

03F-SIW-086

An Architecture for Capturing All Network Data in Any Scale Simulation Event

Steven R. Moore
VisiTech, Ltd.
535A East Braddock Road
Alexandria, VA 22314
757-200-8208
moore@visitech.com

Stephen Kasputis
VisiTech, Ltd.
535A East Braddock Road
Alexandria, VA 22314
703-535-6640
kasputis@visitech.com

Keywords:
AARS, HLA, DVTE, Data Collection

ABSTRACT: *A mandatory requirement for most simulation events is that simulation data be captured and stored for analysis. As simulation events become larger, more complex, and geographically distributed, the collection and synchronization of the large amounts of data presents an increasingly difficult challenge. Millennium Challenge 02 is an example where 30000-35000 entities were sent across the network. Most data capture tools are only capable of capturing a portion of the data. Centralized data collection presents a stressing problem for network bandwidth and network interface of the data collection device. Distributed data collection presents the problem of data synchronization and continuity.*

The Deployable Virtual Training Environment (DVTE), a prototype system designed to support Marine Corps training both on ship and in the field, developed an architecture that is capable of capturing vast amounts of data from disparate sources. The architecture built in DVTE allowed for the synchronized capture and playback of HLA, DIS, and multicast data. This paper describes the architecture for the synchronized capture, playback, and analysis of all the network data for DVTE and the enhancements made to allow this architecture to be easily incorporated into any simulation event.

1. Introduction

The Deployable Virtual Training Environment (DVTE) is a prototype system designed to support Marine Corps training both on ship and in the field [1]. A key component of DVTE is the After Action Review System (AARS), which captures, processes, and replays vast amounts of data from disparate sources. DVTE uses the Distributed Interactive Simulation (DIS) protocol for voice communications and the High Level Architecture (HLA) for simulation data. Although only the HLA simulation data needs to be processed both DIS and HLA data must be captured to be replayed in a synchronized fashion. Running with several disparate simulation protocols is not an issue reserved for DVTE. Several other simulation events have a similar requirement.

Simulation events are becoming larger, more complex, and geographically distributed, therefore the collection and synchronization of the large amounts of data presents and increasingly difficult challenge [2]. Millennium Challenge 02 is an example where 30000-35000 entities were sent across the network. Most data capture tools are only capable of capturing a portion of the data. Centralized data collection presents a stressing problem for network bandwidth and network interface of the data collection device. Distributed data collection presents the problem of data synchronization and continuity. The AARS architecture defined here will help overcome some of the increasing challenges facing current after action reviewing systems today.

2. Objectives

The objectives of this architecture are quite simple:

- Provide a framework for capturing and synchronizing vast amounts of data using disparate simulation network protocols which are simulation event independent.
- Provide a framework to allow live analysis of data during the course of a simulation event.
- Provide a framework to allow users to “bookmark” significant simulation events.

3. Overview

The architecture takes an Object Oriented (OO) approach to implementing an After Action Review System (AARS). The individual components of the AARS are highly cohesive with one another, but are not very tightly coupled. The components which make up the architecture are:

- A. Playback Logger
- B. Data Decoder
- C. AARS Graphical User Interface (GUI)

Each of the key components listed above are individual applications which can reside on a single machine or can be distributed across multiple machines. There can also be multiple instances of each of the components, which will then require synchronization as will be discussed in more detail later. Distributed Interactive Simulation (DIS), multi-cast, and High Level Architecture (HLA) are three commonly used simulation protocols, which will be discussed throughout the paper when describing the various architecture components.

4. Architecture Elements

The core components provide five primary functional elements consisting of the playback logger, data decoder, AARS Graphical User Interface (GUI), distributed control and synchronization, and the simulation bookmark. The architecture does use a commercial off the shelf (COTS) database, but since any database supporting Open Database Connectivity (ODBC) can be used the database is not considered a primary functional element.

4.1 Playback Logger

The playback logger is used to capture data from a network and then replay that data back on the network at the same rate. The logger has similar controls to that of a common DVD player. The logger is able to play, pause, fast forward, rewind, and jump to a particular time. Due to the vast amount of data which may be captured and then replayed the logger is built for speed. To maintain the speed the playback logger records network data in its

raw form. By raw form it is meant that the data is kept in its original binary format and is not decoded. The logger stores the raw network data in a sequential order in a random access file. Each network packet is stored as a record in the random access file not only so data can be read very quickly, but so jumps can also be made. Storing the playback information in a file on the local machine access is much quicker than attempting to query a database over the network. Data is more portable, not relying on any specific database.

Attempting to capture all network data is still very difficult even when trying to capture the data in its raw form. The reason is the different types of network protocols. Capturing broadcast user datagram protocol (UDP) is trivial, because when a packet is sent out over the network it is sent to all of the clients: they “broadcast” the packet. Broadcast is what Distributed Interactive Simulation (DIS) commonly uses. To capture broadcast packets a logger would simply need to sit on the network and listen, and this is transparent to all of the other nodes on that network. Multi-cast is very similar to broadcast, but it uses subscriptions instead of broadcasting all data to everyone. The nodes on the network would subscribe to the data they are interested in. Again a logger would be able to record the data by simply subscribing to the interested multi-cast streams. To make matters difficult however Transmission Control Protocol/Internet Protocol (TCP/IP) establishes direct connections with hosts to communicate. Therefore a logger needs a direct connection from each source of data. This becomes intrusive on the application developer since the application would need to duplicate each packet to each of the hosts it would need to go to. This also becomes an issue for the logger not only because it will need to open a socket for each of the hosts, but during playback the logger would need to make a connection to each of the hosts that were supposed to receive each of those packets as well. The benefit of using TCP/IP is that communication between clients is reliable, while broadcast and multi-cast protocols are based upon UDP which is considered unreliable, but is very fast. There are tradeoffs for using each type of protocol. Some simulations systems may require reliable communications for critical data and may opt to go with a more difficult TCP/IP implementation.

To ease the burden of establishing direct socket communications most distributed simulation systems typically use some type of broadcast or multi-cast methodology. This however does not solve an additional problem. Some applications are designed for extremely low bandwidth networks and may use some packet header information to avoid contention. For instance, some applications may look at the packet header information for the sending host of the packet. When several packets are received at nearly the same time some contention can be resolved by

grouping the packets based upon the sender information in the header of the packet. This, of course, is a problem for the logger, because when playing back the captured information all of the packets will be coming from one machine. The problem is that the packet's header information is usually filled in at a very low level by the Operating System (OS). It is possible to "spoof" the sender of the host by making a low level OS call, but then the logger will need to be able to maintain a list from where the packets were sent.

Use of the High Level Architecture (HLA) adds yet another consideration for data logging. HLA uses a combination of several types of network protocols including UDP, TCP/IP, and multi-cast. One of the features of HLA is that it is very dynamic when it comes to network protocols. It can use one or several different network protocols. Due to HLA being so dynamic there is no easy way to capture the multi-cast, TCP/IP, and UDP in an isolated manner and replay that data. For this reason it is necessary for the logger to become a member of the HLA federation, a federate, in order to record and playback the HLA data. Aside from Time Management HLA uses a publish and subscribe methodology similar to multi-cast. A HLA federate will join a federation and then publish and subscribe to specific objects and interactions as described in the Federation Object Model (FOM). At the end of the exercise the federate will then resign from the federation. There are many different FOMs and each simulation event may have its own. This can lead to a problem with the replay logger since it should not be locked to any particular simulation event. There are even some simulation events which may be running two disparate FOMs in two different federations at the same time. For this reason the replay logger should try to maintain its simulation event specific independence. Doing this in HLA is a little more difficult. As discussed earlier the FOM specifies the data that is sent across the network. The FOM is defined in a .fed, or FED file. The FED file describes the objects interactions and the data distribution spaces [3]. By reading the FED file the logger is able to read the objects and interactions that are available for subscription and publication.

As described above three major simulation protocols are DIS, multi-cast, and HLA. Since DIS is broadcast all clients on a network will receive all of the data, and it is possible that the network will become quickly saturated. Therefore there are methodologies in place to try to reduce the amount of traffic sent across the network. When using multi-cast and HLA however individual clients subscribe to only the data they need so larger simulation events can be supported by a network, especially if multi-cast intelligent network equipment is used. To help possibly overcome the vast amounts of data a logger may receive and need to replay when in a multi-cast or HLA

mode it may be beneficial to allow a user to modify which streams to subscribe to in multi-cast and to choose which objects and interactions to subscribe to in HLA. Depending on the purposes of the replay it may not be necessary to capture and replay all of the simulation data. For instance if the replay is for a visual system during and After Action Review, (AAR) it may not be necessary to capture or replay radio messages if the visual system does not show that information. However, capturing all simulation data may be critical for other more scientific events. If there is too much data for one logger, the load may need to be distributed among multiple loggers. This of course could lead to some synchronization problems between the multiple loggers. Synchronization issues will be discussed below.

Several different simulation protocols were previously discussed, and it would be difficult to attempt to manage several different applications which are very similar. For DVTE the Playback Logger was able to record and playback DIS, Multi-cast, and HLA. A user could simply select in which mode the logger should operate. This allowed all of the file data store code to be reused seamlessly.

4.2 Data Decoder

Maintaining the idea of high cohesion low coupling the data decoder is an application which reads the pertinent information from the network, decodes that data, and stores that data in a database. Since decoding data and then writing to a database may take some time this data decoder is isolated from the logger. The biggest issue concerning the Data Decoder is how to decode the incoming network data. It is very difficult to make this application simulation event independent. There are standards which do exist and can be used to decode some data. For instance DIS has a very well defined standard. The problem however is that DIS as well as other protocols can deviate from the standard, and there is no template to follow for a standard.

HLA is more generalized than DIS, but it does have a template for defining the internal data representation. The objects and interaction sent across the network are defined in the FOM as described above, however the datatypes used for the object attributes and interaction parameter are defined in the Simulation Object Model (SOM). The Object Model Template (OMT) is a template which describes the HLA data [4]. The OMT file can be read to decipher the datatypes which correspond to each of the object attributes and interaction parameters so the data can be decoded and then inserted into a database for later analysis. The SOM, however, only gives a first order level decoding, and it is possible that it may not be sufficient. For instance radio transmissions may be sent over as an

array of octets which would mean absolutely nothing to a user. For that reason there may have to be some second order decoding involved. This responsibility could be placed on the client tool retrieving the information.

The key features of using a database for the data decoder is that many of the technical issues are already handled. When recording for a playback there is no need to query the information, and a playback is usually not done until the simulation event is complete. However for analysis it is possible that users may want to try to obtain information about the exercise as it progresses; possibly for technical purposes to make sure all of the components are responding correctly, or for training purposes to see how individuals are responding to certain types of stimuli. A database system will arbitrate between putting data in the database and retrieving data from the database. This allows data to be written to the database while a user queries data from the database.

4.3 AARS GUI

The AARS GUI is a tool that is used to allow a user to access and analyze the data captured by the Data Decoder. The GUI should give the user a summary but also allow the user to be able to get more detailed information as well. Typically an AARS GUI addresses requirements based upon the simulation event Measurements of Effectiveness (MOEs) and Measurements of Performance (MOPs). Because these differ for each simulation event, the AARS GUI is very difficult to keep data independent. Specific users are typically only interested in specific data. A possible way to make the GUI less specific is to give the user various summary pages, but then also allow the user to developed more advanced database queries. This will allow the user to dynamically enhance the AARS GUI as required.

Another key function of the AARS GUI, in this architecture, is to control the components of the entire AARS system. The AARS GUI controls the loggers and should have buttons to control all of the features of the loggers. A key element of the AARS GUI is that it is able to tie information to the control of the loggers. The AARS GUI is able to analyze the data and determine what a significant event might be, and should therefore be able to control the loggers to go to that significant event. The significant event can be referred to as a "Simulation Bookmark" and that will be discussed in more detail below. Figure 1 shows the AARS GUI used for DVTE.

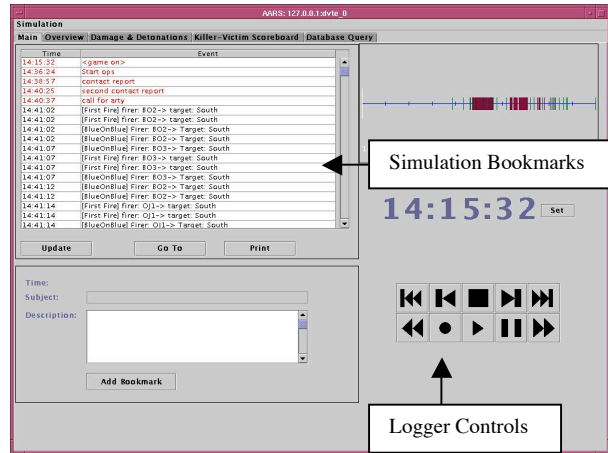


Figure 1. AARS GUI

4.4 Distributed Control and Synchronization

Distributed Control and Synchronization is the key concept of this architecture, and the need for the distributed control and synchronization is crucial. Different simulation protocols were discussed earlier. Imagine a simulation event that uses several of those protocols. For example DVTE used HLA for simulation data and DIS for voice data. This requires at least two disparate loggers to record all of the data for the simulation event. The need for multiple loggers introduces synchronization issues, especially during replay. Loggers can start to record at different times, but the system times must remain synchronized. To maintain consistent system times perhaps the most commonly used tool is Network Time Protocol (NTP). NTP will allow systems to stay in close enough synchronization for most applications. The loggers timestamp packets in receive order. This means when a logger receives a packet it will put a timestamp on it based upon its own current system time. The data decoder timestamps incoming data in an identical fashion.

There are multiple ways to start recording data to allow the most flexibility. Since the data decoder is not used for playback it will start decoding data and writing the data to the database immediately upon initialization. After the loggers have been started they can be told to start recording in one of two ways; either by pressing the record button on each logger individually, which could lead to synchronization problems, or the AARS GUI can issue the record command. The AARS GUI has a panel to control the loggers. Upon initialization the AARS GUI establishes a TCP/IP socket connection with one of the loggers. The AARS GUI will then pass on specific information during initialization such as the filename to use. All of the loggers automatically open and subscribe to a specific multi-cast stream upon initialization. When the logger which communicates with the AARS GUI receives a

command from the AARS GUI it will then send out a command to all of the other loggers who are listening to that multi-cast group. To issue the command for all of the loggers to record, a user would simply click on the record button in the AARS GUI, the record command is sent to the specific logger, and then the specific logger sends out the record command to all of the other loggers. Figure 2 shows how the different pieces of the AARS architecture communicate.

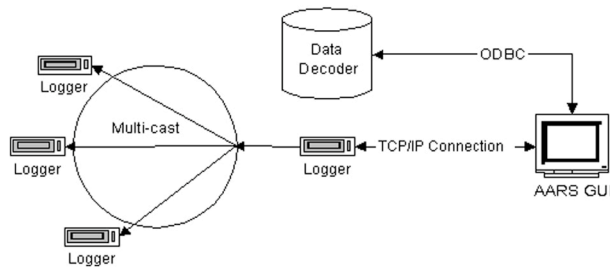


Figure 1. Architecture Communications

Analysis is done by mining the database. By the end of the simulation run, all of the simulation data will be decoded and stored in the database. The AARS GUI communicates to the database using Open Database Connectivity (ODBC). Using ODBC allows the AARS GUI to stay database independent. The AARS GUI does not communicate directly with the data decoder but only with the database.

To replay a simulation event, only the loggers and the AARS GUI are needed. Similar to recording, the loggers are started first and are set to a specific mode (DIS,HLA, or multi-cast). The AARS GUI is then started. The user will tell the AARS GUI to connect to a specific logger and choose a file to play back. The AARS GUI will then send out the filename to the specific logger which will in turn be sent out to all of the other loggers. To start the replay the user then clicks on the play button, and the loggers start playing from the beginning of their log files. Starting a playback from the beginning of a logger file may be an issue if there a lot of down time between the time the loggers started recording and when the actual event started happening. This was one of the reasons the simulation bookmark was introduced.

4.5 Simulation Bookmark

The Simulation Bookmark is designed to be similar to the bookmark feature available in most internet browsers. The Simulation Bookmark is intended to be used by anyone during a simulation event to “bookmark” a significant event. The Simulation Bookmarks are used as reference points, and are used to jump the data loggers to a time at those bookmarks. Bookmarks can be generated manually

or automatically from a database query. The simulation bookmark contains three fields: time, subject, and description. The time is filled in automatically, but it can be modified manually. The subject and description are both defined by the user. The subject is what would be quickly visible in the AARS GUI, where the description should be available if the Simulation Bookmark is highlighted.

The simulation bookmarks can be entered by anyone, but an instructor would probably use the simulation bookmarks to mark significant training events; possibly highlighting good or bad operational student behaviors. An example might be if an aircraft crossed into an artillery firing arch. The simulation bookmarks are used during an AAR event to force all of the loggers to jump to a particular time. During an AAR the instructor would simply highlight the simulation bookmark and click the “Go To” button. This will force all of the loggers to jump to a few seconds before the time in the simulation bookmark. The time to jump to before the simulation bookmark should be user modifiable. This allows time to give a ramp up to the actual event. Bookmarks could also be generated using queries to help an instructor find significant events. Figure 1 shows a list of simulation bookmarks captured during a DVTE event. The red simulation bookmarks are entered by users, while black simulation bookmarks are generated automatically to help users and instructors identify significant events.

5. Current Status

This architecture is currently in use and being enhanced. It is capable of controlling and synchronizing multiple loggers and placing the simulation data in an ODBC compliant database. The system can accept user generated bookmarks as well as automatically generate them based on database queries. The system has a FOM and SOM reader to quickly adapt to any HLA-based federation.

Improvements are continuing to be developed. The architecture requires the management of the separate components identified in Section 3. Therefore, a “One Button Start” capability is currently being developed to facilitate coordination of these components. Additional efforts are continuing to provide a more flexible and simulation event independent AARS GUI. Data Distribution Management (DDM) is used in HLA as a way to reduce the amount of traffic to a particular node by allowing nodes to filter data even more by subscribing and publishing to specific spaces and regions. Most of the information for determining the DDM spaces and regions is located in the OMT file. Efforts are currently underway to extend the architecture to implement DDM while still maintaining simulation event independence.

6. Conclusion

The AARS architecture defined in this paper provides a framework for capturing and synchronizing vast amounts of data from disparate network sources. The architecture is currently being successfully used by DVTE. There are enhancements being made to make this architecture even more usable and robust. As simulation events grow in size and complexity this architecture gives a foundation to allow After Action Review Systems to handle vast amounts of data and be able to offer instructors and users better tools to facilitate their analyses and better reinforce learning objectives.

7. References

- [1] Bailey, Michael P.; Armstrong, Robert: *The Deployable Virtual Training Environment*, I/ITSEC 2002, December 2002.
- [2] Vasend, Gerald: *After Action Review System Development Trends*, Proceedings of the 1995 Winter Simulation Conference, pp. 1262-1266, 1995.
- [3] *High Level Architecture Federation Execution Details (FED) File Specification*, Department of Defense, 31 July 1998.
- [4] *High-Level Architecture Object Model Template Specification Version 1.3*, Department of Defense, 5 February 1998.

Author Biographies

STEVEN R. MOORE is a Senior Software Engineer/Analyst at VisiTech, Ltd. He is the lead software engineer responsible for the AARS and supports other technical areas on the DVTE project. Steven holds a BS from Christopher Newport University.

STEPHEN KASPUTIS is the Chief Scientist with VisiTech, Ltd. in Alexandria, VA. He had held numerous positions in the Undersea Surveillance Program Office including Technical Director of the Fixed Distributed System. He has been the systems engineer for numerous simulation efforts. He has a BS in physics from Penn State, an MS in engineering acoustics from The Naval Postgraduate School, and a doctorate in acoustics from The Catholic University of America.