Journal of Computational Information Systems 1:2(2006) 7-15
Available at http://www.JofCI.org

A HIERARCHICAL TIME MANAGEMENT MECHANISM FOR HLA-BASED DISTRIBUTED VIRTUAL ENVIRONMENT

Yan Zhang†, Zhong Zhou, Wei Wu

School of Computer Science and Engineering, Beihang University
Beijing 100083, China

Abstract

Time management is a key group of services in HLA (High Level Architecture) which has prevailed in distributed simulation and distributed virtual environment. Different from the traditional centralized mechanism, a hierarchical time management is presented in this paper. The lower level named FederateGroupLayer is made up of several FederateGroups. Each FederateGroup is comprised of several federates and deploys an upper time manager among them. In this way, the mechanism can process the time advancement hierarchically in two levels. Experiment results show that it can enhance the performance of time management for HLA-based system with more federates.

Keywords: Distributed Virtual Environment, High Level Architecture, time management, hierarchical.

1. Introduction

With the improvement of networking technology and the requirement for collaboration, the importance of technology in distributed virtual environment is recognized by more and more people, and the technology is applied in many fields including collaboration design, collaboration training and network games, etc. The problem of time synchronization in it is still being discussed by the researchers in this field. Lots of methods were used to resolve this problem, including HLA (High Level Architecture).

HLA is an open, supporting object oriented architecture proposed by United States Department of Defense. The key component is the interface specification, which defines six groups of services, including time manager service. The HLA time management services define a mechanism for federates to advance logical time, and enable federates to send and receive time stamped data. It also allows federates to be time synchronized, an important feature for distributed virtual environment. Runtime Infrastructure (RTI) is the software that provides HLA services used by federates to coordinate their operations and data exchange during an HLA federation execution. Xiaojun Shen introduces a HLA based collaborative environment for electronic commerce on Internet [1]. Tainchi Lu implements a distributed interactive collaboration war simulation game, using HLA technology [2]. Ta Nguyen Binh Duong uses multi-server architecture and HLA to model and simulate a distributed virtual environment, and study the problem of load balance in the system [3].

2. Related Work

HLA time management is based on the PDES (Parallel Discrete-Event Simulation) time management [25]. [11] and [12] proposed the first algorithm to compute LBTS [9] in PDES. It assumes each LP (Logical Process) sends a non-decrement time stamped message, and the communication network promises the messages would be received in the order they sent, which ensures the messages could reach the LP in the

† Corresponding author.
Email address:zy@vrlab.buaa.edu.cn (Yan Zhang)

March, 2006

time stamp order. The received messages are stored in a first-in-first-out link by time stamp order. There is a clock in each link, whose value is the time stamp of the first message in the link. LP chooses the messages in the link whose clock's value is the least to process. If the link is empty, LP blocks it. So the protocol ensures the LP processes the messages in non-decrement time stamp order.

Although the approach can ensure that the local causality constraint will never be violated, it is prone to a deadlock. Therefore, a null message mechanism is used to avoid the deadlock. A null message with time stamp T from LPA to LPB ensures that the LPA won't send any message to B, whose time stamp is less than T. Null message does not mean any activity in simulation system, which is only used to avoid the deadlock. The approach introduces a key property utilized by virtually all conservative synchronization algorithms: lookahead. If a LP with logical time T ensures that it won't send any messages with time stamp less than T+L, then the LP is said to have a lookahead of L. The main flaw of null message mechanism is the excess messages generated in the system. The principal problem is that the algorithm uses only the current logical time and lookahead to predict the minimum time stamp it would generate in the future. In the worst situation, every LP can send any other LP. This will cause an excessive number of null messages, and degrade the system. Besides that, Chandy, Misra proposed to detect the deadlock and break it [13].

Mattern introduces an algorithm to compute LBTS using a distributed snapshot. In his algorithm, each simulator maintains a message counter whose value is the result of subtracting the number of the messages sent from the number of the messages received. When the sum of the entire counter's value is zero, it indicates all the messages sent have reached their destination, and it's the time to compute LBTS. If the sum is not zero, they should wait until the next cycle to compute [14].

Besides that, some researchers have studied how to improve the performance of the system by relaxing ordering constraints. An approach is just ignoring the out of order messages [15] [16]. Richard Fujimoto introduces the concept of approximate time, the principal of which is to use the time intervals to replace the precise time stamps, in order to increase the degree of parallel and improve the efficiency of the system with the lookahead of zero or small lookahead [17]. But the drawback is the length of the time intervals would change in different simulation models. Bu-Sung Lee proposes a causal order based time management model to improve the efficiency of the distributed simulation system [18]. But in the HLA, the causal order is not defined.

At present, some institutions are using the algorithm referred by Fredrick in [26], which uses the output time of each federate in different time state to compute their LBTS. The computation of LBTS in [26] is as follows:

$$LBTS(i) = \begin{cases} Min(T_j + L_j), & F_i \text{ is time constrained} \\ \infty, & F_i \text{ is not time constrtained} \end{cases} \quad (1)$$

F_i is the federate in system, and F_j is the federate who can send messages to F_i. T_j is the logical time of federate F_j, and L_j is the lookahead of federate F_j.

[27] and [28] both use the concept of "the greatest lower bound of TSO messages federate can send", which is the output time referred by Frederick in [26]. In [28] it is called ELT (effective logical time), and in [27] it is called output time. Both of them compute LBTS as following:

$$LBTS(i) = \min\{OUTPUT(j)\}, \quad i \neq j \quad (2)$$

When computing the output time of the federate, which is in NER/NERA state, the latter uses the following equation:

$$OUTPUT(i) = \min\{T(i) + L(i), LETS(i)\} \quad (3)$$

And the former proposed two methods: the first is as following:

$$ELT(i) = Min(T(i), LETS(i)) + Lookahead(i) \quad (4)$$

The other is:

$$ELT(i) = LBTS(i) + Lookahead(i) \quad (5)$$

3. Output time based time management algorithm

3.1. LBTS of federate

For each federate F_i, the equation to compute his LBTS is as following:

$$LBTS(i) = \min\{OUTPUT(j)\}, \quad i \neq j \tag{6}$$

The LBTS of federate is the minimum of the other federates' output time, so the key is how to compute the federate's output time, which is decided by the factors such as time state, logical time, lookahead, et al.

3.2. OutputTime of federate

There is no need to compute the output time of the federate which is not time regulating, because their logical time wouldn't influence the other federates to advance their time. The output time of the time regulating federate is computed as follows:

When the federate is in the Time Advance Request or Time Advance Request Available pending state,

$$OUTPUT(i) = T(i) + L(i) \tag{7}$$

T(i) is the time that federate i wants to advance to. L(i) is the value of his current lookahead.

When the federate is in Next Event Request or Next Event Request Available pending state,

$$OUTPUT(i) = \min\{T(i) + L(i), LETS(i), LBTS(i)\} \tag{8}$$

T(i) is the time which the federate i plans to advance to. L(i) is the value of his current lookahead. LETS(i) is the value of the minimum time stamp of the messages in this message queue. LBTS(i) is the value of his LBTS.

When the federate is in the time granted state,

$$OUTPUT(i) = T(i) + L(i) \tag{9}$$

T(i) is the value of his logical time. L(i) is the value of his current lookahead.

Since the above equations may cause deadlock [23], we replace (9) with (10).

$$OUTPUT(i) = \min\{T(i) + L(i), LETS(i)\} \tag{10}$$

4. Two-level time management

4.1. FederateGroup

In traditional time management, RTI collects the value of all the federates' logical time, then computes each one's LBTS, and then manages the time in the system, as figure 1 shows. But since RTI should communicate with all the federates and compute their LBTS, it may cause RTI to be the bottleneck of the system in two ways. Firstly, as the number of federates in the system grows, the complexity of computing would also grow, and may reach or even exceed the limit that RTI can bear. The result would be degrading the system. Secondly, since all the output time of the federates should be sent to RTI in time, the messages received by RTI would grow quickly in a short time, when more federates join in the system. It also might cause the RTI unable to bear the overload in communication and degrade the system.

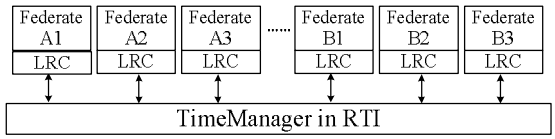

Fig. 1 Traditional time management

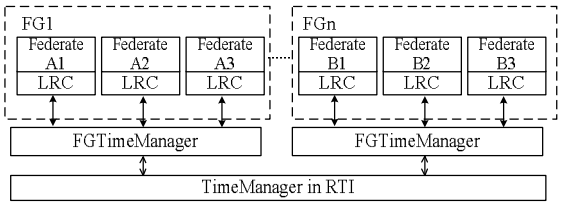

Fig. 2 Two-level time management

We propose the two-level (FG and LRC) time management mechanism based on the traditional time management. As shown in figure 2, the federates are divided into several FG (FederateGroup), in which there is an FGTimeManager, who is responsible for the time of the federates in the FG. And the FGTimeManagers communicate with each other by RTI. The FGTimeManager compute the first and second minimum value of the federates' output time in the FG, and send the result to the others. The mechanism has the following advantages: a) the jobs of computing by RTI are now distributed to the FGTimeManagers, so it decreases the load of RTI, and the probability of degrading the system; b) the messages sent to RTI are divided to groups, and each group of messages are just sent to the FGTimeManager in the FG, so there are no intensive communication in RTI.

4.2. LBTS computation

The equations and denotations in computing LBTS in the mechanism is described below:

FG(i) denotes the FederateGroup i, i∈[1,m], and m is the number of FG. The FGTimeManager in FG(i) is denoted as FGTimeManager(i). F(i) denotes federate i, i∈[1,n], and n is the number of federates. Federation is denoted as {F(1), F(2), ... , F(n)}. The set of the federates in the FG(i) is denoted as SubFederation(i). T(i) denotes the current logical time of F(i), whose lookahead is denoted as L(i), and his output time is denoted as OutputTIme(i). LBTS(i) denotes his LBTS. RMOT(i) denotes the minimum output time in FG(i), and RSMOT(i) denotes the minimum output time in FG(i), besides RMOT(i). FMOT denotes the minimum output time in federation, and FSMOT denotes the minimum output time in federation, besides FMOT.

1. Equation for RMOT(i)

$$\begin{aligned} RMOT(i) &= OutputTime(a) \\ &= \min\{OutputTime(k)\},\ F(k) \in SubFederation(i) \end{aligned} \tag{11}$$

2. Equation for RSMOT(i)

$$RSMOT(i) = \min\{OutputTime(k)\}, k \neq a \tag{12}$$

3. Equation for FMOT

$$FMOT = RMOT(b) = \min\{RMOT(i), RSMOT(i)\} \tag{13}$$

4. Equation for FSMOT

$$FSMOT = \min\{RMOT(i), RSMOT(a)\}, i \neq b \tag{14}$$

5. Equation for LBTS(i)

$$LBTS(i) = \begin{cases} FMOT, & OutputTime(i) < FMOT \\ FSMOT, & OutputTime(i) = FMOT \end{cases} \tag{15}$$

4.3. Algorithm of time advancing

In the algorithm, the requested time advance type of F(i) is denoted as FR(i), and FR(i) ∈{RT(j) | j∈ [1,5]}. RT(1) denotes invoking timeAdvanceRequest() to advance time, RT(2) denotes invoking timeAdvanceRequestAvailable(), RT(3) denotes using nextMessageRequest(), RT(4) denotes using nextMessageRequestAvailabe(), and RT(5) denotes using flushQueueRequest(). The time which F(i) wants to advance is denoted as requestTime(i). TimeAdvanceGrant(i) means RTI grants F(i) to advance his time, requestPending(i) means F(i)'s request to advance time is pending, broadcast(i, A, B, …) means

FGTimeManager(i) sends messages A, B, … to the other FGTimeManagers and upTransfer(i, A, B, …) means FGTimeManager(i) sends the messages A, B, … to the LRC.

When FGTimeManager receives the request to advance time of F(i), he would compute the F(i)'s output time depending on the request type. If the output time of F(i) before sending the request is equal to RMOT(i) or RSMOT(i), FGTimeManager updates the two values, and requests the ones of the other FGTimeManagers. If the output time is at the minimum of all the values, he would update FMOT and FSMOT. Then he request F(i)'s LBTS, and compares it with requestTime(i). The result and the RT(i) would determine whether F(i) could advance his time or be pended. After that, he sends RMOT(i) and RSMOT(i) to the other FGTimeManagers. The algorithm is described below:

1. Compute F(i)'s output time according equation(7), (8), OutputTime(i)' is the updated output time of F(i).

$$OutputTime(i)' = \begin{cases} T(i)+L(i), & FR(i) \in \{RT(j) \mid j \in [1,2]\} \\ \min\{T(i)+L(i), LETS(i)\}, & FR(i) \in \{RT(j) \mid j \in [3,4]\} \\ \min\{LBTS(i), LETS(i), T(i)+L(i)\}, & FR(i) == RT(5) \end{cases} \tag{16}$$

2. Update the value of output time the FGTimeManager holds.

```
if OutputTime(i)∈{RMOT(j), RSMOT(j) | F(i)∈SubFederation(j)}
   { refreshFederateOutputTimeList(j); }     // Update RMOT(j)' and RSMOT(j)'
if OutputTime(i) ∈{FMOT, FSMOT | F(i) ∈SubFederation(j)}
   { refreshRtiOutputTimeList(j);}      // Update FMOT' and FSMOT'
```

3. Decide whether F(i) could advance time.

```
if ((F(i) ∈SubFederation(j)) && (FR(i) == RT(5)))
   { TimeAdvanceGrant(i);}
if ((F(i)∈SubFederation(j)) && ((FR(i)== RT(1)) || ((FR(i)== RT(3))) && (LBTS(i) > requestTime(i)))
   { TimeAdvanceGrant(i);}
else
   {requestPending(i);}
if ((F(i) ∈SubFederation(j)) && ((FR(i)==RT(2)) || (FR(i)==4)) && (LBTS(i)≥requestTime(i)))
   { TimeAdvanceGrant(i);}
else
   {requestPending(i);}
```

4. Send his RMOT and RSMOT.

```
broadcast(j, RMOT(j)', RSMOT(j)').
```

4.4. Algorithm of requesting LBTS

As logical time advancing in federation, the LBTS of each federate is updated continually. There are two ways to update the LBTS. First, the FGTimeManager is responsible for each federate's output time, and compute their LBTS. Second, the federate is responsible for telling FGTimeManager his output time. It means once the output time of federate is changed, he should send the value to the FGTimeManager. Then the FGTimeManager compute the LBTS.

Because the applications of simulation are different, the frequency of advancing time and updating LBTS is different too. Even if the application is the same, the result would be different in different computers and networks. So it is difficult to choose a proper frequency to request the output time. If the frequency is less than the actual one, the system would be degraded. If the frequency is larger than the actual one, it will increase the nonsense overload in computing and communication, and it will degrade the system. So it is wise to choose the latter way to update LBTS, which could avoid the problems in the former one.

4.1.1. Alogrithm in FG layer

When R(j) receives output time of F(i):

1. if OutputTime(i) ∈{RMOT(j), RSMOT(j) | F(i) ∈SubFederation(j)}
   {OutputTime(i)'=T;
   RMOT(j)' is updated according equation(11);
   RSMOT(j)' is updated according equation(12);}
   else
   {OutputTime(i)'=T;}
2. if OutputTime(i) ∈{ FMOT, FSMOT | F(i) ∈SubFederation(j)}
   { FMOT' is updated according equation(13);
   FSMOT' is updated according equation(14);}
3. upTransfer(j; FMOT, FSMOT).
4. if (OutputTime(i)==RMOT || OutputTime(i)==RSMOT)
   { RMOT and RSMOT are updated according equation(11),(12);
   Send results to the other FGTimeManagers;}
   if (OutputTime(i)==FMOT || OutputTime(i)==FSMOT)
   { FMOT and FSMOT are updated according equation(13), (14);
   Send the results to LRC;}
   else
   {record OutputTime(i);}

4.1.2. Algorithm in LRC layer

When LRC(i) received the messages from FGTimeManage:

1. if OutputTime(i) > FMOT
   {LBTS(i) = FMOT;}
   else
   {LBTS(i) = FSMOT;}
2. if OutputTime(i) > minValue
   {LBTS(i) = minValue;}
   else
   {LBTS(i) = maxValue;}

The minValue is the minimum value in the message from FGTimeManager to LRC, and the maxValue is the maximum value in the message from FGTimeManager to LRC.

5. Test results and analysis

We implement time management service in BH RTI 2.2 [24], with the two-level time management mechanism referred above, and design some cases to test the function and performance.

The test environment from 5.1 to 5.3 is described as followed:

2 PCs: CPU Pentium 2.4G Hz, 512 MB-Ram memory, Windows XP Professional. There are 2 federates, and each of them is running in one of these two PCs. The network is 100 Mb/s Ethernet.

5.1. Time Advance Request Service

This case is to test the function of Time Advance Request service in time synchronization and the relationship among the service, lookahead and LBTS. There are only 2 federates A and B, which are both time regulating and time constrained, and with lookahead of 1. They invoke the Time Advance Request service to advance time to logical time 40. Then B invokes Time Advance Request service to request to advance time of 50. After that, A increases his lookahead from 1 to 2, 3… 11. The result in figure 3 shows

that the federates could invoke the Time Advance Request service to advance time synchronously.Because the frequency to advance time in the system reflects the execution efficiency of the whole system, then it can reflect the time management efficiently We use the granted times per second to evaluate the performance of the time management.

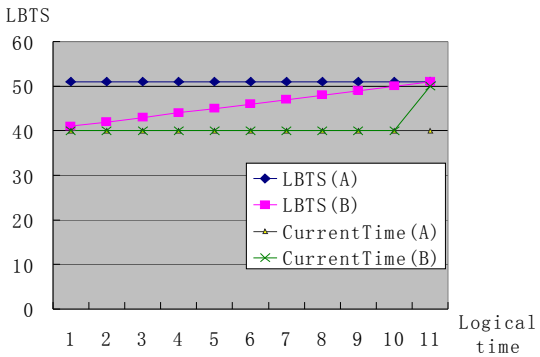

Fig. 3 Lookahead and LBTS when invoking TAR

5.2. Next Event Request Service

This case is to test the function of Next Event Request service in time synchronization and the relationship among the service, TSO messages and LBTS. There are only 2 federates A and B, which are both time regulating and time constrained, and with lookahead of 0.1. They invoke Time Advance Request service to advance time to logical time 20. Then B invokes Next Event Request service to request to advance time of 30. After that, A sends TSO messages with lookahead from 29 to 21 step by step. The result in figure 4 shows that the federates could invoke Next Event Request service to advance time synchronously.

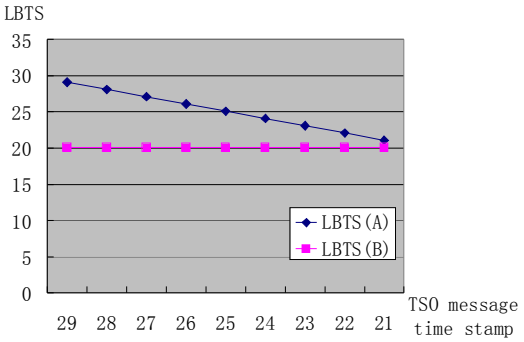

Fig. 4 TSO messages and LBTS when invoking NER

5.3. Performance analysis

Because the frequency to advance time in the system reflects the execution efficiency of the whole system, then it can reflect the time management efficiently We use the granted times per second to evaluate the performance of the time management.

We test BH RTI 2.2, DMSO RTI 1.3NGv6 and pRTI 1516 LE in the same environment:
2 PCs (A, B): CPU Pentium 2.4G Hz, 256 MB-Ram memory, Windows 2000. Network: 100 Mb/s Ethernet.

RTI starts on Machine A, and the federates are running on Machine B. The step of the federates are 10, and the lookahead is 1. They invoke Time Advance Request to advance time. We use the following method to compute granted times per second. We get the logical time T1 before time advancing, and after the

simulation finished, we get the logical time T2, and the time granted times N. Then we can calculate the granted times per second n by equation:

$$n = N/(T2-T1) \quad (17)$$

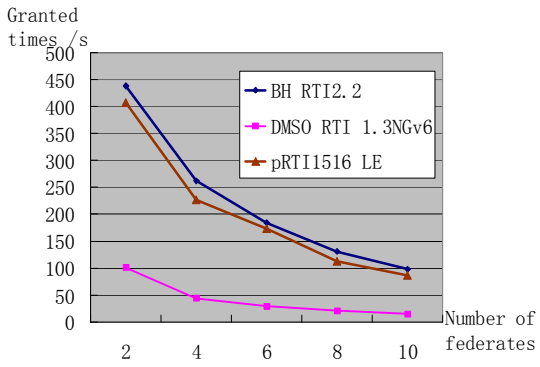

Fig. 5 Granted time per second of BH RTI, DMSO RTI and pRTI LE

The result is shown as figure 5, from which we can see, that the granted times per second of BH RTI are greater than DMSO RTI 1.3NGv6 and pRTI LE. And with the increase of the federates, the burden on RTI increases, so the value of granted times per second drops. But for the two-level time management, the performance of BH RTI is better than DMSO RTI 1.3NGv6 and pRTI LE.

6. Conclusions and future work

This paper proposes two-level time management in HLA-based collaborative environment. In the mechanism, the federates are divided into numbers of groups, so the communication and computation in RTI previously are now distributed in the FGTimeManagers. Each FGTimeManager is responsible for the time management of the federates on its FederateGroup, and sends the messages containing RMOT and RSMOT in its group to each other via RTI. And they could compute the LBTS of the federates in its group by the messages, which decreases the burden on the RTI. The result shows that the efficiency of BH RTI is greater than that of DMSO RTI 1.3NGv6 and pRTI LE, which is implemented by two-level time management mechanism.

We will also be investigating the time management of HLA 1516 standard, and the compatibility between HLA 1516 and HLA 1.3. Because there have been lots of systems which were developed by the HLA 1.3 draft, it's more convenient to make it compatible between the two, without transplanting the systems from HLA 1.3 to HLA 1516.

Acknowledgement

This paper is supported by the National Grand Fundamental Research 973 Program of China under the grant No. 2002CB312105.

References

[1] Xiaojun Shen. Hage, R. Georganas, N. Agent-aided Collaborative Virtual Environments over HLA/RTI. Proceedings of the 3rd IEEE International Workshop on Distributed Interactive Simulation and Real-Time Applications, 128–135, 22-23 Oct, 1999.

[2] Tainchi Lu. and Guanchi Wu. The war-gaming training system based on HLA distributed architecture. Proceedings of International Conference on Computers in Education, 2:889-893, Dec. 2002.

[3] Ta Nguyen Binh Duong. and Suiping Zhou. Effects of inter-server communication in an HLA-based distributed virtual environment. First International Conference on Distributed Frameworks for Multimedia Applications, Page(s):183 – 188, Feb. 2005
[4] Riley, George F. Fujimoto, Richard. Ammar, Mostafa H. Network Aware Time Management and Event Distribution. Proceedings of the Workshop on Parallel and Distributed Simulation, Page(s): 119-126. 2000.
[5] Lee, B.-S. Cai, W. Zhou, J. A causality based time management mechanism for federated simulation. Proceedings of the Workshop on Parallel and Distributed Simulation, Page(s):83-90, 2001.
[6] Thom McLean. Fujimoto, Richard. Predictable Time Management for Real-Time Distributed Simulation. Proceedings of the Workshop on Parallel and Distributed Simulation, Page(s): 89-96, 2003.
[7] Wang, Xiaoguang. Turner, Stephen John. Low, Malcolm Yoke Hean. Gan, Boon Ping. Optimistic synchronization in HLA-based distributed simulation. Proceedings of the Workshop on Parallel and Distributed Simulation, Page(s): 123-130, 2004.
[8] Simulation Interoperability Standards Committee (SISC) of the IEEE Computer Society. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) — IEEE std 1516-2000, 1516.1-2000, 1516.2-2000. New York: The Institute of Electrical and Electronics Engineers Inc., 2000.
[9] High Level Architecture Interface Specification Version 1.3 DRAFT 11, U.S. Department of Defense, 1998.
[10] R M Fujimoto. Time management in the high level architecture [J]. Simulation, 71(6): 388–400, 1998.
[11] Bryant, R. E. Simulation of Packet Communication Architecture Computer Systems. Computer Science Laboratory. Cambridge, Massachusetts, Massachusetts Institute of Technology. 1977.
[12] Chandy, K. M. and J. Misra. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. IEEE Transactions on Software Engineering SE-5(5): 440-452, 1978.
[13] Chandy, K. M. and J. Misra. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. Communications of the ACM 24(4): 198-205, 1981.
[14] Mattern, F. Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation. Journal of Parallel and Distributed Computing 18(4): 423-434. 1993.
[15] Sokol, L. M. and B. K. Stucky. MTW: Experimental Results for a Constrained Optimistic Scheduling Paradigm. Proceedings of the SCS Multiconference on Distributed Simulation. 22: 169-173. 1990.
[16] Rao, D. M., N. V. Thondugulam, et al. Unsynchronized Parallel Discrete Event Simulation. Proceedings of the Winter Simulation Conference: 1563-1570. 1998.
[17] Fujimoto R M. Exploiting Temporal Uncertainty in Parallel and Distributed Simulation [A]. 13th Workshop on PADS[C]. 1999.
[18] Bu-Sung Lee, Wentong Cai, Junlan Zhou. A Causality Based Time Management Mechanism for Federated Simulation [A]. Proc of PADS 2001[C]. 2001.
[19] pRTI 1.3 User's Guide. Pitch AB. 2004.
[20] pRTI 1516 User's Guide. Pitch AB. 2004.
[21] MÄK RTI Reference Manual. MÄK Technologies, Inc. 2003.
[22] DMSO: High Level Architecture Run-Time Infrastructure Next Generation Programmer's Guide, version 1.3v3.2, 2000.
[23] Defense Modeling and Simulation Office. RTI 1.3-Next Generation Programmer's Guide Version 4. 2001. http://www.dmso.mil.
[24] Zhou Z, Zhao QP. Study on RTI congestion control based on the layer of interest. Journal of Software, 15(1):120~130, 2004.
[25] Ouyang LL, Song X, Qing DZ, Hao JB, Wang J. Research of time management in HLA and simulation algorithms of PDES. Journal of System Simulation, 12(3):237~240, 2000.
[26] Kuhl F, Weatherly R, Dahmann J. Fu Zhengjun, Wang Yonghong translate. Creating Computer Simulation Systems An Introduction to High Level Architecture. National Defence Industry Press, 2003.
[27] Liu BQ, Wang HM, Yao YP. A non-deadlock time management algorithm. Journal of Software, 14(9):1515-1522, 2003.
[28] Zhang L, Yin WJ, Chai XD, Liu M. Investigation on time management of RTI. Journal of System Simulation, 12(5):494~498, 2000.
[29] Jian Huang, Kedi Huang. Time Management in HLA. Computer Simulation, 17(4): 69~7, 2000.