Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

DATA CONSISTENCY IN A LARGE-SCALE RUNTIME INFRASTRUCTURE

Buquan Liu
Huaimin Wang
Yiping Yao

School of Computer
National University of Defense Technology
Changsha, Hunan 410073, CHINA

ABSTRACT

In order to support large-scale distributed simulation, we have developed a RTI called StarLink+ with particular architecture which is compliant with IEEE 1516. StarLink+ is composed of a Central RTI server and multiple Local RTI servers. Each Local RTI server manages multiple federates. Data consistency has great influence on RTI's performance and scale. In StarLink+, only a small portion of data must be globally consistent for all Local RTI servers. However, a great amount of data is not consistent for different Local RTI servers. This paper focuses on the research of data consistency about a variety of data in StarLink+. On the one hand, we introduce the fully consistent data such as object name designation and handle assignment; on the other hand, we also study the partly consistent data such as publication and subscription, ownership transfer, and time management.

1 INTRODUCTION

High Level Architecture (HLA) introduces many advanced technologies into distributed simulation and makes distributed simulation develop rapidly. With the scale of distributed simulation applications getting larger and larger , the performance of current Runtime Infrastructure (RTI) can not satisfy increasing need of various applications, and the implementation technology of high performance RTI has become a focus in distributed simulation area.

In distributed architecture, data in RTI is deployed in different positions. Data consistency has an important effect on RTI's performance and scale. Good approaches can decrease the volume of information in network and enhance the efficiency of RTI, thereby solve the performance bottleneck of RTI.

Different from conventional distributed RTIs, StarLink+ is a hierarchical RTI in accordance with IEEE 1516 standards (IEEE 2000a, IEEE 2000b, IEEE 2000c, IEEE 2003), which is composed of a Central RTI server (CRTI) and multiple Local RTI servers (LRTIs). We have successfully deployed StarLink+ into 31 personal computers interconnected by 100 Mbps network and run a federation with 1800 federates. Thousands of federates joined a whole federation within a few minutes. It only cost 900 federates about 5.63 seconds to advance one step all together using the timeAdvanceRequest service, while each federate sent one message subscribed by all other federates within a step.

In StarLink+, only a small portion of data is globally consistent in all Local RTI servers and large amount of data is not consistent. However, any types of data do not prevent concurrent execution of different Local RTI servers. This paper introduces some important technologies about data consistency in StarLink+. In the next section, we explain the unique architecture of StarLink+ and give a few definitions on data consistency. Several experiments and their results are described in this paper, thus these experiments' environment is also introduced there. In the third section, the global data consistency in StarLink+ is discussed, including designation of object name and assignment of handles. While data inconsistency is discussed in the fourth section, such as publication and subscription, ownership transfer, and time advance mechanism. In the fifth section, we explain three issues about StarLink+, which are possibly alternative implementation techniques, standardization and future work. All techniques in this paper shall be useful to a robust RTI's developers for large-scale simulations.

2 DATA CONSISTENCY

StarLink+ was developed on the basis of StarLink (Liu, Wang and Yao 2004). As shown in Figure 1, we know that the whole system is composed of a Central RTI sever and a group of Local RTI servers. The Central RTI server is in charge of all Local RTI servers and each Local RTI server takes charge of multiple federates. CRTI and all Local RTI severs can communicate with one another. Without a Local RTI Component (LRC) (DMSO 2000), a federate can only

communicate with its LRTI. Communications among CRTI, LRTIs and federates are accomplished by the CORBA middleware StarBus (CORBA 2005, Liu 2004, StarBus 2005).

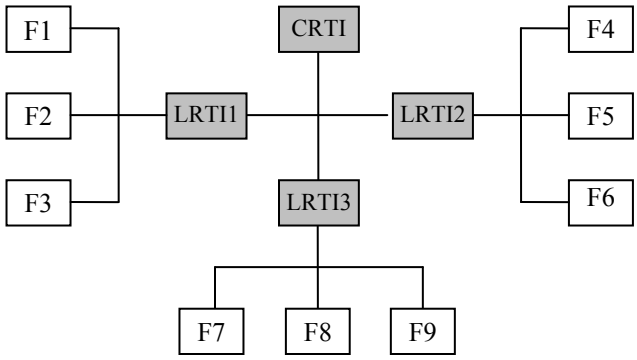

Figure 1: The Hierarchical Architecture of StarLink+

Management and delegate are two roles that a Local RTI server takes on. On the one hand, any federate's information is preserved in its LRTI. A Local RTI server takes charge in its federates and advances simulation in coordination with the Central RTI server and other Local RTI servers. On the other hand, a Local RTI server does not see those federates managed by other Local RTI servers. Any Local RTI server can only join to other Local RTI servers as a common federate. The term delegate is introduced in StarLink+.

Definition 1 *Delegate means that a Local RTI server replaces its federates to communicate with other Local RTI servers.*

A Local RTI server can act as the delegate to its federates. As a delegate, the Local RTI server can join federation execution, publish and subscribe object class attributes, register object instance, update attribute values, transfer attribute ownership, etc.

In StarLink+, the central RTI server is similar to a naming server (OMG 2005a) in CORBA. All LRTIs can interconnect with one another by CRTI. CRTI can be used for creation of federation execution, synchronization point, save and restore in federation management. CRTI is not used in other services such as declaration management, object management, ownership management, time management and data distribution management, etc. The data during federation execution is mainly managed and transmitted by Local RTI servers. Besides all federates' information, the data maintained by a Local RTI server includes federation name, federate list, synchronization point list, object class list, interaction class list, registered object list, object class name and handle pair list, interaction class name and handle pair list, registered object instance name and handle pair list, dimension list, dimension name and handle pair list, reserved object instance name and handle pair list, region specification list, etc.

All data requires to be consistent in each Local RTI server as much as possible, not only static data that can be obtained from a FOM Document Data (FDD) file and an initialization file but also dynamic data that be produced during federation execution such as federate list, synchronization point list, registered object list, registered object instance name and handle pair list, reserved object instance name and handle pair list.

Definition 2 *Strong consistency means that data should be globally consistent in all Local RTI servers.*

To decrease the overhead aroused by global data consistency and enhance simulation efficiency, StarLink+ does not compel each Local RTI server to maintain all data consistently.

Definition 3 *Weak consistency means that data is not globally consistent in all Local RTI severs.*

Both strong consistency and weak consistency are highly efficient in StarLink+. All Local RTI servers can execute simulation concurrently, and there is no synchronization based on lock mechanism. Therefore, StarLink+ is able to be used for large-scale distributed parallel simulations.

In this paper, a few important experiments are described. They were conducted in a large computer room for students to study and review. As in Figure 2, more than 90 personal computers are installed in the room, and they have the same configuration. The configuration for each computer is:

- CPU: Pentium IV 1.7G
- Memory: 256M
- Network: 100Mbps
- Operating system: Windows 2000.

Figure 2: Experiment Environment

In addition, all test programs were written in Visual C++ 6.0. Except for one control program and one result display program, all programs were run in the background. The experiment environment was open. While we made our experiments, students in the room might browse campus network, play network games, listen to music and watch movies via network.

3 STRONG CONSISTENCY

Different from other RTIs which can support multiple federations, StarLink+ only supports a single federation. Mul-

tiple federations should be executed by starting multiple CRTIs. A federation's name and a FDD file's name should be saved in an initialization file. Thus, strong consistency of static data is easy to implement. Whenever CRTI and each LRTI start, they use the same source code to read the same FDD file for initialization, including object class list, interaction class list, object class name and handle pair list, interaction class name and handle pair list, dimension list, etc. The data shall be globally consistent in CRTI and all LRTIs although most data is useless in CRTI..

This paragraph explains the rationale of strong consistency for dynamic data in StarLink+. Representative approaches are the designation of object name and the assignment of handles.

3.1 Object Name Designation

When a federate calls the registerObjectInstance service to register an object instance, each object instance should be designated a unique name. One method is that the object name is registered and designated in CRTI. CRTI either designates a new name for an object instance, or throws an exception if a federate tries to register a name already assigned. However, the method is less efficient and CRTI shall be a bottleneck. In fact, another efficient method is used in StarLink+. Multiple federates can register object instances to their LRTIs simultaneously. The parallel method needn't CRTI to coordinate anything.

When a federate registers an object instance, the federate can appoint a name as one parameter of the registerObjectInstance service. If the federate does not appoint a name, its LRTI shall designate a default name. Then, the LRTI attaches a prefix before each object name as

$$objectName = lrtiName + "\_" + objectName.$$

Thus, the new object name is globally unique. In StarLink+, each LRTI should join to CRTI with a unique name *lrtiName*, and we urgently demand that a federate shall not register an object instance with any LRTI's name. For example, we often use these names to represent LRTIs such as "machine01", "machine02", etc.

After an object instance is successfully registered in a LRTI, the LRTI calls the lRTIregisterObjectInstance service with the parameter of the new name to notify all other LRTIs. These LRTIs register the object instance directly and does not attach a prefix again. The lRTIregisterObjectInstance service is added by StarLink+. We usually call these services prefixing with "lRTI" expanded HLA services, and they are for communication among CRTI and LRTIs.

In StarLink+, CRTI can be started in a single computer or in any computer together with a LRTI because CRTI has nothing to do with the performance of our experiments. We do not mention CRTI in the following experiments. Here is one case of these experiments.

Experiment 1 One LRTI and one federate were started in two computers respectively. Each federate joined to its LRTI within the same computer. Federate F1 called the registerObjectInstance service to register an object instance to LRTI1, and LRTI1 called the lRTIregisterObjectInstance service to register the object instance to LRTI2. Then, LRTI2 called the discoverObjectInstance service to notify federate F2 to discover the object instance. Now F2 registered another object instance to LRTI2 immediately. When F1 found the object instance registered by F2, it started the loop again.

Experimental result When F1 repeats the whole procedure from its registration to discovery for many times such as 1000, 10000, etc., the average time *Toneloop* is easy to compute then. However, the average time *Tmachines* from F1's registration to F2's discovery is more significant, which is a half of *Toneloop*. In our experiments, *Tmachines* is about 0.5 milliseconds.

3.2 Handle Assignment

In StarLink+, all members such as CRTI, LRTIs and federates are appointed a unique handle respectively. The handle is represented by an integer. Here is the efficient approach.

1. The handle of CRTI is equal to zero.
2. When a LRTI joins to CRTI, its ha*ndle* $handle(LRTI)$ is assigned by CRTI as

   $$handle(LRTI) = 10000 * serialNumber.$$

   The serialNumber in the formula is the LRTI's registering order in CRTI. For example, the first LRTI joining to CRTI is assigned the handle 10000, and the second LRTI's handle is 20000.
3. When a federate joins to a LRTI, its handle is assigned by the LRTI as

   $$handle(Federate) = handle(LRTI) + serialNumber.$$

   The *serialNumber* in the formula is the federate's registering order in its LRTI. For example, the first federate joining to a LRTI with handle 20000 is assigned the handle 20001, and the second federate's handle is 20002. Of course, the method requires that the number of federates within a LRTI should not be more than 9999.

Now it is easy to designate an object instance handle. When a federate registers an object instance, its LRTI assigns a handle for the object instance as

$$handle(Object) = (serialNumber * 10000) + handle(LRTI) / 10000.$$

The *serialNumber* in the formula is registering order of object instance. The formula means that an object instance's handle is made up of two parts, the low four digits represent a LRTI while the high digits represent registration order. Of course, the method requires that the number of LRTIs in a federation should not be more than 9999.

In fact, many other types of handles such as region handle have similar results. An alternative way is to define a structure with two member for handle types. One means LRTI, and another means registration order. More details are discussed in section 5.2.

Experiment 2 One LRTI and 60 federates were started in every 30 computers respectively. Each federate joined to its LRTI within the same computer. A federate called the following services to initialize and then waited a synchronization point to advance simulation. These services included createFederationExecution, joinFederationExecution, getObjectClassHandle, getAttributeHandle, publishObjectClassAttributes, subscribeObjectClassAttributes, registerObjectInstance, enableTimeRegulation, enableTimeConstrained, etc.

Experimental results A batch program was used to start 60 federates one by one automatically in each computer. If the interval time between two federates was 1, 2, or 3 seconds, the whole initialization time for 60 federates to reach synchronization point in one computer was exactly 1, 2, or 3 minutes. If all batch programs in 30 computers ran simultaneously, the whole initialization time for 1800 (60*30) federates was also nearly 1, 2, or 3 minutes.

4 WEAK CONSISTENCY

Due to the delegate mechanism, a LRTI does not see any federates that belong to other LRTIs. Thus, there exists large amount of weak consistency in StarLink+.

This paragraph explains the rationale of weak consistency in StarLink+. Representative principles are publication and subscription, ownership transfer, and time advance mechanism.

4.1 Publication and Subscription

In Figure 1, when federate F1 calls the publishObjectClassAttributes or subscribeObjectClassAttributes service to publish or subscribe object class attributes, LRTI1 records the information and also calls the same service to publish or subscribe object class attributes to LRTI2. Thus, if federate F2 publishes or subscribes the object class attributes which have been published or subscribed by F1, LRTI1 shall not notify LRTI2 any more.

In the experiments with 1800 federates, all federates published and subscribed the same object class attributes. Therefore, each LRTI only published and subscribed corresponding attributes to other LRTIs once. When a federate called the updateAttributeValues service to update its instance attributes, its LRTI only called the service once to notify all other LRTIs. Then these LRTIs called the reflectAttributeValues service to notify their federates. The experiments indicate that the number of messages among LRTIs decreases greatly.

4.2 Ownership Transfer

In accordance with HLA standards, an instance attribute shall be owned by at most one joined federate at any given time. The ownership of an instance attribute may be owned by a federate, or unowned by all federates, or by RTI such as an instance attribute in MOM (Management Object Model).

In StarLink+, ownership state of each LRTI keeps consistent at LRTI level, while that may be inconsistent at federate level. That's to say, all LRTIs know which LRTI the owner of an instance attribute belongs to, although they may not know which federate the owner is. This weak consistency can both guarantee correctness of ownership state from overall perspective, and reduce communication among LRTIs.

Therefore, a Local RTI server *X* shall not notify other Local RTI servers if the ownership of an instance attribute is only transferred between two local federates. While a federate in Local RTI server *Y* requests to acquire the ownership, *Y* shall know that *X* is just the Local RTI server that the owner belongs to. Then *Y* notifies *X* that it wants to acquire the ownership. Now *X* notifies the owner to release its ownership. When the ownership is released and owned by *Y*, *X* should notify all other Local RTI servers to change the owner to be *Y*. Next experiment gives a detailed example. In addition, if the ownership of an instance attribute is released by a federate in *X* and no one wants to acquire the ownership, the instance attribute is unowned by all local federates in *X* but all other LRTIs such as *Y* think that the owner is still *X*.

Experiment 3 Three local RTI servers LRTIA, LRTIB and LRTIC ran in three computers. Federate F1 ran in the same computer with LRTIA and joined to LRTIA. Federate F2 ran in the same computer with LRTIB and joined to LRTIB. In Figure 3, an instance was initially registered by F2 and F2 was the owner of its attributes. When F1 called the attributeOwnershipAcquisition service to request to acquire the ownership of an attribute, LRTIA knows that the owner belonged to LRTIB so that LRTIA called the lRTIattributeOwnershipAcquisition service to notify LRTIB. LRTIB called the requestAttributeOwnershipRelease service to notify F2 to release ownership. Then, F2 called the unconditionalAttributeOwnershipDi-

vestiture service to divest the attribute's ownership. LRTIB set the attribute's owner to be LRTIA and called the lRTIsetOwnership service to notify all other LRTIs. Thus, LRTIC set the attribute's owner to be LRTIA while LRTIA set the owner to be F1. Finally, LRTIA called the attributeOwnershipAcquisitionNotification service to notify F1 and F1 acquired the attribute ownership.

Experimental result The average time is 2 milliseconds, which was from F1's requesting to acquire the attribute ownership to receiving the notification of ownership acquisition.

4.3 Time Advance

The computation of Greatest Available Logical Time (GALT) is critical to implement time management services. GALT is also called Lower Bound Time Stamp (LBTS) in HLA 1.3 (DMSO 1998a, DMSO 1998b, DMSO 1998c). For short discussion, we suppose that all federates call the timeAdvanceRequest service to advance logical time, and they do not modify their lookahead (Chandy and Misra 1979, Fujimoto 1988, Fujimoto 1996, Fujimoto 2000).

Definition 4 *The symbol S(i) is defined:*

(a) If i is a federate, we have S(i)=T(i)+L(i). When federate i is in time advancing state, T(i) is the logical time to which the federate request to advance. Otherwise, T(i) is the federate's current logical time. L(i) means the federate's lookahead.

(b) If i is a Local RTI server, we have S(i)=min{T(j)+L(j)}. For any j, j is a federate that belongs to i.

To compare different values visually, we usually call $S(i)$ as i's stature (i is a federate or a Local RTI server). If $S(i)<S(j)$, we say that i's stature is less than j's.

If X is the Local RTI server that federate i belongs to, the federate's GALT is computed as

$$GALT(i)=min\{S(j), S(Y)\}.$$

Where: For any j, $j \in X$ and $i \neq j$;

For any Local RTI server Y, $X \neq Y$.

From the above algorithm, we know that quite few messages occur among LRTIs for computing GALT. Time management in StarLink+ is very efficient. As an example of the federation with 30 LRTIs and 1800 federates, the stature of a LRTI was equal to the minimal stature of its 60 federates'. Only when all 60 federates advanced one step, the LRTI's stature was then changed and it should send a message to notify all other LRTIs. In addition, a LRTI must not send any message to its federates because its federates' information for computing GALT was preserved in the LRTI.

Although the timeAdvanceRequest service is discussed and lookahead is supposed not to be modified during a federation execution, this is similar to other time management services even if lookahead is modified.

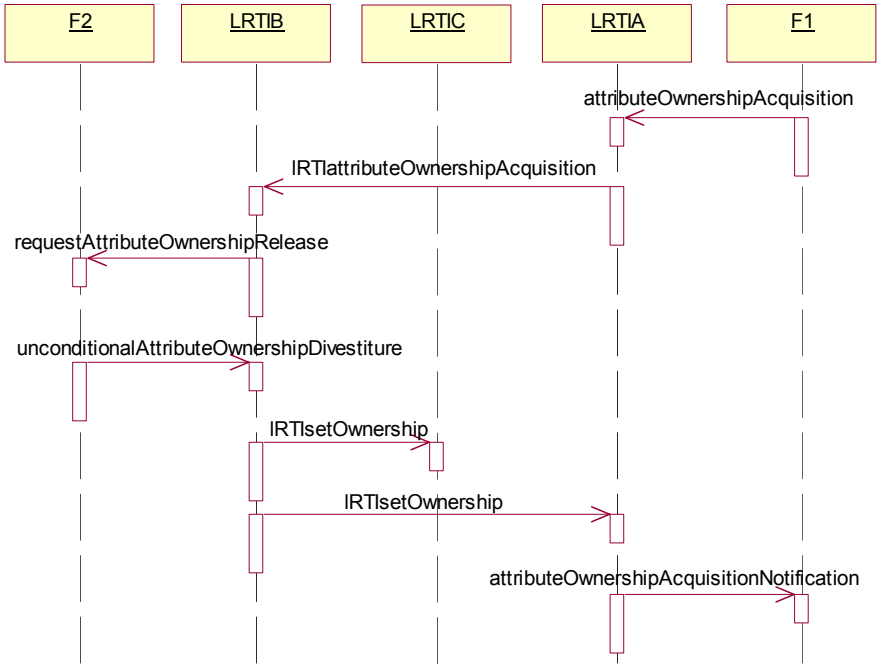

Figure 3: Time Delay of Acquiring Attribute Ownership

Experiment 4 We started 30 LRTIs in 30 computers, and 30, 40 or 60 federates ran in each LRTI. All federates were both time regulating and constrained, and they advanced logical time step by step. Each federate's lookahead was not larger than the interval logical time between two steps. Thus all federates must advance simulation together, and one federate could not go ahead or behind another for the timeAdvanceRequest service. After each federate initialized, it waited for a synchronization point. When all 900, 1200, 1800 federates were started, a control federate in 31st machine registered federation synchronization point. Once the whole federation synchronized, all federates called the timeAdvanceRequest service to advance regularly. In each step, any federate sent a Receive Order (RO) message that was subscribed by all other federates. A LRTI would call the timeAdvanceGrant service to grant each local federate' advance only when all federates in the federation were time synchronized, i.e. they requested to advance to the same logical time.

Experiment results The average time for 900, 1200 and 1800 federates was 5.63, 7.58 and 21.04 seconds, which was from a federate's calling the timeAdvanceRequest service to being granted. In fact, the best result we have obtained for 900 federates was only 4 seconds.

5 MORE CONSIDERATIONS

Three issues are discussed in the section: alternative implementation techniques, standardization and future work about StarLink+.

5.1 Alternative Approaches

A few useful techniques about data consistency have been introduced before. Of course, more complicated approaches may also be used for implementing StarLink+. For example:

1. Let CRTI manage global data. This shall make data consistency rather difficult. Suppose that each federate's GALT be computed by CRTI. CRTI should know each federate's advancing status although a federate's status is already stored in its LRTI. Thus, it is very complicated to confirm each federate's consistent status both in CRTI and its LRTI such as publication and subscription relationship, Time Stamp Order (TSO) message queue, whether a federate is regulating or constrained, etc.
2. Let all federates join to all LRTIs. The approach requires that each LRTI preserves all federates status, which shall also make data consistency complicated.

5.2 Standardization of StarLink+

The interface services in StarLink+ are compliant with IEEE 1516, but a part of data types are in accordance with HLA 1.3 and OMG HLA standards (OMG 2005b, OMG 2005c). For example, most handle types are defined as integer. Both RTI::LogicalTime and RTI::LogicalTimeInterval are defined as double. However, these types are defined as the class type in IEEE 1516. This shall bring more advantages:

1. Simplify the development of StarLink+.
2. Make users develop applications easily. Here is an example. Two different variables x and y can be written as $x+y$ in StarLink+ although one may has the RTI::LogicalTime type and the other owns the RTI::LogicalTimeInterval type. However, the two data types are different in IEEE 1516 and $x+y$ is illegal.
3. Be compatible with earlier RTI versions. The RTIs that we developed before StarLink+ were implemented in the same way.

5.3 Future Work

StarLink+ aims at different groups of users for large-scale simulations. High performance computing is one important research branch in our school (NUDT 2005). Besides large-scale simulations over wide area network and local area network, we shall migrate it to high performance computers made by our school. Therefore, future work in StarLink+ may include:

1. Within a high performance computer, the Message-Passing Interface (MPI) communication mechanism shall replace underground CORBA technology in StarLink+ (Hwang and Xu 1998, MPI 2005). But CORBA is still applied for communication between a high performance computer and exterior machines.
2. In a single node or computer, a LRTI and its local federates shall communicate via shared memory rather than CORBA based on TCP/IP.

6 CONCLUSION

StarLink+ is a hierarchical RTI compliant with IEEE 1516 standards. Data consistency has a considerable effect on RTI's correctness and efficiency. In StarLink+, all local RTI servers can execute simulation concurrently, which can enhance performance of StarLink+ greatly. This paper introduces a few efficient technologies for data consistency such as definition of object class name, and assignment of various handle types. More efficient technologies for data inconsistency are also described such as publication and

subscription, ownership transfer, and time advance mechanism. In addition, we have also presented multiple experiments and their results.

ACKNOWLEDGMENTS

We thank *the National High Technology Research and Development Program of China* (863 program) under the grants of No. 2004AA112020 and No. 2004AA115130 for their support. We also appreciate *the National Natural Science Foundation of China* under the grants of No. 60373024 and No. 90412011.

REFERENCES

Chandy, K. M., and J. Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering* 5 (5): 440-452.

CORBA. 2005. Available via <http://www.corba.org> [accessed March 20, 2005].

DMSO. 1998a. High level architecture rules, v1.3 [online]. Available via <http://hla.dmso.mil> [accessed August 15, 2000].

DMSO. 1998b. High level architecture interface specification, v1.3 [online]. Available via <http://hla.dmso.mil> [accessed August 15, 2000].

DMSO. 1998c. High level architecture object model template, v1.3 [online]. Available via <http://hla.dmso.mil> [accessed August 15, 2000].

DMSO. 2000. RTI 1.3-next generation programmer's guide version 6 [online]. Available via <http://hla.dmso.mil> [accessed April 8, 2002].

Fujimoto, R. M. 1988. Lookahead in parallel discrete event simulation. *1988 International Conference on Parallel Processing* 3: 34-41.

Fujimoto, R. M. 1996. HLA time management: Design document. College of Computing Georgia Institute of Technology Atlanta. Available via <http://www.cc.gatech.edu/computing/pads/papers.html> [accessed October 5, 2002].

Fujimoto, R. M. 2000. *Parallel and distributed simulation systems*. New York: John Wiley & Sons.

Hwang, K., and Z. Xu. 1998. *Scalable parallel computing*. New York: The McGraw-Hill Companies, Inc.

IEEE. 2000a. *Standard for modeling and simulation (M&S) high level architecture (HLA)-framework and rules*. IEEE std 1516-2000. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

IEEE. 2000b. *Standard for modeling and simulation (M&S) high level architecture (HLA)-federate interface specification*. IEEE std 1516.1-2000. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

IEEE. 2000c. *Standard for modeling and simulation (M&S) high level architecture (HLA)-object model template (OMT) specification*. IEEE std 1516.2-2000. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

IEEE. 2003. *IEEE recommended practice for high level architecture (HLA) federation development and execution process (FEDEP)*. IEEE std 1516.3-2003. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Liu, B. Q., H. M. Wang, and Y. P. Yao. 2004. Key techniques of a hierarchical simulation runtime infrastructure-StarLink. *Journal of Software* 14 (1): 9-16.

MPI. 2005. Available via <http://www.mpi-forum.org> [accessed June 14, 2005].

NUDT. 2005. Available online via <http://www.nudt.edu.cn/newweb/research/achievement.htm> [accessed June 14, 2005].

OMG 2005a. Available online via <http://www.omg.org/technology/documents/formal/naming_service.htm> [accessed March 20, 2005].

OMG 2005b. Available online via <http://www.omg.org/cgi-bin/doc?formal/2001-05-01> [accessed June 15, 2005].

OMG 2005c. Available online via <http://www.omg.org/cgi-bin/doc?formal/2002-11-11> [accessed June 15, 2005].

StarBus. 2005. Available online via <http://starbus.nudt.edu.cn> [accessed March 20, 2005].

AUTHOR BIOGRAPHIES

BUQUAN LIU received his B.S. degree in computer science from Nanjing University in 1991. His M.S. and Ph.D. degrees were received in School of Computer from National University of Defense Technology (NUDT) in 1998 and 2004 respectively. He has achieved 2 Provincial Science and Technology Advance Awards and 1 patent of *the design of hierarchical RTI servers based on interoperability protocol*. Now he is an assistant professor of the school and his interests are distributed simulation and high performance computing. His e-mail address is <bqliu@nudt.edu.cn>.

HUAIMIN WANG is a professor in School of Computer at the National University of Defense Technology. He received his Ph.D. degree in computer science in 1992. He is a member of the Editorial Board of *Chinese Journal of*

Computers and *Journal of Computer Science and Technology*. Dr. Wang has served as a member of the Expert Committee for *Computer Software and Hardware of the National High Technology Research and Development Program of China* (863 Program). Since 1990, he has chaired more than 10 research projects under the grants of *the National Natural Science Foundation of China*, 863 Program, and *the National Basic Research Program of China* (973 Program), etc. In 2003, he was awarded one 2nd class National Science and Technology Advance Award. Up to now, he has published more than 90 papers and directed 20 graduate students. His research focuses on distributed object, agent technology, grid computing and network security. His e-mail address is <whm_w@163.com>.

YIPING YAO is a professor of School of Computer in National University of Defense Technology. In this school, he received his M.S. and Ph.D. degrees in 1987 and 2004 respectively. he received his B.S. degree in computer science from Huazhong University of Science and Technology in 1985. He worked in America from March 1996 to July 1997. At present, he has achieved 2 second-class National Science and Technology Advance Awards and 8 Provincial Science and Technology Advance Awards. More than 40 papers and 3 monographs were also published. His research areas are distributed simulation and virtual reality. His e-mail address is <ypyao@nudt.edu.cn>.