

Application of an HLA RTI for Database Access and File Propagation

Douglas Haanpaa
Charles J. Cohen, Ph.D.
Steve Rowe
Cybernet Systems Corporation
727 Airport Blvd.
Ann Arbor, MI 48108
734-668-2567

dhaanpaa@cybernet.com, ccohen@cybernet.com, srowe@cybernet.com

Keywords:

HLA, Database, File Sharing, Data Routing

ABSTRACT: *A task was undertaken to adapt Cybernet's OpenSkies™ RTI massive multiplayer networking software to a networked watershed simulation called PlanIT™, which was written by the Australian-based company Georeality®¹. The PlanIT™ software required a networking infrastructure in order to share a common MySQL database as well as to propagate files from upstream sub-basins/clients to downstream sub-basins. In the pre-OpenSkies PlanIT™, the user manually initiates both of these processes. The database transfer occurs when the user chooses to "update the database" and "request updates from database". The file propagation requires that the files be manually copied and renamed. This paper discusses the problems and solutions for performing distributed database access and file propagation over HLA, thereby automating both processes.*

¹ www.georeality.com.au/

1. Introduction

The need for a data request and reply mechanism is not directly answered by HLA. HLA is, for the most part, a method for broadcasting data. Another task is the serialization and reconstitution of file and database information. Yet another is the need to dynamic rename the transferred files. Finally, the system must somehow represent the rules for file transfer (i.e. who sends what to whom) and then facilitate the efficient network routing of that data from federate to federate. This paper will discuss the added mechanisms that were required to begin this integration as well as an evaluation of its potential performance in terms of network bandwidth requirements as well as interface usability.

2. PlanIT™

Georeality's® PlanIT™ software is a developer toolkit that facilitates the development of networked interaction and collaborative simulation. As an integration of a number of advanced VR software libraries, PlanIT™ provides a 3D application development framework that can be used to create, for example, games, simulation and training software.

While PlanIT™ can be applied to many types of simulation domains, the specific PlanIT™ simulation that Cybernet was brought in to address was to improve the networking capabilities for a watershed simulation application. The PlanIT™ concept was to break up a large watershed into of a number of sub-basins residing within a large virtual environment and associated by a network of interconnecting water flows. These sub-watersheds can then be assigned to wetland engineers/managers who then collaboratively simulate the effects and interdependencies of their decisions and integrate the results to attain a global picture. The geographic location of each of these sub-basins means that it will have upstream and downstream relationships with other sub-basins. For this application PlanIT™ marries the graphical simulation capability of its Auran Jet module with a domain specific simulation for watershed management called SWAT. SWAT, which is short for the *Soil and*

*Water Assessment Tool*², is a river basin scale model developed to quantify the impact of land management practices in large, complex watersheds. In PlanIT™, SWAT is invoked to simulate these watersheds over years, thereby producing output files specifying the results.

PlanIT's solution for collaborative watershed management achieved networked collaboration using two mechanisms: central database access and file sharing. As previously mentioned, these two tasks were somewhat manual, and this is what we attempted to address.

PlanIT™ DB Access

The PlanIT™ system implemented a system for distributing dynamic entities over a network. These entities would be things like markers, borders, and bounding areas. I.e. these are entities that users create, move about the geographic environment, and then perhaps remove. These entities are persistent via a dual-SQL database design that was created at Georeality®. The system utilized two types of SQL-based databases. The first of these is a client MSSQL (Microsoft SQL) database located on each client and which contains all local information that is important to that client. The other database is a server MySQL database that contains common information. That is, entities to which multiple clients, or all clients, require access. There is considerable overlap between these two types of database. In fact, the central MySQL database is, in fact, the central hub for distributing data from client to client. The method that PlanIT™ used to update these databases was fairly manual. The user would click an update button to update the local MSSQL DB with the latest information from the MySQL DB, and vice-versa. Any changes made within the PlanIT™ interface would not be realized on other clients until both transmitting and receiving client performed such an update.

PlanIT™ File Sharing

The other mechanism for data transport is file sharing. As mentioned previously, PlanIT™ uses the SWAT software to simulate a sub-basin over some period of time. After the SWAT software is invoked to simulate the natural/hydrological processes occurring in the watershed, an output file is generated. The output file of one client will, in fact, be the input file for sub-watersheds

² www.brc.tamus.edu/swat/index.html

residing immediate downstream. The original version of PlanIT™ requires the user to save out the file and then copy it (via windows file-sharing, email, ftp, etc.), to each downstream machine, renaming it to the appropriate input file name. Clearly there was opportunity for automation.

Network Requirements

In simplest terms, what was required was a system for automating the database updates and the file sharing. Additionally, a chat system was needed as well as a central lobby server that would provide users with an initial connection address and protocol.

In order to more clearly specify the required system, we needed to ascertain the bandwidth requirements for each of these data transfer modes. Indeed the very nature of how the network communication would work within the PlanIT™ system depended upon how big and how often these updates occurred.

It turns out that updates of DB entities, like areas, buildings, and points, are updates that have a typical size of a few thousand bytes and occur a bunch at initialization and only once every minute or so on average after that. This is because these updates only need to occur when a user changes something in the virtual environment, and worse case this happens at the rate a human can generate mouse click events.

Our networking system (a massive-multiplayer RTI) was intended to handle large numbers of objects moving in 6 degrees of freedom in real-time. Thus, our system can easily handle this human-speed data.

The file-transfer, however, requires more bandwidth. The SWAT output files can get up to 3 MB in size (typical of a 20 year simulation run in a sub-basin, although a typical 1-year run would be much shorter). Interestingly, a more interactive network simulation would update more often and would hence need to transport more of

these files during a simulation/collaboration, but this would imply that each simulation run is over a shorter time span, thus reducing the output file size.

3. HLA DB Interface

Based on the perceived requirements, we decided to create a wrapper interface that resides on top of the existing HLA-based interface. This so-called OSDB (OpenSkies Database) interface is a set of classes that facilitates communication to other clients/federates. This new interface provides the ability to:

1. *Copy and Rename files across the network,*
2. *Send free-form (raw bytes or text) requests to other clients*
3. *Respond to requests with database record-sets*
4. *Receive and extract records/fields from record sets*

Scalability

In order to create a system that provides the functionality listed above and is able to scale to large numbers of players, we are able to leverage our existing massive multiplayer networking technology. The OpenSkies system employs a

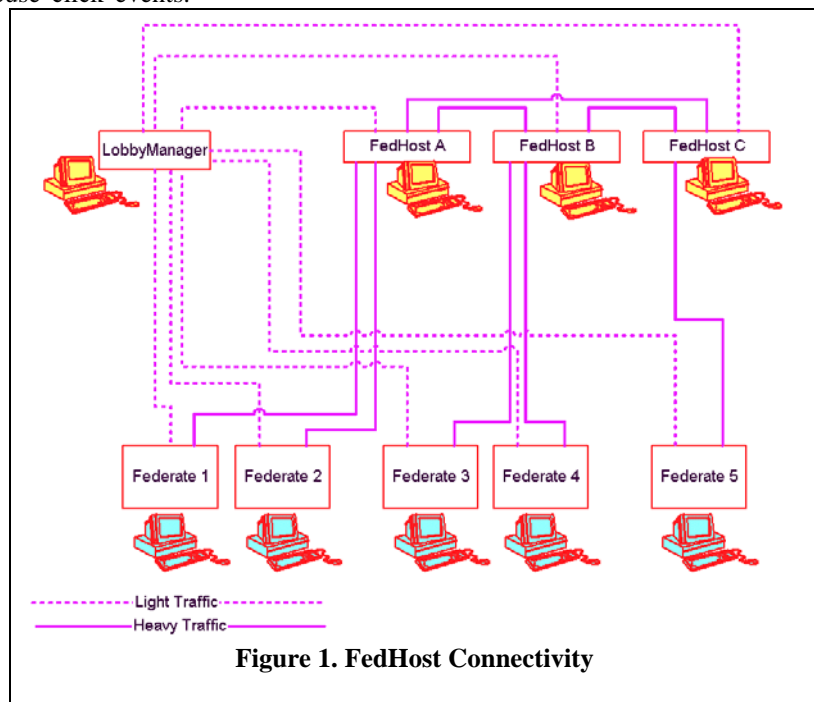


Figure 1. FedHost Connectivity

number of servers

The OpenSkies™ implementation of its Massive Multiplayer Online Gaming (MMPOG) developer kit is based on the HLA specification. An RTI implementation employing peer-to-peer communication between N peers will generate network traffic that is proportional to N^2 . This makes federate scale-up possible only on local areas networks and is limited even in this environment.

Cybernet's OpenSkies™ RTI implementation utilizes a network of servers called FedHosts to mediate network message traffic from the game or simulation clients. Each FedHost routes traffic for a subset of the Federates participating in the Federation. Initial contact for each Federate is to a LobbyManager, which immediately assigns the Federate to a FedHost, load balancing the FedHosts in the process. By itself, this does not eliminate the N^2 traffic problem. For this, OpenSkies™ implements a concept called culling rules, which are applets inserted into the FedHost that are allowed to decode message packets. Based on simulated entity state and packet contents, these culling rules can control messaging quality of service from each client to other clients. One possible culling implementation is an algorithm that culls based on relative distance. As an example, when objects are far away from each other in conceptual simulation space, data is transferred between them more slowly, or with less quality (or not at all) as compared to objects that are close and must interact more quickly. A subset of this capability is the *Data Distribution Management* mechanism that is a standard part of HLA. OpenSkies™, however, is not limited to this specific culling mechanism. Developers are able, using OpenSkies™, to implement their own culling module, which can take advantage of the attributes of a specific application and its requirements. Some examples of this are culling by radio frequency, culling by viewing angle, culling by distance, and culling by rank or security clearance. With culling properly implemented, each FedHost is primarily busy servicing its assigned clients so the number of clients that can be handled by the system scales linearly with the number of supporting FedHosts.

Since the developer is able to code his/her culling module using standard C++, the rules can be arbitrarily simple or complex.

In the context of this effort, the applicability of the culling rules is towards the distribution of database and file information across many servers. As shown in Figure 2 the distribution of multiple servers within an HLA (or similar) federation allows us to split up or replicate information in order to parallelize access. The culling rules can be used to route a particular request to an individual server based on the location of the information, the information types it maintains, and/or load information. For example, consider a data request originating from one of the clients. This request is received at its assigned FedHost server. The FedHost server looks at the request and routes it to Server "A" because:

- The request is for geographic information. Server "A" is the Geographic DB server, so the request is routed to "A".
- The request is for statistical information. All of the servers provide statistical file information, but server "C" is the least occupied at the time of the request, so the request is routed to "C".
- The request is for weather information. All of the servers provide weather information, but server "B" is the closest one (in the network sense) to the requesting client, so the request is routed to "C".

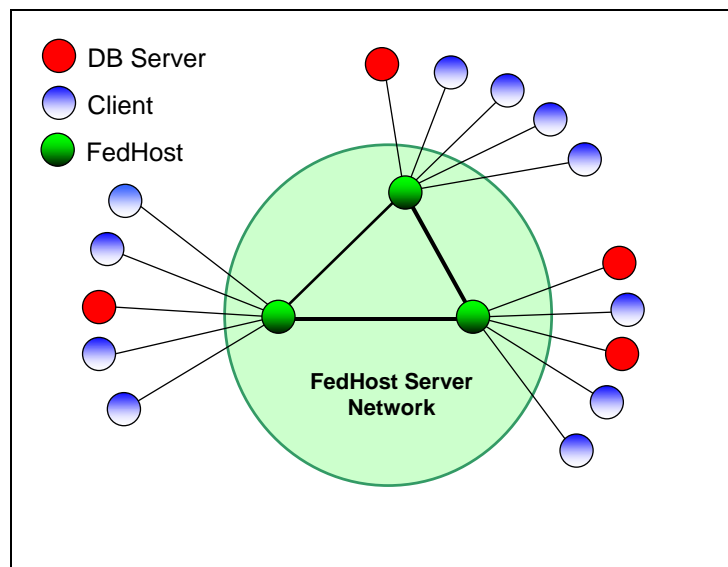


Figure 2: Distributed Database Servers

Distributed DB Access

We have implemented a method for database access over our existing HLA network infrastructure. The system allows each client application to send and request data to/from DB servers. In the network sense, these DB servers are, in fact, also clients/federates. In fact there may be cases every federate is also a DB servers, thereby facilitating a peer DB sharing capability.

The DB access API relies on the fact that all federates publish a standard federate object. This object identifies the federate by name and type. The name is simply a unique name for that federate. The type describes the federate's function. For example, a particular application of this might reserve the string "GeographicServer", so identify DB servers that provide geographic information.

The API for DB access is based on a request/response paradigm. The requestor is able to send requests to either a particular named federate, or to a server of a particular type. In the first case, the system will look for the named federate, and if it exists, relay the request to it. In the latter case, the system will look for a server of that type and then, if one exists, relay the message to it. If there is more than one server of the indicated type, the system will currently choose the first one off the list.

The content of the request is not defined by the OSDB system. It is simply a null-terminated character string that is intended to be filled with standard SQL select text. The system can easily be modified to restrict this, or to even make it more general purpose (by simply sending raw binary data), but for the intended purpose, character strings were the most convenient method.

The response is somewhat more interesting as typical SQL queries return record sets. We have implemented a class wrapper that allows the server to build up record sets using these two functions:

- AddRecord
- AddField

As with the request, the individual fields are filled with character strings. Again the data format could be constrained or generalized, but this was the most convenient method. Once the server application has filled the SQL response

with all of the necessary records and fields, it can then serialize the data and send it to the requestor. When the data response arrives at the requestor, there are utility functions for conveniently extracting the records/fields from the incoming data packet.

This request-response method for data updating is strictly point-to-point, so there's not much we can do to restrict the amount of network traffic generated. Culling rules are used, in this case, merely as a way to correctly direct requests.

In the case, however, where DB updates are sent as updates (i.e. not requested), the culling method may be more applicable. Often such updates need not only be sent to the DB server that maintains a particular type of data. They must also be propagated to other clients/federates that are interested in that type of data. In this case, standard culling methods apply. The method we typically prefer is to create a partitioning of the conceptual space into bins. When an update from a particular federate arrives at the FedHost server, it invokes the associated culling module; the handle for that object is placed in a bin corresponding to some aspect of the object. For example, the conceptual space may in fact be geographic space, divided into 1km by 1km squares. Each object with location information will then be placed into one of these bins and updates from that object will then be relayed to all other clients within that same bin (plus neighboring bins, perhaps). This applies quite readily, to DB updates that encode information about a conceptual space (e.g. geography, radio frequency, security level, etc.).

File Sharing

We have implemented a method for sharing files over our existing HLA network infrastructure. The system allows each client application to send and request files to/from other clients/federates. Similar to the way the DB request/reply is implemented above, this OSDB system allows the application to request named files from other clients and to respond to those requests. These requests can, like the DB requests described above, be made to named federates or by federate type.

The target application we were supporting in this effort, PlanIT™, used a file-based system for

input and output of a simulation. The idea was that many of these simulators would be operating over the network and each represented a link in a chain of simulations. The file output for simulator A, for example would be the input for simulator B. Because the naming conventions of these files did not take into account the possibility of future networking, it was up to us to rename these files appropriately.

We therefore implemented a file-sharing API that allows applications to create rules for file transfer. Such a rule might read as follows:

Send the file named *Aout.txt* from the client named *DocoServer* to the client named *Sim1*, and rename the file *Ain.txt* at the destination.

Such a rule can be invoked immediately or added to the list of rules. The system has a function for invoking all the rules where the source object is owned by the local host. Whether it is a send-once or a batch send, each file is read into memory, sent out to the destination client, where it is reconstituted and saved back out as the new file name.

This system allows applications to send images, movies, audio files, spreadsheets, documents, compressed folders, or anything that can be put into a file, over an existing HLA network.

Both of these APIs, DB access and file sharing, depended on the existence of a request/response facility. This was available within our RTI in two forms. First, the OpenSkies RTI has the ability to add an object reference to update functions, which will cause that update to be sent only to the owner of that object. Second, the culling rules can be used to restrict the destination to one federate rather easily (this would probably be possible using the DDM, but it would be unwieldy). Without this functionality, all messages would have had to be sent to multiple targets, thereby significantly reducing the efficiency of the system. But because this targeted messaging capability was available, the one federate was able to send a request to another “server” federate, and that

server federate could very easily use the incoming request information to form a targeted reply.

4. Conclusion and Future Direction

The current implementation of this library has the ability to request and send text-based SQL updates, and the API for creating these updates makes it simple to use. The file transfer capability was tested with 3MB image files and was able to transfer such files with negligible overhead (i.e. transfer latency was dominated by network transit time). The system is still in testing but seems to perform as expected,

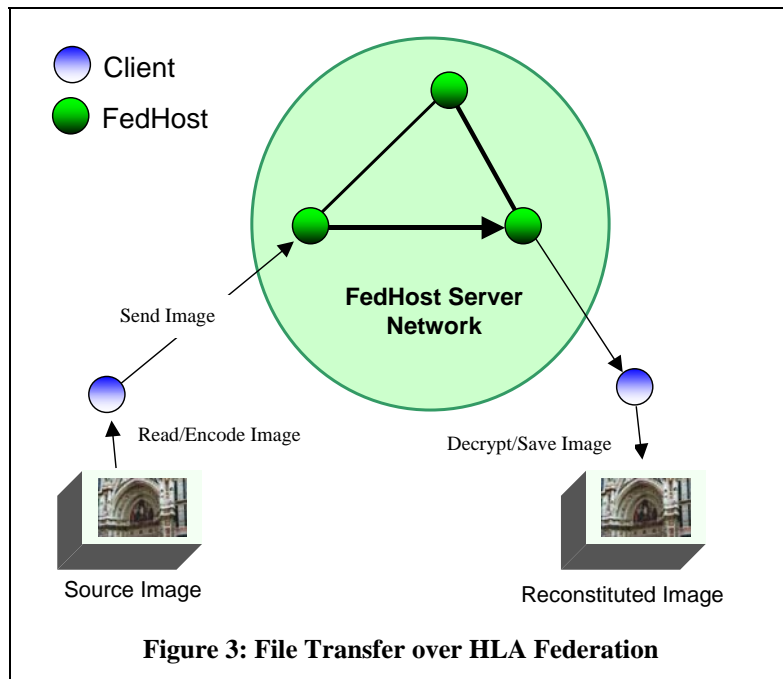


Figure 3: File Transfer over HLA Federation

consolidating file sharing with distributed DB access, chat and HLA connectivity all using one connection. Integration of the new OSDDB libraries into the PlanIT™ software is currently in progress at GeoReality®.

As mentioned in the previous section, the existence of a request/response facility within our RTI made possible the efficient implementation of these functionalities. It is possible that this capability is not within the scope of the intentions of HLA, but it is easy to imagine cases where point-to-point communication might be valuable in a distributed simulation context.

Such functionality could easily be added to the HLA specification. The MOM interface already has a facility for representing clients as objects.

These federate-objects can be subscribed to and discovered by other federates. The only thing missing is the ability to use this information to specify the destination of object updates and interactions.