SYNCHRONIZED DATA DISTRIBUTION MANAGEMENT IN DISTRIBUTED SIMULATIONS

Ivan Tacic
Richard M. Fujimoto
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

KEYWORDS
High Level Architecture, Run-Time Infrastructure, time management

ABSTRACT

A considerable amount of effort in the DIS community has been devoted to developing efficient, scaleable, mechanisms for distributing state updates and interaction information in distributed simulations. By contrast, this question has not received as much attention for distributed simulations using logical time (e.g., analytic simulations). It is observed that data distribution management (DDM) mechanisms used for real-time training simulations such as DIS are insufficient to meet the requirements of logical time-based simulations, and may result in errors such as messages not being delivered to federates that have subscribed for them, even if the network provides reliable delivery. An approach to achieving properly synchronized data distribution is described, and is applied to the data distribution management mechanisms based on routing spaces that has been proposed for the HLA.

1. INTRODUCTION

Data distribution management (DDM) mechanisms are necessary to provide efficient, scalable support for large-scale distributed simulations. Early work in distributed simulation environments in the SIMNET (SIMulator NETworking) project[1] and early DIS systems broadcast each state update to all simulators in the exercise. It is well known that this approach does not scale because the amount of communications increases by $O(N^2)$ where N is the number of processors. CPUs become bogged down processing incoming messages, most of which are discarded (for large N) because they are not relevant to the simulator(s) within the processor. Further, communication bandwidth requirements become excessively large as N increases. It is estimated that 375 MBits per second per platform would be required for a simulation exercise including 100,000 players[2].

Several approaches to attacking this problem have been proposed (see [3] for a survey on this subject). Virtually all use some mechanism to only send messages to the destinations that have need of the information rather than broadcast it to everyone. For example, in the Joint Precision Strike Demonstration (JPSD) [4], federates indicate what information they wish to receive by specifying predicates on entity attributes. Many simulators, e.g., ModSAF[5], CCTT[6], and NPSNET[2] use grid cells to filter information. A two-level hierarchical filtering scheme used in an optimistic parallel simulation is described in [7], and a generalization of grid cells using a construct called routing spaces is used in STOW[8]. Routing spaces are also used in the HLA, and are discussed in greater detail later. The focus on the work described here is on the routing space approach that has been incorporated into the baseline definition of the High Level Architecture (HLA), though many of the techniques and mechanisms are applicable in other contexts.

Thus far, most of the DDM work has focused on large-scale, real-time training simulations. By comparison, only a limited amount of work on this subject has been concerned with distributed logical time simulations[7, 9]. The RTI and the underlying communication infrastructure must maintain some representation of a database indicating which federates should receive what messages. For example, this information might be represented by the memberships of the multicast groups used in the simulation. In training simulations, changes to this database

(e.g., a vehicle might come within the range of a sensor, requiring that future state updates by the vehicle simulator be transmitted to the sensor simulator) take effect at the instant in wallclock time when this condition is detected. By contrast, in logical time simulations, such changes take effect at a specific point in logical time. At any instant during the execution of a distributed simulation, different simulators (federates) will be at different points in logical time. Directly applying DDM mechanisms based on wallclock time semantics to logical time simulations will lead to errors, e.g., some simulators will not receive messages that they should, and unrepeatable executions. This paper is concerned with DDM mechanism for logical time simulations, and specifically, DDM in the High Level Architecture.

The rest of the paper is organized as follows. Section 2 describes HLA issues relevant to data distribution. The synchronization problem in data distribution is described in section 3. A two level architecture for DDM mechanisms consisting of an interest management and a distribution list management layer is described in section 4. Sections 5 and 6 describe the functionality of these two layers and implementation issues. Finally, conclusions and future work are discussed in section 7.

2. DDM IN THE HLA

The HLA provides a common architecture for modeling and simulation within the Department of Defense (DoD). This architecture spans both logical time-based simulations (e.g., analytic simulations) and real time simulations (e.g., for training) such as DIS. HLA's Run Time Infrastructure (RTI) is a software component that provides commonly required services to simulation systems in a way that is analogous to the services a distributed operating system provides to applications. HLA services are grouped into six categories[10]:

- Federation Management (FM)
- Declaration Management (DM)
- Object Management (OM)
- Ownership Management (OWM)
- Time Management (TM)
- Data Distribution Management (DDM)

DM and DDM in the HLA are used to specify which federates should receive messages for each attribute update and interaction. The declaration management (DM) services *Publish* and *Subscribe* allow a federate to update and receive updates to object attributes based solely on object class. The RTI uses information provided in publish/subscribe calls to set up filters that direct data among federates that need them. The object management's (OM) *Update Attribute Values* service call notifies the RTI that one or more attributes have been modified. For example, an object might subscribe to the attribute 'location' of all tanks in the battlefield. However, such filtering will only be appropriate for relatively small federations. DDM services provide more powerful data distribution services enabling value-based filtering[11]. For example a tank might want to receive data from other tanks only if they are in its visible range. Note that since the RTI does not know the meaning of object attributes it cannot provide range-based filtering in this example. Federates must agree on a filtering strategy to do this.

In order to simplify the discussion that follows, only attribute updates are considered. However, the concepts described below also apply to interactions.

The fundamental concept used in the HLA to support value-based DDM is the *routing space.* A routing space is a normalized (coordinate values range from 0.0 to 1.0) multidimensional coordinate system in which federates indicate interest in receiving or providing updates via *subscription* and *update regions*. Subscription and update regions are rectangular (in N dimensions) and are specified by indicating *extents,* with one extent for each dimension. Each extent indicates the portion of that dimension covered by the region. For example, the extents [0.0,0.5][0.0,1.0] specify the left half of a two dimensional routing space. Federates express their interest in receiving updates from other federates through subscription regions defined over a routing space, and are called subscribers (to a specific attribute). On the other hand, federates that are providing updates define their update regions over the routing space, and are called publishers (of a specific attribute). The RTI detects when subscription and update regions associated with an attribute overlap, and provides data transfer from the publisher to all subscribers for all updates to that attribute.

Figure 1 shows a two-dimensional routing space with one update U1 and two subscription regions S2 and S3 that belong to federates F1, F2 and F3 respectively. Since U1 and S2 overlap, the RTI will transmit updates to attributes by F1 that are associated with this update region to subscriber F2, but not to F3. Regions can be changed dynamically by invoking the *Modify Region* service.

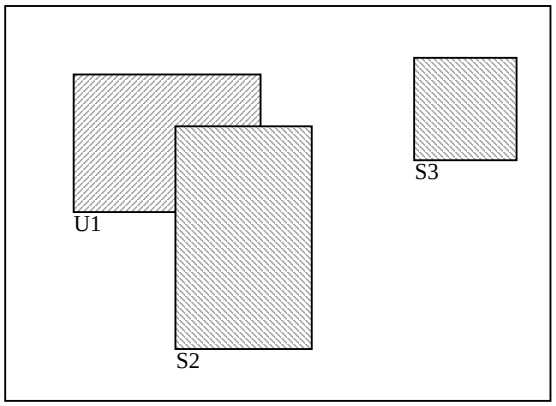

Figure 1. Update region U1 and subscription regions S2 and S3 in a two dimensional routing space.

3. PROBLEM DESCRIPTION

DDM in logical time-based federations must satisfy two principal requirements:

- *federates should receive all messages for information to which they had specified interest, and ideally, no others, and*
- *the RTI must deliver messages in time stamp order and must not deliver messages to a federate with time stamp less than the current logical time of the federate.*

The first requirement can be relaxed somewhat in practice because federates could simply ignore extra messages.

At present, DM and DDM services (e.g., changes in subscription and update regions) take effect throughout the federation at an instant in wallclock time, ignoring delays to actually realize the change. In real-time simulations not using logical time (e.g., DIS exercises), federate time is essentially the same as wallclock time.[1] Thus, all federates are at approximately the same federate time at any instant during the execution. Figure 2 shows a scenario where a blue tank approaches a red tank. The red tank becomes visible to the blue tank at wallclock time 10. This might be realized in the simulation by the blue tank modifying its subscription region at time 10 to overlap the red tank's update region. The RTI will immediately add the blue tank to the list of federates to receive position updates by the red tank. In this scenario, moments after the subscription region has been changed, the simulator for the red tank issues a position update that is then (correctly) received by the simulator for the blue tank.

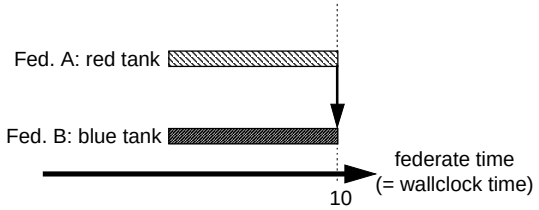

Figure 2. Two tanks approaching each other in a DIS exercise.

Errors result if this same mechanism is used for logical time federations. This is because at any instant during the federation execution, different federates will typically be at different logical times. DDM changes should take effect at a specific instant in logical time. This may result in situations where federates do not receive messages they should have received (or they may receive messages they shouldn't have received). Figure 3 depicts the previous scenario with the blue tank approaching the red tank, but the red tank reaches logical time 10 before the blue tank. The red tank should become visible to the blue tank, however, the blue tank has not yet reached logical time 10, so it has not yet modified its subscription region to receive the state update made by the red tank at time 10.[2] Thus, the simulator for the blue tank fails to receive this message. The blue tank will later modify its subscription region, however, this is too late because the red tank's message has already flowed through the system.

[1] Actually, federate time is derived from wallclock time using an offset and scale factor, however, this is of little consequence in the present discussion.

[2] Actually, the time stamp 10 message is generated by the red tank *before* it reaches logical time 10 because of lookahead constraints.

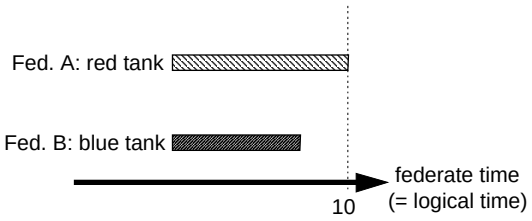

Figure 3. Two tanks approaching each other in a logical time execution, resulting in a lost message.

One approach to solving this problem is to have the RTI maintain a log of messages, and send them to new subscribers as needed (see Figure 4). Here, the position update made by the red tank with time stamp 10 is captured in the log, and re-sent to the blue tank when it subscribes for this information at logical time 10.

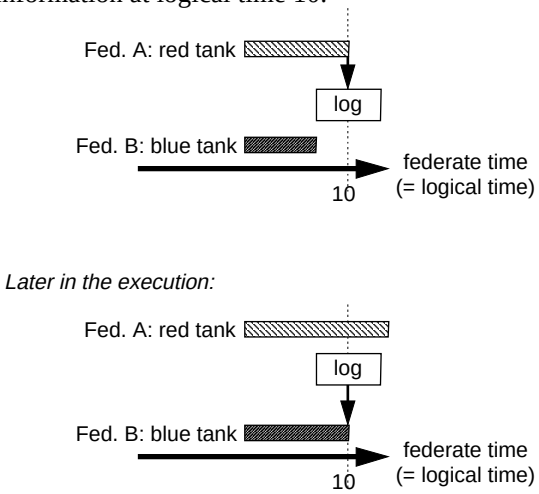

Figure 4. Use of a log to avoid losing messages in logical time federations.

The second problem, receiving a message in a federate's past, is depicted by the scenario shown in Figure 5. This scenario differs from the previous one in that we change the publication region of the red tank at logical time 10 to cause the blue tank to begin receiving its position updates. Here, the blue tank reaches logical time 10 before the red tank, and subscribes to a new region of the battle space. Because the red tank is not publishing any messages that the blue tank could receive, nothing prevents the blue tank from progressing forward in logical time. Later, the red tank advances to logical time 10, updates its publication region so the blue tank will receive its messages, and performs a position update that is sent to the blue tank. This update arrives in the past of the blue tank.

In order to avoid receiving messages in a federate's past, one may conservatively assume any federate using logical time may send a message to any other logical time federate. This would prevent the blue federate from advancing too far ahead of the red federate, regardless of the update and subscriptions made by the federates. This solution is currently implemented in the RTI prototypes that have been developed. This problem is not discussed further in this paper.

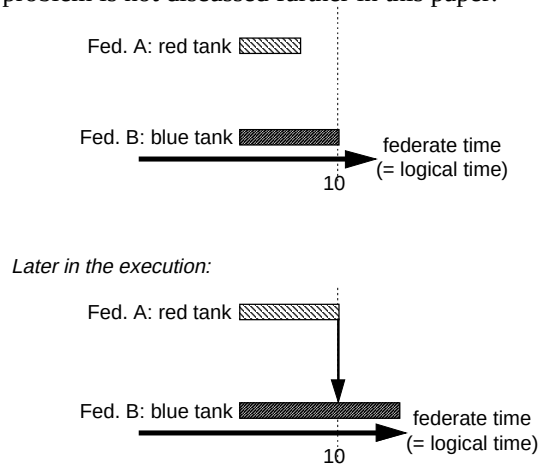

Figure 5. Scenario where a message is received by a federate in its past.

4. A LAYERED ARCHITECTURE

It is convenient to view the DDM system as logically being composed of two layers of software (see Figure 6). The upper layer provides the interface to the federate for specifying its "interests" via interest expressions (portions of the routing space). This interest management software receives and processes interest expressions produced by federates and generates for the lower layer *Add* and *Delete* operations to change the database indicating which federates receive attribute updates and interactions. At this lower layer, distribution list software performs changes to the database and ensures that these changes are properly synchronized with attribute updates and interactions so that each federate receives all of the messages it is supposed to receive, and no others.

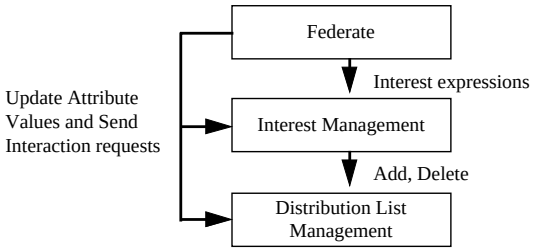

Figure 6. DDM Organization.

The current HLA Interface Specification[10] does not include a time stamp parameter for services that modify subscription and update regions. It is clear that such a specification is necessary for logical time federations to indicate when the changes should take effect, so here, a time stamp parameter has been added for this purpose.

A federate publisher may issue the following operations to the interest management layer:

- **Modify_Region (region_handle H, new_region R, time_stamp T):** informs the RTI that the update region H has been changed to a new region (set of extents) R as of logical time T. The region handle is passed to the federate when the region is created.
- **Update_Attribute_Values(attribute A, new_value V, time_stamp T):** informs the RTI a new value V has been assigned to attribute A at logical time T.[3]

A federate subscriber may issue:

- **Modify_Region(region_handle H, new_region R, time_stamp T):** informs the RTI that the subscription region H has been changed to a new region (set of extents) R as of logical time T.

Federates may also issue DM operations (Publish and Subscribe). These are also processed in the interest management layer, but are not discussed further.

Logically, the RTI can be viewed as maintaining a collection of *distribution lists* indicating which federates should receive attribute updates. The distribution lists collectively form the database mentioned earlier. The distribution lists may not be explicitly represented by the RTI, i.e., they could be realized by the composition of multicast groups in the underlying network. Each change to a distribution list has a logical time associated with it indicating when the change takes effect. Thus, one may conceptually view each distribution list as evolving, one change at a time, over successive logical times.

D(id,T) denotes a distribution list (the id field specifies a particular list) corresponding to federate time T. When an update to an attribute occurs with time stamp T, the interest management layer will map this attribute to one or more distribution lists, and the RTI will send a message to each federate in D(id,T) of all selected lists. The following operations may be performed on a distribution list (see Figure 6):

- **Add(F,id,T):** add federate F to the distribution list identified by id to take effect at logical time T.
- **Delete(F,id,T):** remove federate F from the distribution list identified by id as of logical time T.
- **Update(id, V, T):** send messages to all federates that belong to the distribution list id at logical time T. V indicates the value of the message. This primitive would be invoked when the federate invokes the **Update Attributed Values** (or **Send Interaction**) service.

These operations are invoked within the RTI by the interest manager. They may be invoked when a change in the filter specification occurs that changes the set of federates that should receive messages, when an object becomes "discovered" (or removed), when a new object is instantiated (or removed), or when an attribute is updated or an interaction sent. The semantics of these operations are defined as follows:

- Composing the operations Add(F,id,T) and Delete(F,id,T) with the same parameter values has the same effect as if neither operation were performed. In this case, the two operations are said to be *canceled.*
- Add (F,id,T): let T_D be the smallest time stamped Delete operation (ignoring canceled operations) by federate F on distribution list id such that $T_D > T$. F will receive a message for every update to this distribution list with time stamp in the interval $[T, T_D)$.
- Delete (F, id, T): let T_A be the smallest time stamped Add operation (ignoring canceled

[3] This is a simplification of he HLA interface where multiple attributes may be updated in a single invocation of this service.

operations) by federate F on distribution list id such that $T_A > T$. F will *not* receive any messages for updates to this distribution list with time stamp in the interval $[T, T_A)$.

5. FUNCTIONALITY OF THE INTEREST MANAGEMENT LAYER

Here, we describe one approach to implementing the interest management layer based on partitioning the routing space into non-overlapping cells. In general, the cells need not be the same size, but we will assume identical cells here to simplify the presentation. Using this approach, a distribution list is created for each cell of each routing space. As stated earlier, the IM layer must convert calls by the federate (to modify regions and update attribute values) into calls to the distribution list layer to add federates to (or delete federates from) the distribution lists for individual cells. Update and subscription regions may cover many cells.

Each attribute has a publication region associated with it, indicating the set of cells (distribution lists) that receive update operations when the attribute is updated. Implementing the DDM functionality for publishers is straight-forward. A publisher will issue **Update_Attribute_Values** operations which in turn causes invoking Update in the distribution list layer for each cell that includes a portion of the update region for the modified attribute. This may result in some messages being sent to federates whose subscription region lies outside the update region (but includes one or more cells in common with the update region), however, such messages can be filtered at the destination. The **Modify_Region** operation causes no action to be performed on the distribution list layer, and the only actions necessary in the IM layer are to record the new update region.

When a subscriber issues the **Modify_Region** operation at logical time T, it may cause a change in the distribution lists. This implies a new version of the distribution list is created at logical time T. In general, there may already be another version of the distribution list that has already been created at a logical time greater than T, because **Modify_Region** operations need not be issued in time stamp order across the entire federation.

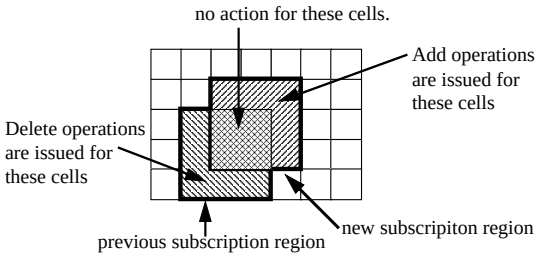

Figure 7. Add and delete operations issued when a subscription region moves if this is the latest (in logical time) change.

If there are no versions of the distribution list at logical times greater than T, the IM simply invokes Add and Delete operations on cells (distribution lists) to define the new subscription region. For example, in the two dimensional routing space shown in Figure 7, the federate is added to five distribution lists, and deleted from five others.

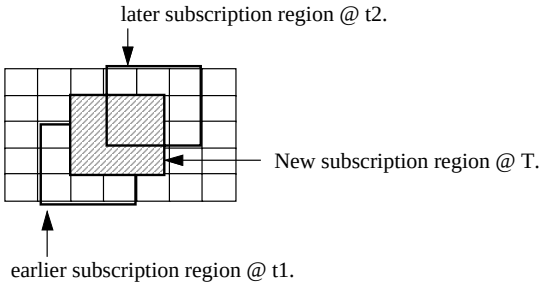

Figure 8. A new subscription region is added, but is not the latest.

More generally, if a version already exists at a time t2 (greater than T), the set of Add and Delete operations that must be issued is more complex. This situation is depicted in Figure 8. The actions that must be performed depend on whether or not the federate is subscribed to the cell at logical time t1, the time corresponding to the most recent version of the distribution list prior to T, as well as whether or not the federate is subscribed at logical times T and t2.

The actions taken for each of the eight possible cases are indicated in Table 1. A '1' in this table indicates the federate is subscribed at the specified time, and a '0' indicates it is not subscribed. For example, the third row in the table indicates that if the federate is not subscribed to a cell at t1 and t2, but is in the new subscription region at T, the IM layer will issue one Add operation @ T and one Delete operation @ t2. This ensures the federate will receive

updates at logical times in the interval from T to t2.

If a federate's subscription region and the update region for an attribute have more than one cell in common, the federate will receive multiple copies of update messages. The IM must therefore eliminate duplicates at the receiving federate.

6. FUNCTIONALITY OF THE DISTRIBUTION LIST LAYER

We now discuss the implementation of the Add, Delete, and Update operations in the DLM layer. If Add, Delete, and Update operations for each attribute were issued sequentially to the RTI in time stamp order, data distribution would be trivial. In that case, Add and Delete operations would simply update the distribution list, and Update operations would transmit messages to the destinations in the list. Because the operations do not arrive in time stamp order, the RTI must maintain different versions of the distribution list corresponding to different points in logical time. An Update operation at logical time T reads the version of the distribution list corresponding to time T for each of the cells in its update region, and Add or Delete operations at logical time T create a new version of the list effective at time T.

Table 1. Actions for a new subscription region at time T

region@t1	region@T	region@t2	IM actions
0	0	0	no action
0	0	1	no action
0	1	0	Add@T, Delete@t2
0	1	1	Add@T
1	0	0	Delete@T
1	0	1	Delete@T Add@t2
1	1	0	no action
1	1	1	no action

The distribution list for cell id corresponding to time T is constructed by determining which federates have subscribed via the Add operation to receive updates with time stamp T. Specifically, define the "subscription function" as follows:

S(F, id, T)

= TRUE if an uncanceled operation Add (F, id, T_A) exists such that $T_A \leq T$ and no uncanceled operation Delete (F, id, T_D) exists such that $T_A < T_D < T$.

= FALSE otherwise

D(id,T) is defined as the set of federates F_i such that S(F_i, id, T) is true.

As discussed earlier, update attribute messages are logged by the RTI so they can be sent to "late" subscribers. A log L(id) for distribution list id is defined for this purpose. This log is defined as a sequence of tuples $<V_i, T_i>$ where V_i is the new value contained in the update message sent for an attribute update with time stamp T_i.

Consider the sequence of Add and Delete operations performed by a *single* federate on a *single* cell. In general, these operations need not arrive at the processor managing the distribution list for that attribute in time stamp order. This is because the federate may not issue these operations in time stamp order, or even if it did, the relative order of the operations may not be preserved as the associated messages are transmitted through the network. Thus, at any instant, there need not necessarily be a perfect pairing of Add and Delete operations. For example, Figure 9 shows a situation where a Delete operation with time stamp 20 has been delayed in the network, giving the temporary appearance that there is an extra Add operation. In this snapshot (i.e., prior to receiving the time stamp 20 Delete operation), the extra Add operation at time 25 is, in effect, a null operation because the federate already subscribed to the attribute at time 15. Because Add and Delete operations with the same parameters cancel, the federate is subscribed to receive updates containing a time stamp in the intervals [5,10) and [15,35).

It is convenient to view the history of Add/Delete operations to an attribute by a federate according to a diagram such as that shown in Figure 9. Consider a new Add or Delete operation with time stamp T. The status of the federate (subscribed or unsubscribed) at time T only depends on the largest time stamped uncanceled Add or Delete operation with time stamp smaller than T. For example, in Figure 9 when the Delete operation with time stamp 20 arrives, the Add

operation at time 15 indicates the federate is subscribed to the attribute at time 20, independent of what other Add or Delete operations occurred with time stamps less than 15. Similarly, the effect of the new operation at time 20 only persists until the next higher time stamped uncanceled Add or Delete operation. In Figure 9, the effect of the new operation at time stamp 20 only persists until time 25. No operation with time stamp larger than 25 is affected by this new operation.

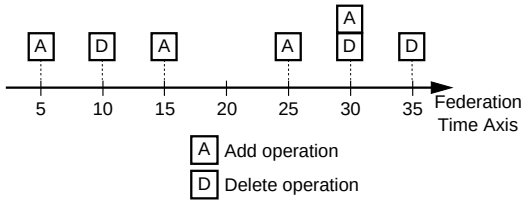

Figure 9. Snapshot of Add and Delete operations by a single federate. The federate is subscribed to receive updates in the intervals [5,10) and [15,35).

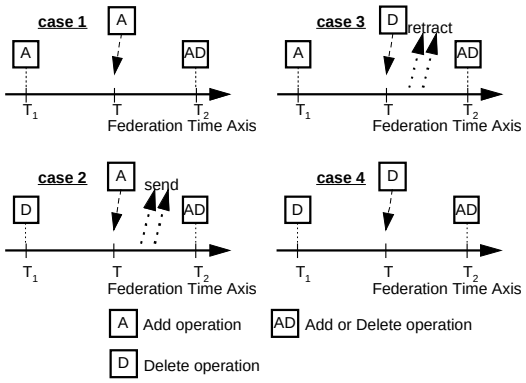

Figure 10. Possible cases when a new Add or Delete operation arrives.

Based on the above observations, one can see that when a new Add or Delete operation arrives with time stamp T, the "effect" of the operation only spans the interval from the time stamp of the immediately preceding (in time stamp order) Add/Delete operation to the immediately following operation. In the discussion that follows, only uncanceled Add and Delete operations are considered. One can enumerate all possible situations. There are eight possible combinations. The first four cases correspond to a newly arriving Add operation. It can be preceded (followed) by an Add or a Delete operation (see cases 1 and 2 in Figure 10). Actually, of these four cases, only two are different because the processing of the new operation does not depend on whether an Add or Delete operation follows. Case 1 in Figure 10 corresponds to a new Add operation preceded by another Add operation, and case 2 corresponds to the new Add preceded by a Delete operation. Similarly, there are effectively two cases to consider when a new Delete operation arrives: it may be preceded by an Add operation (case 3 in Figure 10) or it may be preceded by a Delete operation (case 4 in Figure 10). To account for the smallest and largest time stamped Add/Delete operations (which do not have a preceding or following operation, respectively), we implicitly assume there are Delete operations at times $-\infty$ and $+\infty$ to represent the fact that the federate is not included in any distribution list initially, nor after the end of the execution.

Correct realization of the Add and Delete operations can be derived from the case analysis shown in Figure 10. First consider Add operations. Case 1 corresponds to a new Add operation where the federate is already subscribed to the attribute. This situation of two consecutive (in federate time) Add operations could arise if the Delete operation for the earlier Add operation had been delayed. The new Add operation is not unlike that at time stamp 25 in Figure 10, i.e., this operation has no effect because the federate is already subscribed to the attribute. Therefore, no further action is required other than noting that the Add operation has occurred. Case 2 corresponds to the situation where the federate is not subscribed to the attribute. The federate should, but has not yet received any attribute updates with time stamp in the interval $[T, T_2)$, where T_2 is the time stamp of the following Add/Delete operation, so messages for these updates must be sent to the federate. Updates with time stamp larger than T_2 have already been correctly processed by the operation at time T_2.

Now consider Delete operations. Case 3 corresponds to a Delete operation when the federate is subscribed to the attribute. In this case, the federate has been sent messages with time stamp in the interval $[T, T_2)$, where T_2 is the time stamp of the following Add/Delete operation, so these messages must be retracted (canceled). The retractions are unnecessary if it

is permissible to receive additional messages beyond what the federate had subscribed to receive. Case 4 corresponds to a Delete operation occurring when the federate is not subscribed to receive updates. Like case 1 for the Add operation, no additional messages (or retractions) need to be sent.

Update operations do not directly affect the status (subscribed or unsubscribed) of a federate, so do not impact the Add or Delete operations.

Based on this case analysis, the Add, Delete, and Update operations can be realized as follows:

I. Update (id,V,T):
   A. send V to all federates in D(id,T)
   B. record <V,T> in L(id)

II. Add(F,id,T):
   A. record F has been added to D(id) at time T
   B. if not S(F,id,T) then /* case 2 */
      1. let AD be the smallest time stamped uncanceled Add or Delete operation for F on id with time stamp greater than T, and let T_2 be the time stamp of AD.
      2. For each tupple <V,T_v> in L(id) where T $\leq T_v < T_2$, send V to F

III. Delete (F,id,T):
   A. record F has been deleted from D(id) at time T
   B. if S(F,id,T) then /* case 3 */
      1. let AD be the smallest time stamped Add or Delete operation for F on A with time stamp greater than T, and let T_2 be the time stamp of AD.
      2. For each tupple <V,T_v> in L(A) where T $\leq T_v < T_2$, send a retraction of V to F

6.1. Memory Reclamation

In addition to the above operations, a mechanism is required to reclaim memory used by the distribution lists and logs. The current logical time of the owner of the attribute plus that federate's lookahead provides a lower bound on the time stamp of any future **Update Attribute Values** service request. Assuming Add and Delete operations also adhere to lookahead constraints, the current time of any federate that can Add/Delete the attribute (i.e., any federate in the federation) plus its lookahead gives a lower bound on the logical time of future add and delete requests that will be generated by that federate. Thus, the minimum of these values across all federates (analogous to Global Virtual Time in Time Warp simulations) gives a lower bound on the time stamp of any Add, Delete, or Update operations that will be made on the distribution list in the future. If this minimum time stamp value is T, then all tuples with time stamp less than T can be discarded, provided the RTI retains a copy of the distribution list of the attribute A at time T, i.e., some representation of D(id,T) is required. Value tuples with time stamp less than T may be discarded and their storage reclaimed.

6.2. Simplifications

The mechanism described above requires a message log. This log can be eliminated if certain restrictions are made on federate behavior. Recall that the log was required because Update messages containing a time stamp T may be generated before a subsequent Add or Delete operation containing a time stamp smaller than T. If one makes the restriction that at each instant in the federation execution, a global time stamp value T_{bound} exists such that no update messages with time stamp larger than T_{bound} can be sent, and no Add or Delete operation with time stamp less than T_{bound} can be generated, this situation cannot occur. Realization of this mechanism requires a protocol to advance T_{bound} because all federates must agree to an advance of T_{bound} before it can take effect.

6.3.Multicast Groups

Multiple versions of distribution lists are needed corresponding to different logical times. A form of an abstraction called space-time memory can be used to realize the distribution list[12]. Space-time memory is similar to ordinary memory except a time stamp is specified each time the data structure is read or modified. A read operation returns the most recent version of the data structure as of the time stamp of the read, providing a convenient means for accessing the correct version.

If multicast groups are exploited, however, multiple versions of the distribution list cannot be easily used because the lists are represented within the network infrastructure by membership to multicast groups. This problem can be solved by including in the multicast groups the union of all distribution lists that are currently active. Add

operations result in immediate joins to the multicast group. Delete operations are delayed until the memory reclamation mechanism advances to the time of the operation.

7. FUTURE WORK

A mechanism is described to realize properly synchronized data distribution in distributed simulations using logical time. A two-layer architecture is proposed that includes a distribution list manager that ensures data is routed to subscribers based on logical time semantics, and an interest manager layer that maps interest expressions to the distribution list manager. The DLM is applicable to a variety of data distribution schemes. An approach to implementing the IM layer for routing spaces such as those defined in the HLA is described.

An implementation of the synchronized data distribution mechanisms described here has been developed. Performance measurements of this implementation are just beginning. Future work will concentrate on completing a detailed evaluation of this approach.

8. ACKNOWLEDGMENTS

Work on this project was funded under a contract from DMSO. Comments from the HLA time management and data distribution management working groups are acknowledged.

9. REFERENCES

[1] C. Kanarick, "A Technical Overview and History of the SIMNET Project," in *Advances in Parallel and Distributed Simulation*, vol. 23: Society for Computer Simulation, 1991, pp. 104-111.

[2] M. Macrdonia, M. Zyda, D. Pratt, and P. Brutzman, "Exploiting Reality with Multicast Groups: A Network Architecture for Large-Scale Virtual Environments," in *1995 IEEE Virtual Reality Annual Symposium*, 1995, pp. 11-15.

[3] K. Morse, "Interest Management in Large Scale Distributed Simulations," University of California, Irvine Technical Report TR 96-27, 1996.

[4] E. T. Powell, L. Mellon, J. F. Watson, and G. H. Tarbox, "Joint Precision Strike Demonstration (JPSD) Simulation Architecture," in *14th Workshop on Standards for the Interoperability of Distributed Simulations*. Orlando, Florida, 1996, pp. 807-810.

[5] K. L. Russo, L. C. Shuette, J. E. Smith, and M. E. McGuire, "Effectiveness of Various New Bandwidth Reduction Techniques in ModSAF," in *Proceedings of the 13th Workshop on Standards for the Interoperability of Distributed Simulations*, 1995, pp. 587-591.

[6] T. W. Mastaglio and R. Callahan, "A Large-Scale Complex Environment for Team Training," *IEEE Computer*, vol. 28, pp. 49-56, 1995.

[7] J. S. Steinman and F. Wieland, "Parallel Proximity Detection and the Distribution List Algorithm," in *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*. Edinburgh, Scottland, 1994, pp. 3-11.

[8] D. J. Van Hook, J. O. Calvin, M. K. Newton, and D. A. Fusco, "An Approach to DIS Scalability," in *Proceedings of the*

11th Workshop on Standards for the Interoperability of Distributed Simulations, 1994, pp. 347-356.

[9] T. D. Blanchard, T. W. Lake, and S. J. Turner, "Cooperative Acceleration: Robust Conservative Distributed Discrete Event Simulation," in *Proceedings of the 1994 Workshop on Parallel and Distributed Simulation*. Edinburgh, Scotland, 1994, pp. 58-64.

[10] Defense Modeling and Simulation Organization, "HLA Interface Specification, V. 1.0," U.S. Department of Defense, Washington D.C. August 1996.

[11] Defense Modeling and Simulation Organization, "Data Distribution and Management Design Document, V. 0.2," U.S. Department of Defense, Washington D.C. December 1996.

[12] R. M. Fujimoto, "The Virtual Time Machine," in *International Symposium on Parallel Algorithms and Architectures*, 1989, pp. 199-208.