A Gateway/Middleware HLA implementation and the extra Services that can be provided to the Simulation

Daniel J. Paterson
Erik S. Houglund, Ph.D.
Juan J. Sanmiguel

Research and Engineering
Naval Air Warfare Center Training Systems Division (NAWCTSD)
12350 Research Parkway
Orlando, Florida 32826–3275
patersondj@nvair.navy.mil, houglandes@nvair.navy.mil,

Keywords:
High Level Architecture, Gateway, Middleware, Simulation Middleware Object Classes, SMOC

ABSTRACT: This paper describes a Gateway/Middleware High Level Architecture (HLA) implementation and the extra Services that this implementation provides to a Simulation. This paper relates what Services (Data Distribution Management (DDM), Time Management, Agent, Dead Reckoning, Time Stamping, Update Filtering, Coordinate Conversion, MOM) a Gateway/Middleware HLA implementation could provide, and which are over and above what is available from the Run Time Infrastructure (RTI). This paper characterizes knowledge areas that we re-investigated in order to prepare to incorporate Gateway/Middleware Services into our HLA interface called the Simulation Object Middleware Classes (SMOC) Gateway. A sample Service called "SMOC Agent Class" will be discussed which handles Simulation situations like, "Federates who abruptly leave the federation without correctly resigning and the corresponding state of the undefined federate."

1. Introduction

The Department of Defense (DoD) use of High Level Architecture (HLA) as the basis for distributed, i.e., networked, and interactive DoD simulations created a requirement for tools to convert existing, or legacy, simulations to HLA. [1] Many government and commercial HLA tools have been developed and are listed on the DMSO web site [2].

Among the many tools for HLA development that are available are a class of software called "Middleware." Middleware software is defined as software between the application and the RTI. This Middleware can in some cases be configured as "Gateway’s" and/or bridges between federations running different wire protocols. See Figure 1 for a HLA Bridge example.

Middleware can be a very useful tool in building federations if the software can provide an array of communication services, data transformation services, and management services. As the number of Middleware services increase over and above what is available from the RTI the details of placing data on the wire become of lesser concern to the application. This lets the application designer concentrate on the particulars of their application, making the application more robust, and not having to expend a great amount of effort to become part of a Networked environment.

The most common HLA Middleware including NAWTSD’s Simulation Middleware Object Classes (SMOC) incorporate a common API to provide familiar semantics similar to the DIS PDU structure to allow the application to send and receive data. This common API allows for reuse of the Middleware software across applications.

2. HLA Conversion
Three basic HLA conversion schemes have evolved for simulations requiring HLA compliance. The first scheme is the incorporation of HLA into the applications native code as just another communications class. The second scheme is the development and use of "Gateway" processor and software to convert existing data communication constructs to HLA. An example of the second scheme is a DIS to HLA Gateway for converting Distributed Interactive Simulation (DIS) Protocol Data Units (PDU’s) to native HLA Objects and Interactions. The third scheme is to reuse "Middleware" software, that adds HLA implementation code to the data communications code of the simulation to provide an "Internal Gateway" function on the simulation processor. In effect the Middleware becomes a layer between the HLA and the simulation. The following Table 1 lists the major advantages and disadvantages of each approach.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Processors</th>
<th>Cost</th>
<th>Conversion Time</th>
<th>Speed of operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>1</td>
<td>High</td>
<td>Longest</td>
<td>Fastest</td>
</tr>
<tr>
<td>Gateway</td>
<td>2</td>
<td>Low</td>
<td>Short (~0)</td>
<td>Slowest (multi-processor latency)</td>
</tr>
<tr>
<td>Middleware</td>
<td>1</td>
<td>Medium</td>
<td>Medium</td>
<td>Fastest</td>
</tr>
</tbody>
</table>

Table 1. HLA Conversion Approach

HLA Gateway packages include the Army Simulation, Training and Instrumentation Command’s Gateway and the Naval Air Warfare Center Training Systems Division’s Simulation Middleware Object Classes (SMOC) using the GATEWAY option.

Figure 1 HLA Conversion Approaches with Bridge example, shows the three conversion approaches and the location of the RTI components.

The Bridge approach to Federation development is only used if a common RTI, and/or Federation Object Model (FOM) are not used. In this case the Bridge becomes a federate to two Federations and acts like a translator of RTI’s or FOM’s. A Bridge approach to Federation development should only be used as a last resort due to the possible data latencies [3] introduced by adding a processor into the communications channel. The bridge computer has to receive data, and then store that data, followed by a retransmission of the data across the bridge to a second federation, all of which takes time. Addition federation complexities associated with Federation Bridges are also introduced when dealing with Time Management, Ownership Management, and Data Distribution Management (DDM) across two Federations. When considering Time Management across two Federations consideration of different Time Management schemes must be made. For Ownership Management an Object can be owned by only one Federate, but each Federation requires an owner. Thus the Bridge must become a pseudo owner for one federation and collect updates from the real owner in the second federation. For DDM the bridge must gather up all DDM routing spaces from both federations and combine the spaces in a manner that doesn’t affect federation performance. The issues raised here only touch the complexities of Bridge development and should be addressed before deciding on implementing the Bridge approach between Federations.
3. RTI Services

The RTI provides services to federates in a way that is close to how a operating system provides services to applications. These interfaces are arranged into the six basic RTI service groups given below:

- Federation Management
- Declaration Management
- Object Management
- Ownership Management
- Time Management
- Data Distribution Management.

Generally, a "Middleware" product provides the standard RTI services to an application either in a pass through method or as a hidden service. For the purposes of this paper I have defined a hidden service as, "A service which the Middleware provides using data supplied by the application in a manner that the application does not know about." For example; The Middleware software could read the applications data (Channel Number of a Radio Channel) as it is passed through to the RTI. Using this data the Middleware software can automatically call the RTI’s DDM service with Routing Space set up data. This type of service would provide to an application the benefits of DDM filtering (of Radio Channels) with no additional interface calls or set up requirements.

The "Middleware" product can provide additional services tailored for the application. An example of possible extra "Middleware" Services are illustrated in Figure 2 Gateway/Middleware HLA Services.

All of the 6 RTI services can be supported in a Middleware implementation. However, several design choices on the extent of implementation must be made. These choices are necessary as not all the
functions of each service are likely to be required across a federation.

Gateway/Middleware HLA Services

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Gateway/Middleware Services</strong></td>
</tr>
<tr>
<td>Federation Management</td>
</tr>
<tr>
<td>Object Management</td>
</tr>
<tr>
<td>Time Management</td>
</tr>
<tr>
<td><strong>Runtime Infrastructure (RTI)</strong></td>
</tr>
<tr>
<td>Federation Management</td>
</tr>
<tr>
<td>Object Management</td>
</tr>
<tr>
<td>Time Management</td>
</tr>
</tbody>
</table>

Figure 2 Gateway/Middleware HLA Services

4. Possible extra Middleware Services

This paper is proposing that a set of additional Services can be added to Middleware software for the application. These additional Services are above the HLA API layer and can alleviate some of the cost required by a Federate to become HLA compliant. In addition, several RTI services, such as DDM and Time management, can be implemented in the Middleware application in such a way as to hide the HLA service and gain the benefits of using the service with low implementation costs. For discussions in this paper, the Middleware services reapplying RTI services in a hidden manner to the application are called Middleware RTI hidden service functions, and any additional functions are called Middleware extra service functions.

4.1 Middleware RTI hidden service functions

4.1.1 Data Distribution Management (DDM) pass through service

Data Distribution Management (DDM) Services are defined in chapter 9 of the HLA Interface Specification, Version 1.3 [4], and implemented in the Run-Time Infrastructure (RTI) version 1.3. [5] The service is intended to reduce the network traffic by reducing, or eliminating, the transmission of irrelevant object attribute updates and interaction parameters by filtering on selected data values. To use DDM, a routing space is defined using one or more common variables in each SOM and in the
FOM. Publication and subscription regions are then defined in the routing space. Each region is defined as one or more "extents" of the routing space dimension ranges. Figure 3 shows how, when the publication region of Federate A overlaps the subscription region of Federate B, transmission of object attribute updates and interaction parameters occurs. The publication region of Federate C is not in the subscription regions of Federates A or B, therefore updates and interactions are not transmitted from C to A or B until the Federates move and region overlap occurs. The regions can be static, set at simulation initiation, or they can be dynamic, changing with the federate values of the federation routing space variables.

The RTI implements DDM using network "Multicast" groups based upon routing space dimension value overlap, or non-overlap, for each publication and subscription region [6]. For dynamic regions, too frequent updates can cause the RTI to spend too much time on multicast group updates. [7] Therefore, the frequency of DDM region updates is a federation execution planning parameter.

![Figure 3. DDM Routing Space and Regions](image)
Implementing data filtering on behalf of a client as a hidden service function, requires that the routing space variables used to filter data be compared for the objects and interactions of interest by the Middleware software. These routing space variables are specified in the FOM, or some equivalent, for non–HLA channels, like DIS.

Each Federate must have its regions and their extents defined. Some data is defined as part of the Federation Development Process (FEDEP) and applied to all Federates. Other data are part of each Federate. The cases used to date have been "Static," defined at scenario initialization, and "Dynamic", defined by data at run time. [8]

For the static example, a set of HLA DDM services can be provided by middleware software by using a *.dat file. The Middleware software utilizes this file at run time to initialize the DDM services. By using a static *.dat initialization routine, the Middleware software allows the federate to utilize the filtering provided by HLA DDM service and does not add any code to the federates application. This implementation provides DDM as a new HLA service with no impact on existing federates.

For the dynamic example, a set of HLA DDM services is provided by Middleware software which allows the federate developer to configure the HLA DDM interface in an integrated fashion using a single *.dat file along with simulation controlled data. In this case, the *.dat file is utilized by the Middleware software at run time to initialize the DDM services along with the interface to the simulation to gather data used by the Middleware software to provide filtering. By using a simulation provided data along with the *.dat setup data, the Middleware software allows the federate to utilize dynamic filtering provided by HLA DDM and does not add any code to the federates application. This implementation provides a dynamic DDM implementation as a new HLA service with no impact on existing federates.

### 4.1.2 Time Management pass through services

Time Management (DDM) Services are defined in chapter 6 of the HLA Interface Specification, Version 1.3 [4], and implemented in the Run–Time Infrastructure (RTI) version 1.3 . [5] The service is intended to provide Time Management for the systems being modeled. The HLA Time Management service accommodates a variety of time management policies that must be coordinated with events being delivered to a federate.

In a federation, time always moves forward. For each federate in a federation, the perception of current time may be different. Time management is concerned with the mechanisms for controlling the advancement of each federate along the federation time axis.

A "constrained" federate is a federate that is constrained from progress along the time axis based on the progress of another federate. A "regulating" federate is a federate that regulates the progress, in time of federates that are designated as constrained. In general, there exists "regulating," "constrained," "regulating and constrained," or "neither regulating nor constrained" federates. A federation may be comprised of federates with any combination of time management models.

A real time federation has a simple time management model, namely time must pace the wall clock. Most Real Time federates use a time slicing approach where the federate determines an update frequency and strictly adheres to it. In this approach, if the simulation cannot finish processing in the current slice, it simply drops processing and moves on to the next slice. When a frame is dropped, incoming traffic is not processed and corresponding outgoing traffic is not generated. If the simulation is dropping frames on a regular basis then the simulation is not keeping up with real time.

The challenge for "Middleware" software is to bridge between federates, or federations, running with a different time model. This is a complex area for discussions and the right solution is not known for all real time federations. Middleware software needs to be developed to address this complex issue and allow for regulated, or constrained, federations to run with real time federations or federates.

### 4.2 Middleware extra service functions

#### 4.2.1 Agent

An Agent class service can be provided by the Middleware software layer to provide a fix for anomalies encountered by distributed systems.
An example, Agent class described below, of one such anomaly, was coded into the NAWCTSD SMOC software based upon experiences during recent exercises.

SMOC Agent class – If a federate abruptly leaves the federation without correctly resigning, the state of the federate becomes undefined. Objects published by such federates become zombied and thereby ineligible for acquisition by other federates. This example Agent class was developed for the NAWCTSD’s Simulation Object Middleware Classes (SMOC) software to address this issue.

What causes this problem? When a new federate is connected to a Simulation Object Middleware Classes (SMOC) gateway, the federate will always appear to be alive, even if the federate has not resigned correctly. This can happen when a federate loses its network connection, has a power loss or by not following the federates exit procedure. To solve this problem, a new class was added to the SMOC gateway, the Agent class. The agent class monitors all HLA entities and checks to see if they are still active. If they are not alive, then the entities are removed from the SMOC entity table and not updated for the DIS output channel or for an attached application.

For Middleware software, this example of an Agent class shows what additional functionally might be added to hide the RTI and any network problems from a federate. In this case, a network problem shows up as an Object with an unlimited life span. A real time federation could observe these objects as missiles that fly on forever. A simple Agent class was used to correct the network problem by coding a new Middleware service. This service has no impact on the existing federate except for the removal of unwanted objects.

4.2.2 Dead Reckoning

If a federate requires data smoothing of entity position data points, then Dead Reckoning can be applied at the Middleware software layer. In the NAWCTSD SMOC software, a Dead Reckoning algorithm is applied to each entity contained in its entity table. Common Dead Reckoning algorithms in the commercial literature [9] are highly accurate and involve calculating changes to body coordinates, velocities, accelerations, as well as equations to eliminate effects of centripetal acceleration.

Middleware software, by applying Dead Reckoning to entity positions, velocities, and accelerations, can make an attempt to reduce the network lag and effects of distance between federates. By adding a Global Positioning System (GPS) time stamp to each sent objects coordinates, a very accurate estimate of each federates objects can be maintained.

The net effect of adding Dead Reckoning is to offload the federate from having to perform this important task. The greatest impact will be on Federates with some sort of Viewing System, that tends to jump when no data smoothing is applied.

4.2.3 Coordinate conversion

If a federate requires Coordinate Conversion of entity position data points, this function can be applied at the Middleware software layer. This function corrects the classic problem of how different coordinate based Federates can interoperate. In the SMOC example there are several coordinate conversion function groups as shown below:

- DISs Geocentric and Topocentric coordinate conversion
- Topocentric to DISs Geocentric coordinate system
- Geodetic coordinate latitude, longitude, and height to cartesian geocentric coordinates
- Geocentric coordinates (X,Y,Z) in meters into geodetic coordinates given in latitude, longitude, and height
- Geodetic coordinate given in degrees, minutes and seconds into radians
- Radians into geodetic coordinates (degrees, minutes and seconds)
- Universal Transverse Mercator (UTM) location x,y,z (meters) into Geodetic World Geodetic System (WGS) 84 coordinates
- WGS84 into UTM
- Universal Transverse Mercator (UTM) coordinates to geocentric coordinates
- Geocentric coordinates into Universal Transverse Mercator (UTM) coordinates

For Middleware software, this example of Coordinate Conversion Functions shows how may possible transformations a federation might have to deal with. To a federate requiring
positional interoperability, the sizable set of conversion routines listed above may solve the coordinate conversion problem will little or no impact on code.

4.2.4 Filtering

If a federate requires data filtering of entities or interactions, then this function can be applied at the Middleware software layer. An example of this Middleware function is the filtering of objects to match the maximum number a Visual system can process. A number of delivered visual systems have a limitation of less than 10 objects. Such a restriction requires either the application to pick and choose between the objects presented by a network or have the Middleware layer provide this filtering. By having a Middleware layer provide filtering, the impact on the application software is minimized.

4.2.5 Time Stamping

If a federate requires an accurate time stamp for Federation compliance associated with entity position data points, then this function can be applied at the Middleware software layer. By applying the time stamping during data transfer through the Middleware software layer, the impact on the application software is minimized.

4.2.6 Entity Storage

If a federate requires entity storage of entity parameters, then this function can be applied at the Middleware software layer. An example of entity storage can be found in the SMOC software developed at NAWCTSD. As an entity is discovered and data gathered about the entity, an object table is created. The SMOC object table is then updated at a real time frequency using data gathered off the network or through use of Dead Reckoning algorithms. By having an object table, the Middleware code can provide both the application layer with a constant "smoothed" object data and any output channels (Like a DIS Channel) with updates. Use of entity storage in the Middleware layer provides for a flexible architecture that is able to service an application layer and act as a Gateway, or Bridge, between different networked systems.

5. Conclusion

The building of a Middleware software layer can be used to hide the RTI from an application. By building a set of additional Services into the Middleware software layer for the application, additional functionality can be provided. These additional Services are above the HLA API layer and can alleviate some of the cost required by a Federate to become HLA compliant. In addition, several RTI services such as DDM and Time management, can be implemented in the Middleware application in such a way as to hide the HLA service and gain the benefits of using the service with low implementation costs.

6. References


7. Author Biographies

DANIEL J. PATERSON is an electronics engineer with the Naval Air Warfare Center, Training Systems Division, Simulation & Models Division, Modeling & Simulation Development Branch in Orlando Florida. He is currently the technical leader of the NAWCTSD Interoperability Laboratory. In 1992 he received his MSEE degree from the University of Central Florida. In 1976 he received his BA in Mathematics from Jacksonville University, Jacksonville Florida.

ERIK S. HOUGLAND, Ph.D. is an electronics engineer with the Naval Air Warfare Center Training Systems Division, Battle Force Simulation Branch. His work interests include the implementation of HLA compliance in Navy training systems.

JUAN J. SANMIGUEL is an electronics engineer with the Naval Air Warfare Center Training Systems Division, Air Antisubmarine Warfare/Patrol Branch. His work interests include the implementation of HLA compliance in Navy training systems.