行政院國家科學委員會補助專題研究計畫成果報告

計畫名稱：HLA 環境下電腦兵棋演訓系統所需之時間管理技術研究

計畫類別：☑ 個別型計畫　　□ 整合型計畫
計畫編號：NSC 91-2623-7-032-003
執行期間：2002 年 01 月 01 日至 2003 年 12 月 31 日

計畫主持人：黃俊堯　教授
計畫參與人員：董明智
李銘哲

執行單位：淡江大學資訊工程學系

中 華 民 國 92 年 02 月 17 日

The Time Management Mechanism for the HLA-Based Wargame System -- The Unified Time Management

Jiung-yao Huang[1], *Ming-Chih Tung*[2], *Ming-Che Lee*

Department of CS&IE, Tamkang University, Taiwan

[1]*E-mail: jhuang@mail.tku.edu.tw,* [2]*E-mail: mctung@tkgis.tku.edu.tw*

摘要：本研究提出設計時間管理機制的中介層(稱為 *STM)***來整合以** *HLA* **為基礎的電腦兵棋系統的時間推進方式，包括時間間隔方式及事件驅動方式。此** *STM* **簡化兵棋系統程式所須注意的時間標記動作、維護** *Lookahead* **值、及維持一致的時間推進動作。此外，** *STM* **對** *optimistic* **模擬增加了** *time warp* **機制。此研究所提出之** *STM* **使得系統設計者可以大大簡化其系統設計時所遭遇的時間推進問題，使得設計者可以用一致的方式來設計不同型態的電腦兵棋系統。**

ABSTRACT: *This research describes the addition of a middle layer, called the smart time management (STM), to unify the time management services of time-stepped, event-driven and optimistic time advance approaches in the High Level Architecture (HLA). The capabilities of the STM include taking over event's timestamp tagging work, maintaining a lookahead value and unifying different time advance approaches provided by the RTI. In addition, it adopts the time warp mechanism for the optimistic simulation. The STM proposes a unified and scalable mechanism to allow the user to construct an HLA federation with a consistent time management interface when he is solving the synchronization issue. That is, the proposed middle layer enables the user to employ the conservative and optimistic synchronization mechanisms in a consistent fashion. This paper starts with exploring the fundamental of the time management approaches. A unified, scalable and smart synchronizing solution to design a time-based HLA federation is then fully laid out.*

1. Introduction

Modeling and analyzing the timing behavior of the distributed simulation is of widely interest in various fields of science and engineering. The parallel or distributed simulation refers to the execution of discrete-event simulation programs on a multiprocessor system or network of workstations [1]. The research activities of PADS focus on how to achieve high performance distributed (or parallel) simulation while ensuring that all events to be parallelly processed and remained in causally constraint [2]. There have been two main distributed synchronization approaches proposed over the years. They are the conservative synchronization [3] [4] and the optimistic synchronization [5] [6].

The High Level Architecture (HLA) is initiated by the Department of Defense, USA, to support the interoperability among the distributed simulators. It defines a standard architecture for modeling and simulation of a complex system [7]. The time management service of the HLA Run Time Infrastructure (RTI) includes the synchronization mechanism to ensure the attribute/event ordering among distributed federates (distributed nodes of a simulation) as required. The time management service provides conservative and optimistic approaches to synchronize different federates with the federation (a distributed simulation) [8]. However, it is not an easy task to use these synchronizing services to build parallel/distributed federates. Even though the RTI provides some synchronization interfaces for the distributed federates, several critical design issues such as the time-stepped, event-driven, federate time, logical time advancement, rollback, fossil-collection and state-saving must be carefully studied in the developing phase [9][10]. Different types of federates require distinct synchronization schemes.

Furthermore, the application specific characteristics of a time-based federate such as lookahead, communication patterns among the model components, checkpoints, etc., may have profoundly affected the feasibility of using a specific protocol to simulate a given model. For a given federation, there may be a case that different federates possess conflict characteristics. Hence, a federate can only be effectively simulated by a conservative protocol whereas others may be more amenable to the optimistic method. When we are designing a synchronized federation, there are many critical federates design issues must be carefully studied and evaluated such as the time policies (time-constrained or time-regulating), message ordering definitions (TSO or RO) and logical time advance strategies (time-stepped, event-driven, or optimistic), etc.. Hence, correct design decisions often require experiences in the RTI services and distributed simulation technologies.

This paper describes a mechanism called Smart Time Management (STM) which proposes a unified and consistent synchronizing scheme for the time management services that have been defined in the HLA. Furthermore, the STM extends smart rollback, state-saving and fossil-collection technologies for optimistic federates. This paper first discusses an approach to unify the use of conservative and optimistic time management methods. The proposed approach integrates the conservative and optimistic mechanisms so that the federate developer can easily accomplish the synchronization scheme regardless of the time policies, message ordering methods and the logical time advance strategies being used.

The unified middle layer (wrapper) for the time management interface provided by the RTI is then presented. This middle layer smartly provides the rollback, state-saving and fossil-collection management for the optimistic federates. It helps the programmer to develop the optimistic synchronization mechanism like a conservative one.

2. The Time Management in the HLA

The HLA time management service is concerned with the mechanisms for the simulators to advance their logical time through the simulation time. Time advance of a synchronized federation is coordinated with the object management service by the RTI so that information is delivered to federates in a causally correct and ordered fashion.

2.1 Messages Order and Timestamp

The HLA time management service is strongly related to the services of the message exchange, such as attribute updates and interaction exchange. There are two general types for ordering messages under the HLA. They are receive-order (RO) and timestamp-order (TSO). RO messages are simply placed in an FIFO queue and are immediately eligible for delivery to the federate on their arrival. On the other hand, each TSO message is tagged with a timestamp by the sending federate and is delivered to the receiving federate in the order of non-decreasing timestamps. The incoming TSO messages are placed in a queue within the RTI and will not be delivered to the federate until the RTI can guarantee that there will be no TSO messages out of order for that federate.

To ensure the received TSO messages are in order, the RTI must compute a Lower Bound of the Time Stamp (LBTS) of the future messages that it may receive from other federates. Several algorithms for the LBTS computation have been proposed over the years [10]. To compute the LBTS, the RTI must consider

- the smallest timestamp of any TSO message that any federate might generate in the future, and
- the timestamps of messages within the RTIs and the interconnecting network.

In order to allow the RTI to compute the LBTS, a federate must use the time management services (as appropriate for the internal time advance mechanism of the federate) which will be described in the next section.

2.2 Advancing Logical Time

As mentioned previously, the RTI guarantees a federate will not receive any TSO messages with timestamp less than its current logical time. To realize this capability, federates cannot autonomously advance their logical times. They have to explicitly request for the advancement of their logical times. This time advancement is not allowed to take place before the RTI explicitly grants it. The protocol for advancing logical time is the core of the HLA Time Management services. The complete the time management cycle consists of the following three steps [8]:

1. A federate requests for advancement of its logical time by calling the appropriate RTI service.
2. The federate receives zero or more messages from the RTI (e.g., receives the Reflect Attribute Value or Receive Interaction callback from the RTI)
3. The federate receives a *timeAdvanceGrant* callback from the RTI to indicate that its logical time can been advanced.

There are three different services to request the advancing of logical time [11] [12]: *timeAdvanceRequest* (TAR), *nextEventRequest* (NER), and *flushQueueRequest* (FQR). TAR is well suited for the federate that internally uses time-stepped mechanisms. NER is the preferred alternative for the event-driven federates. FQR can be used for the optimistically synchronized federates to request the out-of-order delivery of events.

3. Time Advancement Analysis

This paper proposes a middle layer, called Smart Time Management (STM), to unify the time management advance mechanisms used by event-driven, time-stepped and optimistic federates. Under the STM, the event-driven and time-stepped federates can use a consistent conservative synchronization method to process "safe" TSO events. Whereas, the STM implements the time warp method for the optimistic federate.

In this section, we first analyze the processing procedure of three different types of the time advance mechanism. The illustrations of how the STM unifies the usage of time management mechanisms are then elaborated in the next following sections. For the following discussions, we symbolize the timestamp of an event *e* as TS(*e*), an event *e* with a timestamp TS(*e*) as *e@TS*, internal event *j* as ie_j, and external event *i* as ee_i.

- **Event-driven.** This type of federate processes both internal events $ie_j@TS$ and external events $ee_i@TS$ generated by other federates in the timestamp order. The federate time is typically advanced to the timestamp of the event that it is processed. Algorithm 1 [9] shows the event processing algorithm for the event-driven federate. It merges external TSO events with internal events so that all events can be processed in the timestamp order.

```
While (simulation still in progress)
{
    invoke **Next Event Request**($TS(ie_j)$)
    The RTI delivers next external TSO event, if any exists, with $TS(ee_i) <= TS(ie_j)$
    The RTI advances federate's **logical time** by callback **Time Advance Grant**
    if (any external event received from above Next Event Request service call)
        process external event(s) delivered to the federate ($ee_i@TS$)
    else
        process next internal event ($ie_j@TS$)
}
```

Algorithm 1. The event processing algorithm for the event-driven federate.

```
While (simulation still in progress)
{
    invoke Time Advance Request(Now+timestep)
    The RTI delivers all TSO events with timestamp <=(Now+timestep) if any exist
    The RTI advances federate's logical time by callback Time Advance Grant
    Merge and order the external event(s) ($ee_i@TS$) and internal events ($ie_j@TS$)
    Process events with the timestamp order
}
```

Algorithm 2. The event processing algorithm for the time-stepped federate.

- **Time-stepped.** Each time advance made by the federate is a fixed duration of simulation time, called a **time step**. The simulator does not advance to the next time step until all simulation activities associated with the current time step have been completed. Algorithm 2 [9] illustrates the event processing algorithm for the time-stepped federate. It first receives all external events with their timestamps no greater than *Now+timestep*. These TSO events are then merged with internal events so that all events are processed in the timestamp order.

- **Optimistic.** This synchronization protocol allows the federate to process the external events out of the timestamp order and also provides a means to recover from such errors, typically through using a roll-back mechanism. As depicted in Algorithm 3 [8], the optimistic synchronization allows each federate to execute as fast as it can without concerning the possibility of causal violations caused by the received external events. If an event with a timestamp less than the current federate time (simulation time) is received (This late arriving event is referred as a "Straggler Event"), the federate processes rollbacks to an earlier saved state, called a checkpoint, with its time tag less than the timestamp of this straggler event and sends anti-messages to revoke the previously sent messages [1] [13]. The rollback distance depends on the state saving period. A short saving period reduces the rollback overhead but increases the state saving overhead. After restoring the checkpoint state, the federate processes all reordered events and then moves forward recklessly once more. For the time warp mechanism, the GVT (Global Virtual Time) is defined to be the timestamp of the smallest unprocessed event in the federation. Hence, any checkpoint states and anti-messages with their timestamps less than GVT can be released from memory [14] [15] [16].

```
while (simulation execution is still in progress)
{
    Next_Event_Time = timestamp of next internal event
    invoke Flush Queue Request (Next_Event_Time)
    honor the RTI requests for Reflect Attribute Values, Receive Interaction or event retractions
        (cancellations). Place these incoming messages to queues of the federate. Process any
        rollbacks or annihilations. Use message retraction to cancel previously sent messages
    the RTI advances federate's GVT by callback Time Advance Grant
    FossilCollect(GVT)
    process the next smallest timestamped message(s) and advance the federate time
}
```

Algorithm 3. The event processing algorithm for the optimistic (time-warp) federate.

From the above studies, we notice that the differences among these three time management services are the ways in which they handle the internal and external events in the timestamp order. Hence, in order to analyze them, we need to formulate the internal and external events first. Let the federate's external event set as EE = $\{ ee_i@TS, \forall i\}$ and its internal event set as IE = $\{ ie_j@TS, \forall j \}$. We can then define the merged and ordered IE and EE event set of a federate as the set E.

$$E = \{ e_x@TS \mid e_x \in \{ee_i \bigcup ie_j\},\ x = 1,2,3,.....,i+j\}$$

with the following property

$$TS(e_a) <= TS(e_b) \text{ when } a <= b,\ \forall e_a, e_b \in \{e_x\}$$

In a federate execution, all events of E are processed in the same sequence, i.e. $e_1, e_2, e_3, \ldots e_{i+j}$, no matter what type of time advance mechanism is used. The difference among distinct types of federates is the time advancing method to request the RTI to callback the received external events. Therefore, the key concept of the STM is to track the internal events of a federate as well as the received external events and regulate the time advancing services in a unified method.

4. The STM Infrastructure

The STM provides not only a unified but also a smart way to use the HLA time management services. The capabilities of the STM include taking over event's timestamp tagging work, maintaining a lookahead value and unifying different time advance approaches provided by the RTI. The STM contains two time advancing modes, the manual-mode and auto-mode, for the developer to design a federate. In the manual-mode, the STM proposes an approach to unify the interface of time-stepped, event-driven and optimistic time advance mechanisms provided by the RTI. The auto-mode, on the other hand, provides an automatic time advancing approach by autonomously requesting external events from the RTI. Significantly, the synchronization among these federates are consistent under the STM. Hence, the STM can help the developer to design federates without spending too much effort in the time synchronization issue.

An abstract view of the STM infrastructure is provided in Figure 1. The STM creates a middle layer to integrate different HLA time management synchronization mechanisms. Furthermore, its runtime contains the HLA extensions to translate event notifications, to create an *RTIAmbassador* object (see [12] for detail RTI ambassador information), and to manage the time. The major concept of the STM is to integrate and maintain internal and external events of a federate. It aids the federate to process all events with the timestamp order. In order to achieve the goal of unified and smart usage of the distinct time advance approaches, the STM contains the HLA extensions to manage the event's timestamp, lookahead and federate time to track the logical time advance situation of the federate time. It can automatically tag the timestamp of outgoing TSO events by maintaining the lookahead value. Furthermore, it can detect and control the time advancement and the handle rollback/recovery process for the optimistic approach by managing the values of incoming/outgoing event's timestamp and the federate time.

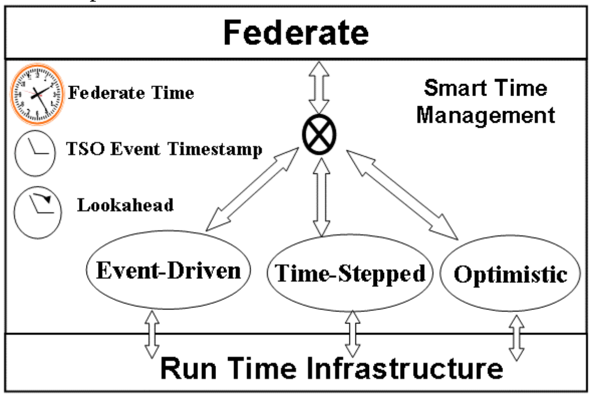

Figure 1. The STM is the middle layer to unify and use the RTI synchronization mechanisms

The basic model of the STM is to integrate all of the time management information. It can be equated by Eq.(1).

TM^{S} = < SS, EE, d, $OE^{\tilde{a}}$, TA, f_{out}, f_{in}, M, Now, L>…(1)
Where:

SS : a saved state variable (checkpoint) [S, T_{ss}] where S is a saved state that includes federate time, Lookahead value and checkpointed ee_i, and T_{ss} is the checkpoint timestamp of S.

EE : a set of external events, i.e. EE = { $ee_i@TS \mid T_{ss} < TS(ee_i) <= Now, \ \forall i$ }

d : the function that generates the anti-messages for the outgoing events. That is, d($oe_x@TS$) = $oe_x^{\tilde{a}}@TS$, where $oe_x@TS$ represents an outgoing event oe_x sent out at time TS(oe_x)

$OE^{\tilde{a}}$: a set of outgoing event anti-messages, i.e. $OE^{\tilde{a}}$ = { $oe_x^{\tilde{a}}$@TS | $T_{ss} < TS(oe_x) <= Now, \ \forall x$ }.

TA: the employed time advance mechanism, which can be event-driven, time-stepped, or optimistic.

f_{out} : the translation function to insert the timestamp into output TSO events and to transform a time advancing call into the method specified by TA.

f_{in} : the translation function to extract timestamp from external TSO events and to read the granted time from the *timeAdvanceGrant* callback.

M: the flag of the STM mode, which is either manual-mode or auto-mode.

Now: the current federate time which is updated by f_{in} when it has receives the *timeAdvanceGrant* callback or external event.

L : the lookahead value.

4.1 Time Maintenance

The Eq.(1) shows that STM maintains the federate time internally. In the HLA specification, notifications from the RTI to the federate are implemented as callback functions. Hence, each federate needs to implement a set of predefined virtual functions and passes them to the RTI via the *FederateAmbassador object* [12]. In the discrete event simulation, each simulation time changed in the system is the result of processing a timestamped event. Therefore, the time-related callback functions from the RTI must be intercepted by the STM translation function f_{in} to extract the timestamps. The callback events must have an associated timestamp that indicates the logical time at which they should be processed. As illustrated in Figure 2, the STM records the largest timestamp of external events that are received from the RTI callback. These external events are then forwarded to the federate. Notice that, the federate time of the event-driven and time-stepped federates are advanced by *timeAdvanceGrant* callback, yet the STM for the optimistic federate advances its federate time according to the timestamp of the processed external/internal events. The GVT of an optimistic federate is the minimum timestamp of the external events or anti-messages that may later arrive. It's the smallest time point where a federate will not rollover.

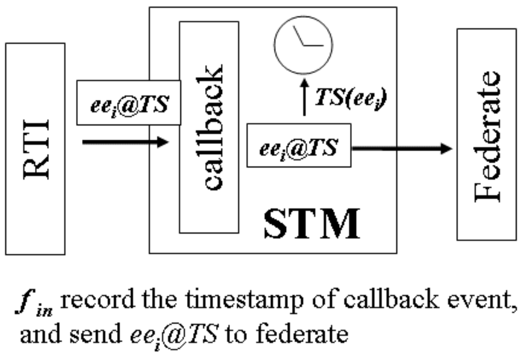
RTI
$ee_i@TS$
callback
$TS(ee_i)$
$ee_i@TS$
Federate
STM
f_{in} record the timestamp of callback event, and send $ee_i@TS$ to federate

Figure 2. The STM receive the external events from the RTI and records its timestamp.

The federate time is the current simulation time of a federate. It is defined as the earliest time that a regulating federate will tag an outgoing message [11]. It is possible for a regulating federate to send a message at time *Now*+Lookahead. As depicted in Figure 3, the STM translation function f_{out} will check every outgoing TSO event, oe_x, and insert the timestamp, *Now*+Lookahead, if it has an empty timestamp. The STM then sends this amended event to the RTI.

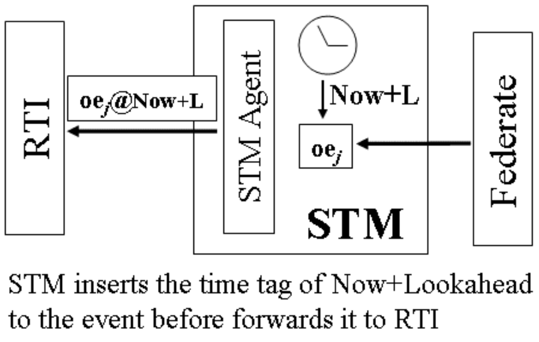
RTI
$oe_j@$Now+L
STM Agent
Now+L
oe_j
Federate
STM
STM inserts the time tag of Now+Lookahead to the event before forwards it to RTI

Figure 3. The STM inserts timestamp to the outgoing events

4.2 Changing Lookahead Value

The Lookahead of a federate is the interval between the current federate time and the earliest time that the federate can use to timestamp a message. It is possible to have a zero-lookahead value [10] [17] [18] such that the effective lookahead of the federate can be an epsilon, which is defined as the smallest possible value of a timestamp, depending upon the employed method to advance its federate time. A lookahead value is necessary to maintain a deterministic causality. In the absence of the lookahead value, it is possible for the outcome of a sequence of events to be affected by other factors such as the network propagation delay. However, if the lookahead value is not mentioned, it is reasonable to assume that the presence or absence of the lookahead will not influent the causality of the received events..

The lookahead value can be changed dynamically during the execution. If a federate increases the lookahead, it can be changed immediately. On the contrary, the lookahead value cannot be instantaneously reduced upon request. At any instance, the lookahead L of a federate indicates to the simulation executive that this federate will not generate any new event with timestamp less than *Now*+L. Notice that the federate time is also maintained by the STM. Any outgoing event with its timestamp less than *Now*+L will be rejected by the STM. If the lookahead is reduced by *X* units, the STM will not take this change into effect until the federate time advance *X* units. In this way, no outgoing events will be inserted timestamp less than *Now*+L.

5. The Manual-Mode STM

Figure 4 sketches the infrastructure of the STM in the manual-mode operation.

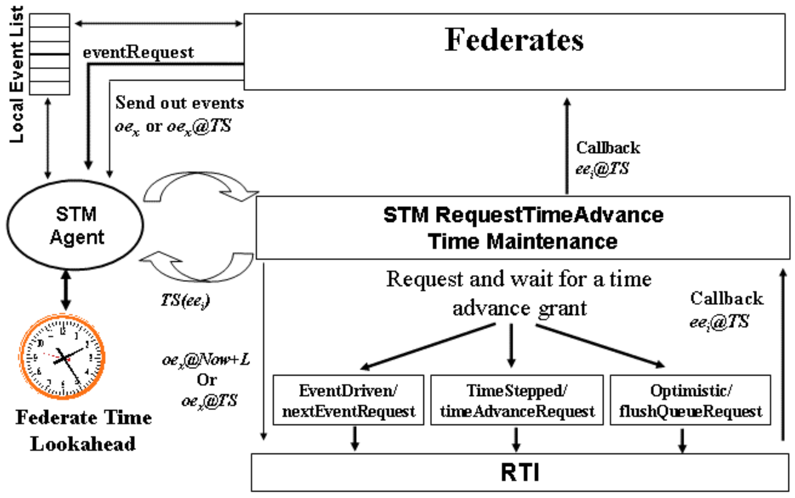

Figure 4. The STM in the manual-mode

The STM provides a unified service, called *eventRequest*, whether it is a time-stepped, event-driven or optimistic, to request for the time advancement. When a federate uses the *eventRequest* service call to request for the time advancement, the f_{out} first checks the TA value, then translates the request into appropriate time advancement service call to the RTI. This procedure is transparent to the federate. The f_{out} will intercept the outgoing TSO events oe_x and insert timestamp *Now*+L, if required, into oe_x first before send it out to the RTI. When the RTI calls back any external event, f_{in} will extract its timestamp and read the *timeAdvanceGrant* value to updates the federate time, *Now*. Notice that, the EE, $OE^{ã}$, SS and d are only used when TA is optimistic, which will be elaborated in section 5.3.

The question of how and when the different types of federates can advance its federate time is then raised. The interplay of the TM^{S} model, Eq.(1), of a federate with the STM affects the conditions of providing the time advance mechanism and will be discussed in the following sections 5.1, 5.2 and 5.3.

5.1 Time-Stepped Simulation

With the HLA specification, the most basic form of time advancement is by calling the *timeAdvanceRequest* (TAR) service [11]. This service can be considered as a time stepping form of the time advancement. The *timeAdvanceRequest* informs the RTI that the federate intends to unconditionally move forward from its current time to the requested time. When a TAR is completed, the requesting federate will receive all timestamped messages with timestamps less than or equal to the requested time and its federate time will be adjusted as requested. All messages received after the TAR will have a timestamp greater than the TAR requested time.

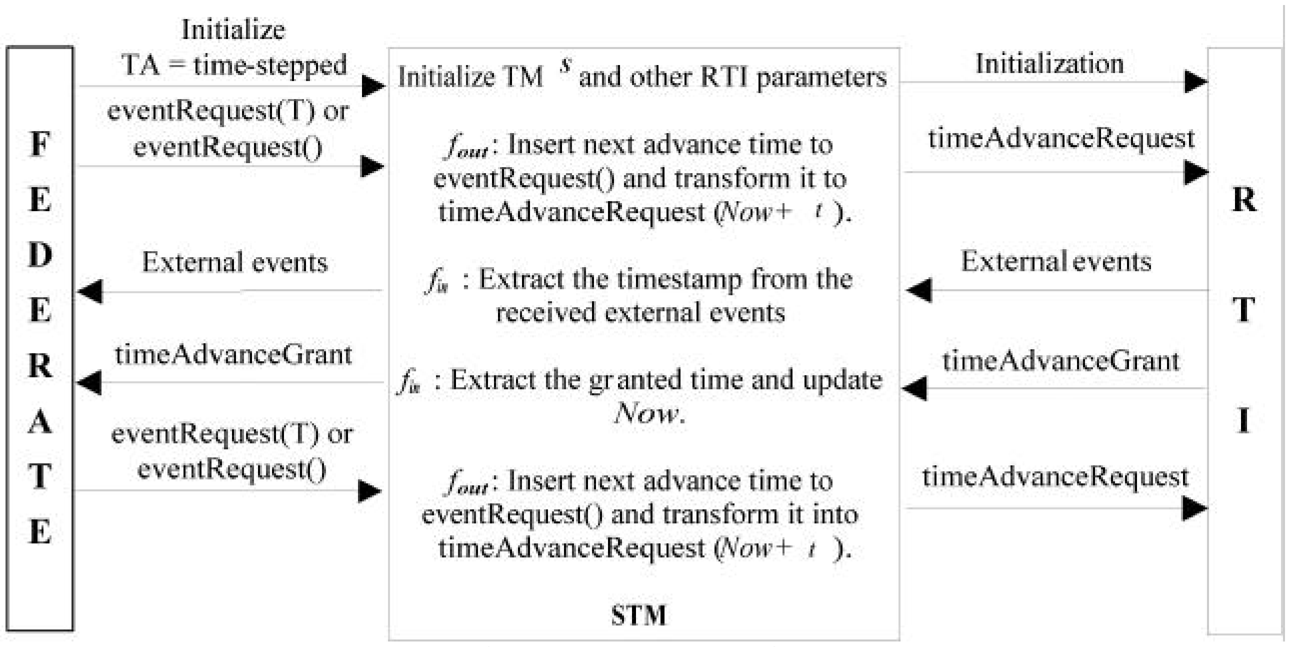
FEDERATE
Initialize TA = time-stepped
eventRequest(T) or eventRequest()
External events
timeAdvanceGrant
eventRequest(T) or eventRequest()
Initialize TM S and other RTI parameters
f_out: Insert next advance time to eventRequest() and transform it to timeAdvanceRequest (Now+ t).
f_in : Extract the timestamp from the received external events
f_in : Extract the granted time and update Now.
f_out: Insert next advance time to eventRequest() and transform it into timeAdvanceRequest (Now+ t).
STM
Initialization
timeAdvanceRequest
External events
timeAdvanceGrant
timeAdvanceRequest
RTI

Figure 5. The time diagram of the manual-mode STM for the time-stepped simulation.

Figure 5 illustrates the time diagram of the STM in the manual-mode to transform the unified time advance function *eventRequest* into the time-stepped advance mechanism. In this case, the STM model of Eq.(1) becomes :

TM^{S} = < f , f , f , f , time-stepped, f_{out}, f_{in}, manual-mode, Now, L>…..(2)

Whereas:

- SS = f , since it is not used when TA = time-stepped.
- EE = f , since it is not used when TA = time-stepped.
- d = f , since it is not used when TA = time-stepped.
- $OE^{ã}$ = f , since it is not used when TA = time-stepped.
- TA = time-stepped. It is initialized by the federate and the federate must give the time step value, $\triangle t$, during the initialization.
- M = manual-mode and it is initialized by the federate.
- Now = the current simulation time and it is initialized by the federate.
- L = the lookahead value and it is initialized by the federate.

In this TM^{S} model, a federate invokes *eventRequest()* or *eventRequest(T)* to the STM. The translation function f_{out} then checks the values of *TA* , *Now* and $\triangle t$ first, and then transforms this call into the time-stepped advancing function, *timeAdvanceRequest*. f_{out} also checks the outgoing event oe_{x} to insert appropriate timestamp, *Now*+L, if it is not timestamped.

The lookahead value L can be dynamically changed during the federate execution. The STM supports zero-lookahead transition as well. When f_{out} detects L becoming zero, it will transform *eventRequest* into the *timeAdvanceRequestAvailable* function.

5.2 Event-Driven Simulation

The *nextEventRequest* (NER) tells the RTI that given the current timing information, this federate wants to move to the requested time unless there exists a received external event with a smaller timestamp. If so, this external event (and all other external events with the same time) will be delivered to the federate and only grant the federate time to the time of this event. The requested time of NER call is used to limit how far ahead in time that the federate can move. Hence, in the absence of a new event in the TSO queue, an NER is treated like a TAR. Upon the completion of the NER, the federate will receive all messages with timestamps equal to or less than the current federate time.

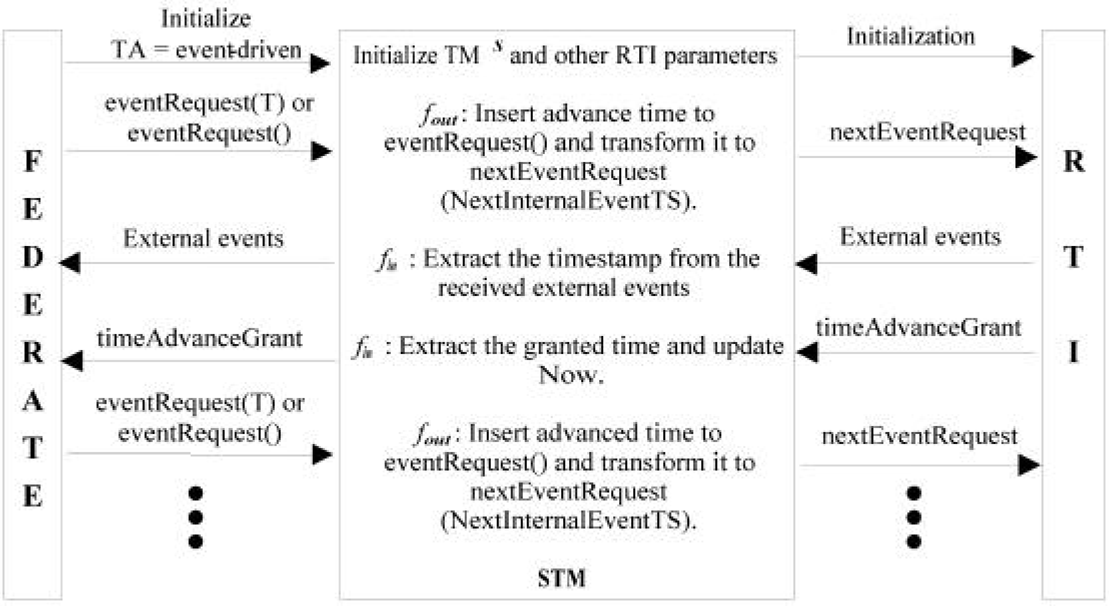
Initialize
TA = event-driven
Initialize TM ^{S} and other RTI parameters
Initialization
eventRequest(T) or eventRequest()
f_{out}: Insert advance time to eventRequest() and transform it to nextEventRequest (NextInternalEventTS).
nextEventRequest
FEDERATE
RTI
External events
f_{in} : Extract the timestamp from the received external events
External events
timeAdvanceGrant
f_{in} : Extract the granted time and update Now.
timeAdvanceGrant
eventRequest(T) or eventRequest()
f_{out}: Insert advanced time to eventRequest() and transform it to nextEventRequest (NextInternalEventTS).
nextEventRequest
STM

Figure 6. The time diagram of the manual-mode STM for the event-driven simulation.

Figure 6 is the time diagram of the STM in the manual-mode for the event-driven simulation. In this case, the STM model becomes Eq.(3) as follows:

TM^S = < f , f , f , f , event-driven, f_{out}, f_{in}, manual-mode, Now, L>…..(3)

Where different from Eq.(2):

SS = f , since it is not used when TA = event-driven.

EE = f , since it is not used when TA = event-driven.

d = f , since not used when TA = event-driven.

OE = f , since it is not used when TA = event-driven.

TA = event-driven and it is initialized by the federate.

When the federate makes the *eventRequest* call to the STM, the translation function f_{out} checks the values of TA, *Now* and L in TM^S first, and then transforms this call into the event-driven function, *nextEventRequest*. The f_{out} will also inspect every outgoing event oe_x and insert appropriate timestamp into it if it is not timestamped.

Furthermore, when f_{out} detects the value of L to be zero, it will transform eventRequest into nextEventRequestAvailable function to support the zero-lookahead.

5.3 Optimistic Simulation

The final mechanism for the time advancement is the optimistic approach, which will cause the RTI to deliver all messages currently in its TSO queue to the requesting federate, regardless of the relative LBTS and the timestamp on the message. The federate will then advance time to the timestamp of the last processed internal/external event. Notice that, since the RTI will deliver all queued messages to the federate, some of them may have their timestamps lager than LBTS and messages with smaller timestamps may arrive in the subsequent time advancing step. This approach is often useful for the loosely coupled federation where actual interactions among federates are relatively infrequent.

An optimistic distributed simulation for the HLA was proposed by [19], [20], [21]. They proposed an extra mechanism, called the rollback manager, to implement the state saving and the rollback management for the optimistic federate in the HLA. This paper adapts and extends this optimistic mechanism for the STM.

Figure 7 illustrates the time diagram of the manual-mode STM for the optimistic simulation that transforms the unified time advance function *eventRequest* into a smart optimistic advance mechanism. In the case, the STM model becomes Eq.(4).

TM^S = < SS, EE, d, OE$^{\tilde{a}}$, optimistic, f_{out}, f_{in}, manual-mode, Now, L>…..(4)

Where different from Eq.(1):

TA = optimistic.

M = manual-mode.

The *eventRequest* function is modified by the translation function f_{out} into *flushQueueRequest* to interact with the RTI to obtain all of the external TSO events. The receiving process of the external events is executed in two phases. The f_{in} first detects and extracts all external events received from the RTI and saved in EE for which its timestamp $TS(ee_i)$ <= LBTS. The STM then sets T_{ss} = max{$TS(ee_i)$ | $TS(ee_i)$ <= LBTS, $\forall$ i}. At this point, the STM receives all "safe" external events which have their timestamp less than LBTS. The time T_{ss} can be considered as the checkpoint time which indicates a point in the simulation time that the state of federate has no risk of rollback. Therefore, the STM saves this state and its current federate time in SS as a checkpoint. Furthermore, the STM uses *saveStateNotification(S, T_{ss})* to notify the federate that the STM has set a checkpoint state S at federate time T_{ss}.

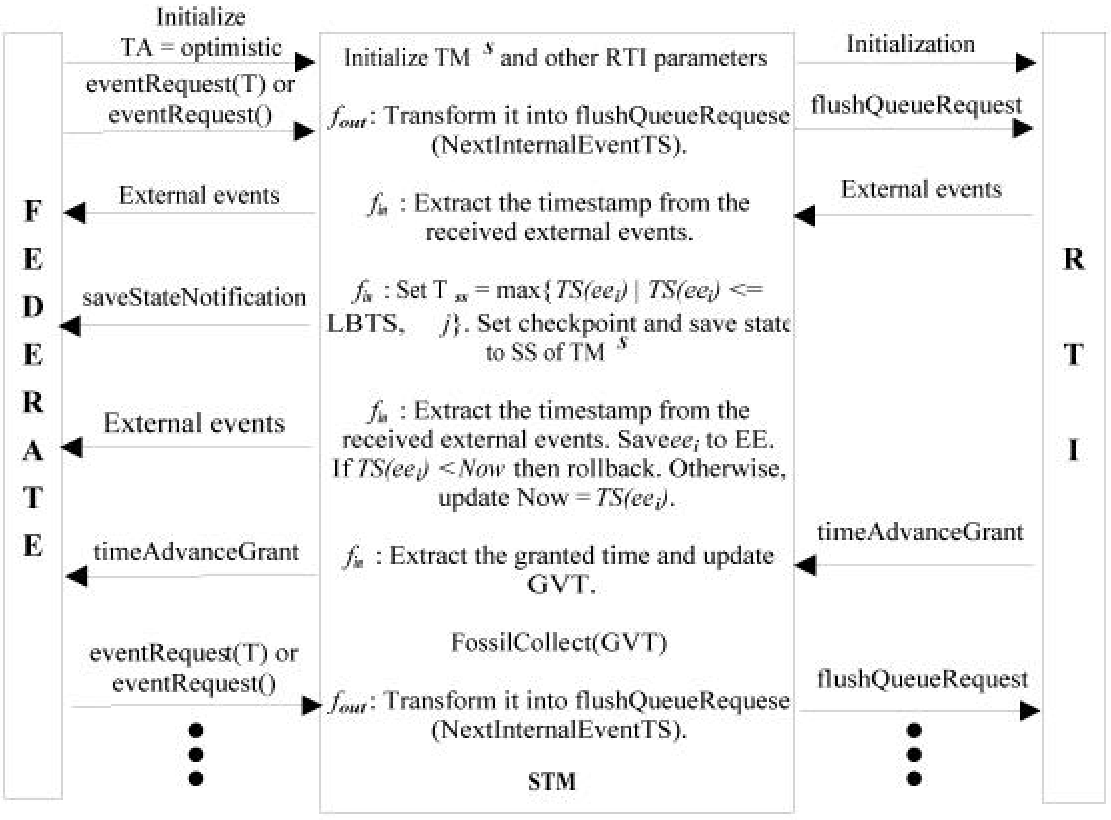
Initialize
TA = optimistic
Initialize TM ^{S} and other RTI parameters
Initialization
eventRequest(T) or eventRequest()
f_{out}: Transform it into flushQueueRequese (NextInternalEventTS).
flushQueueRequest
External events
f_{in} : Extract the timestamp from the received external events.
External events
FEDERATE
RTI
saveStateNotification
f_{in} : Set T $_{ss}$ = max{ $TS(ee_i)$ | $TS(ee_i)$ <= LBTS, j}. Set checkpoint and save state to SS of TM ^{S}
External events
f_{in} : Extract the timestamp from the received external events. Save ee_i to EE. If $TS(ee_i)$ < Now then rollback. Otherwise, update Now = $TS(ee_i)$.
timeAdvanceGrant
f_{in} : Extract the granted time and update GVT.
timeAdvanceGrant
FossilCollect(GVT)
eventRequest(T) or eventRequest()
f_{out}: Transform it into flushQueueRequese (NextInternalEventTS).
flushQueueRequest
STM

Figure 7. The time diagram of the manual-mode STM for the optimistic simulation

The f_{in} then begins to receive the rest of the external events and insert these external events to EE. Since the RTI does not guarantee that the external events with smaller timestamps won't be received in the future, these events are considered unsafe and may suffer from rollback. In this way, if a rollback occurs, the STM contains all of the needed information to cancel the scheduled events and restore the previously saved safe state of the federate.

The STM also keeps tracking all of the anti-messages of the outgoing events with $T_{ss} < TS(oe_x) <= Now$ so that they can be cancelled. If f_{in} receives a straggler event that has a timestamp less than the current time, *Now*, or receives *requestRetraction* callback, the STM then begins to rollback as illustrated in Figure 8.

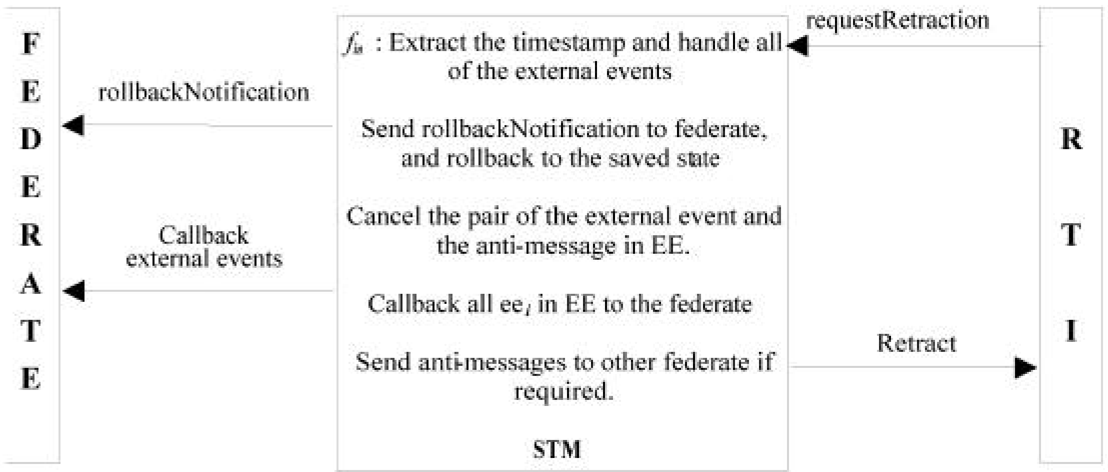
requestRetraction
f_{in} : Extract the timestamp and handle all of the external events
FEDERATE
RTI
rollbackNotification
Send rollbackNotification to federate, and rollback to the saved state
Cancel the pair of the external event and the anti-message in EE.
Callback external events
Callback all ee_i in EE to the federate
Send anti-messages to other federate if required.
Retract
STM

Figure 8. The time diagram of the STM rollback mechanism.

When f_{in} detects a rollback situation, the STM interacts with both the federate and the RTI to perform the rollback. The STM sends *rollbackNotification[S@T_{ss}]* to notify the federate that it should rollback to the saved state *S* at time T_{ss}. The STM then draws out external events recorded in EE and callback to the federate. In this way, the STM can rollback to the saved state and recover externally ordered events to the federate. If the previous processing has resulted in sending some events to other federates, the STM will check its $OE^{ã}$ and send out anti-messages (*Retract*) to the RTI to cancel outgoing events. Therefore, the federate doesn't need to worry about the event cancellation, state saving, and fossil collection. Figure 8 shows the time diagram of the STM rollback mechanism when f_{in} receives *requestRetraction* anti-message. The rollback procedure of the STM is the same as described above, except the STM must cancel the corresponded external event and anti-message saved in EE.

6. The Auto-Mode STM

In the manual-mode, the STM is triggered by the federate to request receiving external events when the federate invokes the *eventRequest* call. On the contrary, the auto-mode allows the STM to automatically request receiving external events. Figure 9 illustrates the infrastructure of the STM in the auto-mode operation, that is, M is set to auto-mode in TM^S. In the auto-mode, the STM agent will constantly monitor the timestamp of unprocessed internal events $TS(ie_j)$ in Local Event List. The f_{out} automatically invokes time advance service call to the RTI when the STM detects the smallest timestamp $TS(ie_j) >= Now$. Therefore, the federate can accomplish the synchronization scheme without knowing how and when it should call the time advance function to the STM. In the auto-mode, the federate only needs to wait for the external events to be callback and then processes them accordingly.

The operation of the auto-mode model of TM^S is similar to its manual-mode, except that it will automatically check the earliest timestamp $TS(ie_j)$ in its Local Event List on whether $TS(ie_j)$ is greater than *Now* or not. If $TS(ie_j) >= Now$, then f_{out} will make an appropriate time advance call to the RTI according to the TA mechanism and receive external events correspondingly.

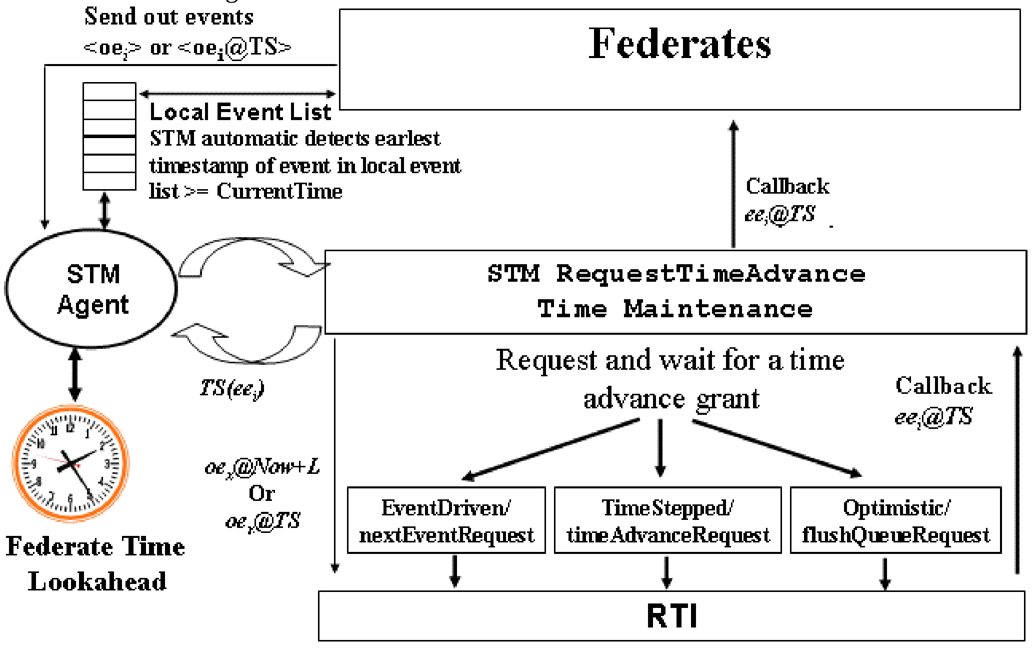
Send out events
<oe_i> or <oe_i@TS>
Federates
Local Event List
STM automatic detects earliest timestamp of event in local event list >= CurrentTime
Callback
ee_i@TS
STM Agent
STM RequestTimeAdvance
Time Maintenance
TS(ee_i)
Request and wait for a time advance grant
Callback
ee_i@TS
oe_x@Now+L
Or
oe_x@TS
EventDriven/ nextEventRequest
TimeStepped/ timeAdvanceRequest
Optimistic/ flushQueueRequest
Federate Time Lookahead
RTI

Figure 9. The infrastructure of the STM in the auto-mode

- **Time-Stepped Simulation**

The auto-mode STM for the time-stepped simulation will automatically trigger the *timeAdvanceRequest* time-stepped function to receive external events. Its TM^S model is basically the same as the manual-mode time-stepped mechanism described in section 5.1, except that f_{out} will automatically make a *timeAdvanceRequest* call to the RTI. In this case, the STM basic model becomes Eq.(5) as follows:

$TM^S = < f , f , f , f$, time-stepped, f_{out}, f_{in}, auto-mode, Now, L>…..(5)

Where different from Eq.(2):

M = auto-mode which is initialized by the federate.

The STM will monitor the earliest unprocessed internal events in the Local Event List when a federate sets M = auto-mode. When the STM detects that the earliest unprocessed internal event has its timestamp $TS(ie_j) >= Now$, f_{out} then checks if TA is time-stepped to make a *timeAdvanceRequest* call to receive external events ee_i. The f_{in} then extracts their timestamp $TS(ee_i)$ before delivers them to the federate through callback functions.

- **Event-Driven Simulation**

The process of the TM^S translation function for the event-driven simulation in the auto-mode is similar to the manual-mode event-driven mechanism described in section 5.2, except that f_{out} will automatically make a *nextEventRequest* call to the RTI for receiving external events. In this case, the STM model becomes Eq.(6) as follows:

$TM^S = < f , f , f , f$, event-driven, f_{out}, f_{in}, auto-mode, Now, L>…..(6)

Where different from Eq.(3):

M = auto-mode which is initialized by the federate.

The STM will watch the earliest unprocessed internal events in the Local Event List when a federate sets M = auto-mode. When the STM detects that earliest unprocessed internal event $TS(ie_j) >= Now$, f_{out} then checks if TA is event-driven to invoke a *nextEventRequest* call to request for external events ee_i. The f_{in} extracts the timestamp $TS(ee_i)$ of the external events before delivers them to the federate with callback functions.

- **Optimistic Simulation (Time-Warp)**

The process of the TM^S function in this type of simulation is similar to the manual-mode optimistic approach described in section 5.3, except that f_{out} will automatically make a *flushQueueRequest* call to the RTI for receiving external events. In this case, the STM model becomes Eq.(7) as follows:

TM^S = <SS, EE, d, OE^{a}, optimistic, f_{out}, f_{in}, auto-mode, Now, L>…..(7)

Where different from Eq.(4):

M = auto-mode which is initialized by the federate.

The STM will watch the earliest unprocessed internal events in the Local Event List when a federate sets M = auto-mode. When the STM detects that the earliest unprocessed internal event has its timestamp $TS(ie_j) >= Now$, f_{out} then checks if TA = optimistic to make a *flushQueuRequest* call to the RTI. The STM will receive all of the “safe” external events, that is, the set $\{ee_i \mid TS(ee_i) <= LBTS, \forall i\}$, first. The f_{in} will extract the timestamp $TS(ee_i)$ of the received external events ee_i before delivering them to the federate with callback functions. The STM then saves checkpoint state SS, where $T_{ss} = \max\{TS(ee_i) \mid TS(ee_i) <= LBTS, \forall i\}$, before notifing the federate about this checkpoint time. Finally, the f_{in} begins to receive the rest of the “unsafe” external events. The f_{in} records these external events ee_i in EE which have their timestamp $T_{ss} < TS(ee_i) <= Now$. Notice that, when it received these “unsafe” external events from the RTI, the STM will advance its current time to $Now = TS(ee_i)$.

The STM will automatically perform the rollback-detection operation. If any out of order external event received or the anti-message callback detected, the STM starts the rollback process. The rollback procedure is the same as that is being described in section 5.3.

7. Conclusion

The STM integrates the conservative and optimistic mechanisms provided by the HLA Time Management. Therefore, that the federate developers can easily employ the time management mechanism regardless of the time policies, messages ordering definitions and the logical time advance strategies of a federate. Furthermore, the STM proposes a smart optimistic synchronization method by creating a middle layer for the time management interface provided by the RTI. The proposed middle layer provides smart rollback, state-saving and fossil-collection management used for the optimistic federates.

Under the STM, distinct time advancement federates of the distribution simulation (federation) can exchange data by using a unified time advance function, *eventRequest*, to communicate with the RTI. The STM proposes a unified and scalable mechanism to allow the user to construct the HLA federates with a consistent time management interface when he is solving the synchronization issue.

The unified and smart synchronizing technique in the STM is very useful. It helps the developer to simplify the process of building conservative and optimistic federates. All the aspect related to the time management issues of the RTI and extended optimistic time warp functions are taken care of by the STM. It relieves the simulation developer from complexity of the synchronizing approach. The STM is capable of identifying what time advancement approach is used by a federate, as well as to taking all suitable actions to ensure that the federate processes internal/external events in the correct causality. The STM will accomplish tasks that are common to every synchronized federate, and does not depend on a specific federate behavior. Hence, the federate program code becomes simpler, because most of the control and management of synchronizing approach are under the STM’s responsibility.

References

[1] Alois. Ferscha: “Parallel and Distributed Simulation of Discrete Event Systems.” Parallel and Distributed Computing Handbook, McGraw-Hill, 1995.

[2] R. M. Fujimoto, “Parallel Discrete Event Simulation”, ACM communication, No. 10, pp. 30-53, 1990.

[3] K. Chandy , J. Misra, “Distributed Simulation: a case study in design and verification of distributed programs”. IEEE Transactions on Software Engineering. Vol. SE-5, No. 5, 440-452, 1979.

[4] R. Bryant, “A Switch-Level Model and Simulator for MOS Digital Systems”. IEEE Transactions on Computers, Feb. 1984, C-33 pp.160-177.

[5] D. Jefferson, “Virtual Time”. ACM Transaction on Programming Languages and Systems, 7(3): 404-425, July 1985.

[6] H. Sowizral, D. Jefferson, “Fast concurrent simulation using the time wrap mechanism”. In Distributed Simulation, SCS, Simulation Councils, pp. 63-69, La Jolla, California, 1985.

[7] Defense Modeling and Simulation Office (DMSO), U.S. Department of Defense, “High Level Architecture Overview”, 1997.

[8] R. M. Fujimoto, “Time Management in the High Level Architecture”, In: SIMULATION, Vol. 71, No. 6, pp. 388-400, 1998.

[9] R. M. Fujimoto, “HLA Time Management: Design Document 1.0”, Defense Modeling and Simulation Office, August 1996

[10] R. M. Fujimoto, “Parallel and Distributed Simulation Systems”, John Wiley & Sons, inc. A Wiley Interscience Publication, 1999.

[11] Frank Hodum, David Edwards, “Time Management Services in the RTI-NG”, 2001 Fall Simulation Interoperability Workshop.

[12] Defense Modeling and Simulation Office, Department of Defense, “RTI 1.3 – Next Generation Programer’s Guide Version 5”, High Level Architecture, Runtime Infrastructure, Feb. 2002.

[13] Paul A. Fishwick, “Simulation Model Design and Execution”, Prentice Hall, New Jersey, 1995.

[14] Jeff S. Steinman, “SPEEDES: A multiple-synchronization environment for parallel discrete event simulation”, The International Journal for Computer Simulation, Vol. 2, No. 3, pp 251-286, 1992.

[15] Jeff S. Steinman, “Breathing Time Warp“, Proceeding 7th Workshop an Parallel and Distributed Simulation, Vol. 23, No. 1, pp 109-118, 1993.

[16] Bernard P. Zeilger, Herbert Praehofer, Tag Gon Kim, “Theory of Modeling and Simulation”, 2nd edition, Academic Press, New York, 2000.

[17] Page H. Ernest, “Zero Lookahead in a Distributed Time-Stepped Simulation”, Simulation Digest, 26(2), pp. 4-13, September, 1997.

[18] R. M. Fujimoto, “Zero Lookahead and Repeatability in the High Level Architecture”, In Proceedings of the 1997 Spring Simulation Interoperability Workshop, Orlando, FL 307, March.

[19] F. Vardanega, C. Maziero, ”Using computational reflection in optimistic distributed simulations”, Computer Science Society, 1999. Proceedings. SCCC '99. XIX International Conference of the Chilean , 1999, Page(s): 1 -8

[20] F. Vardânega, C. Maziero., “Simplifying optimistic distributed simulations in the High Level Architecture”, XI IASTED International Conference on Parallel and Distributed Computing and Systems. Cambridge (MIT) - USA, Nov. 1999.

[21] F. Vardânega, C. Maziero., “A generic rollback manager for optimistic HLA simulations”, Distributed Simulation and Real-Time Applications, 2000. (DS-RT 2000). Proceedings. Fourth IEEE International Workshop on , 2000, Page(s): 79 -85