Developing an HLA Tutorial: Philosophy and Best Practices

Björn Möller, Pitch Technologies, Sweden
Steve Eriksson, Pitch Technologies, Sweden
Åsa Wihlborg, Pitch Technologies, Sweden

bjorn.moller@pitch.se
steve.eriksson@pitch.se
asa.wihlborg@pitch.se

Keywords:
HLA, Tutorial

ABSTRACT: *A standard, like HLA, needs to be exact, complete and unambiguous. This may not be optimal for a beginner wanting to learn HLA. An up-to-date HLA Tutorial document has now been developed. This paper summarizes some of the philosophy of the tutorial. Several best practices on how to use the standard are also covered in the tutorial.*

One of the philosophies of the tutorial is to describe how services from different service groups are used to solve common tasks instead of strictly describing the structure of the HLA standard. Another philosophy is to emphasize how concepts from the HLA Interface Specification and the HLA Object Model Template are used together.

The best practices covered include low-level aspects like optimal memory allocation, handling of HLA service exceptions as well as life-cycle management of shared objects. Selected architectural aspects are also covered, like the use of a federate architecture that separates HLA concerns from domain simulation concerns, federate testing strategies and a basic Federation Agreement sample.

The tutorial, together with C++ and Java sample code, is freely available to industry, academia and anyone interested in learning HLA.

1. Introduction

A publicly available HLA Tutorial [1] has been developed. This paper describes the tutorial and summarizes the philosophy of the tutorial, the best practices presented and finally some thoughts and insights gained during the development.

1.1 Standards versus Tutorials

A beginner wanting to learn HLA, or any standard, may initially be disappointed by the typical standards document. Every feature is described in the utmost detail with a highly specialized terminology. It may take quite a lot of reading to figure out which combination of services that are needed to solve a particular problem. However, the primary purpose of a standards document is to provide a complete, consistent and unambiguous specification, not to give an introduction for a beginner.

Teaching HLA is more similar to telling a story, introducing new concepts, step by step, as they are necessary to fill a particular purpose. Every service should be put into a context of how it is used. The user needs to understand the main principles and central parts of the standard. Additional details can be studied later on when needed.

1.2 Evolving standards and best practices

HLA was incepted in the mid 90's. HLA 1.3 [2] was released in 1998, HLA 1516-2000 [3] in 2000, the SISO DLC [4] standard in 2004 and HLA Evolved [5] in 2010.

There is very little introductory material aimed at developers. The original HLA book [6] was released in 2000 and is based on HLA 1.3. There is also an older programmers guide [7] that came with the DMSO RTI that is also based on HLA 1.3. There are some practical migration guides for migrating to HLA 1516-2000 [8] and HLA Evolved [9]. But in general there is a lack of a practical and up-to-date tutorial for HLA. This may not be a problem for the seasoned HLA developer, but it is a barrier to entry for new persons and organization that need to start using HLA, thus limiting the success of HLA.

Not only the standard itself but also best practices for its usage evolve over time, based on experiences from building federations. Many projects and organizations have reached consensus on how to best use the HLA architecture and services over time. This also needs to be reflected in a tutorial of 2012.

1.3 The HLA Tutorial – why and how

The HLA Tutorial was developed to promote the HLA standard and to make it easily accessible for industry, academia and anyone interesting in the subject. Over the past years there has also been a growing user base, in particular in the civilian domain, that has requested an easily available, up-to-date tutorial. Another issue is that there is currently no tutorial that focuses on how federation agreements, object modeling and HLA services play together.

The tutorial is intended to complement the standard. The tutorial document and the samples represent many man-months of work from experienced HLA instructors and developers. The tutorial is intended to be vendor neutral (although screen shots are generally taken from Pitch products). The source code should work with any HLA Evolved RTI.

The HLA Tutorial builds upon almost 15 years of experience from teaching HLA in courses and seminars in more than a dozen of countries in four continents.

2. Overview of the Tutorial

The tutorial is freely available as a PDF document that may be redistributed. There is also a bigger package, the "HLA Evolved Starter Kit", which contains sample FOMs and federates in C++ and Java as shown in Figure 1. The tutorial can be downloaded separately or as a part of the Starter Kit.

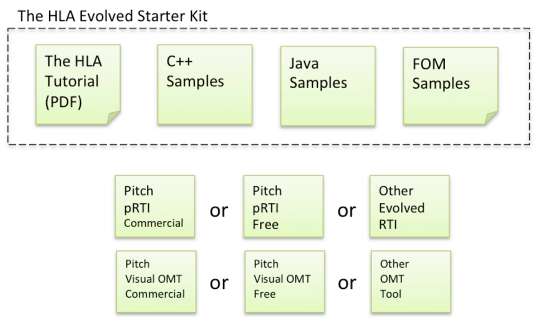

Figure 1: Components of the HLA Evolved Starter Kit and additional HLA software

These samples are intended to work with any HLA Evolved compliant RTI. In case the user does not already have an HLA Evolved RTI or an HLA OMT tool, free versions are available for download.

2.1 Structure of the tutorial

The tutorial consists of two parts but, to date, only the first part has been released. Part one focuses FOM development and the basic HLA services: federation, declaration and object management. Part two focuses on Ownership Management, Time Management, DDM, MOM and other more advanced concepts. Part one contains the following major sections:

Chapter 1 gives an introduction and describes the purpose of HLA (interoperability and reuse), a number of practical applications of HLA and a few words on policy, standardization and how standards can enable a market place.

Chapter 2-3 describes the architecture, topology and services of HLA, provides an overview of the architectural aspects of HLA and introduces the basic terminology without going into detail. It presents HLA as a modern, service oriented, architecture based on a service bus.

Chapter 4-9 provides a step-by-step description of how to build federates for a Fuel Economy Federation. The FOM development and the use of HLA services in the federate code are intertwined to illustrate how they play together. The PDF document contains simplified pseudo-code whereas the C++ and Java samples provide more details such as exception handling.

Chapter 10 describes basic and intermediate techniques for testing and debugging federates and federations.

Chapter 11 describes different types of Object Oriented HLA. It also provides a small amount of sample code.

Chapter 12 summarizes the tutorial, provides a short description of DSEEP and points at some additional HLA services that will be covered in part 2 of the tutorial, for example Time, Ownership and Data Distribution Management as well as the Management Object Model.

Appendix A and B provides a complete Federation Agreement and FOM for the Fuel Economy Federation.

Appendix C provides federate descriptions and file formats (scenario file and federate configuration file) for the Fuel Economy Federation.

Appendix D provides eight lab instructions for readers that have installed the sample federates on his computer. The instructions describes how to run the federates and what code to inspect for each chapter. The advanced user may also modify or extend these federates.

Appendix E describes the classical Restaurant federation and how to run it.

Appendix F gives an overview of the HLA Rules.

2.2 The Fuel Economy Federation

The main example used in the tutorial is the Fuel Economy Federation. Its purpose is to evaluate the fuel consumptions when multiple cars, simulated by different federates, drive along a particular route specified in a scenario file. The interoperability aspects are covered in detail, but the scenario format and fuel consumption models are intentionally simplified.

The federation has three types of federates. The **CarSim** simulates one or more cars. The **MapViewer** visualizes the cars on a map and in a list. The **Master** controls the federation. The federate types matches what is found in many real federations.

In addition to exchanging information about cars and fuel there are two patterns in the federation. The Master controls the scenario selection. Participating federates signal whether they were able to load the selected scenario or not. The Master also controls the start and stop of the scenario time, i.e. the simulation. The scenario is run at scaled real time, for example 5.5 or 1.0 times the real time. These patterns are described in the included Federation Agreement.

2.3 The Restaurant Federation

This federation is provided for the advanced reader. It models a sushi restaurant where chefs prepare sushi and places them on boats that transport them to customers. It was originally developed for HLA 1.3, using an older version of Java. It was originally included in the HLA book [6]. It has now been migrated to HLA Evolved. Some of the code does not always follow what we today consider best practice. This sample is included despite this since it uses a wide range of HLA services.

3. Philosophy

There are many ways to teach HLA. This section summarizes both lessons learned from giving a large number of courses as well as some design choices. Here are the most important philosophies.

A tutorial aimed at the practitioner

HLA and interoperability may well be described from a theoretical or architectural point of view. In this case we have decided to target practitioners that need to develop real HLA federates and federations. This requires more practical how-to advice, and reasonably complete code examples.

Simplify and focus on what´s useful

Many of the chapters have been shortened and simplified several times. More than half of the text of the first draft has been removed. This text was mainly advanced discussions. The text has been simplified as far as possible to make it easy to grasp. New concepts have been introduced only when they can be easily understood.

Present HLA as a modern software architecture

Ten years ago HLA was often presented primarily as a US DoD strategy. While HLA still has a strategic position in NATO and the US DoD, todays developers are more interested in its virtues as a modern software architecture, in particular if they work outside of the defense domain. There are today several related architectural styles that can help people understand HLA, for example Information Bus and Enterprise Service Bus.

Learn by common tasks

It may be tempting to structure an HLA tutorial according to the three standards documents and their chapters. One problem with this is that it takes a long time before the relationships between different parts of the standard starts making sense. The philosophy chosen is instead to show how to perform common tasks. Figure 2 shows an example of this. It illustrates which part of HLA that you need to use to send an interaction.

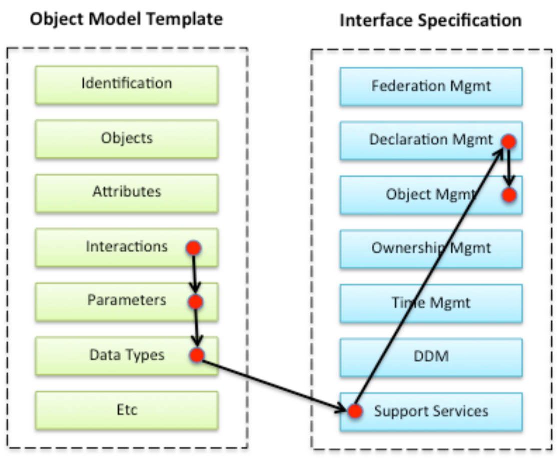

Figure 2: Parts of the HLA specification used to send an interaction

First you need to design an interaction with parameters in the OMT. You also need to define data types for any parameters. Then you need to use the services in the Interface Specification to get handles, publish and subscribe the interaction and finally send it. You may also consider using encoding helpers, which is not shown in figure 2.

The task-oriented approach clarifies how different parts of the standard are used together. It also provides more “instant gratification” to the developer.

Highlight federation agreements and object modeling

Many federate developers today underestimate the value of the FOM concept and the importance of quality FOMs. The approach described in Figure 2 also helps the developer understanding the relationship between the FOM and the federate code.

The FOM, on the other hand, only provides a lower level description of the overarching Federation Agreement, which gives the bigger picture of how to use the information in the FOM.

While other efforts, like FEAT [10] attempts to advance the state of the art with respect to more advanced federation agreement, this tutorial seeks to introduce newcomers to a very simplistic but correct and complete federation agreement.

Use simple pseudo-code in the main text

To clarify how the services are used, the main text of the tutorial uses simplified pseudo-code. The C++ and Java sample code, on the other hand, contains full detail and extensive exception handling. Since many beginners tend to reuse sample code it is important to avoid simplifying for example exception handling. Inferior exception handling often makes federation debugging more time-consuming than necessary.

Use simple english

It is likely that the majority of todays´ HLA users do not have English as their first language. Because of this the tutorial has been written in a simple and friendly tone.

4. Best Practices

HLA offers a large number of services. The HLA tutorial presents how to use them in a simple federation, step by step. In many cases there are several ways to approach a problem, to design a federate and to use the HLA services. The HLA tutorial tries to present a number of best practices that are on an appropriate level for beginners. This section presents some of them.

Note that more advanced or large federates and federation may need to deviate from some of these best practices. Still it is highly useful for a beginners to get their first federates well designed from a general perspective.

4.1 Architecture and design

These best practices apply to the overall architecture and design of federates and federations.

1. Carefully design your federation agreement and FOM before attempting to write any code. If the federation agreement already exists, study it carefully. Starting to design your federates without a proper federation agreement and an agreed upon FOM often leads to costly redesign of the federates. When designing other software it is often possible to “wing it” when problems occur, but since federates that you write need to be compatible with other federates it is not as easy in a federation. Not having a clear view of how everything is supposed to work when starting can lead to different behaviors in different federates depending on the implementers interpretations.
2. Use a coordinated approach for handling scenario time and scenario data across the federation.

   Even if your federation is not time managed the concept of scenario time will exist in the simulators. In many cases federates in a federation will execute in scaled real time. Think through how time will be handled and use a similar approach in all federates when possible. If one federates supports pausing but none of the other federates does, then that feature will not be useful in the federation. Running at a faster or slower pace, pausing and jumping in time, are some things that you should consider even for non time-managed federations.

   Using common scenario data removes the possibility that simulators disagree on the content of the simulation. It also minimizes scenario development work and minimizes the risk for uncorrelated scenario data.
3. Use a specialized federate for starting, stopping and selecting scenario. This follows the principle of separation of concerns. The simulators are experts in simulating a system or parts of a system, not coordinating simulators in a distributed environment. Centralized handling and coordination makes it easier to have clear and well-known states of the simulators in the federation execution.
4. Put the HLA interface code in a separate module. This is also a question of separation of concerns. In software development it is good practice to create modular and loosely coupled system. Encapsulate changeable design decisions. The different modules should have a specific purpose and be as independent as possible from other modules. This approach makes it easier to understand and develop the different modules. It also makes it easier to find faults in the system and changes are easier to make since they won’t affect the whole system. Another advantage of using a modular design is that simulation experts can develop the simulation part and

HLA experts, maybe external developers, can develop the HLA part.

5. Tailor the HLA interface module to the specific subscription needs of each federate for best performance. It is often a good idea to create general and reusable software components. However, when building an HLA module for a certain simulator and federation it might not be the best choice. The major problem is that it may subscribe to more information than needed in a particular federate, causing increased usage of CPU, memory and networking resources. It might also make your system more error prone, harder to understand and harder to maintain. Our recommendation is to optimize for the current environment. Modular design makes it relatively easy to change or switch the tailored HLA interface.

4.2 Program structure

These best practices apply to the use of HLA services within a federate.

1. Register objects without reserving HLA object instance names. Instead use an attribute for naming. HLA object instance names are globally unique. It's relatively costly to register a unique name in the federation. The central RTI component has to check for uniqueness and your application has to handle any error thrown by the RTI should the name already be taken. Instead, have the RTI create a unique name and use an attribute in the object for the name.

2. If you still do reserve names and you need to reserve more than a few HLA object names, use the Reserve Multiple Object Instance Names service.

3. Use a table or hash map for storing the references for discovered object instances. If possible consider a lookup function to quickly locate instances based on name, handle and other relevant keys. Consider having one table or hash map for each object class. One of the functions an HLA module is likely to execute very often is to translate between a simulation objects id and the corresponding HLA object.

4. Implement the Provide Attribute Value Update callback so that other federates, possibly late joiners, can get the mot recent values. Use this in conjunction with the Auto Provide switch, at least for smaller federations. Failing to implement this support can make it impossible for other federates to join the federation when it is already executing. Not supporting late joiners leads to problems even for federations with a fixed federate lineup. For example, if a federate crashes during an execution it cannot rejoin the federation. This might make it necessary to restart the whole federation every time a federate crashes.

4.3 Low level programming

These best practices apply to detailed programming aspects. In most cases they are independent of the programming language used.

1. Allocate memory for objects like encoding helpers in the initial part of the program, not in the main loop. Encoding helpers can be relatively expensive to create and they are likely to be used very often. For optimum performance you should therefore create them once during initialization.

2. Get handles from the RTI in the initial part of the program, not in the main loop.

3. Handle all HLA service exceptions. Instead of just terminating your application if it encounters an exception you should handle it so the application degrades gracefully or at least clean up before terminating. Clean up includes resigning and disconnecting from the federation and perhaps hand over ownership of simulation objects.

4. Be careful with exceptions that are related to any user input or the current technical environment (for example cannot find FOM file, bad IP address). Give the user an opportunity to fix these. Configuration problems related to federation name, FOM file to use and IP address of the RTI central component can easily be remedied by the user or a technician so make sure to give clear error messages.

5. Use encoding helpers to get correct encoding. Utilizing the provided encoder classes is also a good way to be sure that the data is encoded correctly for any operating system, CPU and development environment.

6. Assume that all data received may be incorrect or incorrectly encoded. As with all application development, don't make your code rely upon external components to provide you with correctly formatted data. Failing to do this may open up for security breaches and unnecessary termination of your application.

7. Several callbacks (like Reflect Attribute Values) have several different

implementations, with different parameter sets, in the API. Handle al versions, for example by dispatching them to one common implementation. In most cases it's a good practice to handle update of attributes in a uniform way. Having one implementation for each overloaded version of the callback method makes your code harder to maintain. The more code you write the risk of introducing bugs increase. Subtle differences in the overloaded methods may create hard to find bugs. More code often means more tests and maintenance.

4.4 Testing

These best practices apply to the testing of federates and federations.

1. Verify that all operations, like declarations and object registrations, work as intended by inspecting the RTI user interface.
2. Verify that your federate sends correctly encoded data before trying to use it in a federation. A well behaved federate should only send properly formatted data according to the FOM. Don't assume that the other federates can handle bad data otherwise you may make them terminate ungracefully.
3. Test your federates against known-good federates and recorded data. This is an excellent way to pinpoint where in a federation a problem occurs. For example if your simulated entities don't show up where they are expected in a viewer you can connect it to another proven viewer to determine if it's your federate that sends bad position data or the viewer that displays it incorrectly.

5. Discussion

As can be understood from the previous descriptions a lot of topics have been covered in the tutorial. In this discussion we would like to focus on some topics that we found more difficult to fit into the tutorial.

5.1 The SOM

While the SOM is in no way a difficult concept to explain, it was difficult to find an obvious place to describe how it fits into the federation development process in the tutorial. One approach used to present the SOM is the use of Publish/Subscribe Matrix. Figure 3 shows such a matrix. Each row represents an attribute or an interaction. Each column represents a federate. In each cell we then specify whether the federate publishes or subscribes to that attribute or interaction.

	Fed A	**Fed B**	**Fed C**
Car.Name	Pub	Pub/Sub	Sub
Car.Position	Pub	Pub/Sub	-
Start	Pub	Sub	-

Figure 3: A Publish/Subscribe Matrix

This type of matrix offers a good starting point for a discussion about SOMs.

5.2 The HLA Rules

In most training events we have found that the HLA Rules seem very abstract when presented early in an HLA training event. After a few days of HLA training most participants find them very easy to understand. Several rules are based more or less on common sense, saying that a federate shall hold what it promised to do. Other rules are less trivial and may require some additional discussion, for example that the RTI will only transmit, not store, any data values for attributes and interactions. Several of the rules relate to the SOM, which is not covered in detail in the tutorial. The approach in the the tutorial is to put the HLA Rules in an appendix.

5.3 Getting hold of the HLA specification

The HLA Tutorial makes extensive references to the HLA specification, which is not available for free. Some universities have purchased full access to all IEEE standards, which makes it easy for students and staff to get them. Other readers may be SISO members, which gives them full access to the standard at a modest price. Still a large number of readers will have difficulties getting the HLA specification for various reasons. Students in many parts of the world may consider the price high. Complicated administration may slow down a purchase for a reader in a large organization. Today many readers expect software standards to be freely downloadable from the Internet, which may create some disappointment.

5.4 Understanding commonly used FOMs

Most people learning HLA are interested in understanding not only HLA but also commonly used FOMs. For the defense sector this is usually the Real-time Platform Reference FOM (RPR FOM) [11]. This is a fairly advanced FOM with complex data types. One challenge is that such a FOM is too complicated to be used for explaining basic HLA concepts. It may actually require a tutorial on its own. Another challenge is that many new HLA users come from other domains than defense.

For the HLA tutorial we have chosen to use a very basic FOM that allows us to gradually introduce more and more advances concepts. It is possible that part two of the tutorial may contain overviews of some commonly used FOMs.

6. Conclusions

A freely available HLA Tutorial has been produced. The tutorial is aimed at the practitioner. A software package with sample federates as well as free RTI and OMT software is also available.

Among the most important features of this new tutorial is that it presents HLA as a modern, service-oriented architecture. Another feature is to present the design chain that starts with the Federation Agreement, continues to the FOM and finally uses the HLA services. A third feature is to teach HLA based on tasks, like sending an interaction, rather than on the structure of the standard.

A large number of best practices have also been incorporated in the tutorial. These range from design and architectural practices down to low-level programming and testing.

We believe that the tutorial will have a positive impact on the adoption of HLA over the coming years, not only in the defense domain, but also in civilian applications.

References

[1] "The HLA Tutorial", www.pitch.se, September 2012

[2] "High Level Architecture Version 1.3", DMSO, www.dmso.mil, April 1998

[3] IEEE: "IEEE 1516, High Level Architecture (HLA)", www.ieee.org, March 2001.

[4] SISO: "Dynamic Link Compatible HLA API Standard for the HLA Interface Specification" (IEEE 1516.1 Version), (SISO-STD-004.1-2004)

[5] IEEE: "IEEE 1516-2010, High Level Architecture (HLA)", www.ieee.org, August 2010.

[6] Frederick Kuhl, Richard Weatherly, Judith Dahmann: "Creating Computer Simulation Systems: an Introduction to the High-Level Architecture", Prentice Hall PTR (2000), ISBN 0130225118

[7] DMSO: "RTI 1-3-Next Generation Programmer's Guide Version 3.2", September 2000, US Departement of Defense: Defense Modeling and Simulation Office

[8] "Porting a C++ Federate from HLA 1.3 to HLA 1516" & "Porting a Java Federate from HLA 1.3 to HLA 1516", March 2003, http://www.pitch.se/support/pitch-prti-1516

[9] "Migrating a Federate from HLA 1.3 to HLA Evolved", November 2010, http://www.pitch.se/technology/about-hla-evolved

[10] "FEAT PDG - Federation Engineering Agreement Template", www.sisostds.org

[11] SISO: "Real-time Platform Reference Federation Object Model 2.0 ", SISO-STD-001 SISO, draft 17, www.sisostds.org

Author Biographies

BJÖRN MÖLLER is the Vice President and co-founder of Pitch Technologies. He leads the strategic development of Pitch HLA products. He serves on several HLA standards and working groups and has a wide international contact network in simulation interoperability. He has twenty years of experience in high-tech R&D companies, with an international profile in areas such as modeling and simulation, artificial intelligence and Web-based collaboration. Björn Möller holds an M.Sc. in Computer Science and Technology after studies at Linköping University, Sweden, and Imperial College, London. He is currently serving as the vice chairman of the SISO HLA Evolved Product Support Group and the chairman of the SISO Real-time Platform Reference FOM PDG.

STEVE ERIKSSON is a software developer at Pitch Technologies and has been involved in the development of several commercial HLA products. He received his BSc degree in Computer science from Linköping University in Sweden.

ÅSA WIHLBORG is a Systems Developer at Pitch Technologies and a major contributor to commercial HLA products such as Pitch Visual OMT 2.0. She studied computer science and technology at Linköping University, Sweden.