The Hitchhiker's Guide to Developing OneSAF HLA Interfaces

Jennifer Lewis, Kirk E. Kemmler and Khoi Do
Science Applications International Corporation
Orlando, FL
jennifer.e.lewis@saic.com, kirk.e.kemmler@saic.com, khoi.m.do@saic.com

ABSTRACT

Army Capabilities Integration Center (ARCIC) is integrating OneSAF into its Battle Lab Collaborative Simulation Environment (BLCSE), in an ongoing effort to integrate new technologies into war gaming experimentation. This integration requires modifications to OneSAF's High Level Architecture (HLA) interoperability module to enable OneSAF to interact in the experimental and ever-changing BLCSE federation. This paper introduces OneSAF HLA interface design concepts and provides detailed examples to allow the reader to perform similar development in his own simulation environment. The reader will learn about the OneSAF HLA interoperability architecture and structure, how to use OneSAF's Object Data Model (ODM) and Federation Object Model (FOM) variables, and how to implement converter classes when FOM changes are required. To illustrate these concepts, the paper uses case studies from the BLCSE integration effort, which has resulted in successful, record-run experiments using OneSAF in HLA mode. Specifically, the case studies describe how to implement a new FOM attribute to enable the OneSAF Plan View Display (PVD) to display whether an external individual combatant is armed, and how to implement a new FOM interaction to perform terrain damage effects.

ABOUT THE AUTHORS

Jennifer Lewis is a simulation engineer supporting ARCIC's Battle Lab Collaborative Simulation Environment. She holds a Master of Science degree in Computer Science with an emphasis in Telecommunications and Networking from the University of Texas at Dallas and is a Certified Modeling and Simulation Professional. She has designed and implemented network protocols for the telecommunications and defense industries for the past eight years.

Kirk E. Kemmler is a software engineer supporting ARCIC's Battle Lab Collaborative Simulation Environment. He holds a Bachelor of Science degree in Computer Engineering from the University of Central Florida. Since January 2000 he has participated in the design and development of Man-In-The-Loop, Virtual, and Constructive simulation programs.

Khoi Do is a simulation software engineer supporting ARCIC's Battle Lab Collaborative Simulation Environment. He holds a Bachelor of Science degree in Computer Science from the University of Central Florida. He has developed and integrated military constructive and virtual simulations for the past nine years.

The Hitchhiker's Guide to Developing OneSAF HLA Interfaces

Jennifer Lewis, Kirk E. Kemmler and Khoi Do
Science Applications International Corporation
Orlando, FL
jennifer.e.lewis@saic.com, kirk.e.kemmler@saic.com, khoi.m.do@saic.com

INTRODUCTION

Like many other simulation communities, Army Capabilities Integration Center (ARCIC) is integrating OneSAF into its Battle Lab Collaborative Simulation Environment (BLCSE). BLCSE is a large scale federation, operating primarily under the High Level Architecture (HLA). Because BLCSE uses a Federation Object Model (FOM) not inherently supported by OneSAF, ARCIC extended the OneSAF baseline to support its specific needs. The technical team performing this task resolved numerous integration issues. However, perhaps the most problematic integration issue encountered was a simple lack of expertise in OneSAF development. The intent of this paper is to augment the information provided by PM OneSAF's formal development training by documenting the knowledge of OneSAF architecture and development gained during ARCIC's integration effort. The paper will describe the processes and techniques the integration team used to successfully modify the OneSAF HLA interoperability module. The paper uses specific examples from BLCSE's integration efforts to illustrate these techniques. However, it is written to serve as a general developer's guide, a document the reader can use to perform similar development in his own simulation environment.

DEVELOPMENT ENVIRONMENT SET-UP

The standard OneSAF installation program does not configure the Windows or Linux operating systems for development. Included with the standard OneSAF delivery products are instructions, usually bundled with the source code archive, on how to configure both Windows and Linux operating systems for development. This step has to be accomplished separate from the OneSAF installation. Configuring OneSAF for development usually includes either installing the source code archive or [i]re-naming directories that get installed with the runtime installation, and configuring the operating system with environment variables. The major issues with operating system configuration for OneSAF development have been controlling the installation of C++ system libraries for both Windows (Cygwin) and Linux. These issues often result in C++ compilation errors in OneSAF Environment Runtime Component (ERC) source code. Depending on the exact development environment, simply installing gcc 3.3.2 is not sufficient to correct these problems.

A known issue with the OneSAF Cygwin installation is that it does not provide the correct g++ libraries to compile ERC C++ source code error free. On the Linux development side, the installation of gcc3.3.2 is needed on Red Hat Enterprise 4.x, Fedora Core 5.x (and older), and CENTOS 4.x (and older). For Red Hat 5.x, Fedora Cora 6.x, and CentOS 5.x, the installation of gcc3.3.2 is not needed and should not be done. As of version 1.5.1, OneSAF can be compiled using the native gcc4.1.x compiler that comes with these newer operating systems. To compile the OneSAF JAVA source code, Java JDK 1.5.x is needed. For development on Red Hat 5.x, Fedora Core 6.x and CentOS 5.x, it is recommended to use Java JDK 1.6.x. However, only OneSAF version 2.1(with Java 1.6 OneSAF patch) or newer versions are needed to be able to compile. The step-up to Java 1.6.x is needed to solve OneSAF multi-threaded compatibility issues with newer Linux kernels. Figure 1 summarizes the various operating systems and software needed for each development environment.

Runtime Infrastructure (RTI) Setup

The HLA RTI is an important part of configuring OneSAF to compile in the capability to run the HLA interoperability adapter. The BLCSE federation uses the RTI-1.3NGmatrexVx.x for interoperability testing of

[i] *With the release of OOS Version 2.0 the baseline directories have been standardized for both runtime and development environments.*

OneSAF. Initial HLA interoperability testing began with version 3.1 and has progressed to version 5.5 during the Omni Fusion 2008 experiment. The integration team identified several issues with the version 3 releases of the MATREX RTI, mainly related to the stability of the RTI when federates improperly left the federation or when a network connection was too heavily loaded. The team worked around these issues by maintaining strict control over when and how a federate joined and resigned from the federation. However, the team has found recent releases of the MATREX RTI to be robust and stable enough for use in the BLCSE federation, even under high network loads.

In order to compile OneSAF HLA interoperability source codes, the RTI and its Java binding files must be installed. Proper environment variables also must be set, specifically RTI_HOME, RTI_BUILD_TYPE, RTI_JAR_PATH, and RTI_JAR_NAME.

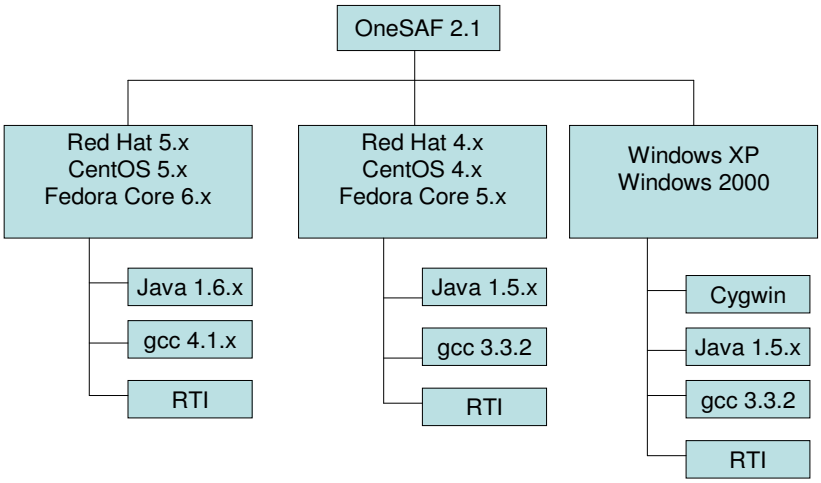

Figure 1. OneSAF Development Environment

Building OneSAF Source

Once the RTI is installed and environment variables are set, the user can perform either a top-down standard OneSAF build, per the OneSAF build instructions, or a more pointed HLA build. The HLA build can be done by building only the HLA interoperability directories in both the core and any external development directories being used that contain additional HLA core inheritances and extensions. For BLCSE purposes, the integration team developed additional object and interaction converters, which are located in the extension directory "SWR/src/net/onesaf/ext/blcse". It is recommended that all external OneSAF development be done under the extension directory. In order for OneSAF to recognize the new extension directory and be included in the top down standard build, the extension needs to be added to the "SWR/confdb.csv" file and build with the "-e <extension_name>" parameter. For example, "./build –l build.log –e blcse", where "blcse" is the BLCSE extension. The BLCSE extension provides specific functionality for the BLCSE FOM, which is a derivative of the MATREX FOM. The BLCSE federation is in continuous change and development. Therefore, the BLCSE FOM rarely coincides exactly with the latest released MATREX FOM. BLCSE engineers are continuously collaborating with MATREX to incorporate BLCSE FOM additions and changes back into the MATREX FOM through the MATREX FOM Management Group's change request process.

Configuring OneSAF's HLA

Configuring OneSAF to operate with an HLA interoperability adapter starts with the OneSAF property "net.onesaf.core.services.sim.interopmgt. interfaceConfig". Adding this property to the "onesaf.properties" file will enable OneSAF to read in all the interoperability configuration properties needed

for HLA or DIS, depending on how the configuration file is configured. The HLA interoperability interface configuration file details and parameterizes various OneSAF HLA settings such as the FOM to use, federation name, federate name, and Data Distribution Management settings. The file also lists the OneSAF HLA converter classes which are the primary elements to converting OneSAF interactions and objects to their FOM counterparts and vice versa. The various OneSAF interoperability configuration files are located under "PAIR/interop" directory. Figure 2 shows HLA federate settings, and Figure 3 shows the converter classes currently being used in the BLCSE interoperability file.

HLA INTEROPERABILITY ARCHITECTURE

OneSAF was developed to enable the interoperability adapter to communicate natively as either HLA or DIS. Architecturally, there is a clearly defined line drawn between OOS and the interoperability interfaces, whether it is HLA or DIS. It is best that this line is only crossed via the provided interfaces in the Object Data Model (ODM). The public ODM interfaces, found in net.onesaf.core.services.odm, provide visibility into the Runtime Data Model (RDM) objects and classes and give the developer nearly complete access to needed data for proper OneSAF synchronization with the federation.

Core and Extension Packages

The core OneSAF baseline provides MATREX compatible object and interaction classes needed for basic federate operation. Most of the basic data type conversion methods are provided allowing the developer to easily convert from and to the OneSAF ODM and FOM data types, eliminating the need for each extension package developer to re-create these converters. Not completely provided are the interfaces for the interactions, called converters. Often what the core interoperability baseline lacks are converters for specific or customized functionality. Using standard object-oriented principles and standards, it is relatively straightforward to extend from and use all of the existing OneSAF HLA architecture in order to create specifically tailored converters.

```
<?xml version="1.0" encoding="UTF-8" ?>

<InteropConfig refID="1" >
  <interfaceConfigurations>
    <HLAConfig refID="2">
      <name>MATREX HLA Adapter</name>
      <className>net.onesaf.core.services.sim.interopmgt.hla.HLAInterface</className>

      <federationName>BLCSE_HLA_Federation</federationName>
      <federateName>OOS_windows</federateName>
      <FOMName>matrex_fom_v4.0_blcse</FOMName>
      <autoJoin>false</autoJoin>
      <autoJoinSimState>RUNNING</autoJoinSimState>

      <siteID>8</siteID>
      <applicationID>5</applicationID>

      <timeRegulating>false</timeRegulating>
      <timeConstrained>false</timeConstrained>
      <timeLookahead>1</timeLookahead>

      <!-- 100 ms (default value) -->
      <RTITickDelay>100</RTITickDelay>
      <!-- 1 ms -->
      <RTIMinTickTime>1</RTIMinTickTime>
      <!-- 5 sec -->
      <RTIMaxTickTime>5000</RTIMaxTickTime>

      <!-- 200 ms (default value) -->
      <OOSTickDelay>200</OOSTickDelay>
```

Figure 2. BLCSE OneSAF HLA federate settings

```
  <converters>
    net.onesaf.ext.blcse.services.sim.interopmgt.hla.converters.matrex.blcse_v40.AircraftConverter:
    net.onesaf.ext.blcse.services.sim.interopmgt.hla.converters.matrex.blcse_v40.AssetConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.AmmunitionInventoryConverter:
    net.onesaf.ext.blcse.services.sim.interopmgt.hla.converters.matrex.blcse_v40.GroundPlatformConverter:
    net.onesaf.ext.blcse.services.sim.interopmgt.hla.converters.matrex.blcse_v40.IndividualCombatantConverter:
    net.onesaf.ext.blcse.services.sim.interopmgt.hla.converters.matrex.blcse_v40.MunitionConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.ScenarioDataConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.UniformAtmosphereConverter
  </converters>
  <converterData>
    defaultGroundPlatformComposition=entity/mr/COMBAT/ARMOR/Tank_M1A2_Abrams_Armor:
    defaultAircraftComposition=entity/lr/COMBAT/AVIATION/ROTARY_WING/RWA_AH64_ApacheD:
    defaultICComposition=entity/mr/COMBAT/INFANTRY/ICFullyLoaded_M240_Gunner:
    defaultMunitionComposition=entity/hr/flyoutMunitions/Hellfire_Radar_Guided:
    defaultUnitComposition=unit/external/generic_lr:
    platformEnum=PlatformTypeEDT:
    icEnum=IndividualCombatantEDT:
    munitionEnum=MunitionTypeEDT:
    tagGenerationMethod=AUTOGEN
  </converterData>
</ObjectHandlerConfig>
<InteractionHandlerConfig refID="4">
  <converters>
    net.onesaf.ext.blcse.services.sim.interopmgt.hla.converters.matrex.blcse_v40.CallForFireConverter:
    net.onesaf.ext.blcse.services.sim.interopmgt.hla.converters.matrex.blcse_v40.DamageReportConverter:
    net.onesaf.ext.blcse.services.sim.interopmgt.hla.converters.matrex.blcse_v40.WeaponFireConverter:
    net.onesaf.ext.blcse.services.sim.interopmgt.hla.converters.matrex.blcse_v40.FeatureDamageConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.AmmunitionInventoryConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.CIDFriendResponseConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.CIDInterrogateConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.FireMissionConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.FireMissionResponseConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.FireSolutionConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.MoveConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.MunitionDetonationConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.RequestFireSolutionConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.TriggerPullConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.RequestLAMFireConverter:
    net.onesaf.core.services.sim.interopmgt.hla.fom.hlal_3.matrex.converter.v30el.RequestPAMFireConverter:
```

Figure 3. BLCSE OneSAF HLA converter settings

Converter Class Structure

Standard OneSAF HLA converters contain at least two methods, *processOOSFiredInter (SimulationEvent)* and *processFOMReceivedInter(InteractionClassRTIHandle)*. The former method is called by the Interaction Handler to convert OneSAF ODM outbound or published interaction data to the FOM. The latter method is called to convert inbound or subscribed FOM interaction data into OneSAF ODM. Another required method within the converter classes is the *initialize(InteractionHandler)* method. This method adds the subscription and publish classes for the converter to the publish and subscribe list for the instance of the HLA interoperability adapter.

Data Type and Enumeration Mapping

OneSAF relies on the enumerated entity and munitions names in the Object Model Template (OMT) file to perform enumeration mappings between OneSAF entity and munitions compositions to the FOM entity and munitions data. It is important to note the difference in the OMT and Federation Execution Data (FED) files. Both are representations of the FOM. However, the FED includes the subset of the data in the OMT, which the RTI requires at runtime to set up proper publication and subscription routes. Therefore, only the OMT file contains the enumerated data type (EDT) definitions OneSAF requires to perform proper enumeration mapping.

In order for OneSAF to map corresponding entity and munitions FOM data to OneSAF entity and munitions compositions HLA entity and munitions enumeration mapping files are utilized. Listed in the OneSAF HLA configuration file are references to FOM mapping data files for entities and munitions. These files list the FOM enumeration description and the OneSAF composition. Also listed in these files is the intended direction of the mapping. A value of "inbound" would be a FOM to OneSAF mapping, a value of "outbound" is a OneSAF to FOM mapping, and a value of "bidirectional" would be for both directions.

FOM CONVERTER CLASSES

As released, OneSAF version 2.1 comes with various FOM support. It supports Entity Resolution Federation (ERF) FOM version 4.0, Multi-Resolution Federation (MRF) FOM version 4.0, and MATREX FOM versions 2.0 through 4.0. Since MATREX has released numerous versions of their FOM, OneSAF kept its converter classes compatible with all major FOM releases. Each MATREX FOM release has its own directory structure with supporting converter classes within the OneSAF HLA framework. With each new MATREX FOM release, the interaction and object converter classes are updated as an extension of the previous FOM converter classes. For example, the fire interaction converter for MATREX FOM version 4.0 extends → version 3.1e extends → version 3.0 extends → version 2.2 extends → version 2.0. Along with the interaction and object converters, OneSAF also includes a set of utility converter classes that convert commonly used complex FOM data types, such as State Vector, to simple FOM data types.

Converters get called when one of two conditions are triggered. One condition is when a subscribed OneSAF interaction is created or a subscribed OneSAF object attribute is changed, e.g., when a OneSAF fire interaction is created or a OneSAF entity's position changes. When this condition occurs, OneSAF converts interaction/object data to FOM interaction/object data. The other condition that triggers converters is when a subscribed FOM interaction or object attribute update is delivered to OneSAF. In this case, the corresponding converter translates the FOM interaction/object data to OneSAF interaction/object data that gets posted to OneSAF's sim engine. If the FOM object update is for a newly discovered entity, the subscribed entity converter class creates a OneSAF entity to represent that external entity in OneSAF. This entity is marked as transient. It has no physical or behavioral model associated to it. It is important to note that the OneSAF checkpoint process does not save transient entities, as OneSAF currently only supports light-weight checkpoints during an HLA or DIS exercise.

All OneSAF complex data types are broken down to basic data types. OneSAF supports the External Data Representation (XDR) standard used by most HLA federations. OneSAF reads and writes to the FOM data buffer through its two XDR classes. "XDROutputStream.java" writes OneSAF basic data types to the FOM data buffer. "XDRInputStream.java" reads from the FOM data buffer to populate OneSAF data types. These classes extend Java's DataOutputStream class, adding new functionality specific for writing and reading FOM compliant data, such as padding shorts, Booleans and strings to 4-byte boundaries.

MATREX FOM converters that come included in OneSAF cover most HLA interactions and objects translation needs. However, over the course of BLCSE experiments, new interoperability requirements with other simulations have driven the needs for new converters to translate new interactions and objects. Some of the converters needed are in the areas of mounting and dismounting, supply transfer, and maintenance casualty care and/or evacuation.

CASE STUDY: NEW FOM ATTRIBUTE

Functional Requirement

To support the Complex Web Defense (CWD) experiment conducted in March 2008, BLCSE had a requirement to portray civilians as hostile entities if they are visibly carrying one or more weapons. However, the FOM used by the BLCSE federation only contained enough information to describe the weapon state of a single weapon being carried by a civilian.

Implementation

To implement the requirement, the team created a new FOM attribute called SecondaryWeaponState in the existing IndividualCombatant object. In conjunction with the existing FOM attribute, PrimaryWeaponState, the FOM can describe the states of both weapons being carried by a civilian or soldier entity. The FOM is a simple text file written in OMT format. Therefore, it can be modified using a standard text editor or using a FOM-specific application tool, such as Aegis's Object Model Development Tool (OMDT). Figure 4 shows the

FOM detail of the modified Individual Combatant object from a standard text editor.

In order for OneSAF to be able to populate this new FOM attribute, the integration team created an IndividualCombatantConverter in the BLCSE extension package. Because the FOM used in the BLCSE federation is a variant of the MATREX FOM, the actual file is located at services\sim\interopmgt\hla\converters\matrex\blcse\IndividualCombatantConverter.java. This file creates the necessary FOM and ODM variables. It also adds these variables to the lists of attributes which later will be sent to the RTI for publication and subscription.

```
(Class (ID 35)
   (Name "IndividualCombatant")
   (PSCapabilities PS)
   (Description "Indicates a single soldier")
   (SuperClass 12)
   (Attribute (Name "PrimaryWeaponState")
              (DataType "ICWeaponStateEDT")
              (Cardinality "1")
              (Accuracy "perfect")
              (AccuracyCondition "always")
              (UpdateType Conditional)
              (UpdateCondition "On change")
              (TransferAccept N)
              (UpdateReflect UR)
              (Description "Current state of an individual combatant\'s weapon")
              (RoutingSpace "PlatformSpace")
   )
   (Attribute (Name "SecondaryWeaponState")
              (DataType "ICWeaponStateEDT")
              (Cardinality "1")
              (Accuracy "perfect")
              (AccuracyCondition "always")
              (UpdateType Conditional)
              (UpdateCondition "On change")
              (TransferAccept N)
              (UpdateReflect UR)
              (Description "Current state of an individual combatant\'s weapon")
              (RoutingSpace "PlatformSpace")
   )
```

Figure 4. FOM detail of new Individual Combatant object attribute

The handleODMPropertyChanged function populates the FOM variable from internal ODM variables by calling the handleODMWeaponState function, adapted from the existing handleODMPrimaryWeaponState function. The ODM value, obtained from the Lifeform class, is translated to the proper FOM enumeration values. This value is then written in XDR format to the FOM data buffer, from which it will be transmitted to the RTI through the setAttributeInClassHandle function.

The handleFOMAttrRequest function populates the ODM variable for data coming in from the RTI by calling the handleFOMSecondaryWeaponState function, adapted from the existing handleFOMICWeaponState function. This function decodes the XDR-formatted data and sets the Lifeform object with the correct weapon state value. The function also determines if the Individual Combatant (IC) is carrying a visible weapon, by evaluating the LifeForm's weapon states. If the IC is not carrying a visible weapon, the function uses the Lifeform's Entity base class to default the IC's composition name to a standard civilian. This triggers the OneSAF Plan View Display (PVD) to show the IC with a green, neutral icon, effectively hiding the IC's true identity. In addition, by modifying the composition name, the user cannot see the true composition by using the icon's tool tip information.

Results

The converter classes worked effectively for the CWD experiment, preventing operators from knowing if an apparent civilian might, at some point, pick up a gun and become a hostile entity. Another option for implementing the same functionality is to create an attribute called ICWeaponState that has cardinality 0+. This would allow an IC to have any number of weapons without requiring additional FOM or converter class changes.

CASE STUDY: NEW FOM INTERACTION

Functional Requirement

To support CWD, BLCSE had a requirement to provide a visual indication on the PVD when an Ultra High Resolution Building (UHRB) becomes destroyed. This required OneSAF to interoperate with a Dynamic Terrain Server (DTS). The DTS determines when a UHRB becomes destroyed based on detonations that occur within the area of the UHRB. Information is passed from the DTS to OneSAF via a new HLA interaction called FeatureDamage, which contains a parameter indicating the center of mass location of the affected UHRB.

Implementation

Once the team added the Feature Damage interaction to the FOM, OneSAF needed a way to utilize the new data that would be delivered via the interaction. As with the new FOM attribute, a converter class was written using existing OneSAF super classes to convert the byte stream of data into useable values. The data converted are double values resolving the three points of the coordinate specifying the UHRB's center of mass. From the location provided in the interaction, a one square meter bounding box is created. Using the bounding box, a call is made to get the UHRB within the boundary, and the damage fraction is then set to 100%. Within OneSAF when a UHRB damage fraction is set to anything above zero, a visual PVD indicator is drawn. In this case, a red line outlines the UHRB to identify that it has been damaged or destroyed.

Since ODM classes have not been developed yet for damage fraction level access to features like UHRB's the only way gain access to this level of data was to access the FeatureAPI class and EnvironmentHandle class directly. Most of the work to allow ODM access to the ERC architecture is not due to be complete until around OneSAF version 4.0. Meanwhile, accessing the lower level class is a risky but viable option. Using these lower level classes limits the ease of access and may cause some modification of data that was unintended.

Results

The effects of this implementation were visual only. In no way was the traffic ability of the buildings affected. The architecture to provide this level of interface from the HLA converter classes is just not available yet and is difficult to implement. Detailed knowledge of the ERC architecture and a familiarity with the PVD interfaces is required. Most of the damage level and rubble capability exists in the low resolution buildings, but the ability to rubble or largely degrade UHRBs is not yet available.

Great care should be exercised when using the OMDT to modify the OMT and FED files. When the FOM is created, each object and interaction is given a numerical identifier. Some applications, including OneSAF, use these identifiers, rather than the interaction name, to access the FOM's interaction data. When adding a new interaction to the FOM, the OMDT does not necessarily add the interaction to the end of the list of identifiers. Instead, it inserts the new interaction into the sequence based on the new interaction's class hierarchy. This can cause a cascading re-numbering of all existing interaction IDs and cause problems for applications that rely on pre-defined interaction IDs.

DIS INTEROPERABILITY ARCHITECTURE

Besides from supporting HLA, OneSAF also natively supports DIS. Switching which interface to use can easily be done by changing the interoperability interface within the "onesaf.properties" file. The DIS interface is similar to the HLA interface on how it communicates with OneSAF Object Database (ODB). It also contains converter classes for of standard IEEE 1278 PDUs. However, the underlying architecture for subscribing and translating the OneSAF ODB is different from the HLA implementation. It is also possible to enable both interfaces at the same time. By doing so, the OneSAF node can act as a DIS-HLA gateway. DIS data will be translated into the OneSAF ODB and then translated to HLA and vice versa. The Advanced Concepts Research Tool (ACRT) program uses this architecture to allow OneSAF to receive simulation traffic via HLA and to send data to it 3-D visual system via DIS.

SUMMARY

OneSAF interoperability is currently a major objective of many simulation communities. The details in this paper are intended to serve as a guide for developing HLA interoperability for specific needs within a simulation environment. The BLCSE OneSAF integration team has resolved many technical and developmental issues during the past two years, which have allowed OneSAF to participate in successful, record-run experiments within a large-scale, highly distributed federation in HLA mode. These issues range

from network configuration to performance. More issues and resolutions to this work can be found the 2007 I/ITSEC paper entitled "Modernizing Army Experimentation Using OOS" and the 2008 I/ITSEC Paper entitled "OneSAF Testing in Complex Web Defense Experiment".