Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

APPLYING TEMPORAL DATABASES TO HLA DATA COLLECTION AND ANALYSIS

Thom McLean
Leo Mark
Margaret Loper
David Rosenbaum

College of Computing and Georgia Tech Research Institute
Georgia Institute of Technology
Atlanta, GA 30332, U.S.A.

ABSTRACT

The High Level Architecture (HLA) for distributed simulations was proposed by the Defense Modeling and Simulation Office of the Department of Defense (DOD) in order to support interoperability among simulations as well as reuse of simulation models. One aspect of reusability is the ability to collect and analyze data from simulation executions, including a record of events that occur during the execution, and the states of the simulation objects. Several approaches have been developed for data collection in distributed simulation environments. The HLA presents an interesting new paradigm within which to design effective data collection and analysis techniques. The capabilities of the Run-Time Infrastructure (RTI) can be exploited to design efficient and flexible data collection tools.

Recent research on the efficient log-based implementations of temporal databases may enable more efficient collection and analysis of data from simulation executions. Using a distributed real-time temporal database approach, we may be able to expand run-time analysis opportunities.

In this paper we present a list of important research questions regarding the utility and viability of a temporal database data collection and analysis approach. The questions address issues of how and where to collect the data, how to resolve temporal issues in a distributed system, what functionality must be supported by the temporal database, and what the relationship to the Run-Time Infrastructure (RTI) should be.

1 BACKGROUND

1.1 Distributed Simulation

For this analysis, we consider a distributed simulation to be a set of logical processes which communicate by passing messages. The processes communicate *updates* to the states of objects they own. The processes also communicate events, or *interactions*, which the receiving processes may use to alter their execution. The fundamental difference between an update and an interaction is that updates are changes to the state of data in objects which persist in the simulation, while interactions occur at an instant, and their data does not persist. The update and interaction messages may be passed with or without a timestamp indicating the simulation time at which the data within the message is valid. The timestamp, if present, may be used to order the messages for delivery.

The processes also may collaborate to maintain synchronization. The synchronization may be an explicit scheme of time management, such as the conservative and optimistic schema described in (Fujimoto 1990), or a set of common barriers or checkpoints which halt the execution at a specified point. In a real-time distributed simulation, the processes are required to keep pace with an external clock. Real-time simulations may choose not to synchronize at all, due to the processing overhead required. For this reason, real-time simulations frequently do not apply a timestamp to messages they send. The lack of a timestamp implies that the message is relevant at simulation time t = NOW, or upon being sent. Practically, this means that the message should be processed as soon as possible. The lack of a timestamp also implies receive-order delivery of the message.

1.2 HLA

The purpose of the development of the HLA is to provide a technical means to achieve interoperability, and to promote reuse of various M&S components. (DMSO 1995) The HLA was designed to provide a common context for describing the interfaces between simulation *federates* and the data they exchange during an execution.

There are three parts to the HLA: the Object Model Template (OMT), the Interface Specification (I/F Spec), and the HLA Rules. (DMSO 1997) The OMT provides a means to describe the data which will be used and passed between federates. The I/F Spec describes the mechanisms for performing the transfer of that data. The concrete implementation of the I/F Spec is the Run-time Infrastructure (RTI). The HLA Rules are a set of basic principles which unify the architecture, and provide high-level guidance to appropriate implementation of a federate or federation.

The RTI provides all the basic services necessary to conduct the simulation execution. The RTI provides synchronization and time management services to govern the execution. The RTI also provides for the integration of federates which do not use the same time management scheme. This permits messages to be delivered in either time-stamp-order or receive-order, as appropriate for the federate.

The HLA does not specifically address data collection, either through the Rules, or the I/F Spec. There are, however, several support services, in the RTI Management Object Model (MOM), which facilitate data collection. These have been exploited by various loggers and analysis applications. (Loper, McLean et al. 1997)

1.3 Temporal Databases

A *temporal database* supports the storage and retrieval of temporal data objects. Each entry in the database has one or more associated timestamps. In a *bi-temporal* database (Jensen, Clifford et al. 1992), two types of *timestamps* are associated with each state of an object, the *transaction-time timestamps* and the *valid-time timestamp.* The transaction timestamps record when the state of the object is current in the database. The valid timestamps record when the state of the object is current . Two *timeslice operators* are supported in a bitemporal database, transaction timeslicing and valid timeslicing.

The transaction-time timeslice, DB_T(tt), where tt <= NOW, returns the state of the database as it was at time tt. The valid time timeslice, DB_V(vt), returns the state of the world modeled by the database at time vt as it is known NOW. In addition to the two timeslice operators, traditional database operators support the retrieval of objects in the timeslices.

In a log-based implementation of a temporal database (Jensen, Mark et al. 1991; Jensen, Mark et al. 1993), changes to a set of objects are stored in logs in transaction-time order. Transaction-time timeslices are computed from previously computed and cached timeslices by incrementally updating the cached timeslice with later updates from the log, or by decrementally downdating the cached timeslice with earlier updates from the log. Log-based temporal databases also support direct retrieval of change information from the logs .

A survey of temporal and real-time database research was presented in (Ozsoyoglu and Snodgrass 1995).

For temporal databases to be useful in support of Real Time Distributed Simulations, they must support storage and retrieval of simulation *interactions* in addition to object state *updates*. Interactions, in the context of temporal databases, occur at an *instant*. This means that the begin and end valid time for an interaction are the same value. Moreover, since interactions do not persist (i.e., their life-span is zero), there is no notion of change to an interaction. Instantaneous entries in temporal databases have peculiar implications to timeslice operators. This aspect of interactions is important and will be addressed in subsequent sections.

2 Motivation

In keeping with the goals of HLA, it is appropriate to explore ways to more effectively collect, store and use the data from a simulation execution. When we examine the nature of this data, we note properties of distributed simulation which parallel certain concepts in temporal databases. Most notable is the common dominant theme of Time. Other interesting parallels exist between ownership management and concurrency control; federate interest management and replication; and execution synchronization and transaction serializability.

Temporal databases have been developed to deal with notion of the changing state of data. Recent work in real-time databases underscores the requirement to collect data from real-time systems, without interfering with their operation. (Snodgrass 1990) Data collection in distributed simulations may be a useful application of temporal databases.

In addressing the application of temporal database technology to distributed simulation data collection, there are several questions which must be answered. We address these questions from four broad perspectives: the data collection problem, handling the temporal ambiguities, the required functionality, and implementation approaches specific to HLA.

3 Data Collection

Data collection can be used to support after-action analysis, replay, or can be used for runtime monitoring. For this paper, we limit our definition of data collection to the process of gathering of information from the various federates during the simulation execution. Data collection in this manner results in a record of the *federation execution* which can be permanently preserved. The Federation Object Model provides each federate a complete list of public object and interaction classes which may be exchanged during the federation execution. We are interested in preserving both the state of the federates (which relates to the object classes) , and the events (which relates to the interaction classes).

To begin to address the data collection problem we observe that, at any point in a simulation execution, there is a set of federates that are producing data of a specific class. These federates have *published* the specified classes, and, in the case of objects, have registered instances with the RTI. At the same point, there is a complementary set of consumers of the data, who have expressed interest by *subscribing* to the classes. The RTI must transport the data from the producers to the consumers that have subscribed to it. This process may be further refined by the specification of *regions*, which define finer granularity to the interest expression.

We have described the goals of the data collection and the system from which the data will be collected. The following questions must be answered before we can understand the requirements for a good data collection tool design.

3.1 How much do we need to collect?

The first question regarding data collection is, intuitively, related to the amount of data collected. There are several useful alternatives to consider. Before we can assess the viability of a temporal DB application, we must be able to express the collection requirement more clearly. We can describe at least three alternatives for a data collection set with implications on the amount of data they collect.

- Comprehensive Set. We can easily define the set of all data within the federation execution. This is a straightforward definition, and is intuitively appealing. However, some federations produce very large amounts of data. Moreover, the distribution of the data is dynamic, which may change the required configuration of data collection points. Attempting to collect a comprehensive set violates some of the spirit of HLA simulation, in that it requires data to be produced which is not necessary for the simulation execution. (Bachinsky, Tarbox et al. 1997)

- Partial Set. An alternative to comprehensive data collection is the definition of a partial set of the execution data which will be recorded. This may resolve issues regarding scale, but is an arbitrary approach.

- Dynamic Set. There may be ways to express data collection requirements in a way which can be interpreted during the execution. The collection requirements may be predicated upon the state of the simulation execution. This may allow direct expression of the collection requirement in terms of federation goals (measures of performance, etc.).

- Sufficient Set. During the execution, the RTI is only required to pass data in which some federate has expressed interest. A federate's *interest set*, is the set of object instance attributes and interaction classes to which the federate has subscribed. We may be able to take advantage of this observation about the RTI in our data collection approach. We observe that a data collection mechanism may possibly interfere with the execution by generating additional communications within the RTI, not required by the federates. By defining the *federation sufficient set,* the union of all federate interest sets, we describe all the data which is passed during the federation execution, given no additional data collection interference. We note that the federation sufficient set will contain the data necessary to recreate the ordered delivery of all messages received by any federate during the execution. The notion sufficient set is being used effectively in current implementations of distributed debugging tools such as Instant Replay and BEE to create repeatable execution environments (LeBlanc and Mellor-Crummey 1987; Bruegge 1991).

3.2 What about the private data?

In addressing general data collection requirements, we find that HLA has no notion of private data space, data which is not defined in the FOM. However, in addressing reusability, we must note that private data is important also. If, for example, the RTI is used to support save and restore operations, then the federate must be able to restore itself to its former internal state, which may involve restoration of private data. There may be data collection functionality for storing private data which is useful in after-action review (AAR) or other post-hoc analysis. The ability to store and index private data in the same manner as the data received through the RTI could provide a useful capability.

3.3 How do we minimize the impact of data collection?

Regardless of the amount of data collected, there will be some additional network, and processing load placed upon the simulation system. Moreover, a centralized, or single collection point logging approach is likely to be impractical for many federations. If a distributed data collection scheme is developed, simulation network will be subjected to some load imposed by the required coordination. Methods to minimize the impact of data collection on the system under test will an important contribution.

3.4 Can current technology plausibly support data collection requirements?

Even at a basic level, it is yet to be proven that a temporal database approach can support the collection requirements. As has been noted in (Bachinsky, Tarbox et al. 1997), a moderate sized federation can have storage requirements in excess of .5 terabytes, per 48 hours execution. We reasonably can expect that the size of federations will grow. We may assume that the volume of data will grow as well. It is unclear whether a temporal database implementation can scale well.

4 TEMPORAL ISSUES

The application of a temporal database to distributed simulation data collection presents certain issues related to the order of delivery of update and interaction messages. In real-time simulations, many of the messages between federates will be delivered in receive-order. The RTI, in this case, can make no guarantees with regard to delivery order of these messages. Additionally, in simulations where each federate's time is managed externally, there may not be perfect synchrony between respective simulation clocks. The data collection mechanism itself may have an external (wallclock) time. These facts serve to complicate the direct application of temporal database technology, and lead to the following open questions.

4.1 Can we apply the concept of transaction time and valid time to a real-time simulation?

Transaction time and valid time are explained in Background, Section 1.3. The transaction time relates to the availability of the data in the database. However, if we assumed a distributed, or *federated* database approach, we may have several transaction times for the same data, one for each point at which the data is received. We may want to have a special notion of transaction time for the producer of the update.

The notion of valid time transfers well to the HLA as the timestamp of the update or send. However, since we are dealing with real-time simulations, where a timestamp is frequently t = NOW, it is likely that updates may arrive after the beginning of valid time. This means that for some period of time, the correct data was not available. No current temporal database research addresses such an occurrence.

4.2 Can we handle runtime query processing?

Processing a query to the database may be an important part of federations which guide their execution by runtime analysis. To this end, it may be desirable to develop ways to process queries while the data is being collected. The overhead of the query processing implementations will help determine the overall efficacy of run-time query support.

Query processing is complicated by the network latency, as mentioned above, and the lack of a universal time clock. If, for example, a query is processed for t = NOW, it should return the current attribute values of all the simulation objects. If, however, an update is pending, having been sent at some time before, say t = NOW - 10 (10 milliseconds ago), and the valid time for the update is time t = NOW - 5 (has a timestamp of 5 milliseconds ago), but it has not been received or posted to the database, the query cannot possible return the correct attribute values. In a real-time simulation, where no timestamps are used, there are no temporal database semantics which can tell us what the return values should be. (Wang, Jajodia et al. 1993; Clifford, Dyreson et al. 1997)

4.3 What is the state of the database at a given point of time?

If data is collected using a federated temporal database, there are no models for determining the state of the database at a particular moment. It would be desirable to have the state of the database mirror the state of the simulation itself. Because the individual database instances will receive updates and interactions at differing times, there is no straightforward way to consolidate the data. Therefore, it appears that the database state will always lag the simulation state. This is further complicated by message drop out, where a subscriber is not always guaranteed to receive all messages transmitted using a particular transport service. This means that the database will have an imperfectly mirrored state with respect to the simulation.

5 FUNCTIONALITY

If we presume the viability of a distributed temporal database data collection scheme, we may begin to define basic criterion for its utility. In this section we ask questions relating to the functional requirements for data collection and analysis, and relate them to the proposed temporal database implementation. We observe that there are three basic requirements: data logging, log management, and post-hoc review. We also note that this implementation may allow for additional capabilities in non-linear review, data mining, and in the construction of repeatable execution environments.

5.1 What is needed to support run-time logging?

The temporal database will need to have some control functions which apply to the federation execution. We may need commands to start and stop the collection process. We may need commands to establish the schema, if it cannot be derived from the federation object model. There must certainly be a means to establish the collection set, as described in section 3. There may be a need to modify the collection set during the execution.

At a fundamental level, the implementation must be able to obtain and store the execution data at the rate it is produced. An efficient approach to this will take advantage of monotonically increasing local transaction time. A log-based temporal database maintains append-only data structures. The log-based approach may offer some computational simplifications over other storage techniques. The state of the database, at a particular point in time is computed by knowing a previous cached state, and incremental changes from the transaction log up to the desired point. This technique may be a more efficient than a traditional storage model in event log applications such as a distributed simulation.

5.2 How can support log management?

The data collected during the federation execution is a single record of the simulation. If the data is collected at several physical points, it may be desirable to reference the data as a unified log. To do this, the database must support either logical or physical merging of the database. This process may be done during the execution, or afterwards. The merge is complicated by the same temporal issues, mentioned previously. We must also deal with the need to remove duplicate entries in the log. Because of the differences in local transaction time, it may be difficult to identify when duplicates actually exist. Moreover, it is not clear how one would establish which entry was the best, or correct one where duplicate entries have differing timestamps. We can't expect to solve the general problem of how to merge logs with no common wall clock, but we may be able to determine logical approaches for HLA.

5.3 How will the data be reviewed?

At the heart of the reusability issue for this type of simulation data is the notion that some analysis will be performed on the data after it is collected. It is important that research be directed to provide useful analytic functions. Several notional capabilities are proposed here.

Since the real-time simulations are frequently used to create virtual environments for humans, a VCR is a natural metaphor for recording execution data. Using visualization applications, one can monitor the progress of the execution. In DIS simulations, logs have been used to provide VCR-like functions.

In HLA simulations, we apply the VCR metaphor to the review of the state of the simulation objects and the occurrence of interactions. Several review functions may be appropriate for temporal database support, and have been explored in previous log-based database research. These include at least:

- Play, pause and stop. The play function could be in real or scaled time.
- Rewind and reverse. The idea of a reversal of the simulation state is not straightforward in DIS simulations, but can be accomplished with temporal data base support by decremental downdate, as mentioned in the introduction.
- Frame advance. Since updates are only sent when change occurs, the period of time between changes can be referred to as a frame.
- Zoom and focus. When focusing on a particular portion of the record, it may be desirable to reduce the working set of data. This is analogous to a database filter.

5.4 Can non-linear review be supported?

It may be useful to review the recorded data in a non-linear manner. The analyst may find larger portions of the data irrelevant, or may be reviewing multiple logs simultaneously. There is no established research in this area. However, previous research has provide ways to provide quasi-random access to database states. Through periodic storage of the current state cache, the state of the database can be rapidly determined for any timeslice. This research would have to be extended to support general non-linear review.

5.5 Can data mining be supported?

Simulation executions would appear to be target rich opportunities for innovative data mining techniques. One might use inferential techniques to discover patterns of execution which are not intuitive to an outside observer. (Jensen and Mark 1992) However, to support such operations, we may need to expand current concepts of data mining.

In this type of simulation, for instance, there is an interesting duality between object states and interaction events. This is due to the fact that object instances persist in the simulation data space, whereas interactions are transient. When querying the database about its state, we can specify a time instant, and be returned a timeslice. This timeslice is the state of all the objects in the database at the specified time. Since interactions occur at an instant, and posses not state information, there is no natural way to express them in a timeslice. It is obvious that, in order to deal with interactions, one would specify a period, rather than a single time value. There are no convenient semantics for dealing with this. In order to include interactions in data mining of simulation, timeslice and other temporal operators may need to be modified.

When an object changes its state, and updates its attribute values, the new values are recorded. Techniques are being developed for mining based upon queries on change. Although this applies directly to attribute value updates, it is unclear how one might adapt this to interactions.

5.6 Can this data be used to support repeatable execution?

Achieving a repeatable distributed is difficult. At least two general approaches for producing repeatable executions have been developed: log based approaches and message ordering approaches. Log-based approaches record needed simulation events as they occur, such that a replay will provide any needed information, in a repeatable fashion. A message ordering approach assumes that the logical processes (LPs) or federates are repeatable except to the degree that relative message timing yield non-determinism. This is discussed in detail in (Bruegge 1991). It may be possible to use temporal replay based on local transaction time discussed here in an approach to repeatability.

6 IMPLEMENTATION ALTERNATIVES

When we consider ways to implement the functionality described, we must decide how much of the RTI should we use to support this data collection. In this section we raise issues regarding the use of the RTI by the temporal database system.

6.1 What should the architectural relationship of the temporal database be to the RTI?

We have several choices in this regard. A logical first choice might be to implement the database as one or more federates. This has the advantage of using the existing RTI services. We may instead choose to implement the database completely outside the RTI, monitoring the raw communications to collect data. This has the advantage of mitigating the penalty on the rest of the federation, but, since there are no current "on-the-wire" standards, it may be difficult to implement. Thirdly, we may choose to implement the database within the RTI itself. We believe that it makes sense to implement the collection as a set of collaborating analysis federates, as explained in (Jackson and Wood 1997).

6.2 How will the RTI services be can we use?

There are several services which are intuitively useful in the implementation of the temporal database. Certainly, if the temporal database is implemented as an HLA federate, the basic services of join, subscribe, unsubscribe, etc., will be useful. In this way the data collection connection sill be established with the data producers. One notably useful function is the RequestObjectAttributeUpdate service. This service may find natural use in processing queries for NOW.

There are several open questions with respect to use of the RTI services to support temporal database functions. Each of these questions present interesting research opportunities.

How do we manage time within the federated databases? Even though the simulation may be running real-time, there is no reason why the databases themselves can't maintain a time based synchronization.

How do we ask a query? If we support run-time query processing, how can a query be formulated and expressed using the RTI.

How do we process a query? The process gathering the data together to reply to a query is a non-trivial problem. We need to determine whether query processing using the RTI is viable.

How do we monitor the load on the database management system? When the production and consumption of data shifts, or changes its characteristics, there will a change in the load on the various database instances. We need a means to monitor or derive the change in the load to any database instance.

How can we implement a logger protocol? The implementation will need an intra-database protocol to dynamically shift collection responsibilities from one database instance to another. We may be able to implement this protocol by extending the MOM concept.

Can we extend the RTI MOM? Some of the data needed to manage the data collection process are already available in the RTI MOM. The data collection application can determine a federates subscription set, for example. Other information in the MOM may provide much of the needed federate data.

7 SUMMARY

The ability to collect, and manage and review simulation execution data is a practical aspect of reuse. Temporal databases appear to support flexible data collection and analysis of simulation executions. There are, however, several issues regarding implementation of a temporal database in the real-time distributed simulation which will need to be resolved.

In this paper, we have identified several open areas for research in the application of temporal database concepts to HLA data collection. We have noted the opportunity to extend some temporal database concepts to embrace distributed simulation concepts, and how analysis may be enhanced through application of temporal database techniques.

REFERENCES

Bachinsky, S., G. Tarbox, et al. 1997. Data Collection in an HLA Environment. In *Proceedings of the 1997 Spring Simulation Interoperability Workshop*, Orlando, FL, UCF-IST.

Bruegge, B. 1991. A Portable Platform for Distributed Event Environments. .

Clifford, J., C. Dyreson, et al. 1997. On the Semantics of "Now" in Databases. *ACM Transactions on Database Systems* 22(2): 171-214.

DMSO 1995. Modeling and Simulation Master Plan. Alexandria, VA, Defense Modeling and Simulation Office.

DMSO 1997. HLA Interface Specification, Version 1.1, Defense Modeling and Simulation Office.

Fujimoto, R. M. 1990. Parallel Discrete Event Simulation. *Communications of the ACM* 33(10): October 1990.

Jackson, L. and J. R. Wood 1997. A Federate for Analysis in Advance Distributed Simulation. In *Proceedings of the 1997 Fall Simulation Interoperability Workshop*, Orlando, FL, IST-UCF.

Jensen, C. S., J. Clifford, et al. 1992. A Glossary or Temporal Database Concepts. *ACM SIGMOD Record* 21(3): 35-43.

Jensen, C. S. and L. Mark 1992. Queries on Change in and Extended Relational Model. *IEEE Transactions on Knowledge and Data Engineering* 4(2): 192-200.

Jensen, C. S., L. Mark, et al. 1993. Using Deffierential Techniques to Efficiently Support Transaction Time. *The VLDB Journal* 2(1): 75.

Jensen, C. S., L. Mark, et al. 1991. Incremental Implementation Model for Relational Databases with Transaction Time. *IEEE Transactions on Knowledge and Data Engineering* 3(4): 461-473.

LeBlanc, T. J. and J. M. Mellor-Crummey 1987. Debugging Parallel Programs with Instant Replay. *IEEE Transactions on Computers* C-36(4): 471-481.

Loper, M. L., T. McLean, et al. 1997. The HLA Federate Compliance Testing Process, Revised. In *Proceedings of the 1997 Fall Simulation Interoperability Workshop*, Orlando, FL.

Ozsoyoglu, G. and R. Snodgrass 1995. Temporal and Real-Time Databases: A Survey. *Transactions on Knowledge and Data Engineering* 7(4).

Snodgrass, R. 1990. Temporal Databases Status and Research Directories. *SIGMOD Record* 19(4): December 1990.

Wang, X., S. Jajodia, et al. 1993. Temporal Modules: An Approach Toward Federated Temporal Databases. *ACM SIGMOD* 5: 227-236.

AUTHOR BIOGRAPHIES

THOM MCLEAN is a Research Scientist II at the Georgia Tech Research Institute and a graduate student in the College of Computing. He received a BS in Aerospace Engineering from the U.S. Naval Academy in 1984, and a MS in Computer Science from Texas A&M University in 1992. His thesis topic is the Analysis of Real-Time Distributed Simulations. His other research interests include scientific computing, meta-databases, and message sequence charts.

LEO MARK has been Associate Professor in the College of Computing at the Georgia Institute of Technology since 1992. He received a MS and PhD from Aarhus University, in Denmark. From 1986-1992, he was an Assistant Professor at the University of Maryland, College park. His research interests include temporal databases, incremental computation methods, self-describing databases, database generators, metadata management.

MARGARET LOPER is a Research Scientists II at Georgia Tech Research Institute and a graduate student in the College of Computing at Georgia Tech. She received a BS in Electrical Engineering from Clemson University in 1985, and a MS in Computer Engineering from University of Central Florida in 1991. Her thesis topic is Causality in Distributed Simulations.

DAVID ROSENBAUM is a Research Scientist at the Georgia Tech Research Institute. He received an MS in Computer Science from Georgia Tech in 1990 and a BS in Mathematics from Michigan State University in 1983. His research interests include software architecture, distributed simulation systems, virtual environments, and object-oriented data technologies.