

A DECOUPLED FEDERATE ARCHITECTURE FOR DISTRIBUTED SIMULATION CLONING

Dan CHEN¹, Stephen John TURNER², Boon Ping GAN¹, Wentong CAI², Junhu WEI²

¹ Singapore Institute of Manufacturing Technology
Singapore 638075

E-mail: {dchen, bpgan}@simtech.a-star.edu.sg

² School of Computer Engineering
Nanyang Technological University
Singapore 639798

E-mail: {assjturner, aswtcai, asjhwei}@ntu.edu.sg

KEYWORDS

High Level Architecture, Runtime Infrastructure, distributed simulation cloning, decoupled federate architecture, fault tolerance.

ABSTRACT

Distributed simulation cloning technology is designed to perform “what-if” analysis of existing High Level Architecture (HLA) based distributed simulations. The technology aims to enable the examination of alternative scenarios concurrently within the same simulation execution session. State saving and recovery are necessary for cloning a federate at runtime. However it is very difficult to have a generic state manipulation mechanism for any existing federate, as these can be developed independently and freely. The correctness of replicating a running federate significantly depends on the Runtime Infrastructure (RTI) software. The distributed simulation also needs fault tolerance to provide robustness at runtime. This paper proposes a decoupled federate architecture to address the above issues. A normal federate is decoupled into two processes, which execute the simulation model (virtual federate) and the local RTI component (physical federate) respectively. The decoupled approach interlinks the two processes together via Inter-Process Communication. The virtual federate interacts with the RTI through the standard RTI service interface supported by a customized library. The decoupled architecture ensures the correct replication of federates and facilitates fault tolerance at the RTI level. At the same time, it provides user transparency and reusability to existing federate codes. Benchmark experiments have been performed to study the extra overhead incurred by the decoupled federate architecture against the normal federate. The encouraging experimental results indicate that the proposed approach has a performance close to the normal one in terms of latency and time synchronization.

1. INTRODUCTION

Distributed simulation is an important technology that facilitates simulation programs executing in a distributed environment. Geographically distributed simulation models can be linked together to construct a large-scale simulation federation. Distributed simulation technology has a variety of applications, one of which is supply-chain simulation. It meets the pressing need of simulating a supply-chain, as this often involves multiple companies across enterprise boundaries and simulation models that are developed independently (Gan et al. 2000; Turner et al. 2001).

The High Level Architecture (HLA) defines the rules and specifications to support reusability and interoperability amongst the simulation federates. The Runtime Infrastructure (RTI) software supports and synchronizes the interactions amongst different federates conforming to the standard HLA specifications (Dahmann et al. 1998). HLA-based distributed simulation provides interoperability and reusability of the independent simulation federates. However in the context of a traditional distributed simulation, one simulation session can only yield one single set of results for analysis. To perform “what-if” analysis, one has to repeat the execution of the simulation to examine alternative scenarios or decision strategies using different rules and parameters. Therefore the simulation analyst may choose some best solutions based on all the possible results. Basically it is a time-consuming and onerous task in which a lot of computation is repeated unnecessarily.

During the simulation, a federate will meet some points (decision points) at which there occurs a critical change of system states, and it is faced with different choices to proceed (Chen et al. 2003a). Instead of executing all the choices one by one in a linear way, the simulation cloning approach offers users the flexibility to examine these different choices concurrently. At a

predefined decision point, cloning of a federate may be triggered at runtime, following which the federate can replicate itself into multiple clones to explore different possibilities (Chen et al. 2003a). Each clone explores one particular path together with its partner clones spawned from the other federates in the original scenario. Thus, users are able to analyze multiple alternative results concurrently using the same simulation models within a single simulation run.

However it is challenging work to ensure correct and efficient simulation cloning especially in the context of distributed simulation. For example, it needs state saving and recovery at both the simulation model level and the RTI level, rather than simply starting another federate instance. A distributed control mechanism is needed to coordinate the federates within the same distributed simulation session. Furthermore, the correctness of cloning significantly depends on the platform and RTI software the simulation federates use. As the local RTI component is not designed to be replicated, direct cloning of a federate can lead to unpredictable and uncontrollable failure at the RTI level. Thus it requires us to design a reliable and correct federate cloning approach.

One of the key benefits of HLA-based simulation is reusability (Dahmann et al. 1998), which raises another critical issue of reusing the existing code of user federates while adopting simulation cloning technology. Considering the complexity and variety of simulation models, it is difficult to have a generic cloning solution that keeps the consistency of any simulation federate while cloning. However a middleware approach makes it possible to monitor the system state of a federate at the RTI level (Chen et al. 2003a).

Moreover simulation federates running at different locations are liable to failure, and the failure of one federate can lead to the crash of the overall distributed simulation. Cloning more and more federates inside one single federation may increase the risk of such failure steadily, thus we need to investigate the fault tolerance issue in simulation cloning and apply it to the distributed simulation technology.

This paper introduces the idea of decoupling the local RTI component from a normal HLA federate. Basically a normal federate contains the simulation model and the local RTI component. 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 a “virtual” RTI component with it, 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. An Inter-Process Communication channel bridges the two processes together into a simulator in a general sense. All the RTI calls employed by a virtual federate call services via the corresponding physical federate. This approach ensures the intactness of existing simulation federates. Cloning a federate means replicating multiple virtual federates and starting new physical federates with recovered system states at runtime. As the virtual federate contains no real RTI component, the decoupled approach avoids the risks incurred by copying a running federate.

The decoupled architecture isolates the failure of local RTI components from the simulation federates. In the case of an RTI component crash, a new federate or even a new federation can be resumed according to the stored states at the RTI level. One does not need to start the whole simulation from scratch, thus the decoupled approach provides fault tolerance in this sense. All these advantages can be achieved without interrupting the execution of the user’s simulation.

To investigate the overhead incurred by the decoupled approach, this paper presents a set of benchmark experiments on latency and synchronization performance. Results are reported and compared between the decoupled federates and normal federates. The experimental results indicate a promising performance of the decoupled federates in both benchmarks.

The rest of this paper is organized as follows: Section 2 outlines the distributed simulation cloning technology and related work. Section 3 discusses the decoupled approach in detail for both design and implementation. Section 4 describes the benchmark experiments and analyzes the results. In section 5, we conclude with a summary and proposals on future works.

2. DISTRIBUTED SIMULATION CLONING TECHNOLOGY

2.1 Related Work

Hybinette and Fujimoto first employed the simulation cloning technology as a concurrent evaluation mechanism, in the context of parallel simulation (Hybinette and Fujimoto 2001). The motivation for this technique was to develop a parallel model that supports an efficient, simple, and effective way to evaluate and compare alternative scenarios. The method was targeted for parallel discrete event simulators that provided the simulation application developer with a logical process (LP) execution model.

Schulze et al introduced a cloning approach to extend the flexibility of system composition to run-time (Schulze

et al. 2000). Their approach included the parallel management of different time axes in order to provide forecast functionality. Internal cloning and external cloning techniques were suggested to clone the federates at run-time.

As our design targets the users who may have their own existing complex simulation models, we have the additional aim to provide reusability and transparency while enabling simulation cloning. Our research and discussion are based on HLA-compliant distributed simulations. Providing easy utilization and deployment is another major concern. Distributed simulation cloning technology should be a much more powerful and flexible decision support tool than traditional “linear” simulation. Our approaches focus on the control of a large-scale distributed simulation using the cloning technology.

2.2 Distributed Simulation Cloning

Simulation cloning technology involves research issues such as trigger conditions, cloning operation, distributed coordination, state saving and recovery, scenario management and interactive control, etc. We define some terms in distributed simulation cloning as follows.

Cloning of a distributed simulation may happen at some critical points that are defined by a simulation analyst. At one of those points, a federate may face different choices that perform alternative actions. These points are known as **decision points**, which comprise trigger conditions and candidate actions for a federate to perform. A decision point usually represents the location in the execution path where the states of the system start to diverge in a cloning-enabled simulation. From the decision point onwards, a federate may spawn multiple executions to exploit alternative scenarios concurrently.

A federate is said to perform **active cloning** if it makes clones on its own initiative. As there exist multiple interoperating federates in distributed simulations, when one federate splits into different executions, the partners who interact with this federate may have to spawn clones as a result of the active cloning, thus **passive cloning** happens. The clones created from the same root federate are referred to as **sibling clones**. Each clone is an independent simulation federate, and it cooperates with some clones of other federates to form an independent simulation scenario. Those clones within the same scenario are known as **partners**.

In order to save computation, our proposed approach merely requires cloning the federates whose states will change at a decision point and keeping other federates intact. Thus the simulation is replicated incrementally;

such an **incremental cloning** approach shares computation between federates in multiple scenarios. Although new scenarios have been created due to the active cloning, a clone is capable of executing in multiple scenarios, and these are known as **shared clones**.

When a federate is cloned, we can create multiple federations to meet the demand of executing alternative scenarios or generate new federates within the original federation without intervening in the execution of any other scenario. The former approach is called a **Multiple-federation Solution**, and the latter approach a **Single-federation Solution**. A single-federation solution offers advantages in cloning control and cloning sharing and is adopted for our research. In order to manage concurrent scenarios within a single federation, we propose to use the Data Distribution Management (DDM) (Morse and Petty 2001) mechanism to partition scenarios. To provide reusability to existing simulation federates, a middleware approach is adopted to hide the implementation of any cloning related modules. Thus transparency to simulation federates is achieved in distributed simulation cloning.

3. CLONING FEDERATES

3.1 Problems in Cloning Federates

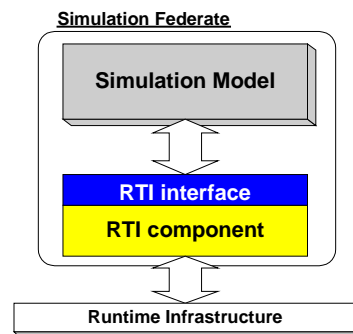


Figure 1: Abstract Model of a Simulation Federate

A normal simulation federate can be viewed as an integrated program consisting of a simulation model and local RTI component, as shown in Figure 1. As mentioned above, active cloning of a federate occurs at a decision point to enable different candidate actions to be performed. “Cloning” implies that the new clones of one particular federate should 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 simulation state consistency. 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 active 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 recovery 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 recovery mechanism can be provided that will be suitable for any simulation federate. 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 recover the RTI states while enabling transparency.

However some operating systems enable the user to duplicate a running process. In UNIX, some system calls such as **fork** can create a new process that is an exact copy of the calling process (Stevens 1999). This suggests the possibility of cloning a federate at runtime using such a process duplication mechanism. Thus the correctness of cloning depends on the platform and RTI software that the simulation federates adopt. However, the local RTI component is not designed to be duplicated, thus forking a federate can lead to unpredictable and uncontrollable failure at the RTI level. The failure of the local RTI component prevents the simulation execution from proceeding correctly.

In HLA-based distributed simulation, the crash of one federate can result in the failure of the overall simulation federation. As more and more clones will participate in the existing federation as a result of simulation cloning, fault tolerance becomes another important concern. From the above discussion, we can see that the simulation model and the local RTI component have very different characteristics. Therefore, it seems that we can make a distinction between these two modules for cloning a federate.

3.2 Decoupled Federate Architecture

To tackle the problems involved in replicating running federates, this section introduces the decoupled federate architecture to separate the simulation model from the

local RTI component. The design and implementation of this approach will be covered in detail.

3.2.1 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 without altering the semantics of RTI services (Chen et al. 2003a). Figure 2(B) gives the abstract model of the virtual federate. Compared with the original federate model illustrated in Figure 1, the only difference is in the module below the RTI interface, which remains transparent to the simulation user.

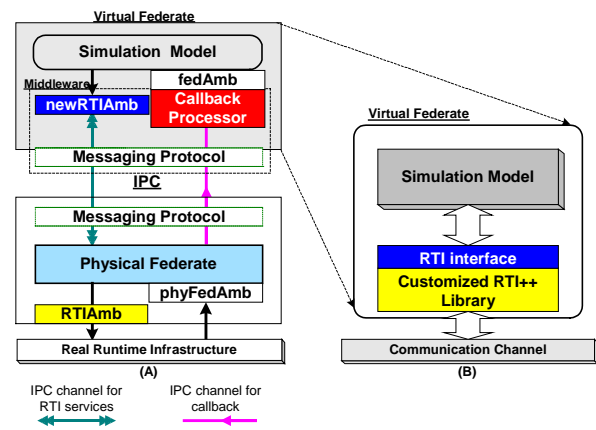


Figure 2: Decoupled Federate

A physical federate is specially designed as shown in Figure 2(A). The physical federate associates itself with a real local RTI component. Physical federates interact with each other via a common RTI. Both virtual federates and physical federates operate as independent processes. Reliable Inter-Process Communication (IPC) or other out-of-band communication mechanism bridges the two entities into a simulator in a general sense (Stevens 1998). Using the decoupled approach, cloning of a simulation federate can be done by forking the virtual federate process and starting an additional physical federate instance with restored system state at the RTI level.

All the components inside the dashed rectangle form a **Middleware** module between the simulation model and the IPC. Within the virtual federate, the **newRTIAmb** contains customized libraries while presenting the standard RTI services and related helpers to the simulation model. This module is also designed to contain all other management modules for cloning purpose (Chen et al.

03b). The **fedAmb** serves as a common callback to the user federate, which is freely designed by the user and independent of the decoupled approach. The **newRTIAmb** handles the user's RTI service calls by converting the method together with the associated parameters into IPC messages via the **Messaging Protocol**. The protocol defines a mapping between an IPC message type and the RTI method it represents. For example, an RTI_UPDATE message indicates that the virtual federate has invoked the RTI method *updateAttributeValues()*. The IPC conveys these messages to the physical federate for processing in a FIFO manner immediately.

The physical federate is designed to convert 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 IPC. 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 IPC messages. The IPC 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 the normal simulation federate using the same code in 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 clone or a federate contains a virtual federate process and a physical federate process.

3.2.2 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.

Figure 3 depicts the procedure where a simulation model initiates an RTI call and waits for a return from the real RTI, using the *updateAttributeValues* method as an example. The procedure is as follows:

- The virtual federate invokes the redefined *updateAttributeValues* method.

- Inside the *updateAttributeValues* method, the *packMsg* routine extracts the data stored in the *AttributeHandleValuePairSet (AHVPS)* and packs them together with other parameters such as the associated timestamp, object instance handle and tag into an RTI_UPDATE message.
- The IPC enqueues the RTI_UPDATE message to the physical federate. The virtual federate switches to waiting mode for the returned message.
- Once the physical federate receives the IPC message, it invokes the *unpackMsg* routine to process it according to the associated type, RTI_UPDATE.
- A new *AHVPS* object and related parameters are recovered based on the IPC message and passed to the *RTI::updateAttributeValues*, which invokes the real RTI service.
- On the accomplishment of this *RTI::updateAttributeValues* call, the physical federate acknowledges the virtual federate with an IPC message containing the returned value.
- The *updateAttributeValues* call finishes and the data retrieved from the acknowledgement message is returned to the simulation model.

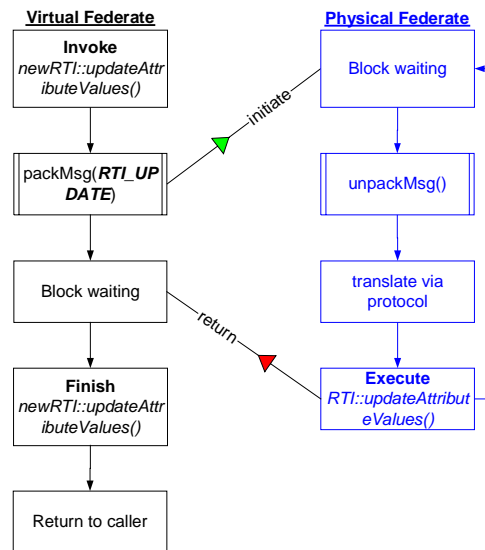


Figure 3: Executing an RTI Call in the Decoupled Architecture

From the user's point of view, the initiation and accomplishment of an RTI call are identical to 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 IPC mechanisms have limitations in message size and buffer size. For example, the **Message Queue** based on Solaris defines the maximum queue length as 4096 bytes (Stevens 1998). The message sender and receiver must agree with each other on the same message length. If a fixed message size is defined for IPC 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. Thus a 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 are to be sent for a single data block. Figure 4 gives the messaging details based on this simple protocol.

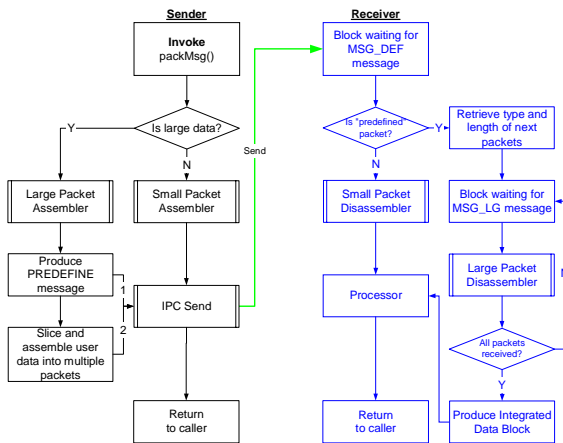


Figure 4: Messaging between Virtual Federate and Physical Federate

The RTI communicates with a federate via its federate ambassador provided by the user (DMSO 2002). 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. Figure 5 depicts how to realize these functionalities. To ease discussion, we

assume the physical federate will get the callbacks shown in Figure 5. This procedure is illustrated by the following steps:

- The Virtual federate invokes the routine *newRTI::tick()* and the latter sends out an RTI_TICK message to the physical federate.
- The Physical federate calls the real RTI *tick()* according to the RTI_TICK message.
- 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 data sent by the RTI is enqueued to the callback IPC channel. The routine inside the *newRTI::tick()* accesses the queue for the virtual federate.
- As long as the RTI_TICK_DONE message is not detected, the callback processor continuously processes the messages in a FIFO order while activating the corresponding method in the fedAmb module based on the messaging protocol.
- At the physical federate side, once the RTI finishes its current work and passes control to the physical federate, the latter returns an RTI_TICK_DONE message to the virtual federate.
- On receiving the RTI_TICK_DONE message, the virtual federate accomplishes the *newRTI::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 *newRTI::tick()* can only return when the real RTI finishes its work. The communication channels linking the virtual federate and physical federate work in a FIFO manner. Thus 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 the equivalent operations to the normal federate. It guarantees consistency in presenting interactions from the real RTI to the simulation model and also ensures user transparency.

The decoupled architecture requires an additional IPC 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 reports and analyzes the experimental results in terms of event transmission latency and synchronization efficiency.

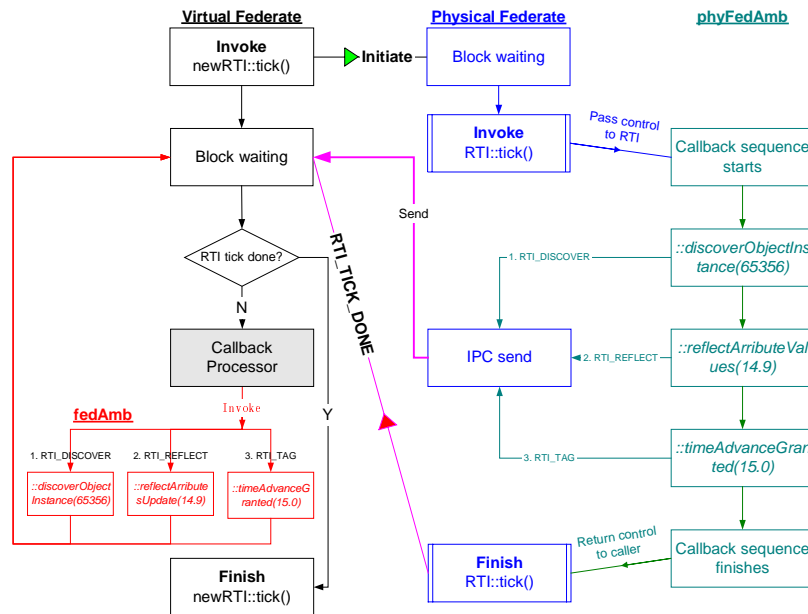


Figure 5: Conveying Callbacks to the Virtual Federate

3.2.3 Fault Tolerance

In an HLA-based distributed simulation, the participating federates running at different locations are liable to failure. A lot of factors may contribute to the failure of a federate, for example, network congestion, platform crash of the RTIEXEC process (DMSO 2002) etc. It is also difficult to handle such failure during runtime because most RTI implementations operate as a black box. In a large-scale distributed simulation, the crash of one federate can lead to failure of the overall simulation. More and more federates will participate in the same federation with the cloning of federates, which increases the possibility of simulation failure. Although users model the simulation federate properly, an RTI failure still induces simulation collapse. Simply restarting a new federate to substitute the crashed one is not applicable since the consistency of the overall simulation state is lost. Considering the complexity and distribution of the individual simulation models and the number of federates in a large-scale distributed simulation, it is costly to restart the overall distributed simulation. As fault tolerance (Danami and Garg 1998) is needed in a distributed simulation, we propose using the decoupled approach to address the potentially unpredictable faults at the RTI level. In this study, the fault tolerance aims to minimize the wasted distributed computation and to provide user transparency. In other words, the user does not have to intervene into the running simulation to deal with RTI failure.

The middleware approach enables the interception of RTI services invoked by the simulation model. The

system state at the RTI level is accessible using the middleware approach (Chen et al. 2003a). Thus we can retrieve the “features” of a federate, such as the object classes subscribed and published as well as whether the federate is time constrained or time regulating. Furthermore we can log the “operation” history of a federate. The middleware can track the object instances registered and each attribute update to any object instance. All these operations are hidden beneath the newRTIamb interface. Based on the stored information, a crashed federate can be replaced by a new federate with inherited system states from the system state log. By “plugging” the new federate back to its virtual federate, the distributed simulation can continue without being interrupted by the previous RTI failure. The approach can also take advantage of the federation save and restoration mechanism provided by RTI services. This mechanism is indicated as in Figure 6. The same model of state saving and recovery used in cloning federates can also provide this fault tolerance.

Figure 6 gives a model of how fault tolerance can be achieved with the decoupled architecture, in which one of the running federates (marked as *m*) is highlighted for study. As illustrated in Figure 6(A), at runtime the middleware intercepts the invocation of each RTI service method. The interceptor logs all the RTI system states into stable storage. Some RTI states are relatively static, such as the federate identity, federation information, the aforementioned declaration data and time features. The static states also include the registered or deleted objected instances. Some other RTI states are highly dynamic, such

as granted federate time, sent and received interactions, updated and reflected attribute values of object instances, etc.

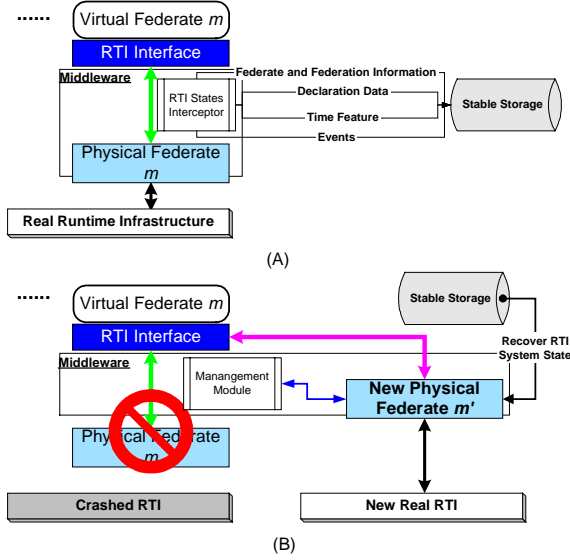


Figure 6: Fault Tolerance using Decoupled Federate

As soon as the middleware detects the RTI failure, no matter whether it is due to a local physical federate or incurred by other federates or for some other unpredictable reasons, the management module within the virtual federate will cut off the connection from its physical federate and terminate it (as shown in the left side of Figure 6(B)). Subsequently the management module will initiate a new physical federate instance m' and have it join the existing federation or possibly a new federation with another RTIEXEC process. When the whole federation fails, other virtual federates can also perform the same action and form a new workable federation together in the same way. After that, the physical federate reads in the RTI state from the stable storage. It invokes the corresponding RTI services with restored parameters to recover the features of the old federate and resumes the dynamic system states in the snapshot obtained from the stable storage. Finally the virtual federate continues execution with the new physical federate. Thus, the physical federate works as a plug-and-play component, it can be replaced and transplanted at runtime.

4. BENCHMARK EXPERIMENTS AND RESULTS

In order to investigate the overhead incurred in the decoupled architecture, we perform a series of benchmark experiments to compare the decoupled federate with a normal 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.

4.1 Experiment Design

The experiments use three computers in total (two workstations and one server), in which the server executes the RTIEXEC and FEDEX processes. The federates that run at one independent workstation are enclosed in a dashed rectangle. In our case $fed A[i]$ and $fed B[i]$ ($i \geq 0$) occupy workstation 1 and workstation 2 respectively. The computers are interlinked via a 100Mbps-based backbone.

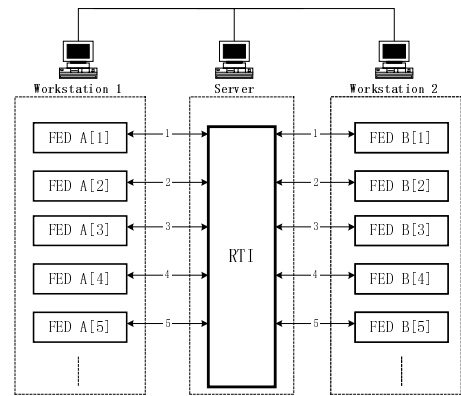


Figure 7: Architecture of Benchmark Experiments

The server (Ultra-Enterprise) has following specification:

- sparcv9 processor (* 6) operating at 248 MHz
- 2048 Mbytes of RAM
- Sun Solaris OS 5.8
- GCC 2.95.3
- DMSO RTI NG 1.3 V6 for the **SunOS-5.8** operating system and the gcc-2.95.3 compiler

The workstations (SunBlade 1000) have the following specification:

- sparcv9 processor operating at 900 MHz
- 1024 Mbytes of RAM
- Sun Solaris OS 5.8
- GCC 2.95.3
- DMSO RTI NG 1.3 V6 for the **SunOS-5.8** operating system and the gcc-2.95.3 compiler

The experiments emulate the simulation cloning process by increasing the number of identical federates. As shown in Figure 7, $fed A[1]$ and $B[1]$ form a pair of initial federate partners, which represent the federates to be cloned. $Fed A[i]$ and $B[i](i > 1)$ stand for the i th clones of the two original federates respectively. The architecture

is used through all the benchmarks experiments and for both normal federates and decoupled federates.

A DDM based approach is used to partition concurrent scenarios (Chen et al. 2003b). For the latency benchmark, each pair of federates have 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 acknowledgement from its partner (from *fed A[i]* to *fed B[i]*, and vice versa) for 5,000 times with a **payload of 100, 1000 and 10,000 bytes**. The time interval in the ping-pong procedure will be averaged and divided by 2 to give the latency in **milliseconds**. A federate merely reflects the events with identical region to itself. In other words, *fed A[i]* only exchanges events with *fed B[i]*.

As for the time advancement benchmark, all federates are time regulated and time constrained. Each federate has lookahead 1.0 and advances the federate time from 0.0 to 5,000.0 with timestep 1.0 using *timeAdvanceRequest* (DMSO 2002). The results report the rate that the RTI issues *timeAdvanceGranted* (TAGs/Second).

4.2 Latency Benchmark Results

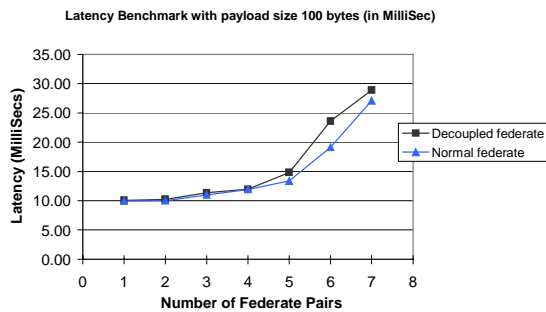


Figure 8: Latency Benchmark on Decoupled Federate vs Normal Federate with Payload 100 Bytes

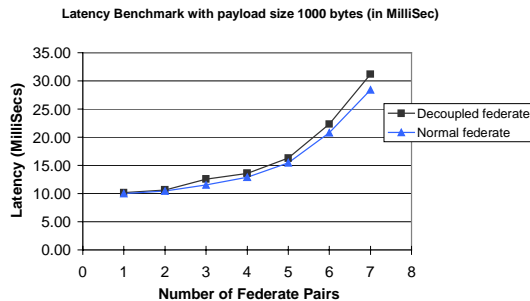


Figure 9: Latency Benchmark on Decoupled Federate vs Normal Federate with Payload 1000 Bytes

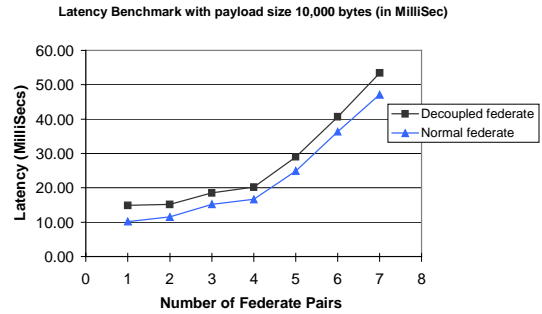


Figure 10: Latency Benchmark on Decoupled Federate vs Normal Federate with Payload 10,000 Bytes

The latency benchmark experiments report the latency with three different payload sizes. From Figure 8 to Figure 10, we can see that no matter whether the payload size is small or large, the 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 Figure 8 and Figure 9, when the payload is not greater than 1000 bytes, the latency varies from about 10 milliseconds for one pair of federates to about 30 milliseconds 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 Figure 10, when a bulky payload as large as 10,000 bytes is applied, the decoupled federates incur about 5 milliseconds 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.

4.3 Time Advancement Benchmark Results

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 these two types of federates is very similar in terms of time advancement.

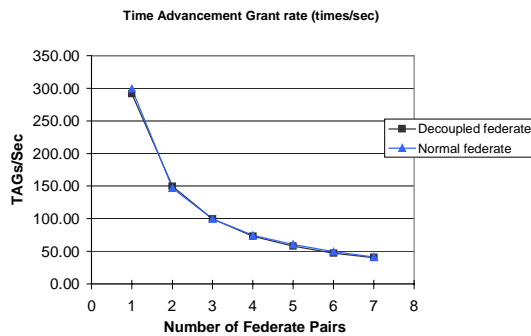


Figure 11: Time Advancement Benchmark on Decoupled Federate vs Normal Federate

5. Conclusions and Future Work

In this paper, we have investigated some issues in cloning federates in distributed simulations. In order to overcome the problems of replicating a federate instance and providing fault tolerance to a distributed simulation, a decoupled federate architecture is proposed. This approach decouples the simulation model from the RTI local runtime component in a normal simulation federate. 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 at the virtual federate, while the original RTI component is substituted with a customized library. The decoupled architecture enables a relatively generic method of replicating the simulation model. It also facilitates state saving and recovery at the RTI level for cloning a federate and fault tolerance. The proposed approach guarantees the correctness of executing RTI services calls and reflecting RTI callbacks to the simulation model.

Benchmark experiments have been performed to investigate the overhead incurred by a decoupled federate architecture. The experimental results are compared for a decoupled federate and normal federate in terms of latency and time advancement performance. The 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 architecture can provide other advantages to distributed simulation technology. The potential application avenues are as follows:

- Using a communication channel between the virtual federate and physical federate, we are able to

distribute the computational complexity of one federate with a heavy load in a cluster computing environment

- Physical federates can be more independent, which allows the further optimization of the computation. For example, the physical federates can monitor the federation and optimize the computation by migrating the virtual federates to other nodes

For our future work, we need to further explore the mechanism of the cloning operation to ensure simulation consistency. Another challenge is the interactive manipulation of cloning-enabled simulation in a distributed environment, where users are offered the flexibility to control and update the cloning online.

REFERENCES

- Chen, D.; B. P. Gan; S. J. Turner; W. Cai; N. Julka; and J. Wei. 2003. "Evaluating Alternative Solutions for Cloning in Distributed Simulation". *Proceedings of the 36th Annual Simulation Symposium* (Orlando, Florida, USA, Mar), 201-208.
- Chen, D.; B. P. Gan; S. J. Turner; W. Cai; and J. Wei. 2003. "Data Distribution Management in Distributed Simulation Cloning". *Proceeding of 2003 European Simulation Interoperability Workshop*, (Stockholm, Sweden, June), paper no. 03E-SIW-024.
- Dahmann, J. S.; F. Kuhl; and R. Weatherly. 1998. "Standards for Simulation: As Simple As Possible But Not Simpler, The High Level Architecture for Simulation". *Simulation*, 71:6 (Dec), 378-387.
- Danami, O. P. and V. K. Garg. 1998. "Fault-Tolerant Distributed Simulation". *Proceedings of the 12th Workshop on Parallel and Distributed Simulation* (Banff, Albert, Canada), 38-45.
- DMSO. 2002. RTI 1.3-Next Generation Programmer's Guide Version 5 (Feb), *DoD, DMSO*.
- Gan, B. P.; L. Liu; S. Jain; S. J. Turner; W. Cai; and W. Hsu. 2000. "Distributed Supply Chain Simulation Across Enterprise Boundaries". *Proceedings of the 2000 Winter Simulation Conference* (Orlando, Florida, USA), 1245-1251.
- Hybinette, M. and R. M. Fujimoto. 2001. "Cloning parallel simulations". *ACM Transactions on Modeling and Computer Simulation*, Volume 11, (Oct), New York, USA, 378-407.
- Morse, K. L. and M. D. Petty. 2001. "Data Distribution Management Migration from DoD 1.3 to IEEE 1516". *Proceeding of the Fifth IEEE International Workshop on Distributed Simulation and Real-Time Applications* (Cincinnati, Ohio, USA, Aug), 58-65.
- Schulze, T.; S. Straßburger; U. Klein. 2000. "HLA-federate Reproduction Procedures In Public Transportation Federations". *Proceedings of the 2000 Summer Computer Simulation Conference* (Vancouver, Canada, Jul).
- Stevens, W. R. 1999. "UNIX Network Programming, Inter-Process Communications". Vol. 2, 2nd Edition, Prentice Hall.
- Turner, S. J.; W. Cai; and B. P. Gan. 2001. "Adapting a Supply-chain Simulation for HLA". *Proceeding of the Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications* (San Francisco, California, USA), 67-74.