An Evaluation of the High Level Architecture (HLA) as a Framework for NASA Modeling and Simulation

Michael R. Reid[1]
Computer Sciences Corporation, Lanham, Maryland 20706 USA

Abstract

The High Level Architecture (HLA) is a current U.S. Department of Defense and an industry (IEEE-1516) standard architecture for modeling and simulations. It provides a framework and set of functional rules and common interfaces for integrating separate and disparate simulators into a larger simulation. The goal of the HLA is to reduce software costs by facilitating the reuse of simulation components and by providing a runtime infrastructure to manage the simulations. In order to evaluate the applicability of the HLA as a technology for NASA space mission simulations, a Simulations Group at GSFC conducted a study of the HLA and developed a simple prototype HLA-compliant space mission simulator. This paper summarizes the prototyping effort and discusses the potential usefulness of the HLA in the design and planning of future NASA space missions with a focus on risk mitigation and cost reduction.[2]

1 Introduction

Modern simulation systems often reside on networks of computer systems with numerous individual simulators concurrently running on separate and disparate computing platforms. The concept of distributed and remote component-based computing lies at the core of the High Level Architecture (HLA). Its main function is to integrate and facilitate the interoperability of numerous different simulators running collaboratively on a variety of platforms and to provide advanced time and data management for the overall simulations.

The U.S. Department of Defense (DoD) Defense Modeling and Simulation Office (DMSO)[3] sponsored the development of the HLA as part of an effort to control the costs and increase the capabilities of DoD simulations. The Institute of Electrical and Electronics Engineers (IEEE)[4] has approved the HLA as a standard as well (IEEE-1516). The HLA was originally designed to fit the extensive simulation needs of the DoD and allied defense organizations, but it is by no means limited to military applications. It is a suitable core technology for diverse applications ranging from games designed purely for amusement to the serious simulation and modeling of complex and sophisticated processes and workflows.

Although the missions of the National Aeronautics and Space Administration (NASA) and the DoD are fundamentally different, their technological needs often overlap. In many ways, NASA's simulation needs mirror those of the DoD. Like the DoD, NASA operates complex operational systems and relies on simulators to reduce both risk and cost and will likely do so to an even greater extent in the future. Several recent mission failures, schedule delays, and cost overruns underscore the need for the high-fidelity simulation of missions prior to their launch. The HLA is best suited for game-like simulations in which numerous "actors" come and go and interact and other types of simulations that involve a number of interacting parts. In the NASA realm, future large simulation systems that model the interplay of multiple spacecraft, ground systems, natural objects, and natural phenomena are potential candidates for an HLA-based design.

In order to evaluate the suitability of the HLA as a technology for NASA simulations, the CSOC[5] Simulations Group[6] at NASA Goddard Space Flight Center (GSFC) conducted a study of the HLA and

[1] Correspondence: Mike.Reid@gsfc.nasa.gov.
[2] All opinions expressed in this paper are solely those of the author.
[3] See http://hla.dmso.mil.
[4] See http://standards.ieee.org.
[5] NASA Consolidated Space Operations Contract. See http://www.csoconline.com.
[6] See http://cmex.gsfc.nasa.gov.

developed a simple prototype HLA-based space mission simulator. This paper describes that study and the prototype.

2 The Purpose of the HLA

The HLA provides a standard that will hopefully reduce the cost and development time of simulation systems and increase their capabilities by facilitating the reuse and interoperability of component simulators. By adhering to a common standard, simulators need not be limited to the application for which they were specifically designed and will be available for use in other simulation systems. Although the DoD sponsored the development of the HLA, this technology is now available to the wider simulations and modeling community.

3 What the HLA Is and Is Not

The HLA is a standard architecture upon which one may design and integrate application-specific simulators. Kuhl *et al.* [1] succinctly describe it as "...the glue that allows you to combine computer simulations into a larger simulation." The HLA consists of a set of rules to which all compliant applications must adhere, a common model for sharing data and documenting shared data, and an application program interface (API) for a *Runtime Infrastructure* (RTI) software system that integrates the individual simulators. The RTI provides, among other things, a communications and data sharing protocol, mechanisms for managing time and data, and a method of synchronizing the interplaying simulators. The effort required to integrate HLA-compliant simulators into larger simulation systems is far less than would be required to integrate similar simulators that were not designed in accordance with a common architecture and interface.

It is important to emphasize that the HLA is an architecture for simulations and modeling applications and not in itself, a simulator or a modeling tool. It is not a rapid application development system for simulations and it does not provide any data display capabilities or user interfaces. It is also not a software package, although the interface specification of the RTI software is part of the larger HLA specification.

4 Nomenclature

Like many technologies, the HLA has its own terminology. Some of the most important terms are given and explained below. See IEEE [3] for a more complete list of terms:

Federate: An individual simulator application or executable component. These are the independent simulators, which the HLA integrates together into a larger collaborative simulation.

Federation: A simulation composed of two or more (often many more) federates integrated together.

Federation Execution: A session in which a federation is running, usually as a distributed system.

Federation Object Model (FOM): The common object model that describes the data shared between federates within the federation. See section 5.2 for a discussion of the FOM.

Simulation Object Model (SOM): The object model that describes the data that an individual federate shares with the federation. It also contains some other interfacing information about the federate. In some ways, the FOM is a subset of the collection of SOMs defined for the various federates in the federation. See section 5.3 for a discussion of the SOM.

Object: In a conceptual sense, an HLA object is an entity that the simulation models. HLA objects represent "actors" that play in the simulation. In a literal sense, an object is a container for shared data that are created by a federate during the federation execution and persist for the duration of the federation execution or until deliberately destroyed. The FOM defines all classes of object and any federate that wishes to publish or subscribe to an object must also compatibly define that object in its SOM. HLA objects store their data in *attributes*.

Interaction: An HLA interaction is essentially a broadcast message that any federate playing within the federation execution can send or receive. A publishing federate sends out an interaction and subscribing federates receive it (or miss it). If no subscribing federate receives the interaction, the data it carries are lost. The FOM defines all classes of interaction and any federate that wishes to publish or subscribe to an interaction must also compatibly define that interaction in its SOM. Interactions carry their data in *parameters*.

Runtime Infrastructure (RTI): The software that implements the HLA interface specification and runs the federation execution (i.e., the overall simulation). See section 5.4 for more on the RTI.

Object Model Template (OMT): The standard template used for defining the form, type, and structure of the data shared within the federation and for some other interfacing information. All FOMs and SOMs are documented in accordance with the OMT. The OMT defines two formats for describing data and interfacing protocols: a tabular form and a Data Interchange Format (DIF). The OMT tabular format consists of a set of tables that humans can easily read. OMT DIF is a text format that one uses for the Federation Execution Data (FED) file. The RTI software reads the FED file to learn about the FOM. Supporting tools may exchange information using DIF as well.

5 An Overview of the HLA

A brief overview of the HLA is given here. See Kuhl *et al.* [1] for a far more complete and detailed description of this technology. The HLA consists of three basic components: the federation rules, the RTI interface specification, and the Object Model Template (OMT) [2]. The IEEE-1516 [3] specification for the HLA defines the set of rules that govern the design of compliant federations and individual federates. These deal mainly with data ownership and exchange. It also defines the interface to the RTI [4] and the OMT [5].

For sharing data among applications within a simulation, the HLA specifies an object-oriented variation on the basic *publish and subscribe* paradigm. Persistent data are stored within the attributes of HLA *objects*. HLA objects are similar to objects in an object-oriented programming language in that they contain attributes and are based on classes from which one may derive other classes with full attribute inheritance. Unlike objects in object-oriented programming languages, HLA objects do not contain member functions. Applications that provide data *publish* the relevant attributes of the appropriate objects and update them. Applications that receive data *subscribe* to those attributes and read them. Ephemeral data are distributed through HLA *interactions*, essentially broadcast messages. These too are based on classes from which one may derive other classes. Data producing applications send interactions on a one-time basis and data consuming applications receive them. Applications can be, and commonly are, both publishers and subscribers of data.

5.1 The Role of Objects

Objects form the core of an HLA-based simulation. HLA objects represent the "actors" in a simulation and they have state. Federates can create any number of instances of the objects defined in the FOM and they can change their state. Therefore, actors can enter the simulation and leave as needed. The federates drive the objects and the interplay among objects forms the simulation. The RTI provides the federates with the tools to create, destroy, sense, and manipulate the objects.

5.2 The Federation Object Model (FOM)

The Federation Object Model (FOM) describes the "universe." That is, the universe that the federation models. It is an abstract, but fundamental component of the HLA. The RTI and the other federates need a precise description of the type and form of every object and interaction that they share. The FOM specifies every class of object and interaction known to the federation and provides a complete list of every attribute contained in every object and every parameter contained in every interaction. It also precisely defines the forms and data types of all attributes and parameters. This information is documented in accordance with the OMT.

5.3 The Simulation Object Model (SOM)

The Simulation Object Model (SOM) describes the data that a particular federate shares with the federation. It can also contain some information related to how it interfaces with the RTI. The SOM is specific to the given simulator application and every federate must have a defined SOM that is documented in accordance with the OMT. A federation's FOM is composed of parts of the SOMs of all of its participating federates.

5.4 Services Provide by the RTI

Federates communicate with the RTI through the *RTIambassador* and the *FederateAmbassador*. These are classes (in the programming language sense) supplied as part of the RTI software. The RTIambassador contains a large set of public member functions that an application calls in order to request services from the RTI. The FederateAmbassador is a base class that contains virtual public member functions. The application developer derives a local class from the FederateAmbassador base class and implements the virtual public member functions to serve as callbacks. When the federate joins a federation execution, it passes an instance of an object of the locally derived FederateAmbassador class to the RTI. During the federation execution, the RTI calls the public member functions in the FederateAmbassador object whenever it needs to communicate with the federate.

It is left to third parties to actually implement and supply the RTI software in accordance with the HLA specification. DMSO has a solid and full-featured RTI implementation that is available on a variety of computing platforms. They currently provide their RTI software to external users free of charge, but impose some restrictions on its distribution. We used the DMSO RTI in our prototype. Hopefully in the near future, commercial vendors will offer users more choices in RTI implementations. As specified by the IEEE-1516 standard [4], the RTI provides six categories of services to federates. These are briefly outlined below.

5.4.1 Federation Management

Through the set of functions and callbacks categorized into its Federation Management service, the RTI facilitates the creation of federations, the joining and resigning of federates, federate synchronization, and the overall management of the federation.

5.4.2 Declaration Management

The Declaration Management service handles the publication and subscription of classes of HLA objects and interactions. In order to access any of the shared data as defined in the FOM, a federate must make use of this service as provided by the RTI.

5.4.3 Ownership Management

Through the Ownership Management service, the RTI controls the ownership of HLA objects and their attributes. Only the owning federate may update the attributes of a given object instance. However, the RTI can transfer the ownership of objects and attributes between federates and different federates can own and update different attributes within the same object instance.

5.4.4 Object Management

Object Management is the RTI service that facilitates the sharing of data within a federation. The RTI functions grouped into this service category allow calling federates to update and read HLA object attributes and to send and receive interactions. This service also provides a means of informing federates of the registration and deletion of object instances from the federation execution.

5.4.5 Time Management

Through its Time Management service, the RTI controls when federates can advance their positions on the federation's time-line. This controls when they receive time-stamped events, such as object attribute updates, interactions, or notifications of other changes to the state of the federation. The RTI can manage either clock-driven (time-step) simulators or event-driven simulators.

5.4.6 Data Distribution Management (DDM)

The functions and callbacks in the DDM service allow federates to apply routing spaces that associate HLA objects with regions of interest. A federate can subscribe to classes of events, but direct the RTI to deliver only those specific event instances that it is interested in. Through the DDM services, one can control the scope of visibility of HLA objects, interactions, and other events. Our prototype did not test or make use of this RTI service.

6 Objective of the Study

The purpose of this study was to determine if the HLA is a potential core technology for use in NASA simulations. The Simulations Group and its customer at GSFC are interested in modeling overall space missions. This involves numerous separate applications simulating the complex and multifarious aspects of a space mission. Examples of these aspects are spacecraft, on-board instruments, telemetry, tracking stations, ground systems, flight dynamics, and in the case of science missions, the natural objects and phenomena under study. Many future missions will involve constellations and formations of spacecraft working together. The collaborative interactions of multiple spacecraft will add a new and complex, but exciting, dynamic to missions. Modeling them early in their development cycle will be crucial to their ultimate successes. The HLA is possibly one of the technologies that could make these sophisticated new simulation systems realities. The HLA study and the development of the prototype space mission simulator were performed during the Fall 1999 through Summer 2000 time period.

7 Methodology

We decided that the best way to learn about the HLA and to evaluate its suitability to our problem domain was to design and build a prototype simulation system based on this technology. The prototype would consist of several fully HLA-compliant simulators "playing" together as federates within a federation. The emphasis would be on studying the HLA as a technology with a view towards space missions. Creating a simulator that would be of use in a real mission was not a goal of this effort.

7.1 The Prototype Simulator

The prototype space mission simulator very simplistically models a rudimentary constellation of two Earth-orbiting spacecraft collecting science data. A simulated "tracking station" monitors the positions of the two spacecraft in simulation time and a simulated "Earth" generates science data for their on-board instruments to collect. In its present form, this simulator does nothing that would be of use to any real mission or application, but it does illustrate the overall concept of how a useable space mission simulator might work and it demonstrates the role that the HLA technology could play in such a system.

The prototype space mission simulator consists of four small, specialized simulators: a Spacecraft Controller, an Orbit Calculator, an Earth simulator, and a Tracking Station. These four simulators are all fully HLA-compliant and are completely separate applications. They run concurrently and separately, but not independently of each other, in individual processes and optionally on separate machines. The four simulators in the prototype are federates and they communicate exclusively through the RTI to make up an HLA federation. As is required for HLA-compliance, the four federates each define the data they share with the rest of the simulation in a SOM documented according to the OMT specifications. Collectively, the SOMs of the four federates make up the federation's FOM. This is also documented in accordance with the OMT.

7.1.1 The Federates

The simulation models two spacecraft in Earth orbit. The FOM defines a spacecraft class of object and the Spacecraft Controller federate creates two instances of the class and names them "Landsat 7" and "Terra." The Orbit Calculator federate, running as a completely separate application and typically on a separate machine, acquires ownership of those attributes of the two objects, which store the current positions of the two "spacecraft." These it updates with each advance of its clock using positions computed earlier from the actual orbital elements of the real Landsat 7 and Terra spacecraft (The relationship between the two simulated spacecraft and the real spacecraft of the same names begins and ends with the orbital parameters). The Tracking Station federate, also running as a completely separate application, reads the positional attributes of both "spacecraft" and plots their paths on a simple screen display as a function of simulation time. The Earth simulator generates realistic Earth magnetic field data. It acquires the attributes that represent an imagined on-board magnetometer science instrument and updates them as the spacecraft pass over points on the Earth. The result is a rudimentary simulation of a ground station tracking a constellation of two Earth-orbiting satellites collecting magnetic field science data.

7.1.2 Time Management

The prototype space mission simulation is clock-driven. The federates use the RTI's Time Management service to stay together. Each federate maintains its own internal clock and asks permission from the RTI before advancing forward in simulation time. All of the federates, except for the Tracking Station, are both *time regulating* and *time constrained.* This means that each federate can both pace the federation execution and be paced by it. The Tracking Station is time constrained only. This means that it cannot pace the federation execution, but the federation execution can pace it. The RTI assures that time constrained federates do not advance their internal clocks until all of the time regulating federates have caught up with them. And equally, it assures that the federation execution does not run ahead of any time regulating federates.

In the case of the prototype, the Orbit Calculator federate sends out an interaction containing the simulation start time and time increment at the beginning of the federation execution. The other federates subscribe to this interaction and set their internal clocks accordingly. The Orbit Calculator also sends out an interaction informing the other federates of the end of the simulation.

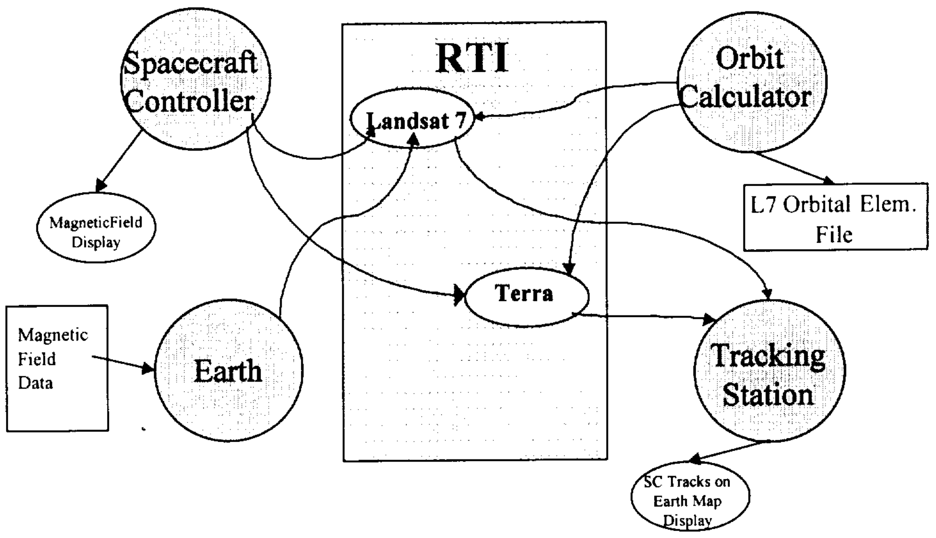

Figure 1 A diagram of the data flow within the prototype Space Mission Simulator.

What these prototype simulators actually do is not particularly important, but how they do it is. Internally, they do little more than move data around, but they are fully HLA-compliant, run as federates participating in an HLA federation, and communicate exclusively through the RTI. Again, the purpose of the prototype was not to realistically model a space mission, but to study and evaluate the HLA and to get an idea of how a mission planner or engineer might eventually use the HLA to build a useful space mission simulator in the near future.

7.1.3 Platforms and Software Used

The prototype space mission simulator was developed on a Pentium II™[7]-based computer running the Linux™[8] operating system and it used the DMSO RTI, version 1.3NG. It is coded mainly in C++. Except

[7] Pentium II™ is a trademark of Intel® Corporation.
[8] Linux™ is a trademark of Linus Torvalds.

for the RTI, the system was built using entirely free open-source software. Linux proved to be an excellent platform for this application.

8 Reuse Potential

Reusability is a key feature of component-based architectures such as the HLA. At the federate level, HLA-compliant applications have great potential for reuse beyond the specific systems for which they were developed. In the development of the prototype, an effort was made to carry the potential for reusability beyond the level of executable components and down to the code level. This was done by developing, in C++, an abstract base class that contains all of the code that a federate needs in order to interface with the RTI and to comply with the HLA rules. Each of the simulators is implemented as a C++ class derived from the base federate class. The derived classes contain the code that does all of the application-specific simulation work. Likewise, abstract base classes were developed for the objects and interactions in the simulation. Each HLA object defined in the FOM is mirrored in C++ by creating a data-specific class derived from the base HLA object class. The C++ abstract base class contains the member functions needed to create, register, and delete the object and to publish, subscribe, acquire, and release its attributes. The attributes themselves and the functions that manipulate them are data-dependent and are left to application-specific classes that are derived from the base object class. The HLA interactions defined in the FOM follow a similar design paradigm. These base classes are all compiled into a library to which the specific applications dynamically bind at run-time. This design worked reasonably well.

Once the base classes were developed, the amount of code needed to implement all of the application-specific federates, objects, and interactions was only about 37% of the total. Table 1 gives the number of lines of code written for each specific application and the percentage of the total (i.e., common library plus application-specific code) for the application that that number represents. In other words, the percentage of custom code is the number of lines of custom code divided by the sum of that number and the amount of code in the common base class library, times one hundred percent. This value is the percentage of the total amount of code used to implement each individual application that had to be written specifically for that application. Conversely, subtracting the percentage of custom code from one hundred percent gives the percentage of code reused by the application. This yields an average of about 87% code reuse for the applications (i.e., the federates).

Table 1 The amount of code needed to implement the applications.

Application Name	Lines of C++ Code	Percentage new code[9]	Percentage Code reuse[10]
Spacecraft Controller	262	12.2	87.8
Orbit Calculator	303	13.9	86.1
Earth	262	12.2	87.8
Tracking Station	274	12.7	87.3
common base classes library	1,881	N/A	N/A
Total	2,982	36.9[11]	63.1[12]

One should not construe these figures to mean that all or even most simulations based on the HLA will achieve such high levels of code reuse, but the data do suggest that there is the potential for a significant amount of code reuse and a commensurate cost savings. This system contains only one federation and all of the components are similar and were written specifically for this project. Therefore, one should be cautious in using these numbers as a basis for estimating the potential for code reuse on other HLA-based projects. Our experience with the prototype provides no data on the potential for reuse possible at the level of the executable component or federate. As with any software statistics, it is important to consider these numbers within the context of the project for which they were collected.

[9] The percentage of code written specifically for the application (i.e., not including the common library).
[10] The percentage of code used to implement the application that resides in the common library.
[11] The number of lines of custom code divided by the total lines of code for the whole federation times 100%.
[12] The percentage of custom code subtracted from 100%.

9 Considerations

There are some facts that one should take into consideration before adopting the HLA as the core architecture for a system. The HLA comes with a fairly steep learning curve. The effort required of a system designer or programmer to become familiar enough with the technology to use it effectively is tantamount to that required to learn an entirely new programming language.

In order to reduce costs, it is often desirable to make use of commercial off-the-shelf (COTS) software components and to interface with proven legacy systems. Since most COTS products and legacy systems are not HLA-compliant and may never be, one would need to develop custom interfacing applications in order to integrate them into an HLA-based simulation system. Although quite doable in most cases, the tasks would not be trivial. The HLA facilitates the reuse of preexisting simulators, but not as "plug and play" modules. The system integrator must still reconcile the preexisting application's SOM with the new simulation system's FOM and this will nearly always require changes to one or the other. The project manager should take these factors into account when developing the project plan so that adequate time and resources are available to address them. Furthermore, it is important to keep in mind that although the HLA is now a standard and a stable technology, it is still a maturing one.

10 Conclusions

The HLA could find a role as a core technology in certain types of space mission simulations, including those involving multiple spacecraft flying in formation. It could also serve as an architecture for scientific applications that model natural systems. The HLA will not increase the fidelity of simulators, but its modular and distributed design will make it easier to build higher-fidelity simulation systems.

Although intended to make integrating disparate and remote simulators more practical, the HLA cannot make "plug and play" a reality. To advertise it as such is to oversell it. Even within the context of an HLA-compliant simulation, the persons responsible for the various simulators must still negotiate common data types in order to make their systems compatible. The HLA provides fully compliant simulators with the mechanisms for communicating, but the individual applications must still have specific knowledge of the data that they share.

The HLA is especially well suited as a basis for game-like simulations in which a little "universe" is being simulated with multiple actors coming and going and interacting. It also has potential as a framework for integrating numerous separate simulators into a larger distributed simulation system. The fact that HLA-based simulators are designed around data classes from which any number of instances may be created give them great potential for easy expandability. The HLA's inherent distributed nature should also allow for essentially unlimited scalability.

Future space missions will often involve constellations or formations of spacecraft working in unison. In order for simulators to model such missions, they will need to simulate multiple instances of spacecraft and ground stations and possibly multiple instances of natural objects in the space environment. They will also need to integrate different simulators together, usually as a distributed system running on multiple platforms. In the case of missions that rely on groups of spacecraft flying in formation, the simulators will need to simulate interactions between multiple spacecraft, ground stations, and the natural environment. The designers of such simulations should consider the HLA as a potential core technology upon which they could base their systems.

Future space mission simulations will likely require the services of commercial products to provide, among other things, mission and flight modeling and telemetry generation. These applications are probably not presently HLA-compliant and would require custom interfacing software in order to participate in HLA-based simulations. Such custom HLA interfacing software should be straightforward, but not trivial, to develop. Therefore, it will be important to allocate time and resources to them in the project plan.

The HLA is probably not an appropriate architecture for simulations that just generate data for some single, external system or for simulations that cannot define their problem space as a collection of interacting objects. In these cases, the HLA would just be an unneeded extra layer and would provide little or no additional value. The HLA would also not be a good architecture for simulators that must send data using a specific protocol, such as IP, or through a specific type of interface, such as a hardware bus. Bypassing the RTI for external communications violates one of the primary rules of the HLA and would largely defeat its purpose. This is not to say that such simulators could not serve as components of larger HLA-based simulations. They certainly could.

One can envision an HLA-based simulation system that supports a space mission from the earliest phases in its life cycle through launch and beyond. Mission planners and scientists start with one or two interacting simulators and use them in their studies and planning. As development progresses and the requirements become better defined, the simulation system grows. More and more applications are added to the simulation system and others are refined or replaced. It evolves into a sophisticated modeling tool that can try out "what if" scenarios and produce simulated data products. As the mission reaches the advanced stages in its development, the simulation system has expanded with it and takes on a system test and integration role by generating high-fidelity telemetry that is fed into the actual ground systems or even into the real spacecraft. After launch, the simulation system provides a platform to safely test out responses to anomalies and other deviations from the original mission plan. It also serves as a training platform for the operations personnel. By this phase, the system has become large and complex and it requires computing power well beyond the capabilities of any single computer. Since the HLA is component-based, the complexity is manageable. Since it fully supports distributed computing, one simply adds processors as needed; some of them at remote locations.

As a technology, the HLA is well suited for the types of simulations for which it was designed. It is solid, well designed, and should have a place within NASA and the wider simulations and modeling community.

Acknowledgements

The author wishes to acknowledge the important contributions made to this study and to the development of the prototype by his colleagues and teammates Dr. Neil A. Ottenstein and William G. Parlock of Computer Sciences Corporation. Janice E. Swope of Computer Sciences Corporation managed the project. This study was performed for NASA/GSFC in Greenbelt, Maryland under the Consolidated Space Operations Contract (CSOC).

References

[1] Kuhl, F., Weatherly, R., and Dahmann, J., *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*, Prentice-Hall PTR, Upper Saddle River, NJ, 1999.

[2] Defense Modeling and Simulation Office (DMSO), *High Level Architecture Run-Time Infrastructure, RTI 1.3-Next Generation Programmer's Guide*, October 1999. See http://hla.dmso.mil.

[3] IEEE, *IEEE P1516/D4 Draft Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules*, April 1999.

[4] IEEE, *IEEE P1516.1/D4 Draft Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface Specification*, April 1999.

[5] IEEE, *IEEE P1516.2/D4 Draft Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template (OMT)*, April 1999.