998.

[9] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*, John Wiley and Sons, Inc, 2000.

[10] B. Samadi, "Distributed Simulation, Algorithms and Performance Analysis," Ph.D. Thesis, University of California, Los Angeles (UCLA), 1985.

[11] F. Mattern, "Efficient Distributed Snapshots and Global Virtual Time Algorithm for Non-FIFO System," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 18. no. 4, pp. 423-434, August 1993.

[12] B. D. Lubachevsky, "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks," *Communications of the ACM*, 32(1): 111-123, 1989.

[13] J. Steinman, "Discrete-Event Simulation and the Event Horizon Part 2: Event List Management," *10th Workshop on Parallel and Distributed Simulation*, pp. 170-178, 1996.

[14] R. M. Fujimoto, "FDK Users Guide," Version. 3.0, 2003, http://www.cc.gatech.edu.