J. Parallel Distrib. Comput. 68 (2008) 1487–1503

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A decoupled federate architecture for high level architecture-based distributed simulation

Dan Chen [a,*], Stephen John Turner [b], Wentong Cai [b], Muzhou Xiong [b]

[a] Institute of Electrical Engineering, Yanshan University, Qinhuangdao, Postcode: 066004, China
[b] School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore

ARTICLE INFO

Article history:
Received 25 February 2007
Received in revised form
15 May 2008
Accepted 25 July 2008
Available online 6 August 2008

Keywords:
High level architecture
Runtime infrastructure
Decoupled federate architecture
Distributed simulation cloning
Fault tolerance
Web enabled architecture
Grid enabled architecture

ABSTRACT

A number of research issues arise in executing large scale High Level Architecture (HLA) based distributed simulations. Among these issues distributed simulation cloning, fault tolerance and Grid enabled architecture are particularly important and challenging. This paper presents a Decoupled Federate Architecture as the underlying infrastructure to facilitate a solution to each of the above issues.

Distributed simulation cloning has been designed to improve the performance of "what-if" analysis, by enabling the examination of alternative scenarios concurrently within the same execution session. Furthermore, simulation federates running at different locations are liable to failure. Such risk increases with the scale of a distributed simulation. The architecture has also been exploited to form the basis of a generic framework to support runtime robustness. The development of complex simulation applications often requires geographically distributed huge computing resources and data sets. Using the architecture, Web and Grid technologies have been successfully combined with existing simulation technologies thus to enhance interoperability and execution flexibility for such simulations.

Benchmark experiments have been performed to study the extra overhead incurred by the Decoupled Federate Architecture against a normal federate. Encouraging experimental results indicate that a well designed architecture can achieve performance close to the normal one in terms of latency and time synchronization. Additional experiments have been presented for evaluating the three solutions supported by the architecture.

1. Introduction

Modeling and simulations are essential tools in many areas of science and engineering, for example, for predicting the behavior of new systems being designed or for analyzing natural phenomena. Simulation of complex systems has long been placed in the highly computation intensive world with computational requirements which far exceed the capabilities of a single computer. The last decade has witnessed an explosion of interest in distributed simulation technology. It not only aims at speeding up simulations, but also serves as a strategic technology for linking simulation components of various types (e.g., discrete or continuous, numerical or discrete event etc.) at multiple locations to create a common virtual environment (e.g., battlefields, virtual factories and supply chains, agent-based systems, games etc). The culmination of this activity is the advent of the IEEE 1516 standard, namely the High Level Architecture (HLA), to facilitate interoperability among simulation federates and promote reuse of simulation models. Runtime Infrastructure (RTI) software supports and synchronizes the interactions amongst different federates conforming to the standard HLA specifications [10,17].

In the case where the problem domain is particularly complex, or involves multiple collaborative parties, analysts often need to construct a large scale federation with individual simulation federates interacting over the Internet. These applications are usually time consuming and computation intensive, which require vast distributed computing resources. There are a number of important research issues to be investigated in the area of large scale HLA based distributed simulations, such as simulation cloning, fault tolerance and Web or Grid enabled architectures.

In the context of a conventional simulation, an analyst chooses the best solutions by repeating the simulation multiple times, to examine alternative decision policies and strategies. Simulating each of these choices from the start would require much computation to be repeated unnecessarily. Distributed simulation cloning technology can clone replicas of one or more federates at the decision point, to examine alternative scenarios concurrently [3]. The analyst can quickly obtain multiple sets of simulation

* Corresponding author.
E-mail addresses: chendan@pmail.ntu.edu.sg, dan.chen@ysu.edu.cn (D. Chen), assjturner@ntu.edu.sg (S.J. Turner), aswtcai@ntu.edu.sg (W. Cai), mzxiong@ntu.edu.sg (M. Xiong).

0743-7315/$ – see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2008.07.010

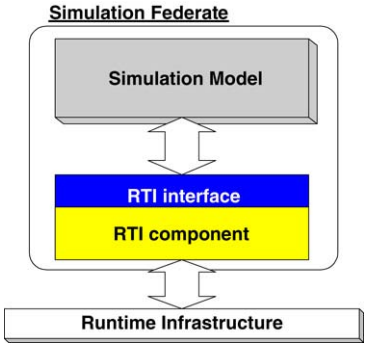


Fig. 1.1. Abstract model of a simulation federate.

results that represent the impacts of alternative decisions in the physical system. Thus, the technology facilitates an effective decision support tool for simulation.

Simulation federates running at different locations are liable to failure: as the current IEEE 1516 HLA does not support a formal fault tolerant model [17], crash of a federate or a part of a federation will lead to the failure of the whole federation. When failure occurs, even if it is feasible to restart the simulation from a previous checkpoint [19], repeating the execution could either be costly or lose the functions of the failed simulation (for example, a rare event may not be regenerated in the new "recovered" simulation execution). The risk of such failure increases with the number of federates inside one single federation. Although fault tolerance support has been informally proposed in the latest HLA Evolved specification, and some design patterns for fault tolerant federations were suggested in [22], there are only a few preliminary and non-standard implementations for this purpose. Hence, there exists a pressing need for a mechanism to support runtime robustness in HLA-based distributed simulations.

Executing HLA-based simulations normally requires users to allocate computing resources to federates and the RTIEXEC (DMSO RTI) statically, which lacks flexibility. Interoperation amongst federates over the Internet is also restrained by security requirements (e.g., firewalls) or heterogeneity of the computing resources in different organizations. Web services provide interoperability between software applications distributed over the Internet. The emergence of Grid technologies provides an unrivalled opportunity for large-scale distributed simulation. The HLA enables the construction of a large-scale simulation using distributed (possibly existing) simulation components. Meanwhile, Grid technologies provide mechanisms for the collaboration and management of distributed computing resources where the simulation is being executed, while facilitating access to geographically distributed data sets as well.

A normal simulation federate can be viewed as an integrated program, consisting of a simulation model and Local RTI Component (LRC), as shown in Fig. 1.1. The simulation model executes the representation of the system of interest, whereas the LRC services it by interacting and synchronizing with other federates. In a sense, the simulation model performs local computing, while the LRC carries out distributed computing for the model. It is very difficult to directly enable any of the above three technologies in a normal federate, due to the tight coupling of the simulation model and the LRC. For example, there is no straightforward method to replicate a federate to support simulation cloning. Providing fault tolerance to federates requires an approach to "isolate" the error of the LRC from the simulation model in addition to the challenge to develop a generic state saving and recovery mechanism. This paper introduces the idea of decoupling the LRC from a normal HLA federate, in order to facilitate solutions to the above research issues.

The proposed approach separates these two modules into two independent processes, namely a virtual federate and a physical federate. The virtual federate inherits the code of the original simulation federate, while associating with it a "virtual" RTI component, which still provides the simulation model with a standard interface of RTI services. The real RTI services are accessed through the physical federate working in the background. All the RTI calls employed by a virtual federate call services via the corresponding physical federate. Various communication channels may be used to bridge the two processes together into a simulator. These processes may execute on the same processor, on different processors of a shared memory multiprocessor or even computers on a cluster or network. Low level communications (Socket, IPC, Fast Message, MPI etc.) may ensure efficiency. In contrast, high level communications (COBRA, Web service etc.) can enhance the system's interoperability, and provide easy utilization, management and deployment. No matter what communication channels are used, the approach ensures the intactness of existing simulation federate codes.

To investigate the overhead incurred by the decoupled approach, this paper first presents a set of experiments to benchmark the latency and efficiency of synchronization. Experimental results for the decoupled federates and normal federates are compared. The results indicate a promising performance of the decoupled federates in both types of benchmarks. The paper then presents solutions to the three technologies, i.e., distributed simulation cloning, fault tolerance as well as Web and Grid Enabled Architecture. The solutions are established upon the Decoupled Federate Architecture. Several series of experiments have been carried out to evaluate these solutions.

A shorter version of this paper was presented in the 15th European Simulation Symposium [4] in the context of a Decoupled Federate Architecture for distributed simulation cloning. The rest of this paper is organized as follows: Section 2 outlines related work on the research issues addressed by the Decoupled Federate Architecture approach. Section 3 introduces the background of this study and analyzes the problems. Section 4 details a design and implementation of the decoupled approach. This section also presents the benchmark experiments on the architecture and analyzes the results. Section 5 discusses the solutions to resolving the above research issues and gives the experimental results on system performance. Section 6 concludes with a summary and proposals on future works.

2. Related work

Morse et al. proposed an Extensible Modeling and Simulation Framework[1] (XMSF) to allow various simulations to take advantage of Web-based technologies [23,24]. XMSF makes use of Web-based technologies, applied within an extensible framework, that enables a new generation of modeling & simulation (M&S) applications to emerge, develop and interoperate. One of its important initiatives is to develop a web enabled RTI. Within the XMSF framework, multiple federates reside as web services on a WAN and the federation's Federation Object Model[2] (FOM) is mapped to an XML tagset, allowing interoperation with other distributed applications supported by web services. The federates communicate using the Simple Object Access Protocol (SOAP) and the Blocks Extensible Exchange Protocol (BEEP). Several obvious advantages are identified, such as: platform crossing, language crossing and interoperability

[1] http://www.movesinstitute.org/xmsf/xmsf.html.

[2] The primary function of a FOM is to specify, in a common standardized format, the nature of the data exchange among federates.

of federates (for example, breaking though the constraints of firewalls etc.).

Straßburger et al. presented different solutions, including a gateway program approach to adapt models built using Commercial-off-the-shelf (COTS) simulation packages to the HLA [27]. Under this approach, a simulation application consists of a COTS-based model and a gateway program that couples the model with the RTI. Thus, previously standalone models may interoperate with each other using the HLA specification, with user transparency maintained. Similarly, McLean and Riddick designed a Distributed Manufacturing Adapter to integrate legacy simulations with the HLA/RTI [21]. This design aims to simplify the integration while gaining capabilities of HLA-based distributed simulations.

Hybinette and Fujimoto proposed using simulation cloning technology as a concurrent evaluation mechanism in the parallel simulation domain [16]. The motivation for this technique was to develop a cloning mechanism that supports an efficient, simple, and effective way to evaluate and compare alternate scenarios. The method was targeted for parallel discrete event simulators that provide the simulation application developer with a logical process (LP) execution model.

Fault tolerant techniques often employ redundant/backup components to achieve system robustness, such as the Replica Federate approach proposed in [1]. Their approach produces multiple identical instances of one single federate, and failures can be detected and recovered upon the outputs of those identical instances. However, redundancy is liable to result in lower system performance. Furthermore, extra federate replicas in a distributed simulation increase the probability of overall system failure incurred by an LRC failure.

Significantly different from above work, this project focuses on developing a generic infrastructure, which addresses three independent issues: (1) enabling distributed simulation cloning, (2) supporting fault tolerance and (3) incorporating Web/Grid technologies in the HLA. However, this is difficult as these technologies are heterogeneous to the HLA. Reusability is one of the key benefits of HLA-based simulation [10]. This study also aims at providing reusability, thus it minimizes the effort for users to adopt these technologies in developing federates or taking advantage of legacy federates. Undoubtedly, this requirement poses more complications in this task.

3. Background and problem statement

Section 3.1 introduces the concepts related to High Level Architecture (HLA) followed by the way in which HLA-based simulations form and operate with the support of Runtime Infrastructure software. In the rest of Section 3, we elaborate the background and the major problems with respect to the focus of this study, i.e., to facilitate solutions based the Decoupled Federate Architecture to distributed simulation cloning, fault-tolerant distributed simulation and Web/Grid enabled simulation. Section 3.2 presents the idea of distributed simulation cloning, and presents the crucial problems to be tackled for cloning federates. Section 3.3 analyzes the risk of RTI failure and discusses several research challenges to support a generic fault-tolerant mechanism to HLA-based simulations. In Section 3.4, we discuss the potential benefits and feasibilities of integrating the Web or Grid technologies with the HLA technologies.

3.1. High level architecture and runtime infrastructure

The HLA defines a software architecture for modeling and simulation, which is designed to provide reuse and interoperability of simulation components. The simulation components are referred to as federates. A simulation federation can be created to achieve some specific objective by combining simulation federates. The HLA supports component-based simulation development in this way [11]. The HLA federation is a collection of federates (observers, live participants, simulators etc.) interacting with each other for a common purpose, for example wargaming. These federates interact with each other with the support of the Runtime Infrastructure (RTI), and the use of a common FOM. In the formal definition, the HLA standard comprises four main components: HLA Rules, Object Model Template (OMT), Interface Specification [17] and Federation Development and Execution Process (FEDEP) [18]. The HLA rules define the principles of HLA, in terms of responsibilities that federates and federations must uphold. Each federation has a FOM, which is a common object model for the data exchanged between federates in a federation. The OMT defines the metamodel for all FOMs [20]. The HLA Interface Specification identifies the RTI services available to each federate and the functions each federate must provide to the federation. The FEDEP mainly defines a process framework for federation developers, including the necessary activities to build HLA federations.

The HLA is an architecture defining the rules and interface, whereas the Runtime Infrastructure (RTI) is the software conforming to the HLA standard, that is used to support a federation execution. Fig. 3.1 gives an overview of an HLA federation and the RTI. The RTI provides a set of services to the federates for data interchange and synchronization in a coordinated fashion. The RTI services are provided to each federate through its Local RTI Component (LRC) [12]. The RTI can be viewed as a distributed operating system providing services to support interoperable simulations executing in distributed computing environments [14]. A total of six service categories are defined in the specification, namely Federation Management, Declaration Management, Object Management, Ownership Management, Data Distribution Management and Time Management [12]. The RTI services are available as a library (C++ or Java) to the federate developers. Within the RTI library, the class *RTIAmbassador* bundles the services provided by the RTI. A federate may invoke operations on the interface to request a service (federate-initiated service) from the RTI. The *FederateAmbassador* is a pure virtual class that identifies the "callback" functions each federate is obliged to provide (RTI-initiated) service. Federate developers need to implement the *FederateAmbassador*. The callback functions provide a mechanism for the RTI to invoke operations and communicate back to the federate.

3.2. Problems in cloning federates

When reaching a decision point, a federate has different choices to examine. Instead of simulating each choice from the start, distributed simulation cloning technology can be used to replicate the federate, and allow the replicas to examine these choices concurrently from the decision point onwards. A decision point represents the location in the execution path where the states of the system start to diverge in a cloning-enabled simulation. "Cloning" differs from simple "replication" in the sense that clones of the original federate execute in different "paths" rather than simply repeat the same executions. Even though the computation of clones is identical at the decision point, from the decision point onwards, each clone explores one particular path together with its partner federates to form an independent scenario which represents a unique decision in the simulated physical system [2]. This provides the user with the opportunity to evaluate multiple

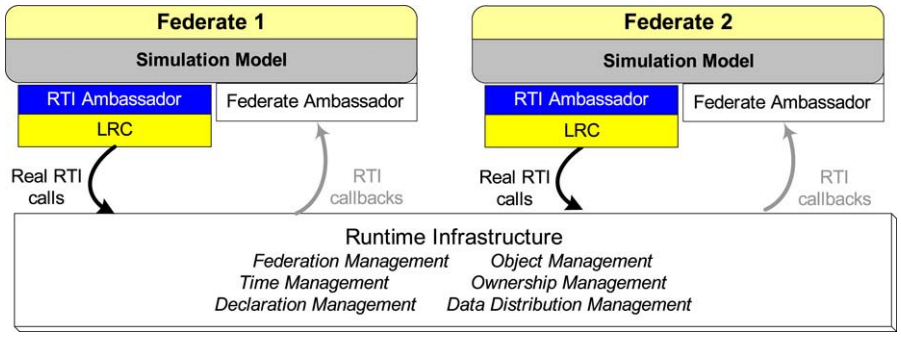


Fig. 3.1. Architecture of HLA-based distributed simulation.

alternative results concurrently, using the same simulation models within a single session. Thus execution time can be reduced, and the analyst may quickly obtain multiple sets of results that represent the impacts of alternative decisions.

The new clones of one particular federate should initially have the same features and states as the original federate, both at the RTI level and at the simulation model level. This is to ensure the consistency of simulation state. For example, at the RTI level, clones must have subscribed to the same object classes and registered the same object instances etc. At the simulation model level, the clones should have the same program structure, data structures, objects and variables; all these program entities should have identical states. Immediately after the cloning, the clones will be given some particular parameters or routines to execute in different paths.

One possible solution is to introduce a state saving and replication mechanism to the simulation federates, allowing the simulation federate to store snapshots of all the system states. When cloning occurs, new federate instances are started and initialized with stored states. However, users model their simulations in a totally free manner. It is unlikely that a generic state saving and replication mechanism can be provided that will be suitable for any simulation federate. Besides, as the LRC is not designed to be replicated, direct cloning of a federate can lead to unpredictable and uncontrollable failure at the RTI level.

Even given such a mechanism, it is unlikely that all simulation developers will use the same standard package to model their simulations. Without the ability to customize the user's simulation code, it is almost impossible to make snapshots of all system states of any federate. Furthermore, the principle of reusing existing federate code increases the difficulty of this task. On the other hand, the standard HLA specification makes it relatively easy to intercept the system states at the RTI level. Using a middleware approach, one may save and replicate the RTI states while enabling transparency. Thus we can see that the simulation model and the LRC have very different characteristics. Therefore, it suggests a distinction should be made between these two modules for cloning a federate.

3.3. Problems in enabling fault tolerance

Many factors can contribute to the failure of a federate, for example, network congestion, platform crash of the RTIEXEC process [12] etc. People have developed many technologies for facilitating fault tolerance in distributed applications. Cristian pointed out some principles about fault-tolerance in distributed system architectures [9], and these are: understanding failure semantics, masking failure and balancing design cost.

The checkpoint and message-logging approach is an alternative to the replication methods described in Section 2. As proposed in [19], a process records each message received in a message log while the state of each process is occasionally saved as a checkpoint. Message logging can be pessimistic, assuming the occurrence of a failure at any time, or optimistic, assuming that a message will always be properly delivered before process failure. A failed process can be restored using some previous checkpoint of the process and the log of messages. This approach assumes the process execution to be deterministic between received input messages.

Although the RTI supports the federation save and restore services, the HLA specification does not support or propose a fault tolerance mechanism for HLA-compliant simulations [11]. In the context of a checkpoint or simulation migration approach, it means a federate has to be specially modeled. For example, given that a checkpoint method is adopted, a federate should have the additional functionalities to save its simulation model's states and recover from the last checkpoint. It is desirable to allow users to model their federates freely while enabling robustness. To optimize execution, it is preferable that the fault tolerance approach can handle the failure of the RTI immediately, without repeating or disrupting the execution of the simulation model.

A normal federate usually exists as a single process at runtime, and the simulation model shares the same memory space with the LRC (see Fig. 1.1). In the case where the RTI crashes or meets congestion, the failure of any LRC prevents the simulation execution from proceeding correctly even though the simulation model contains no error at all. Besides the challenge to develop a generic state saving and recovery mechanism, the fault-tolerant model requires an approach to "isolate" the error of the LRC from the simulation model. On the other hand, the HLA standard makes it relatively easy to intercept the system states at the RTI level using a middleware approach. Furthermore, we can see that the simulation model and the Local RTI Component have very different characteristics. Therefore, it suggests a distinction should be made between these two modules when dealing with failure.

3.4. Web or grid enabled architecture

The Grid infrastructure provides dependable, consistent, pervasive, inexpensive and secure access to high-end computational capabilities [13]. The Open Grid Services Architecture (OGSA) extends the Web services to include additional functionalities. With these features, Web and Grid services offer a promising approach to enhance the flexibility and interoperability for large scale HLA-based simulations.

The HLA does not define security mechanisms for using RTI services, while the Grid provides a built-in security architecture to support authentication and secure communication in accessing Grid services [15].

The HLA supports interoperability among HLA-compliant federates. However, this feature is often hampered in the context of large scale distributed simulations over the Internet. This is due to various security requirements on the networking resources

belonging to different organizations. There is a need to combine the HLA and Web or Grid technologies, in order to achieve the advantages of both. Directly utilizing Web or Grid services will incur extra effort from the developers in modeling simulations. An alternative approach is to use a *Web* or *Grid Enabled Architecture*, which manages the RTI services at the backend, while presenting dynamic access of the RTI services to the simulation model or the end users via Web or Grid services. The architecture can be used to relieve the developers from the complication of coding Web or Grid services in existing simulation models.

4. Decoupled federate architecture

To address the issues discussed in Section 3, a Decoupled Federate Architecture has been designed to separate the simulation model from the Local RTI Component. Section 4.1 presents the basic idea of the Decoupled Federate Architecture with notations defined. Section 4.2 details a candidate design of the architecture and describes the operation of a federate supported by the architecture. Section 4.3 depicts a RTI++ middleware, i.e., the implementation of the architecture. Section 4.4 introduces a series of experiments to benchmark the performance of the architecture using the candidate design.

4.1. Virtual federate and physical federate

In the context of the decoupled architecture, a federate's simulation model is decoupled from the Local RTI Component. A virtual federate is built up with the same code as the original federate. As HLA only defines the standard interface of RTI services, we are able to substitute the original RTI software with our customized RTI++ library (also see Section 4.3) without altering the semantics of RTI services [2]. Fig. 4.1(B) gives the abstract model of the virtual federate. Compared with the original federate model illustrated in Fig. 1.1, the only difference is in the module below the RTI interface, which remains transparent to the simulation user.

A physical federate is specially designed as shown in Fig. 4.1(A). The physical federate associates itself with a real LRC. Physical federates interact with each other via a common RTI. Both virtual federates and physical federates operate as independent processes. Reliable external communication channels, such as Inter-Process Communication (IPC), Sockets, SOAP or other out-of-band communication mechanisms, bridges the two entities into a single federate executive [28]. To ease discussion, this paper uses the abbreviation "*ExtComm*" to denote those alternative communication channels and assumes messages are delivered/received strictly in FIFO manner.

All the components inside the dashed rectangle form a *Middleware* module between the simulation model and the real RTI. Within the virtual federate, the *RTIPlus* contains customized libraries while presenting the standard RTI services and related helpers to the simulation model. This module can be designed to contain all other management modules for the objectives mentioned previously. The *fedAmb* serves as a common callback (FederateAmassador) to the user federate, which is freely designed by the user and independent of the decoupled approach. The RTIPlus handles the user's RTI service calls by converting the method together with the associated parameters into ExtComm messages via the *Messaging Protocol*. The protocol mainly defines a mapping between an ExtComm message and the RTI method it represents. For example, an RTI_UPDATE message indicates that the virtual federate has invoked the RTI method *updateAttributeValues()*. The protocol can also be extended for other purposes, such as manipulating the physical federate. The ExtComm conveys these messages immediately to the physical federate for processing in a FIFO manner.

The physical federate converts an RTI call message generated from the virtual federate into the corresponding RTI call through its own messaging protocol layer. The *RTIAmb* module executes any RTI service initiated by the simulation model prior to passing the returned value to the ExtComm. The *phyFedAmb* serves as the callback module of the physical federate to respond to the invocation issued by the real RTI. Within the phyFedAmb module, the messaging protocol is employed to pack any callback method with its parameters into ExtComm messages. The ExtComm enqueues the callback message to the *Callback Processor* module at the virtual federate. Through the messaging protocol, the callback processor activates the corresponding fedAmb method implemented by the user. From the simulation users' perspective, a combination of one virtual federate and its corresponding physical federate operates as a simulation federate in the context of the decoupled architecture. The federate combination performs an identical execution to a normal simulation federate with the same code as the virtual federate. In future discussion, we will explicitly use "normal federate" to refer to a traditional federate that directly interacts with the RTI. By default, in the discussion of this paper a federate contains a virtual federate module and a physical federate module.

4.2. Inside the decoupled architecture

As discussed above, the decoupled approach interlinks a virtual federate and the physical federate into a simulator that performs an identical simulation to the corresponding normal federate. This section gives the details of how an RTI service call is executed and the callback is invoked in the Decoupled Federate Architecture.

Fig. 4.2 depicts the procedure where a simulation model initiates an RTI call and waits for a return from the real RTI. For example, the virtual federate invokes the redefined *updateAttributeValues()* method. The data associated with this call are packed into an RTI_UPDATE message, and sent over to the physical federate. The latter then recovers a new *AHVPS* [12] object and related parameters, and those are passed to the *RTI::updateAttributeValues()* accordingly, which invokes the real RTI service. On the accomplishment of this *RTI::updateAttributeValues()* call, the physical federate acknowledges the virtual federate with an ExtComm message containing the returned value. Immediately after that, the initial *updateAttributeValues()* call finishes and the returned value is conveyed to the simulation model. From the user's point of view, the initiation and accomplishment of an RTI call are same as the original normal federate. The semantics of RTI services are kept intact in the decoupled approach.

The RTI software has an interface that provides flexible methods to the user for packing update data, and leaves the transmission details transparent. The user can create update data of variable lengths. However, most low level communications do not offer flexible methods to handle message delivery without agreement of lengths between source and destinations. For example, most IPC mechanisms have limitations in message size and buffer size. The *Message Queue* based on Solaris defines the maximum queue length as 4096 bytes [28]. The message sender and receiver must agree with each other on the same message length. If a fixed message size is defined for ExtComm messaging, it may incur some unnecessary overhead. A fixed large size is inefficient in transmitting small messages. On the other hand, a fixed small size increases the overhead for packing, delivering and unpacking a large number of small packets in the case of processing large messages. Therefore a simple protocol is proposed for messaging between the virtual federate and physical federate. We define a small message size (MSG_DEF) and a large message size (MSG_LG) for assembling user data into packets. A special "PREDEFINE" packet is used to notify the receiver if large or multiple packets

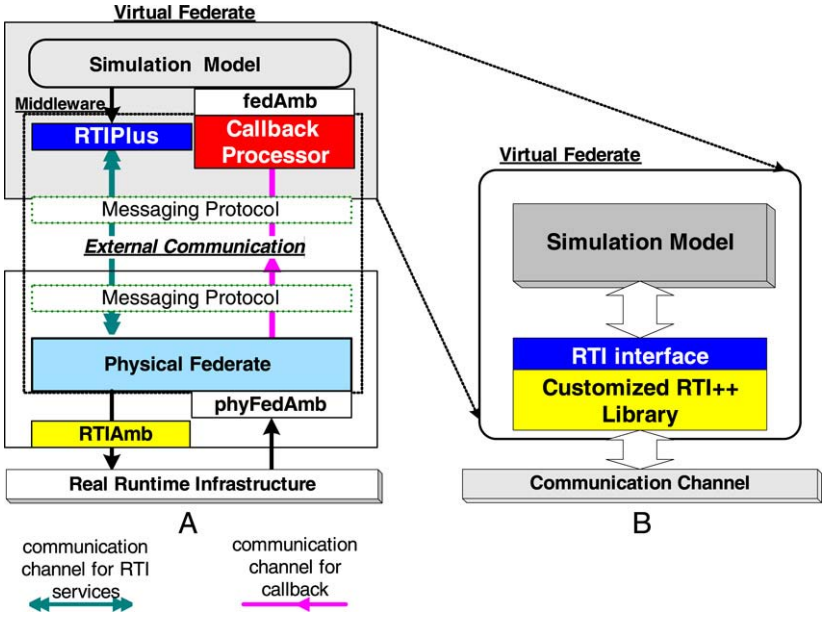


Fig. 4.1. Decoupled federate architecture.

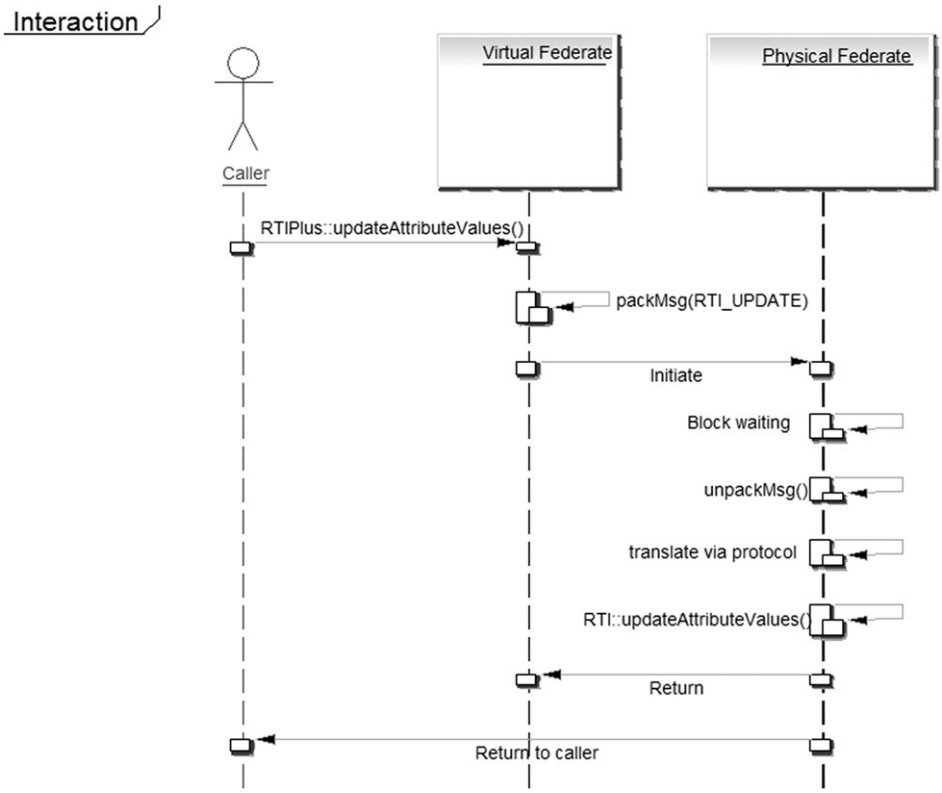


Fig. 4.2. Executing an RTI call in the decoupled architecture.

are to be sent for a single data block. Fig. 4.3 gives the messaging details based on this simple protocol.

The RTI communicates with a federate via its Federate Ambassador provided by the user [12]. In the DMSO RTI, a federate must explicitly pass control to the RTI by invoking the *tick()* method. For example, the RTI delivers the Timestamp Order (TSO) events and Time Advance Granted (TAG) to a time-constrained federate in strict order of simulation time, which coordinates event interchange among time regulating and time constrained federates in a correct and causal manner. Therefore, the decoupled architecture should guarantee that (1) the Federate Ambassador at the user federate works in a callback like manner and (2) callback methods are invoked in the correct order. Fig. 4.4 depicts how to realize these functionalities. To ease discussion, we assume the physical federate will get the callbacks shown in Fig. 4.4.

The Virtual federate invokes the routine *RTIPlus::tick()*, and an RTI_TICK message is sent to the physical federate which triggers the real RTI *tick()* call. The Local RTI Component acquires control and delivers events to the *phyFedAmb* module of the physical federate in a strict order. In each callback method invoked, the

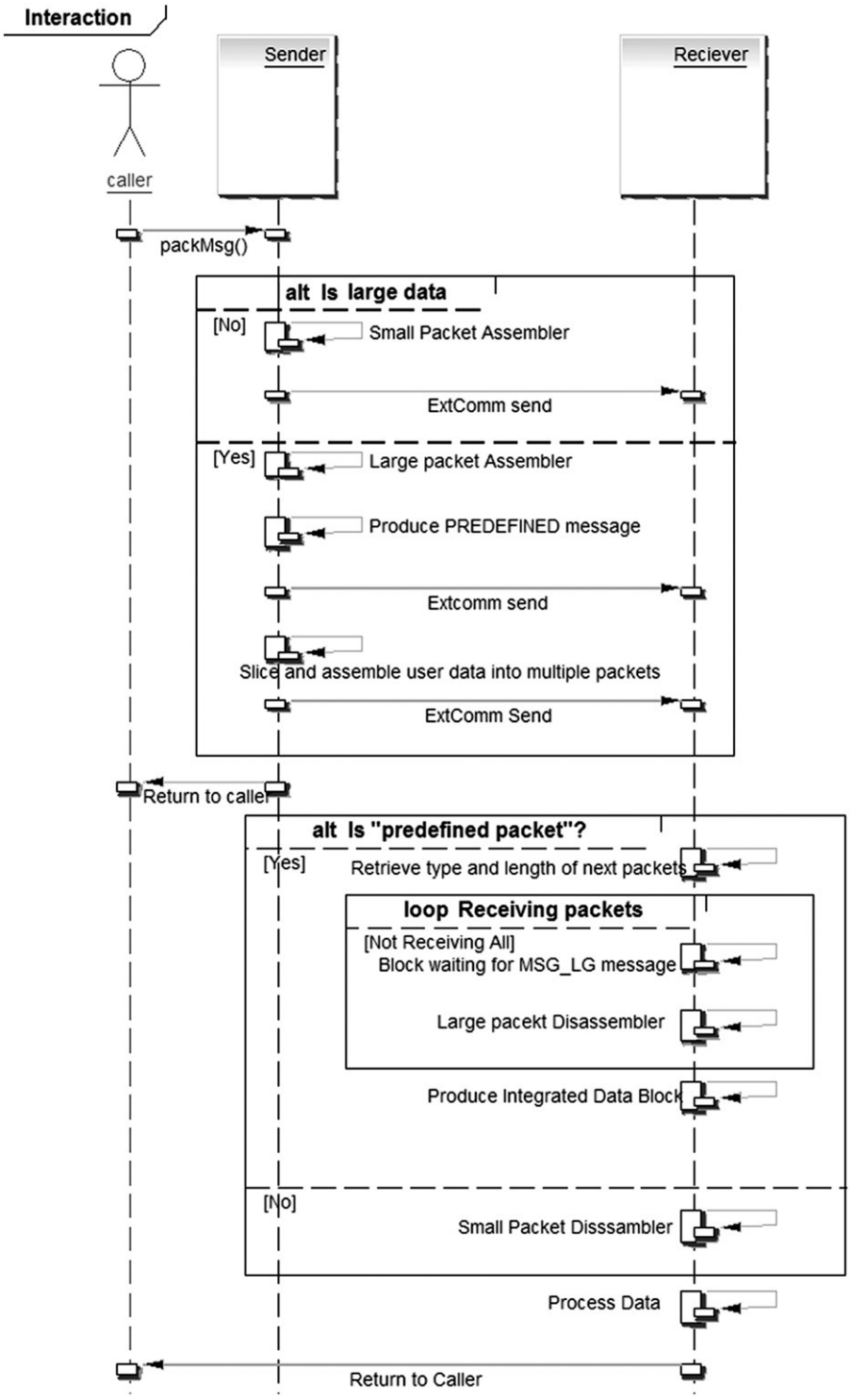


Fig. 4.3. Messaging between virtual federate and physical federate.

data sent by the RTI is enqueued to the callback *ExtComm* channel. The Callback Processor continuously processes the messages in a FIFO order, while activating the corresponding method in the *fedAmb* module. At the physical federate side, once the RTI passes control to the physical federate, the latter returns an RTI_TICK_DONE message to the virtual federate. Thus, the virtual federate accomplishes the *RTIPlus::tick()*, and control is returned to the caller immediately.

The real RTI starts to take charge only when the physical federate explicitly invokes *RTI::tick()*. On the other hand, the *RTIPlus::tick()* can only return when the real RTI finishes its work. As the *ExtComm* channels work in a FIFO manner, the order of each callback method invoked at the physical federate is identical to the sequence in which the callback processor at the virtual federate processes the data. From the user's perspective, the callback mechanism based on the decoupled approach executes

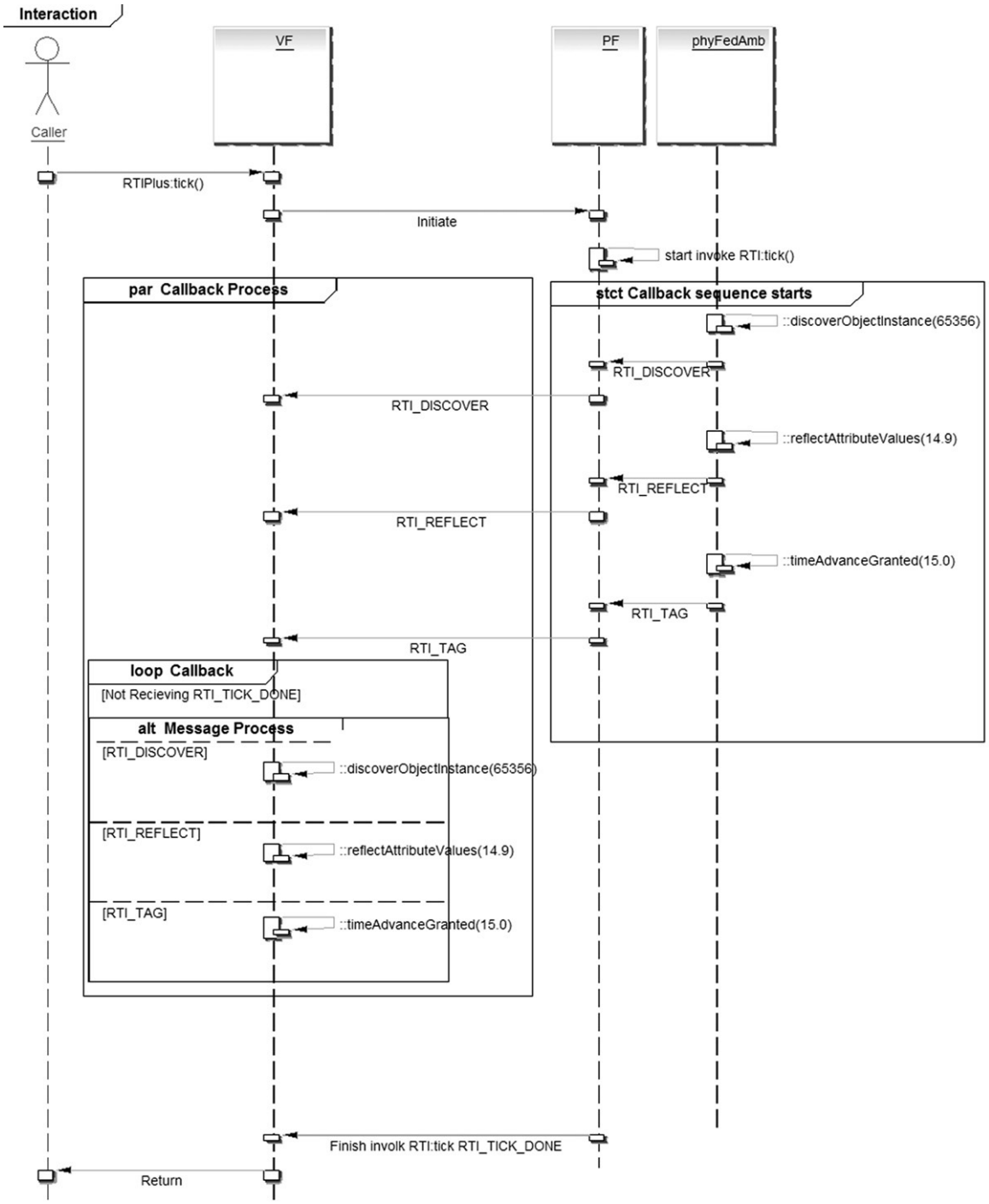


Fig. 4.4. Conveying callbacks to the virtual federate.

the equivalent operations to the normal federate. It guarantees consistency in presenting messages from the real RTI to the simulation model and also ensures user transparency.

The decoupled architecture requires an additional ExtComm communication layer, although it performs exactly the same computation as the corresponding normal federate. The external communication may incur some extra overhead. To investigate the overhead incurred by the decoupled approach, a series of benchmark experiments has been performed to compare with the normal federates. Section 4.4 reports and analyzes the experimental results in terms of communication overhead and synchronization efficiency.

4.3. Middleware and the RTI++ Library

Conforming to the critical principle of reusing the user's programs, our decouple approach minimizes or avoids the modification to the user's existing code. As also indicated in Fig. 4.1, a middleware approach has been adopted to hide the complexity. The middleware is designed as an RTI++ library which extends the standard RTI to encapsulate the enhanced functionality and all operations directly related to the RTI while presenting the standard RTI interface to the user. The physical federate is also regarded as a component of the middleware.

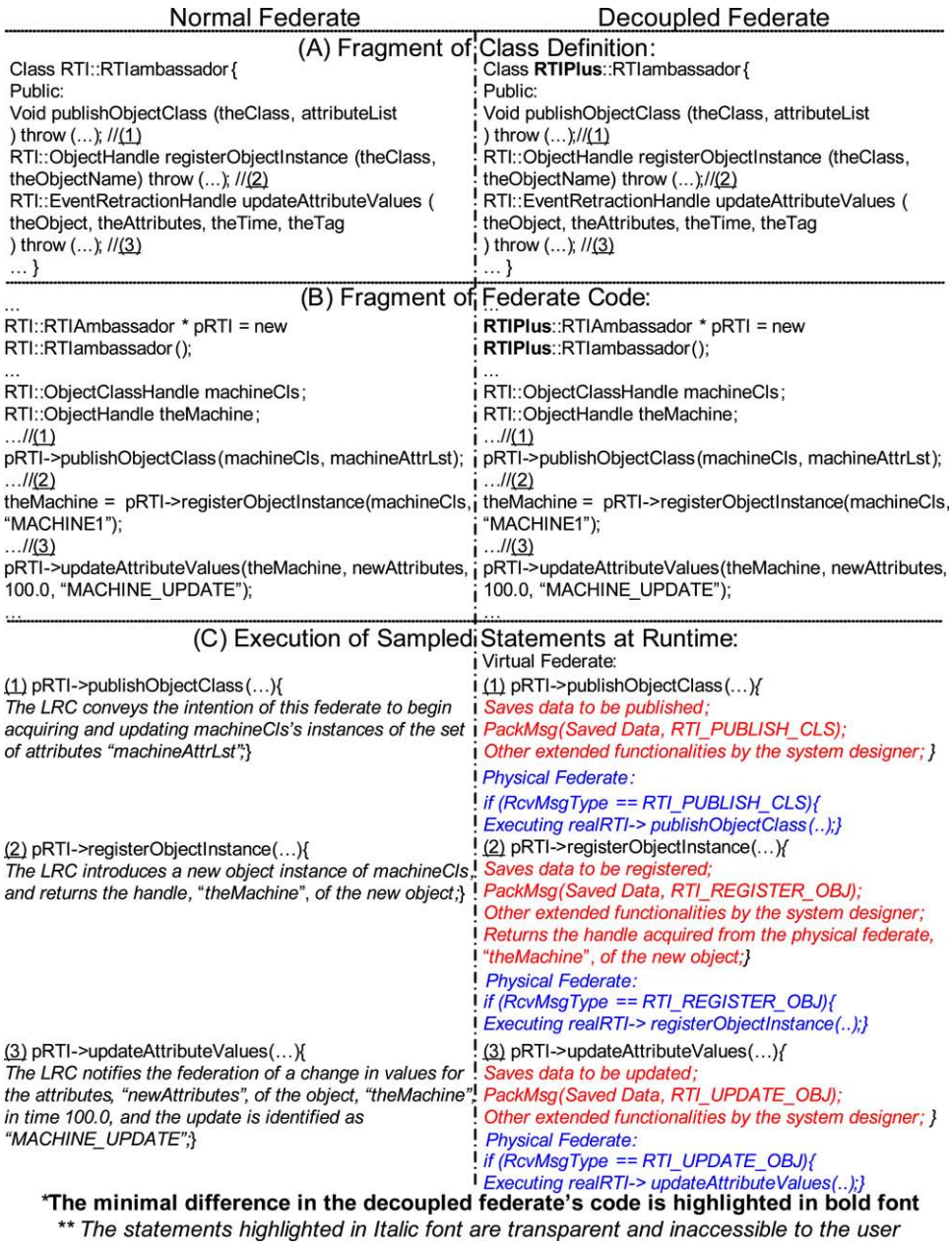


Fig. 4.5. Illustration of the RTI++ library and execution of the middleware.

Any of the solutions to be discussed in Section 5 should work as an underlying control mechanism transparent to federate codes. Hiding the implementation details while enabling the management of scenarios can maximize the reusability of the user's existing simulation model. Fig. 4.5 gives the pseudo code of an implementation of middleware. The left side represents the code for a normal federate while the right column illustrates the counterpart for the corresponding decoupled federate.

Fig. 4.5(A) shows the fragment of the definition of RTI services for a normal federate and the decoupled federate, and the RTI++ class is different only in the name. As the RTI++ adopts an identical definition of FederateAmbassador class to the normal RTI, introduction to this class is therefore omitted in this paper. Fig. 4.5(B) illustrates the implementation of a federate of two versions. The difference[3] lies in the declaration of the RTIAmbassador instance.

Three RTI calls are highlighted for a comparison at runtime between the normal federate and the decoupled federate, as shown in Fig. 4.5(C) with indexes (1), (2) and (3). For example, once the 2nd call *registerObjectInstance()* invoked by the normal federate, LRC takes control and introduces a new object instance named "MACHINE1"of class *machineCls* into the federation and returns the object handle to the variable *theMachine*. As for the decoupled federate, the virtual federate invokes the call, and the RTI++ middleware takes control and starts following private routines inaccessible to the simulation developer: (a) saving arguments passed to the call as system states, (b) sending an "RTI_REGISTER_OBJ" message to the physical federate, (c) the physical federate being instructs to execute the RTI call *registerObjectInstance()* to obtain the object handle, (d) extended functionalities by the designer of the decoupled architecture and (e) returning object handle to the variable *theMachine*. Routine (b) may adopt alternative communication backbones, and routing (d) can be customized by us for various purposes, i.e., support of cloning, fault tolerance and Web/Grid enabled architecture in this study.

[3] These minor differences do not exist in our Java implementation of Grid Enabled Architecture.

Table 4.1
Configuration of experiment test bed

Specification	Computers	
	Workstation1&2	Server1
Operating System	Sun Solaris OS 5.8	Sun Solaris OS 5.8
CPU	Sparcv9 CPU, at 900 MHz	Sparcv9 CPU * 6, at 248 MHz
RAM	1024M	2048M
Compiler	GCC 2.95.3	GCC 2.95.3
Underlying RTI	DMSO NG 1.3 V6	DMSO NG 1.3 V6
Processes running on	*FedX*[*i*] & *Y*[*i*]	RTIEXEC & FEDEXE

The executions of the two versions of the federate are the same from the simulation user's point of view.

4.4. Benchmark experiments and results for a candidate design

The decoupled architecture separates the simulation model and the LRC into independent processes. In contrast, in a normal federate these operate within the same memory space. In order to investigate the overhead incurred in the proposed architecture due to decoupling, we perform a series of benchmark experiments to compare the decoupled federate with a normal federate. The experiments study the scalability, by emulating the simulation cloning procedure based on the decoupled architecture using the IPC *Message Queue* as the external communication to bridge the virtual and physical federate. The performance is compared in terms of latency and time advancement calculation. Latency is reported as the one-way event transmission time between one pair of federates. The time advancement performance is represented as the time advance grant rate.

4.4.1. Experiment design

The experiments use three computers in total (two workstations and one server), in which the server executes the RTIEXEC and FEDEX processes (see Fig. 4.6). The federates that run at one independent workstation are enclosed in a dashed rectangle. In our case *Fed X*[*i*] and *Fed Y*[*i*] ($i \geqslant 0$) occupy workstation 1 and workstation 2 respectively. The computers are interlinked via a 100 Mbps-based backbone. Table 4.1 gives the configurations of the test bed.

The experiments study the scalability by increasing the number of identical federates. As shown in Fig. 4.1, *Fed X*[1] and *Y*[1] form a pair of federate partners, which represent the initial federation scenario. *Fed X*[*i*] and *Y*[*i*] ($i > 1$) stand for the *i*th clones of the two original federates respectively and form an individual scenario. The architecture is used through all the benchmarks experiments and for both normal federates and decoupled federates.

A Data Distributed Management (DDM) based approach is used to partition concurrent scenarios [2]. For the latency benchmark, each pair of federates has an exclusive point region associated to any event being exchanged. The federates are neither time regulating nor time constrained. In one run, each federate updates an attribute instance and waits for an acknowledgment from its partner (from *Fed X*[*i*] to *Fed Y*[*i*], and vice versa) for 5000 times with a **payload of 100, 1 k and 10 k bytes**. The time interval in the ping-pong procedure will be averaged and divided by 2 to give the latency in **millisecond**s. A federate merely reflects the events with identical region to itself. In other words, *Fed X*[*i*] only exchanges events with *Fed Y*[*i*].

As for the time advancement benchmark, all federates are time regulated and time constrained. Each federate has a lookahead 1.0 and advances the federate time from 0.0 to 5000.0 with timestep 1.0 using *timeAdvanceRequest()* [12]. The results report the rate (TAGs/Second) that the RTI issues *timeAdvanceGranted()*.

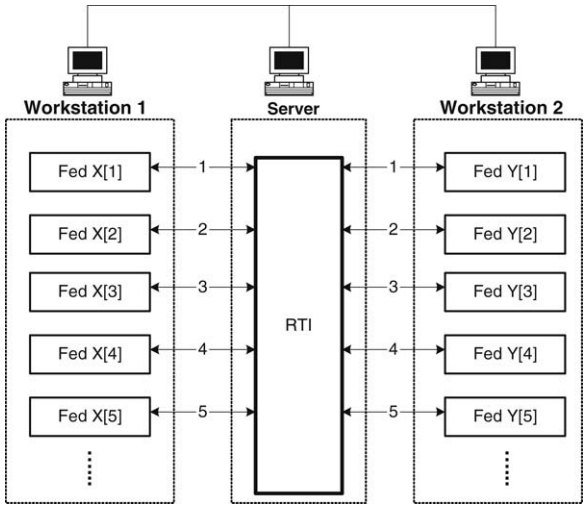


Fig. 4.6. Architecture of benchmark experiments.

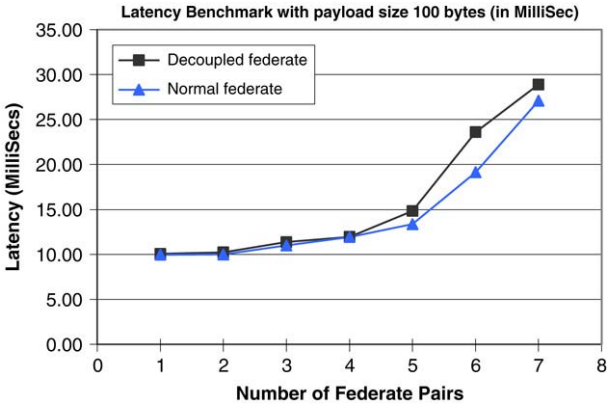


Fig. 4.7. Latency benchmark on decoupled federate vs. normal federate with payload 100 bytes.

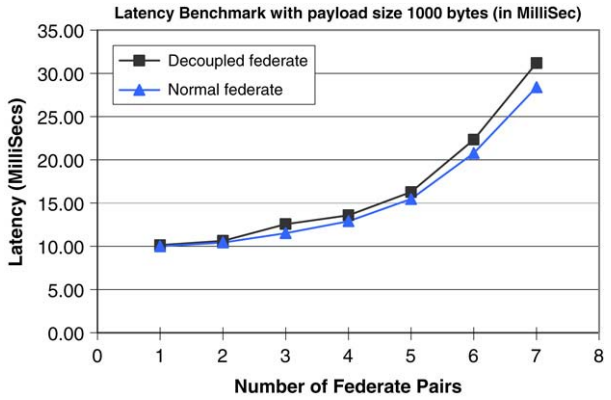


Fig. 4.8. Latency benchmark on decoupled federate vs. normal federate with payload 1000 bytes.

4.4.2. Latency benchmark results

The latency benchmark experiments report the latency with three different payload sizes. From Figs. 4.7 to 4.9, we can see that no matter whether the payload size is small or large, latency increases steadily with the number of federates. The increment becomes obvious when the number of federates exceeds 4 pairs (8 federates in total).

As indicated in Figs. 4.7 and 4.8, when the payload is not greater than 1000 bytes, the latency varies from about 10 ms for one pair

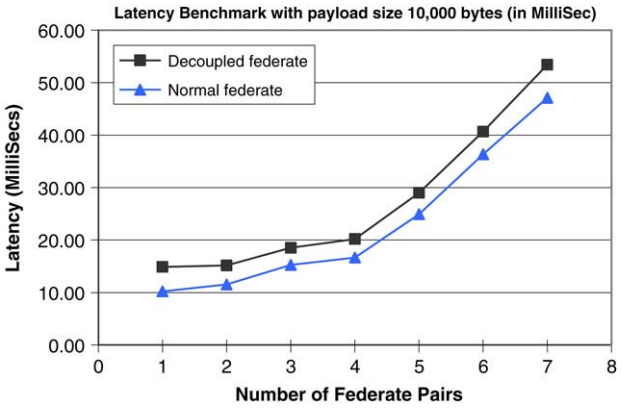


Fig. 4.9. Latency benchmark on decoupled federate vs. normal federate with payload 10,000 bytes.

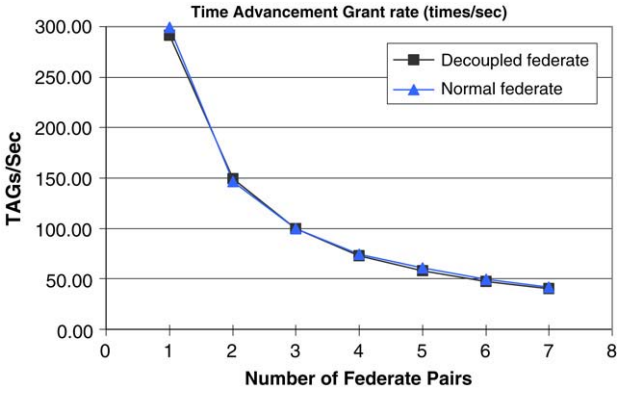


Fig. 4.10. Time advancement benchmark on decoupled federate vs. normal federate.

of federates to about 30 ms for 7 pairs of federates. The decoupled federate and normal federate show similar results in this situation, and the decoupled federates incur only slightly more latency than the normal ones.

As shown in Fig. 4.9, when a bulky payload as large as 10,000 bytes is applied, the decoupled federates incur about 5 ms extra latency to the normal ones. However the extra latency remains nearly constant with the number of federate pairs. The latencies for both types of federates increase more rapidly than the small payload cases. This is due to the extra overhead incurred by Inter-Process Communication, which becomes obvious with bulky data transmission between the physical federate and virtual federate. When the payload size and the number of participating federates are not too large, the decoupled federate has a similar performance to the normal federate in terms of latency.

4.4.3. Time advancement benchmark results

Another series of experiments is carried out to compare the decoupled federates and normal federates in terms of the speed of time synchronization. Fig. 4.10 presents the experimental results reported as the TAG rate. In the time advancement benchmark, the TAG rate decreases with the number of federates for both decoupled and normal federates. The rate decreases less rapidly when the number of federate pairs is greater than 4 (8 federates in total). The TAG rate is about 300 times per second for one pair of federates down to about 40 times per second for 7 pairs of federates. The results indicate that the performance of the decoupled and normal federates is very similar in terms of time advancement.

5. Solutions developed upon the architecture

This section has a general discussion on the major uses of the Decoupled Federate Architecture. An independent solution together with performance results are presented for each issue. Section 5.1 introduces a design for distributed simulation cloning. A simply supply chain simulation has been used to examine the effectiveness of the design. Section 5.2 discusses a generic fault tolerant mechanism. Experiments have been conducted to measure the overhead of enabling fault tolerance. Section 5.3 describes the design of Web or Grid enabled architectures. Two series of benchmark experiments on Grid enabled federates and Grid enabled federations respectively have been performed to investigate the performance and scalability of the Grid Enabled Architecture.

5.1. Distributed simulation cloning

Distributed simulation cloning technology aims to provide much more powerful and flexible decision support than traditional "linear" simulation. Using the decoupled approach solves the problem of making copies of LRC at runtime in the case of executing traditional federates. There exist alternative approaches to cloning a federate. We can replicate a simulation model, using COTS packages which support state saving/recovery. Another way is to fork the virtual federate process to create its exact replicas. Each of the replicas executes an instance of the simulation model with identical initial states. New physical federate instances can be started with restored system state at the RTI level to serve the replicated models.

As illustrated in Fig. 5.1, at runtime the middleware intercepts the invocation of each RTI service method. The RTI States manipulator saves RTI states immediately before passing an RTI call to the physical federate to execute it.[4] After that, the RTI states Manipulator logs all the RTI system states into stable storage. Some RTI states are relatively static, such as the federate identity, federation information, the published/subscribed classes and time constrained/regulating status. Other states include the registered or deleted objected instances, and granted federate time. Some event data may also need to be saved, such as sent and received interactions, updated and reflected attribute values of object instances, etc.

As soon as a federate reaches a decision point, cloning conditions are checked to decide whether cloning is required or not. If necessary, the middleware spawns clones of the federate immediately, to explore alternative execution paths. From the perspective of the federate making clones, the process of a simulation cloning can be described as follows (see Fig. 5.2).

- **Copying simulation model**. The cloning manager within the middleware makes the specified number of clones of the virtual federate (simulation model).
- **Initiating physical federates.** The cloning manager initiates an individual physical federate for each clone of the virtual federate and hooks up the two new processes into a whole.
- **Replicating states.** A clone's physical federate is initialized with the stored system states from the parent federate, after which a new clone of the original federate is formed.

[4] For example, when the virtual federate invokes *publishObjectClass()*, the RTI States Manipulator intercepts this call and saves the information, after which it will call the physical federate via ExtComm.

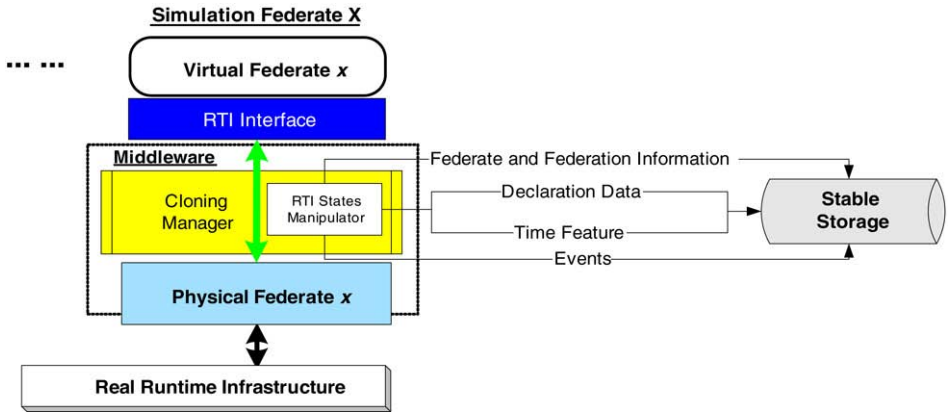


Fig. 5.1. Saving RTI states.

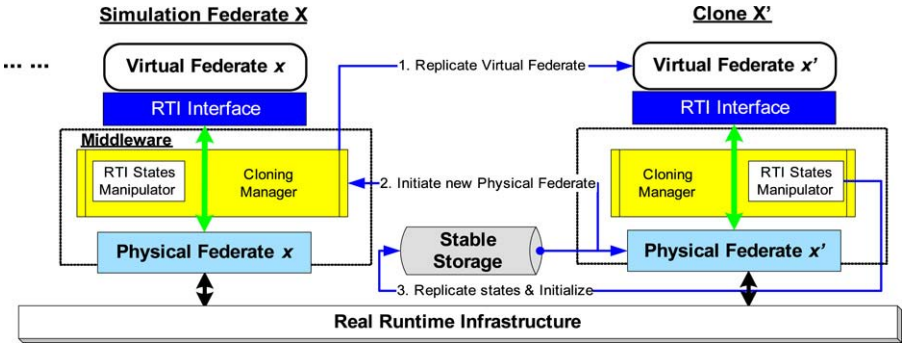


Fig. 5.2. Federate cloning.

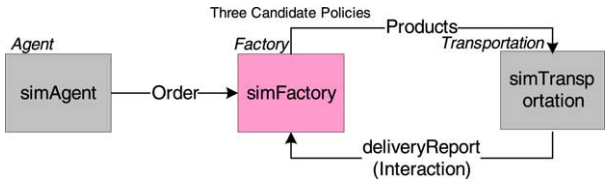


Fig. 5.3. A simple distributed supply chain simulation.

Those COTS packages that already provide state saving of the simulation model can easily employ the Decoupled Federate Architecture to perform model cloning. In the future when standards for interoperation amongst different COTS packages are established, simulation cloning can be a desirable scheme to optimize execution for those COTS packages that support state saving [30].

The theory and algorithms of distributed simulation cloning are detailed in [5,6]. Experiments on a distributed supply chain simulation in [5] have been repeated to show the execution efficiency of the cloning technology. As illustrated in Fig. 5.3, a supply chain with three nodes can be modeled as three federates, namely *simAgent*, *simFactory* and *simTransportation*. At a decision point, the *simFactory* has three candidate policies to proceed with. The simulation executes the supply chain's operation for a whole year (from simulation time 0 to 361).

Using the same codes for the simulation models, the federates are built into two different versions by linking to: (1) the DMSO RTI library directly (NORM) and (2) an RTI++ middleware library supporting cloning (CLON_EN). We also configure the federates of the 2nd version to execute each of the policies without using cloning (CLON_DIS). For normal and CLON_DIS federates, we first execute the policies one by one (sequential scheme), after that we initiate three federations in parallel to study an individual policy in each federation (parallel scheme). As for CLON_EN federates, federate *simFactory* is triggered to make active cloning at simulation times 80, 203 and 320. The average time of executing one single scenario per run is 584 and 589 s using normal and CLON_DIS federates accordingly (see Fig. 5.4). Compared with the experiments not using cloning technology, the ones using cloning can reduce execution time significantly (∼15%, ∼35%, and ∼60% for cloning at three different points). The more common computation there is between scenarios, the more execution time can be reduced using cloning.

5.2. Supporting fault tolerance

The federation save and restore services provided by the RTI can be used to recover a crashed federation. In order to handle failure, we can save the RTI states with these services at some checkpoints. In the case of failure, a new federation can be created to "restore" the federation with the saved states. This approach requires the simulation model to have the functionality to manipulate the states at the model level, and it repeats the computation from one of the checkpoints onwards. The overhead for executing federation save and recovery can also be significant [26,32]

We propose using the decoupled approach to design a fault-tolerant model, which aims to minimize the cost for providing robustness. The model is designed to handle the fail-stop failures which may affect the global execution [29]. Our study does not consider federate crashes due to the incorrect implementation of its simulation model. We assume also that the messages that are sent and received in the network are not corrupted.

As shown in Fig. 5.5, the model contains a Failure Detector and a Management Module approach in the middleware. The management module comprises an RTI States Manipulator (see Section 5.1). The failure detector monitors the status of the physical federate or even the RTIEXEC if necessary. As soon as an RTI

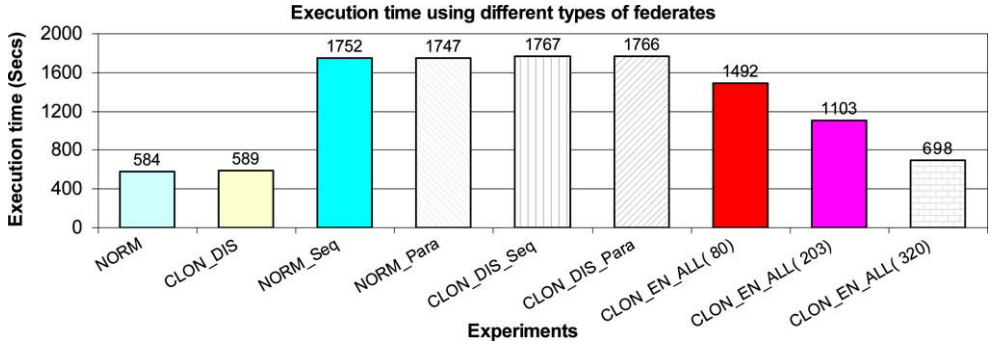


Fig. 5.4. Execution time of experiments using different types of federates.

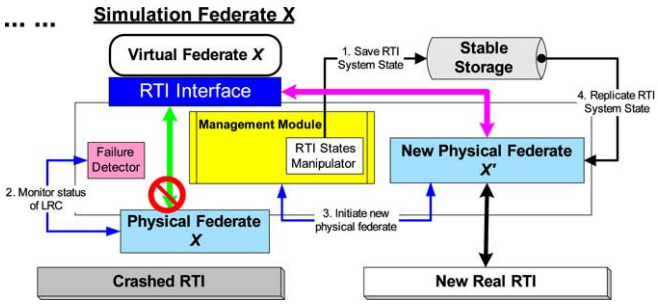


Fig. 5.5. Fault-tolerant model upon dynamic LRC substitution.

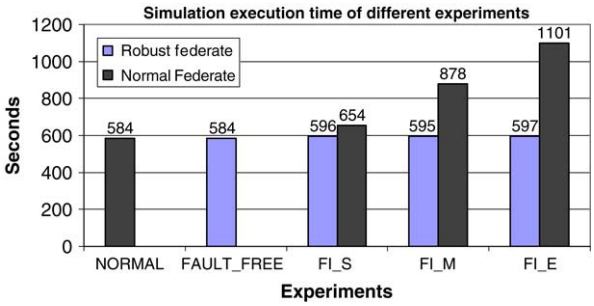


Fig. 5.6. Execution times for benchmarking fault tolerant model.

failure is detected, the management module within the virtual federate will cut off the connection from its physical federate and terminate it.

The fault tolerant model provides user transparency, which frees the user from including extra "fault tolerant codes" in modeling federates, or intervening into the running simulation to deal with RTI failure. The model exploits a "dynamic LRC substitution" method, which allows a simulation to be resumed from exactly the point where the failure occurs. This means the simulation model's execution is immune from the LRC failure. The model enables robustness to legacy federates and avoids extra effort for facilitating rollback to the simulation models.

The Failure Detector monitors the status of the LRC or even the RTIEXEC if necessary. An RTI failure in the current implementation can be: (1) time-out of an RTI invocation, (2) a critical RTI exception,[5] (3) any other unknown error from the RTI or (4) crash of the physical federate or RTIEXEC. The first three cases can be detected passively via the physical federate while the fourth requires the failure detector actively checking the status of the physical federate or RTIEXEC. Subsequent to confirming an occurrence of an RTI failure, the Management Model will start a failure recovery procedure (from the perspective of the *first failed federate(s)*) in the following steps:

- The Management Module cuts off the connection from its PhyFed and terminates it, while other federates' middleware attempt to extract received events before doing this.
- The Management Module creates a new physical federation and initiates a new PhyFed instance. Other federates' middleware also perform exactly the same operation. All virtual federates switch to the new PhyFeds and form a new workable federation together.
- All RTI States Manipulators recover RTI states from stable storage to the PhyFeds.
- All Buffer Managers ensure in-transit events are delivered properly to the subscribers.
- The Management Module synchronizes the recovered federation to guarantee that all federates are fully reinitialized and ready to proceed.

Finally, the virtual federates obtain control again and continue execution with the support of a new physical federation. Therefore, physical federates work as plug-and-play components, and they can be replaced at runtime. The fault-tolerant model functions as a firewall to prevent failure of local or remote LRCs from stopping the execution of the simulation model.

In [7], we have investigated the performance of the fault tolerant model, via experiments on the overall execution time using normal and robust federates (same federate codes linking to an RTI++ middleware library supporting fault tolerance) to execute the distributed supply chain simulation (Fig. 5.3). For normal federates, we have a number of runs, and the average execution time of these runs is referred to as the NORMAL time of executing one simulation session. As for the robust federates, we first repeat the FAULT_FREE experiments (no fault encountered during execution) then carry out a number of the FAULT_INCURRED (fault deliberately introduced) experiments. From FAULT_INCURRED experiments, we select three runs in which the failure of federate *simFactory* occurs at the start (FI_S), middle (FI_M) and end (FI_E) stages respectively. The simulation execution times are reported in Fig. 5.6. The normal simulation execution time is ~584 s using normal federates, which is almost the same as the average simulation execution time in FAULT_FREE experiments. In the FAILURE_INCURRED experiments, the simulation execution time is only 11–13 s longer than the normal case. When failure occurs, we assume that the normal federates have to start from the beginning. Obviously, the later the failure occurs, the more execution time can be saved (up to 50%).

5.3. Supporting web or grid enabled architecture

Using a Decoupled Federate Architecture can ease the combination of Web or Grid technologies with the HLA. Fig. 5.7 gives

[5] e.g. *RTI::RTIinternalError* or any other exception specified as critical by the user.

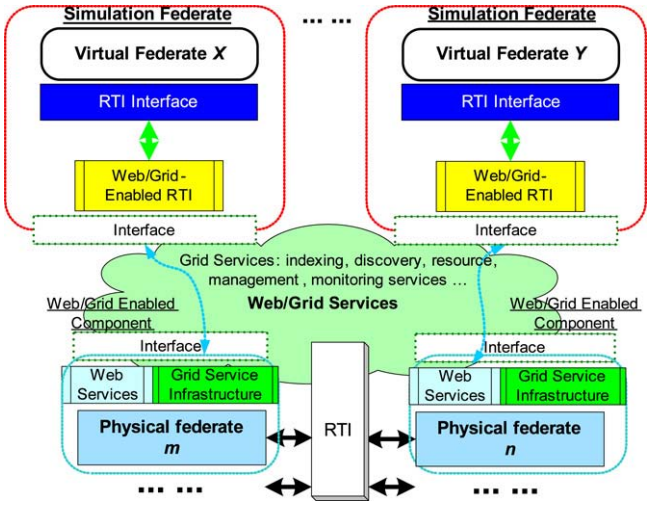


Fig. 5.7. Model supporting web or grid enabled architecture.

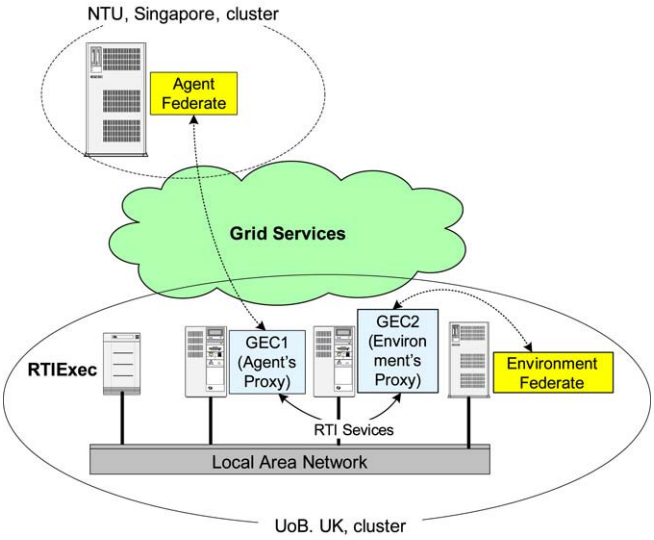


Fig. 5.8. Tileworld experiment configuration in a WAN environment.

the abstract model which supports Web or Grid enabled architectures. The model treats physical federates and the RTIEXEC process as components managed by the Web or Grid services. Therefore, each LRC (or RTIEXEC) is encapsulated as a Web or Grid service and becomes a Web or Grid-enabled component (GEC) accessible via web-based communications (such as SOAP, BEEP). A Web or Grid-enabled RTI library is embedded in the middleware to bridge the model and the physical federate.

The model has significant advantages compared with traditional simulation, using the HLA RTI in the sense that (1) it can cross firewalls, (2) it enhances the interoperability and connectivity among simulation federates and (3) it eases the interoperation between HLA federates and non-HLA simulations. Communication between the virtual federate and the physical federate is via Web services which can pass through most firewalls. Using Web services allows interoperation of federates physically located at various organizations, while connecting them over a normal RTI may encounter network barriers. The RTI component becomes a Web service in the context of the architecture. Non-HLA simulations can also be wrapped as Web services. Therefore, HLA simulations can easily interoperate with non-HLA ones through Web services.

Using Grid services has the added benefits over using pure Web services as follows:

- Grid services provide dynamic creation and lifecycle management. The RTI and federate instances are dynamic and transient service instances under the management of the Grid service infrastructure.
- The model also eases the discovery of the Grid-enabled RTI and federate services using the Grid Index Service [15].
- The Grid provides security in accessing services using its built-in mechanisms such as authentication, secured communication etc.
- With the notifications feature of the Grid, the simulation model does not have to block for the completion of RTI calls and may operate asynchronously from the physical federate.

With the nature of Grid, this model has the potential in providing fault tolerance and load balancing. The model supports the reusability of legacy federate code, and it also allows federates developed upon the Web or Grid Enabled Architecture to interact with other traditional federates.

5.3.1. Experiments with grid enabled federates

A Grid Enabled Architecture [31] from our research group has been developed in Java using Globus Toolkit 3. Experimental results for latency benchmark and efficiency of time advancement calculation (see Section 4) have been collected. Normal federates and federates supported by Grid Enabled Architecture are executed over a Linux Cluster and WAN as shown in Fig. 5.8 (between University of Birmingham, UK and Nanyang Technological University, Singapore). The experimental results are shown as in Table 5.1.

Table 5.1
Benchmarks over LAN and WAN

		Normal federates	Grid enabled architecture
Latency	Cluster	10 ms	19 ms
	WAN	122 ms	935 ms
Time advancement	Cluster	680/s	150/s
	WAN	2/s	0.41/s

The latency benchmark over LAN shows that the Grid Enabled Architecture incurs about twice the overhead of the normal federate in a cluster. In the case of WAN environment, such communication using GT3 becomes very costly; it can also be observed that the latency of the Grid Enabled Architecture is ~7 times more than that of the normal federate. The time advancement benchmark indicates that the Grid Enabled Architecture incurs about 3–4 times more overhead than the normal federate case in terms of synchronization efficiency.

We have also developed an agent-based federation using *Tileworld* [25], as a test case. Tileworld is a well established testbed for agent-based systems. It includes a grid-like environment consisting of tiles, holes and obstacles, and agents whose goal is to score as many points as possible by pushing tiles to fill in the holes. In general, an agent-based federation can be constructed by one *environment federate* containing the tiles, holes and obstacles of the model and one or more *agent federates*. The complexity of the simulated system increases considerably with the number of agents.

Experiments for evaluating the performance of normal federates and Grid enabled federates have also been conducted in a WAN environment [8]. In these experiments, the RTIEXEC and the environment federate are in Birmingham while the agent federate is in Singapore, as illustrated in Fig. 5.8. Two GECs are also hosted in Birmingham for the environment and agent federates respectively. Only one agent federate is used with the number of agents ranging from 1 to 2048. The results for a WAN environment are depicted in Fig. 5.9 (logarithmic scale applies in both X-axis and Y-axis). It can be observed that the Grid enabled agent simulation federates perform much worse than the normal agent simulation federates in the WAN environment. Nevertheless, when the number of agents

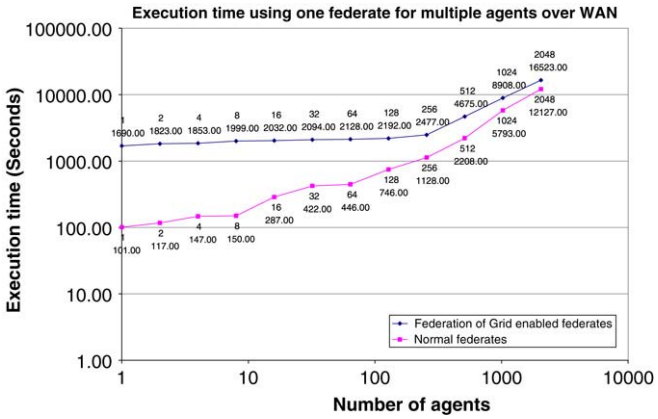


Fig. 5.9. Execution time of grid enabled and HLA repast based simulations with a single agent federate over WAN.

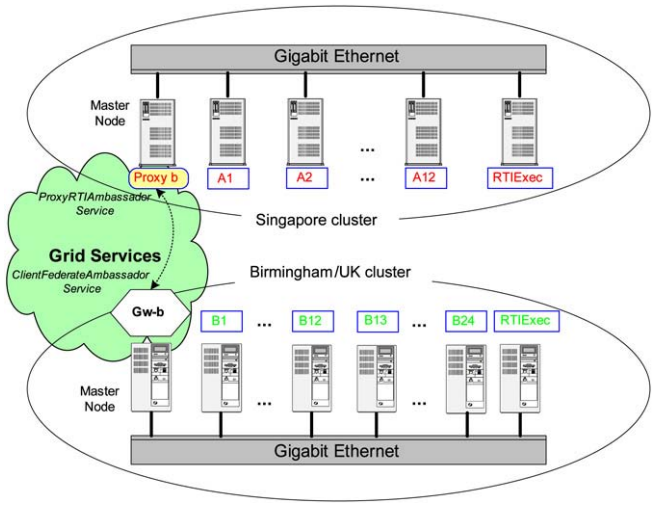


Fig. 5.10. An example of grid enabled federation over WAN.

becomes larger, the total execution time with Grid enabled federates tends to be closer to that with the normal federates.

5.3.2. Grid enabled federation

Considering the need of supporting the reusability of federates at executive level, it is desirable to present a whole federation as services. In a HLA-based simulation, federates interact only through the RTI. Federates are not aware that other federates are connected in the same federation. From a federate's point of view, its correct execution can be guaranteed as long as the events it consumes from other federates, and the events it generates to them have correct content and order. Thus, we can use a gateway federate [2] to collect the events of the federates from the intranet and present the gateway federate as a set of Grid services on a public node using the Grid Enabled Architecture. Eventually, the resources inside the administrative domain and the internal simulation federates are exposed to authorized external users.

The configuration of a Grid enabled federation can be illustrated as in Fig. 5.10. The federation is formed by a number of synchronization benchmark federate. Each benchmark federate occupies an individual work node separately. The benchmark starts with one federate at Singapore cluster and another at Birmingham cluster. We then increased the number of federates equally until twelve federates were involved at each side (denoted as A1–A12 at Singapore and B1–B12 at Birmingham). *Proxy b* directly interacts with federates B1 to B24 through an RTI session at Birmingham

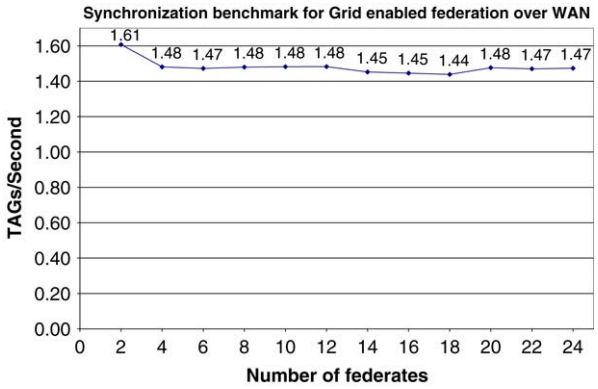


Fig. 5.11. Synchronization benchmark results for Grid enabled federation over WAN.

while *Gw-b* interacts with federates A1–A12 through another RTI session at Singapore. A Grid-enabled gateway between the two RTI sessions was formed by gluing *Gw-b*together with *Proxy b* through Grid invocations over the Grid. Fig. 5.11 presents the benchmark results. The TAG rate is 1.61 times per second for two federates down to about 1.48 times per second for four federates. The TAG rate remains almost constant when the number of federates is not greater than twenty four.

6. Conclusions and future work

In this paper, we have investigated a number of research issues in large scale HLA-based distributed simulations that can be resolved using a Decoupled Federate Architecture. A federate is separated into a virtual federate process and a physical federate process, where the former executes the simulation model and the latter provides RTI services at the backend. A standard RTI interface is presented to support user transparency, while the original RTI component is substituted with a customized library. Benchmark experiments have been performed to investigate the overhead incurred by the Decoupled Federate Architecture. The experimental results are compared for a decoupled federate and normal federate, in terms of latency and time advancement performance. Results indicate that the decoupled architecture incurs only a slight extra latency in the case of a bulky payload and has a very close performance of time advancement compared with a normal federate.

The Decoupled Federate Architecture can be exploited in supporting distributed simulation cloning, providing fault tolerance and introducing Web and Grid technologies to HLA-based simulations. In this paper, we have presented three solutions to address these issues respectively. In these efforts, the Decoupled Federate Architecture has been shown to offer three advantages: (1) it guarantees the correctness of executing RTI services calls and reflecting RTI callbacks to the simulation model, (2) it facilitates state saving and replication at the RTI level, and (3) it incurs minimal overhead when being utilized to sustain distributed simulation cloning and fault tolerance.

In summary, the HLA-based distributed simulations benefit from the Decoupled Federate Architecture in the following ways:

- Distributed Simulation Cloning. The architecture enables replicating federates at runtime without affecting the correctness of RTI and other simulation federates. Experimental results indicate that the cloning technology based on the architecture can optimize the computation of distributed simulations effectively compared with the ones using traditional federates.

- Fault tolerance. Experiments demonstrate that faults of a part of a federation can be successfully isolated, and fault recovery provided without having to rollback the simulation execution. Experimental results show that the robust federates have a very close performance to normal federates.
- Web or Grid Enabled Architecture. The architecture has been engineered to combine the advantages of Web/Grid technologies and HLA simulations to achieve a Web/Grid Enabled Architecture, which provides flexible resource management and enhanced interoperability to HLA simulations. Experimental results show that the current implementation of Grid Enabled Architecture incurs more overhead than using the normal RTI software, and it is suitable for coarse-grained applications.

One of the advantages of executing distributed simulations over a network is the capability of exploiting sharable computing power. This can be achieved by migrating load from the congested host to other more lightly loaded hosts. For our future work, we need to design and implement the models to support load balancing. Using the Decoupled Federate Architecture, we aim to avoid users' efforts to handle RTI states and in-transit events and the overhead incurred in saving/restoring RTI state when adopting a federate migration approach to balance the federates' load.

Acknowledgments

This research was supported under the project "Technology for Secure and Robust Distributed Supply Chain Simulation" funded by the Agency for Science, Technology and Research, Singapore.

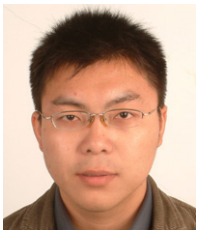
References

[1] C. Berchtold, M. Hezel, An architecture for fault-tolerant HLA-based simulation, in: Proceedings of the 15th International European Simulation Multi-Conference, ESM, 2001, Prague, Czech Republic, 2001, pp. 616–620.
[2] D. Chen, B.S. Lee, W. Cai, S.J. Turner, Design and development of a cluster gateway for cluster-based HLA distributed virtual simulation environments, in: Proceedings of the 36th Annual Simulation Symposium, IEEE Computer Society, Orlando, Florida, USA, 2003, pp. 193–200.
[3] D. Chen, S.J. Turner, B.P. Gan, W. Cai, N. Julka, J. Wei, Alternative solutions for distributed simulation cloning, Simulation: Transactions of the Society for Modeling and Simulation International 79 (5–6) (2003) 299–315.
[4] D. Chen, S.J. Turner, B.P. Gan, W. Cai, J. Wei, A decoupled federate architecture for distributed simulation cloning, in: Proceedings of the 15th European Simulation Symposium, Delft, the Netherlands: TU Delft, Oct, 2003, pp. 131–140.
[5] D. Chen, S.J. Turner, B.P. Gan, W. Cai, J. Wei, Management of simulation cloning for HLA-based distributed simulations, in: European Simulation Interoperability Workshop 2004, Edinburgh, UK, June, 04E-SIW-010, 2004.
[6] D. Chen, S.J. Turner, W. Cai, B.P. Gan, M.Y.H. Low, Algorithms for HLA-based distributed simulation cloning, ACM Transactions on Modeling and Computer Simulation 15 (4) (2005) 316–345.
[7] D. Chen, S.J. Turner, W. Cai, A Framework for robust HLA-based distributed simulation, in: Proceedings of the 20th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulations, Singapore, May, 2006 pp. 183–192.
[8] D. Chen, G.K. Theodoropoulos, S.J. Turner, W. Cai, Y. Zhang, Y. Xie, Large scale agent-based simulation on the grid, Future Generation Computer Systems 24 (2008) 658–671.
[9] F. Cristian, Understanding fault-tolerant distributed systems, Communications of the ACM 34 (2) (1991) 57–78.
[10] J.S. Dahmann, F. Kuhl, R. Weatherly, Standards for simulation: As simple as possible but not simpler, the high level architecture for simulation, Simulation: Transactions of the Society for Modeling and Simulation International 71 (6) (1998) 378–387.
[11] J.S. Dahmann, The high level architecture and beyond: Technology challenges, in: Proceedings of the 13th Workshop on Parallel and Distributed Simulation, Atlanta, Georgia, USA, 1999, pp. 64–70.
[12] DMSO. RTI 1.3-Next Generation Programmer's Guide Version 5, Feb, *DoD, DMSO*, 2002.
[13] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid, International Journal of High Performance Computing Applications 15 (3) (2001) 200–222.
[14] R. Fujimoto, 2007. http://www.cc.gatech.edu/computing/pads/tech-highperf.html.
[15] Globus, 2004. http://www.globus.org/.
[16] M. Hybinette, R.M. Fujimoto, Cloning parallel simulations, ACM Transactions on Modeling and Computer Simulation 11 (4) (2001) 378–407.
[17] IEEE 1516, IEEE Standard for High Level Architecture, 2000.
[18] IEEE1516.3, IEEE Recommended Practice for High Level Architecture Federation Development and Execution Process, FEDEP, 2003.
[19] D.B. Johnson, Distributed system fault tolerance using message logging and checkpointing, Ph.D. Thesis, Rice University, Texas, USA, 1989.
[20] F. Kuhl, R. Weatherly, J. Dahmann, Creating Computer Simulation Systems: An Introduction to High Level Architecture, Prentice Hall, ISBN: 0-13-022511-8, 1999.
[21] C. McLean, F. Riddick, The IMS Mission Architecture for Distributed Manufacturing Simulation, in: Proceeding of the 2000 Winter Simulation Conference, Orlando, Florida, USA, Dec, 2000, pp. 1539–1548.
[22] B. Möller, B. Löfstrand, Mikael Karlsson, Developing fault tolerant federations using HLA evolved, in: Proceeding of 2005 Spring Simulation Interoperability Workshop, San Diego, CA, USA, April 2005, Paper No. 05S-SIW-048.
[23] K.L. Morse, M.D. Petty, Data distribution management migration from DoD 1.3 to IEEE 1516, in: Proceeding of the Fifth IEEE International Workshop on Distributed Simulation and Real-Time Applications, Cincinnati, OH, USA, Aug, 2001, pp. 58–65.
[24] K.L. Morse, D. Drake, D. Brunton, Web Enabling an RTI — an XMSF Profile, in: Proceedings of the 2003 European Simulation Interoperability Workshop, Stockholm, Sweden, Jun, E03-SIW-034, 2003.
[25] M.E. Pollack, M. Ringuette, Introducing the tileworld: Experimentally evaluating agent architectures, in: National Conference on Artificial Intellegence, 1990, pp. 183–189.
[26] K. Rycerz, M. Bubak, M. Malawski, P. Sloot, A framework for HLA-based interactive simulations on the grid, Simulation: Transactions of the Society for Modeling and Simulation International 81 (1) (2005) 67–76.
[27] T. Schulze, S. Straßburger, U. Klein, HLA-federate Reproduction Procedures In Public Transportation Federations, in: Proceedings of the 2000 Summer Computer Simulation Conference, Vancouver, Canada, July, 2000.
[28] W.R. Stevens, UNIX Network Programming, Inter-Process Communications, 2nd ed., vol. 2, Prentice Hall, ISBN: 0-13-081081-9, 1999, pp. 129–155.
[29] A.S. Tanenbaum, M. van Steen, Distributed Systems: Principles and Paradigms, Prentice Hall, ISBN: 0-13-088893-1, 2002, pp. 362–367.
[30] X. Wang, S.J. Turner, W.Y.H. Low, B.P. Gan, A generic architecture for the integration of COTS packages with the HLA, in: Proceedings of the 2004 UK Operational Research Society Simulation Workshop, Birmingham, UK, Mar, 2004, pp. 225–233.
[31] Y. Xie, Y.M. Teo, W. Cai, S.J. Turner, Service provisioning for HLA-based distributed simulation on the grid, in: Proceedings of the 19th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulations, Monterey, CA, USA, Jun, 2005, pp. 282–291.
[32] K. Zając, M. Bubak, M. Malawski, P. Sloot, Towards a grid management system for HLA-based interactive simulations, in: Proceedings of the 7th IEEE International Symposium on Distributed Simulation and Real Time Applications, Delft, Netherlands, Oct, 2003, pp. 4–11.

Dr. Dan Chen is a Professor with the Institute of Electrical Engineering, Yanshan University (China). He was a Postdoctoral Research Fellow with the School of Computer Science at University of Birmingham (UoB, UK) and the School of Computer Engineering at Nanayang Technological University (NTU, Singapore). He received a B.Sc. in applied physics from Wuhan University (China) and a M.Eng. in computer science from Huazhong University of Science and Technology (China). After that, he received another M.Eng. and a Ph.D. from NTU in year 2002 and 2006 respectively. His research interests include: computer-based modelling and simulation, distributed computing, multi-agent systems and grid computing. Recently, he has been working in large scale crowd modelling & simulation and Neuroinformatics.

Dr. Stephen John Turner joined Nanyang Technological University (NTU, Singapore) in 1999 and is Director of the Parallel and Distributed Computing Centre in the School of Computer Engineering. Previously, he was a Senior Lecturer in Computer Science at Exeter University (UK). He received his M.A. in Mathematics and Computer Science from Cambridge University (UK) and his M.Sc. and Ph.D. in Computer Science from Manchester University (UK). His current research interests include: parallel and distributed simulation, distributed virtual environments, grid computing and multi-agent systems. He is steering committee chair of the Principles of Advanced and Distributed Simulation conference and advisory committee member of the Distributed Simulation and Real Time Applications symposium.

Dr. Wentong Cai is an Associate Professor with the School of Computer Engineering at Nanyang Technological University (NTU, Singapore), and head of the Computer Science Division. He received his B.Sc. in Computer Science from Nankai University (PR China) and Ph.D., also in Computer Science, from University of Exeter (UK). He was a Post-doctoral Research Fellow at Queen's University (Canada) before joining NTU in February 1993. Dr. Cai's research interests include Parallel & Distributed Simulation, Grid & Cluster Computing, and Parallel & Distributed Programming Environments. His main areas of expertise are the design and analysis of scalable architecture, framework, and protocols to support parallel & distributed simulation, and the development of models and software tools for programming parallel/distributed systems. He has authored or co-authored over 180 journal and conference papers in the above areas. Dr. Cai is a member of the IEEE. He is currently an associate editor of *ACM Transactions on Modeling and Computer Simulation*(TOMACS), editorial board member of *Multiagents and Grid Systems — An International Journal*, and editorial board member *of International Journal of Computers and Applications*.



Mr. Muzhou Xiong is a Ph.D. candidate in Huazhong University of Science and technology, PR China. Also he currently is a Research Associate in Nanyang Technological University, Singapore. He obtained his B.S. degree in computer science from Huazhong University of Science and Technology in 2002. His research interests include grid computing, network storage, automatic storage management, and parallel and distributed simulation. His recent research has focused on modeling of human behavior model for high dense crowds.