Management of Simulation Cloning in HLA-based Distributed Simulations

Dan CHEN[1], Stephen John TURNER[2], Boon Ping GAN[1], Wentong CAI[2], Malcolm Yoke Hean LOW[1], Junhu WEI[2]
[1] Singapore Institute of Manufacturing Technology
Singapore 638075
E-mail: {dchen, bpgan, yhlow}@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.

ABSTRACT: *Distributed simulation cloning technology is designed to analyze alternative scenarios of a distributed simulation concurrently within the same simulation execution session. One important goal of the technology is to optimize execution by avoiding repeated computation amongst independent scenarios. This paper introduces the design and functionalities of the modules handling the cloning of High Level Architecture (HLA) based distributed simulations. The federate cloning mechanism is established upon a decoupled federate architecture to replicate federates as described in a previous paper.*

This paper first presents the process of cloning at decision points and the method of coordinating and synchronizing clones, to ensure the state consistency of the overall simulation. A mapping approach is designed for the clones to map the new HLA entities created in state replication to the existing ones known to the simulation model. This entity mapping method helps in supporting user transparency and reusability of existing federate codes. An example of a simple supply-chain simulation indicates that the proposed approach provides correct distributed simulation cloning and can significantly reduce the execution time for evaluating different scenarios of distributed simulations.

1. Introduction

Distributed simulation is an important technology that facilitates the execution of simulation programs in a distributed environment. Geographically distributed simulation models can be linked together to construct a large-scale simulation federation. 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 [1].

For a traditional simulation, one simulation session can only yield one single set of results for analysis. An analyst chooses some best solutions by repeating the simulation multiple times to examine alternative decision policies and strategies. Basically it is a time-consuming and onerous task in which a lot of computation is repeated unnecessarily. In the context of a large-scale distributed simulation, considering the complexity and distribution of the individual simulation models and the number of federates, it is costly to reconfigure and re-execute the overall distributed simulation again and again.

When the simulation advances to a decision point, a federate (simulation component) has different choices to examine. Instead of simulating each of these choices from the start, the federate can make replicas of itself to examine them concurrently using distributed simulation cloning technology. Thus the execution time can be reduced and the analyst may quickly obtain multiple sets of simulation results that represent the impacts of alternative decisions. One important goal of this technology is to optimize execution by avoiding repeated computation amongst independent scenarios. Distributed simulation cloning technology also has some other applications, such as providing fault tolerance.

In this project, we aim to enable HLA-based distributed simulation cloning. When performing distributed simulation cloning, it is desirable to replicate only those federates whose states will alter at a decision point while the rest remain intact, thus such an incremental simulation cloning [2] mechanism is expected to reduce unnecessary computation.

Our cloning technology adopts a decoupled federate architecture [3] to enable the replication of federates. The cloning management mechanism is introduced to ensure correct federate replication, and its task includes creating clones, manipulating states and coordinating the federation.

A clone is required to inherit identical states from the original federate, including the RTI entities known to the simulation model. For example, the clone has to "re-register" the object instances owned by the cloned federate. The cloning mechanism needs to ensure that the clones use the same reference to the original entities at the RTI level as before cloning, to keep the state consistent and federate code transparent.

The simulation cloning procedure can affect the execution of the remaining federates, for example, the time advancement of other federates may be affected. Therefore, the technology needs to guarantee consistency of the system states in the overall distributed simulation. The method of coordinating and synchronizing clones during cloning is designed to avoid disrupting the execution and to maintain the state consistency.

The technology allows clones (shared clones) to operate in multiple scenarios, which requires accurate control to achieve correctness and efficiency. The cloning mechanism for shared clones ensures they require cloning only when absolutely necessary. It guarantees the simulation results of alternative scenarios achieved using distributed simulation cloning technology are the same as those obtained by repeating simulation executions.

The rest of this paper is organized as follows: Section 2 addresses related work and introduces some basic concepts. Section 3 introduces the overall cloning architecture. Section 4 details the algorithms for distributed simulation cloning, including active cloning and passive cloning. A supply-chain simulation example is presented in section 5, which can be cloning-enabled with the proposed technology. In section 6, we conclude with a summary and proposals on future work.

2. Related Work and Concepts

2.1 Related Work

Hybinette and Fujimoto proposed using simulation cloning technology as a concurrent evaluation mechanism in the parallel simulation domain [4]. This technique aimed 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 provide the simulation application developer with a logical process execution model.

Schulze et al introduced a cloning approach to extend the flexibility of system composition to run-time [5]. 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.

Our design targets users who may have their own existing complex simulation models; thereby 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. We also need to support easy utilization and deployment. Our approaches focus on optimizing and controlling a large-scale distributed simulation using the cloning technology.

2.2 Concepts in Distributed Simulation Cloning

Some terms are defined in our proposed distributed simulation cloning technology. A federate may make clones on its own initiative to explore different scenarios when it reaches a decision point, and such a federate is said to perform **active cloning**. An active cloning results in the creation of new scenarios. Other federates who interact with this federate may have to spawn clones to perform proper interaction with each of the replicas, and those federates are said to perform **passive cloning**. For those federates whose states are not affected by the active cloning, they are capable of operating in the new scenarios in addition to the original one as **shared clones**. The clones created from the same root federate are referred to as **sibling clones**. Those federates within the same scenario are known as **partner federates**.

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 that do not intervene in the execution of any other scenario. In the context of the latter solution, multiple scenarios (created in active cloning) coexist within the same federation session. In order to manage concurrent scenarios within a single federation, we use the Data Distribution Management (DDM) [6] mechanism to partition scenarios. Each scenario is specified with an exclusive **characteristic point region** which is associated to the clones that operate in the respective scenario [2]. To provide reusability to existing simulation federates, a **middleware** approach is adopted to hide the implementation of any cloning related modules. To tackle the problems involved in replicating running federates, the **decoupled federate architecture** is used to separate the simulation model from the local RTI component [3]. A **virtual federate** is built up with the same code as the original federate, while a **physical federate** associates itself with a real local RTI component serving the virtual federate with RTI services.

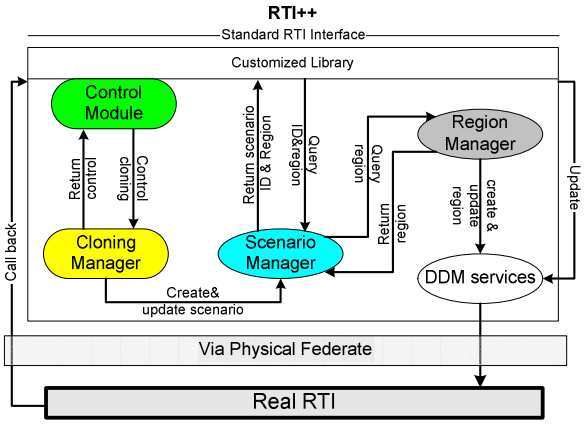

Figure 3.1: RTI++ and internal modules

3. Overall Simulation Cloning Architecture

A RTI++ library to enable simulation cloning is built as the middleware between the simulation model and the real RTI. The middleware contains several major modules which perform the necessary functionalities related to simulation cloning while presenting a standard RTI interface to the simulation model (as shown in Figure 3.1).

Prior to using simulation cloning, the user can specify the conditions according to which the cloning should be triggered and the different actions to be taken. The **Control Module** monitors the states in which the user is interested and evaluates the conditions for cloning the federate at a decision point. The **Cloning Manager** module **(CMM)** creates new clones for the request issued by the Control Module, and it initiates the creation and update of the scenarios. The **Scenario Manager** module creates and stores the scenario tree, from which the identity and corresponding DDM region of each scenario can be fetched. The Scenario Manager keeps the history of the overall cloning procedure. The **Region Manager** module creates DDM regions and manages the regions. The Region Manager services the Scenario Manager by answering enquiries and providing the clone specific region. The Region Manager also deals with any request for modifying a region. The Scenario Manager presents the region to the middleware for partitioning event transmission. The RTI++ Object Management services invoked by a federate are executed via the corresponding DDM methods by associating the region obtained from the Scenario Manager. Eventually it is the physical federate who calls the real RTI services and conveys the callbacks to the RTI++ middleware.

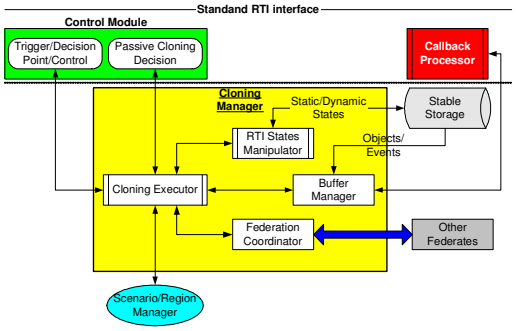

Figure 3.2: Cloning Manager Module

In the context of the middleware approach, the CMM provides services directly to the control module that handles the decision point and cloning trigger. It also interacts with the scenario management module [2] to update/fetch scenario information, such as region and scenario ID etc. Once the condition for cloning is met, the cloning trigger invokes the cloning manager module and the latter performs cloning of the federate accordingly. External stable storage is used to store RTI states of the federate. Figure 3.2 illustrates the internal components inside the CMM.

The Cloning Executor works as the key component of the CMM. It answers the cloning request issued by the Control Module. Upon cloning it interacts with the scenario management module to update and retrieve scenario and characteristic region information [2]. The Cloning Executor makes replicas of the simulation model. In the meantime, it initiates the same number of new physical federate instances. The saved RTI states are loaded from the external stable storage to initialize the physical federates. Once the state of each clone is initialized to the snapshot of the parent federate at this decision point, the Cloning Executor resumes the whole simulation after applying the action defined for the Decision Point to each clone.

The RTI States Manipulator saves RTI states and replicates them when needed. The RTI states include the static states, such as published object classes, and dynamic states, such as federate time and generated events.

The Cloning Executor directly replicates static states for new clones, while the Buffer Manager takes charge in copying dynamic states. A new clone needs to register object instances that represent the same entities known to the simulation model in the parent federate. However, the RTI assigns different handles [7] to the “cloned” objects which are unknown to the simulation model. The Buffer Manager should deal with this problem using an entity mapping approach. We name the object instances registered by the original federates prior to cloning as

Original Object Instances whereas we use **Image Object Instances** to denote those object instances re-registered by the clones in the state recovery procedure. The approach maintains the relationships amongst original entities and "cloned" entities at the RTI level. All incoming events will be mapped (processed) by the Callback Processor [3] prior to conveying them to the simulation model. The Buffer Manager also handles the in-transit events on cloning using the RTI federation save services, which forces all in-transit events to be delivered to all receivers.

During cloning of a federate, the federation coordinator should synchronize other federates within the whole simulation run including the sibling clones. The coordinator is designed to ensure that prior to continuing the simulation run, every clone finishes initialization and all federates have updated their states for the current cloning operation. Only when these conditions are met, can the simulation be reactivated and resumed.

4. Algorithms for Distributed Simulation Cloning

As soon as a federate reaches a decision point, the Control Module evaluates the states in which the user is interested. Once the cloning conditions are met, the Cloning Manager module will make clones immediately to explore alterative executions as predefined in the cloning triggers. Then an active simulation cloning occurs. The partner federates may perform passive cloning or operate as shared clones.

4.1 Active Simulation Cloning

From the perspective of the federate (parent federate) making clones, the process of an active simulation cloning can be described as follows (see Figure 4.1). In this process, the RTI services used by the middleware are directly executed by the physical federate, remaining transparent to the simulation model.

- **Initiating cloning**. The Cloning Executor calls *cloningManager::createClone(number of clones)* to initiate the active cloning. The federate being replicated enters cloning mode, in which control is with the RTI++ middleware.

- **Synchronizing federation**. The Federation Coordinator notifies and synchronizes the remaining federates by requesting a federation save. The federation save label is coded to contain the following information: (1) whether the current simulation cloning is an active one or passive one (2) the number of new clones to be created (3) the handle of the federate making clones, and (4) the scenario within which cloning occurs (cloning scenario). The remaining federates retrieve this label via their callbacks. When they have identified this synchronization is for cloning purpose, their Callback Processors [3] extract the coded information for reference. When the whole federation has completed synchronization, all federates enter cloning mode.

- **Handling in-transit events**. The Buffer Manager ensures all in-transit events reach their destinations prior to cloning. The current design utilizes the federation save services to achieve this goal.

- **Updating scenario tree**. The Scenario Manager of every federate updates its scenario tree [2] based on the cloning scenario ID and clone number. The scenario updating algorithm ensures all federates maintain identical scenario trees. Those federates remaining intact check whether they operate in this cloning scenario as a partner federate or not. A partner federate will work as a shared clone in the current and new scenarios; the region manager needs to merge the new scenarios' point region extents [2] into its existing region and notify the RTI about this region update. The updated region can be written as $region(Scen[1]) \cup \ldots \cup region(Scen[n])$, in which $region(Scen[i])$ represents the characteristic point region associated with the ith scenario in which the shared clone will execute. Non-partner federates directly block until the end of the current cloning.

- **Making clones**. The Cloning Executor makes the specified number of replicas of the simulation model and initiates an individual physical federate for each replica. Within each replica, the Cloning Executor hooks up to the corresponding physical federate and makes it join the existing federation. Thereafter a sibling clone of the parent federate comes into being, which leads to the construction of a new scenario.

- **Replicating system states**. A clone's physical federate needs to be initialized with the stored system states from the parent federate. The system states to be replicated include: (1) object classes/interactions that have been published/subscribed, (2) registered object instances, (3) federate time and (4) buffered events. The state replication of the new clones will be detailed later. The parent federate and partner federates block for the new clones to complete state replication. Each clone reports to the parent federate when fully initialized with its inherited system states. All clones' identities will be collected and forwarded to the partner federates by the parent federate.

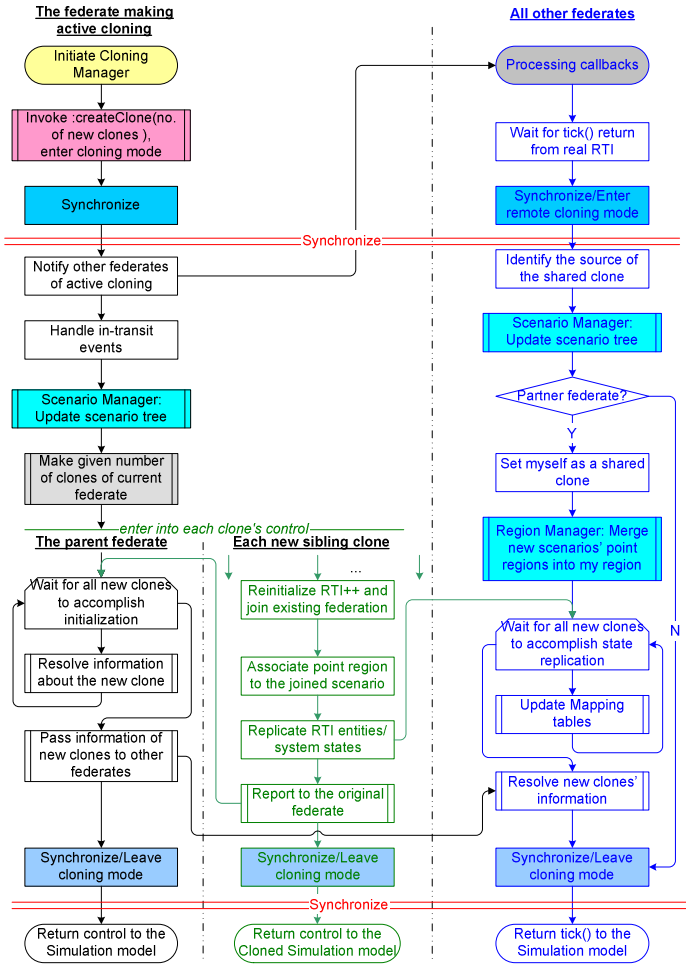

Figure 4.1: Active simulation cloning

- **Leaving cloning mode**. After all clones announce that they are ready, the Federation Coordinator of the parent federate initiates another federation synchronization to declare the termination of the cloning process. The synchronization label codes the relationship between each clone's federate handle and the scenario in which it will operate. The partner federates' Callback Processors can decode the mapping relationships and use them to distinguish the source of incoming events in the future. As soon as synchronization is achieved, the parent federate, its clones and the remaining federates obtain control from the middleware.

After leaving cloning mode, the paused simulation execution resumes with new scenarios starting operation. Each of the new clones and the parent federate execute in a single scenario while the partner federates of the parent initially become shared clones that execute in the original scenario and the new ones. All federates work in normal mode until the next occurrence of simulation cloning.

Figure 4.2 depicts the state replication procedure of a new clone. This procedure involves the parent federate, the new clones and the partner federates, but does not affect the remaining federates. Prior to creating clones, the parent federate sends a "system" interaction (namely "*cloningNotification*", defined for cloning) to the partner federates. This interaction encapsulates the published object/interaction classes and the list of registered objects of the parent federate. The receiver's Callback Processor can simply compare the original object instances discovered previously with this remote object list, and then it can predict what image object instances it should discover from each new clone. Hence information contained in this interaction provides criteria for the partner federate to detect the state replication status of each clone. Furthermore the receiver's Callback Processor will refer to this remote publication information in processing events from the parent federate's sibling clones.

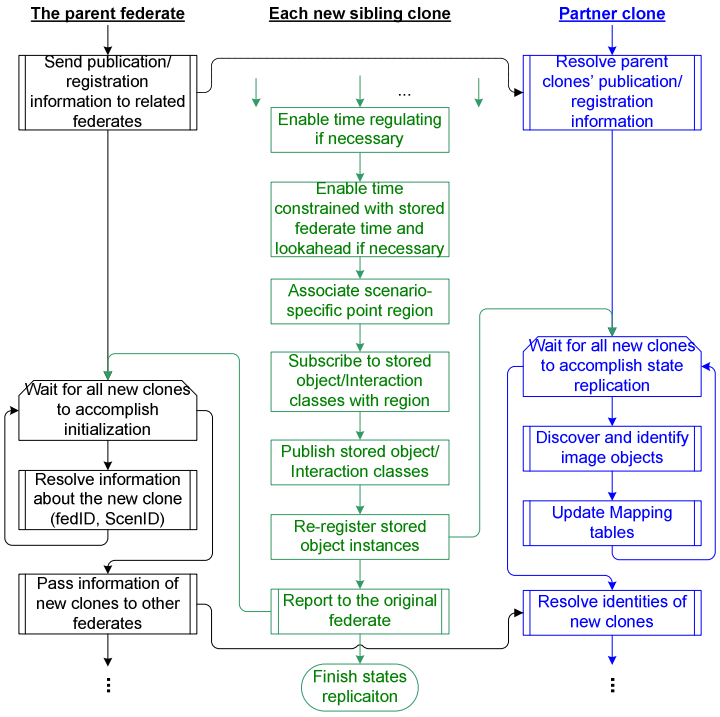

Figure 4.2: Replicating States for new clones

When a clone is created, first it enables time constrained and/or time regulating depending on the parent federate's settings. The federate time and lookahead recorded on cloning should be specified accordingly.

Secondly, the clone's Cloning Manger associates a scenario-specific point region to the physical federate, which is obtained from the scenario tree according to the scenario in which the clone will execute. The States Manipulator instructs the physical federate to subscribe/publish the same object/interaction classes as the parent federate has subscribed/published, but with the newly retrieved region. The partner federates (shared clones) merge the designated point regions of the new scenarios into their currently associated ones.

Thirdly, the **new clone** needs to register image objects to the objects owned by the parent federate. The physical federate invokes the RTI method *RTI::registerObjectClassInstanceWithRegion(objectClass, Name, …)* [7] and conveys the returned object handle to the middleware, and the latter maps the original objects to the image ones properly. The specified "name" of an image object follows the format "<federate handle>&&<original object instance handle>". The middleware of other **partner federates** which are subscribers will discover these image objects (their regions overlap with the clone's) and can map them to the original ones from their names. The mapping relationship (recorded in Mapping Tables) will be used by the Callback Processors to process events from different scenarios. The Callback Processor blocks until all image objects have been discovered, and hides these object discoveries from the simulation model to keep correct HLA semantics. On the other hand, the **new clones** may also detect objects registered by the partner federates before cloning (due to the overlapped region). As the cloned simulation model has "already" discovered those objects, this also should be hidden to avoid repeated discovery. The **parent federate** waits for new clones to be created and initialized. As soon as each clone completes initialization, it reports to the parent federate with a "*cloningReady*" message containing its federate handle and scenario ID. This design puts constraints (see Figure 4.3) on the parent federate and partner federates before leaving cloning mode. Only when all clones finish replicating states may the whole federation resume normal execution. Thus the state consistency of the overall simulation can be achieved.

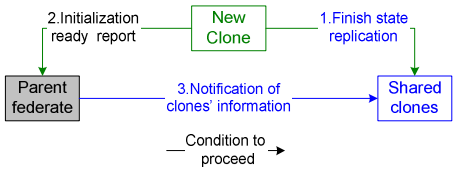

Figure 4.3: Coordinating federates in state replication

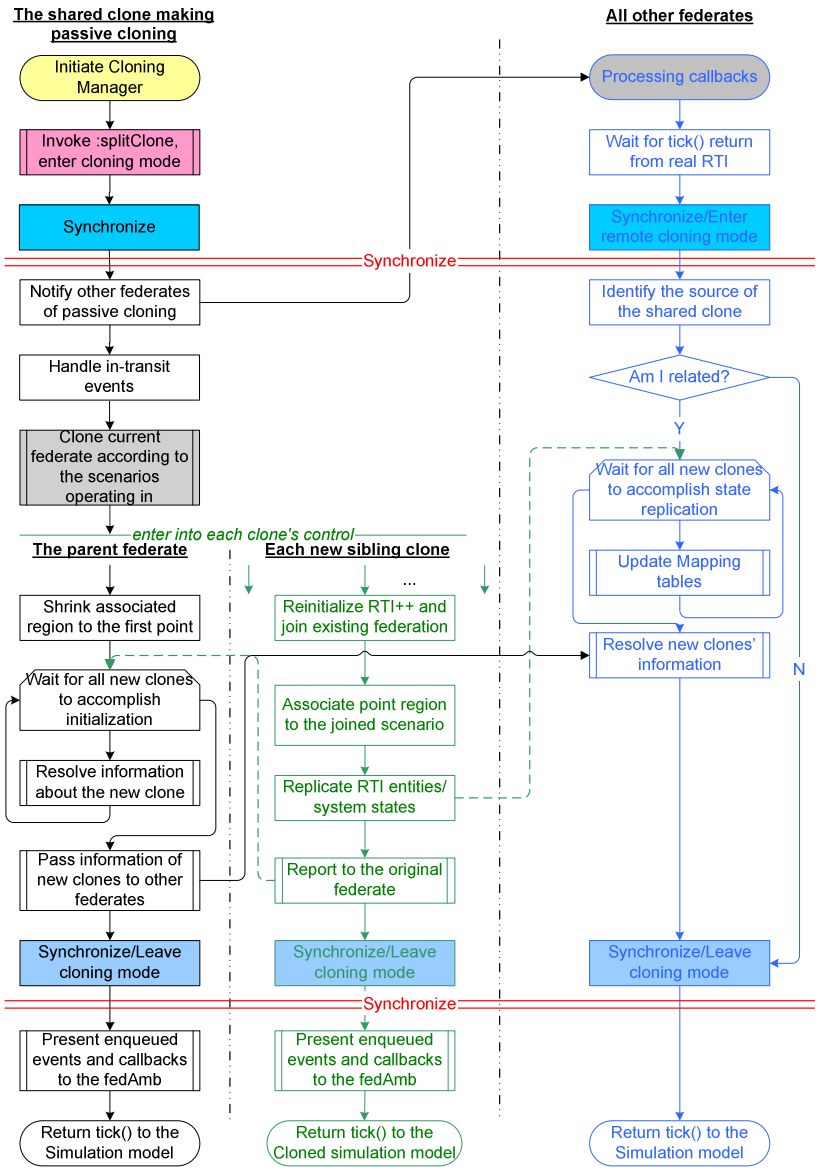

Figure 4.4: Passive simulation cloning

4.2 Passive Simulation Cloning

The partner federates of the federate making active cloning may remain intact and become shared clones in related scenarios, given that their system states are not affected by the cloning immediately. Incoming events of a shared clone are checked by the Callback Processor. The result of checking an event decides whether the shared clone remains shared or requires passive cloning. The event checking algorithm determines which events and how these events should be conveyed to the simulation model as well as when the passive cloning should be triggered. The algorithm checks whether the events from sibling clones are identical or not. It ensures that the Federate Ambassador will reflect only one single event for one full set of identical events obtained from all scenarios involved. This design helps shared clones hide the complexity of processing events from multiple scenarios. As a result, shared clones operate in multiple scenarios as if they only interact with one single scenario independently.

Once different events have been detected from the scenarios in which a shared clone operates, the RTI++ middleware prepares for the forthcoming passive cloning; for example, it sorts the received events according to their

source scenarios. All callbacks are prevented from being delivered to the simulation model until the passive cloning is completed. Such a design aims to keep the semantics of the HLA specification.

As mentioned previously, a shared clone performs passive cloning to handle different events from existing scenarios rather than to explore new scenarios on its own initiative. The passive cloning does not cause creation of new scenarios, and this is different to active cloning. A passive cloning procedure is illustrated in Figure 4.4 from the shared clone's point of view, and can be described as follows:

- **Initiating passive cloning**. The Cloning Executor calls *cloningManager::splitClone()* to initiate the passive cloning, and the shared clone enters cloning mode. The in-transit events are handled in the same way as for active cloning.

- **Synchronizing federation**. The Federation Coordinator notifies and synchronizes with the remaining federates by requesting a federation save. These federates retrieve the federation save tag and identify the current passive cloning from the extracted information. When the whole federation has completed synchronization, all federates enter cloning mode.

- **Creating clones and updating region**. The Cloning Executor creates new clones, and the number of clones is set depending on the number of scenarios in which the parent federate operates beforehand. The original region of the parent federate (denoted as $region(Scen[1]) \cup \ldots \cup region(Scen[n])$) should be split into n individual point regions. The Region Manager of the parent federate and each clone associates an exclusive point region to the physical federate according to the individual scenario in which it may participate. The parent federate becomes a normal federate and so do its new clones, while the remaining federates continue to operate as normal or shared clones. Existing scenarios are kept intact without generating new ones.

- **Replicating system states** (see Figure 4.2). The mapping tables of partner federates will be updated by their Callback Processors according to the re-registered image object instances.

- **Leaving cloning and conveying buffered callbacks**. After each clone finishes initialization, another federation synchronization will be initiated to denote the ending of passive cloning. The Callback Processor of the parent federate or each new clone needs to convey the buffered events to the Federate Ambassador. The events are inherited from the parent federate (for new clones). Only those events with timestamp not greater than current federate time have to be conveyed. Subsequently, each Callback Processor returns control to the simulation model and the latter obtains control again. Thereafter the whole federation resumes normal execution.

5. An Application Example

This section presents a simulation example to verify the correctness and investigate the performance of the proposed distributed simulation cloning technology. The example studies a simple supply chain that comprises an agent company, a factory and a transportation company. The agent keeps issuing orders to the factory, and the latter processes these orders and plans production accordingly. The transportation company is responsible for delivering products of the factory and reports the delivery status. The factory has a limited manufacturing ability, which means it needs to adjust its daily operation policy in the case of fulfilling an extra large order. One of the following candidate policies can be chosen:

a) Keeping normal operation policy (NORMAL policy), thus the factory may get penalized since the order is not fulfilled in time.

b) Encouraging overtime work to ensure on time delivery (OVERTIME policy), the factory has to pay for the extra manpower.

c) Sharing a partial order with other competing factories (SUBCONTRACT policy), this also incurs profit loss and negative impact on future business development.

The analyst focuses on optimizing the extra cost and the profit loss of the factory operation in processing large orders. Three different scenarios need to be constructed to examine the candidate policies.

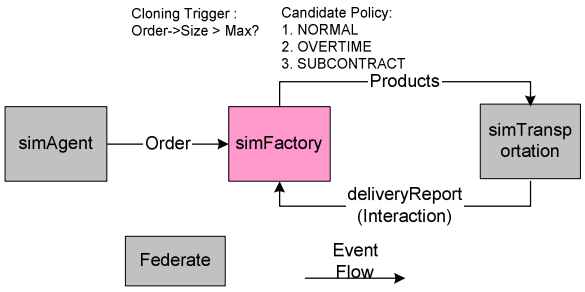

Figure 5.1: An example of distributed supply chain simulation

Table 5.1: Declaration information of the federates

Federate	Object Class and Attribute		Interaction Class and Parameter
	Order	Products	deliveryReport
	Index, Size	*Amount, Index, Date*	*Index, Status*
simAgent	Publish	NIL	NIL
simFactory	Subscribe	Publish	Subscribe
simTransportation	NIL	Subscribe	Publish

Table 5.2: Configuration of experiment test bed

Specification	Computers			
	Workstation1	Workstation2	Server1	Workstation3
Operating System	Sun Solaris OS 5.8	Sun Solaris OS 5.8	Sun Solaris OS 5.8	Win2000 Professional
CPU	Sparcv9 CPU, at 900 MHz	Sparcv9 CPU, at 900 MHz	Sparcv9 CPU * 6, at 248 MHz	Intel 1700 MHz Pentium IV
RAM	1024M	1024M	2048M	256M
Compiler	GCC 2.95.3	GCC 2.95.3	GCC 2.95.3	MS VC++ 6.0
Underlying RTI	DMSO NG 1.3 V6	DMSO NG 1.3 V6	DMSO NG 1.3 V6	DMSO NG 1.3 V6
Processes running on	simFactory	simTransportation	simAgent	RTIEXEC & FEDEX

5.1 Configuration of Experiments

The three nodes in the supply chain can be modeled as three federates as shown in Figure 5.1, namely *simAgent*, *simFactory* and *simTransportation*. These federates form a simple distributed supply chain simulation. Two object classes "*Order*", "*Products*" and one interaction class "*deliveryReport*" are defined in the FOM [7] to represent the types of events exchanged amongst the federates. Table 5.1 gives the classes being published and/or subscribed by the federates. A cloning trigger is predefined for federate *simFactory*, which contains a cloning condition " $Order \rightarrow Size > MAX$? " and three candidate policies. The simulation emulates the supply chain operation of one-year duration. The *simFactory* reports the extra cost incurred in each order and in the whole year at the end of simulation. The simulation starts at time 0 and ends at 361 with one time unit representing one day in reality. The experiments use four computers in total (three workstations and one server), which are interlinked via a 100Mbps-based backbone. Each federate (together with its clones if any) occupies one individual computer, with the RTIEXEC and FEDEX processes running on another computer.

The experiment architecture and platform specification are listed in Table 5.2. Using the same codes, the federates are built into two different versions by linking to: (1) the DMSO RTI library directly (denoted as TRADITIONAL) and (2) the RTI++ middleware library supported by the real RTI library. We can also configure the federates of the 2nd version to execute each of the policies without using cloning (CLONING_DISABLED) or examine multiple policies concurrently using cloning (CLONING_ENABLED).

5.2 Simulation Execution with Cloning Enabled

Figure 5.2 depicts the simulation execution using the CLONING_ENABLED federates. *SA[0], SF[0]* and *ST[0]* are the abbreviations of federate *simAgent*, *simFactory* and *simTransportation* accordingly. Each scenario is marked as *Scen[i]* (i = 0, 1, 2, ...), in which *Scen[0]* denotes the initial scenario. The incremental simulation cloning occurs along the time axis as follows:

At time 0, the simulation initializes with a single scenario *Scen[0]*. When simulation progresses to time T_1, *SF[0]* performs active cloning due to an order with extra large volume, which results in the birth of clones *SF[1]* & *SF[2]*, and new scenarios *Scen[1]* & *Scen[2]*. The remaining federates do not need to be cloned immediately, and they only need to expand their associated region to enable interacting with *SF[1]* & *SF[2]*. Thus *SA[0]* and *ST[0]* become shared clones in both scenarios. The event flow from *SF[i] (i = 0,1,2)* to *C[0]* is named as *ev_F[i] (i = 0,1,2)*. *ST[0]* keeps intact as long as *ev_F[i](i = 0,1,2)* remain identical.

At simulation time T_2, *ev_F[0]* deviates from *ev_F[1]* and *ev_F[2]*, this triggers a passive cloning of *ST[0]* and results in the birth of clones *ST[1]* & *ST[2]*. This passive cloning does not trigger any change of existing scenarios. *SA[0]* persists as a shared clone after that.

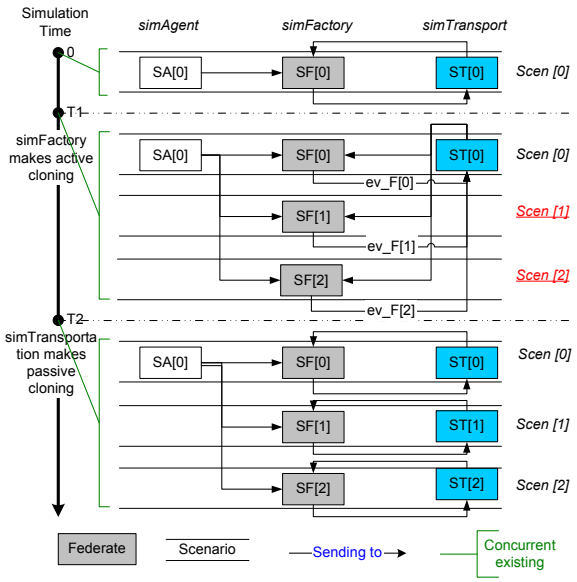

Figure 5.2: Executing simulation using cloning-enabled federates

Clones are created incrementally according to the changing external conditions. We always have: *Total no. of federates* $\leq \sum$ *No. of federates executing in each scenario*. For example, from simulation time T_0 to T_1 there exists only 5 federates simulating 3 scenarios whereas there has to be 9 federates examining the same scenarios in the context of traditional distributed simulations.

5.3 Correctness of Distributed Simulation Cloning

To verify the correctness of the cloning mechanism, we specify federate *simAgent* to generate the same set of orders in different runs. We first execute the TRADITIONAL federates, in which the collected results are used as a reference in subsequent experiments. The second round exploits CLONING_DISABLED federates to execute the three policies one by one. The third round adopts CLONING_ENABLED federates to explore all policies in a single session. The seven experiments in total are indicated in table 5.3.

Table 5.3: Experiments for verifying the correctness of cloning technology

Type of Federates	Candidate Policies		
	NORMAL	OVERTIME	SUB-CONTRACT
TRADITIONAL	*T_Norm*	*T_Over*	*T_Sub*
CLONING_DISABLED	*D_Norm*	*D_Over*	*D_Sub*
CLONING_ENABLED	*E_ALL*		

The outputs obtained using TRADITIONAL federates are summarized as follows:

- *simAgent* issues 240 orders, in which the first extra large order and the last order carries timestamp 203 and 362.5 respectively
- *simfactory* receives 239 orders and makes multiple batches of products according to the policies adopted. The extra costs are reported as $\$1.28678E+7$, $\$6.66488E+6$ and $\$5.99839E+6$ in experiment *T_Norm*, *T_Over* and *T_Sub* respectively
- *simTransportation* receives all product updates issued earlier than the end time and sends *deliveryReport* interactions with respect to these updates.

We have following observations from the experiments using other types of federates:

- Outputs in the experiments with CLONING_DISABLED federates (*D_Norm*, *D_Over* and *D_Sub*) match exactly those using TRADITIONAL federates
- In experiment *E_ALL*, *simFactory* makes two new clones at simulation time 203 to execute the three policies concurrently. When the simulation progresses to time 211, passive cloning occurs in *simTransport* and results in spawning two clones of *simTransport*. Each clone of *simFactory* reports the same results to those in *T_Norm*, *T_Over* and *T_Sub* respectively.

Outputs of the above experiments indicate that the technology provides a correct cloning mechanism to HLA-based distributed simulations. This also proves that the technology ensures complete fidelity in enabling cloning to traditional distributed simulations. In other words the simulation cloning technology does not introduce variation to the simulation results.

5.4 Efficiency of Distributed Simulation Cloning

To investigate the performance of the cloning technology, we carry out another series of experiments to collect the overall execution time using different types of federates to executing all policies. For TRADITONAL and CLONING_DISABLED 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 CLONING_ENABLED federates, we let federate *simAgent* generate different orders, thus federate *simFactory* may trigger active cloning at different stages in each run. The experiments are listed as in table 5.4.

Table 5.4: Experiments for studying the efficiency of cloning technology

Type of Federates	Execution scheme	
	Sequential	Parallel
TRADITIONAL	*T_Seq*	*T_Para*
CLONING_ DISABLED	*D_Seq*	*D_Para*
CLONING_ ENABLED	*E_ALL(multiple records)*	

For the sequential scheme, the summation of the execution time of the three runs is calculated. For the parallel scheme, the arithmetic average of three runs' execution time is recorded. As for experiments with CLONING_ENABLED federates, we select three runs in which cloning of *simFactory* occurs at time 80, 203 and 320. These points represent cloning at the start, middle and end stages respectively. The CPU utilization of one federate (in workstation 1&2) is reported as ~93%. In the case of three federates running on a single workstation, each federate occupies ~32% CPU time. Physical federates have a CPU utilization as low as ~1%. Experimental results are recorded in seconds in Figure 5.3. The average time of executing one single scenario per run is 497 and 504 Seconds using TRADITIONAL and CLONING_DISABLED federates accordingly. The percentage of time saved using cloning technology is indicated in Figure 5.4.

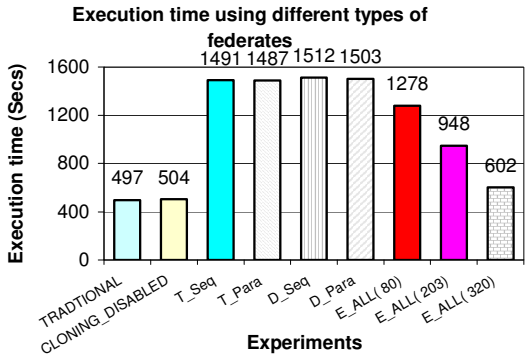

Figure 5.3: Execution time of experiments using different types of federates

Compared with the experiments not using cloning technology, the ones using cloning can reduce execution time significantly. The experimental results indicate that the more common computation there is between scenarios the more execution time can be reduced using cloning. It proves that the proposed technology can help in optimizing computation for HLA-based distributed simulations.

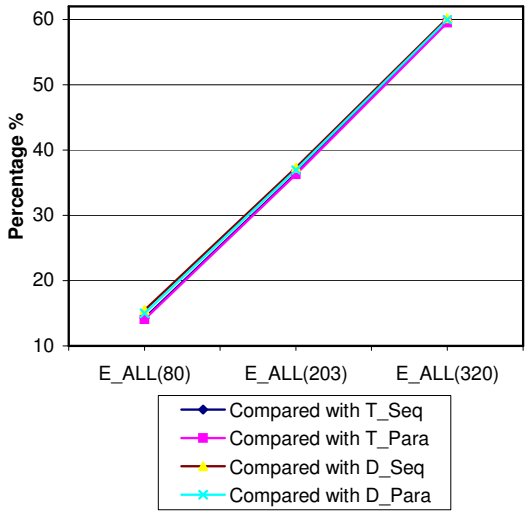

Figure 5.4: Percentage of saved execution time

6. Conclusions and Future Work

In this paper, we have investigated the mechanism and design of managing HLA-based distributed simulation cloning. Our design enables an incremental simulation cloning mechanism. The cloning management module is developed to ensure correct replication of distributed simulations when preset conditions are met at decision points. On cloning simulations, coordination and synchronization are introduced to maintain the state consistency. The management mechanism for shared clones supports the sharing of federates amongst different scenarios, and it minimizes redundant computation based on an event checking algorithm.

A series of experiments has been performed to investigate the correctness and performance of the cloning technology. The experimental results are compared for traditional, cloning-disabled and cloning-enabled federates in terms of outputs' uniformity and computing efficiency. The experimental results show that the technology provides a correct cloning mechanism to existing distributed simulations. The technology can optimize the computation of distributed simulations effectively compared with the ones using traditional federates.

For our future work, we need to further minimize the overhead incurred in the cloning procedure, which includes investigating an alternative approach to dealing with in-transit events on cloning. Another issue is to facilitate fault tolerance using cloning technology.

References

[1] 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*, pp. 378-387, Vol. 71, No. 6, Dec. 1998.

[2] D. Chen, S. J. Turner, B. P. Gan, W. Cai, J. Wei, N. Julka: "Alternative Solutions for Distributed Simulation Cloning" *Simulation: Transactions of the Society for Modeling and Simulation International*, Vol. 79, No. 5-6, pp. 299-315, May-Jun. 2003.

[3] D. Chen, S. J. Turner, B. P. Gan, W. Cai, J. Wei: "A Decoupled Federate Architecture for Distributed Simulation Cloning" *Proceedings of the 15th European Simulation Symposium,* pp. 131-140, Oct. 2003, Delft, the Netherlands.

[4] M. Hybinette and R. M. Fujimoto: "Cloning parallel simulations" *ACM Transactions on Modeling and Computer Simulation*, pp. 378-407, Vol. 11, Oct. 2001.

[5] T. Schulze, S. Straßburger, U. Klein: "HLA-federate Reproduction Procedures in Public Transportation Federations" *Proceedings of the 2000 Summer Computer Simulation Conference,* Jul. 2000, Vancouver, Canada.

[6] K. L. Morse and M. D. Petty: "Data Distribution Management Migration from DoD 1.3 to IEEE 1516" *Proceedings of the Fifth IEEE International Workshop on Distributed Simulation and Real-Time Applications,* pp. 58-65, Aug. 2001, Cincinnati, Ohio, USA.

[7] F. Kuhl, R. Weatherly, J. Dahmann: "Creating Computer Simulation Systems: An Introduction to HLA" ISBN 13-022511-8, Prentice Hall, 1999, USA.

Author Biographies

DAN CHEN is a Research Engineer with the Productions and Logistics Planning Group at Singapore Institute of Manufacturing Technology in Singapore. He received a Bachelor of Science in Applied Physics from Wuhan University, China in 1994, a Master of Engineering in computer science from Huazhong University of Science and Technology, China in 1999, and a Master of Engineering in Computer Engineering from Nanyang Technological University, Singapore in 2002. His research interests are distributed simulation, networking and other relative technologies.

STEPHEN JOHN TURNER joined Nanyang Technological University (Singapore) in 1999 and is currently an Associate Professor in the School of Computer Engineering and Director of the Parallel and Distributed Computing Center. Previously, he was a Senior Lecturer in Computer Science at Exeter University (UK). He received his MA in Mathematics and Computer Science from Cambridge University (UK) and his MSc and PhD 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.

BOON PING GAN is a Research Engineer with the Production and Logistics Planning Group at Singapore Institute of Manufacturing Technology in Singapore. He received a Bachelor of Applied Science in Computer Engineering and Master of Applied Science from Nanyang Technological University of Singapore in 1995 and 1998 respectively. His research interests are parallel and distributed simulation, parallel programs scheduling, and application of genetic algorithms.

WENTONG CAI is currently an associate professor at School of Computer Engineering (SCE), Nanyang Technological University (Singapore). He received his B.Sc. in Computer Science from Nankai University (P. R. China) and Ph.D. also in Computer Science from University of Exeter (U.K.). He was a Post-doctoral Research Fellow at Queen's University (Canada) from Feb 1991 to Jan 1993, and joined SCE as a lecturer in Feb 1993. Dr. Cai has served as program committee members in many international conferences (e.g., PADS, DSRT and PDCS). He is a member of IEEE and his current research interests are mainly in the areas of Parallel & Distributed Simulation and Grid Computing.

MALCOLM YOKE HEAN LOW is a Research Engineer with the Production and Logistics Planning Group at the Singapore Institute of Manufacturing Technology. He received his Bachelor and Master of Applied Science in Computer Engineering from Nanyang Technological University, Singapore in 1997 and 1999 respectively, and a D.Phil. in Computer Science from Oxford University in 2002. His research interests include adaptive tuning and load-balancing for parallel and distributed simulations, and the application of multi-agent technology in supply chain logistics coordination.

JUNHU WEI is working with Nanyang Technological University (Singapore) as a Research Fellow. He received his BE in Automatic Control and ME in System Engineering and PhD in Control Engineering from Xi'an Jiaotong University (China). His current research interests include parallel and distributed simulation, Simulation, Planning and Scheduling of Manufacturing.