Extended FOM Module Merging Capabilities

Björn Möller – Pitch
Andy Bowers – Mitre
Mikael Karlsson – Pitch
Björn Löfstrand – Pitch

bjorn.moller@pitch.se
bowersa@mitre.org
mikael.karlsson@pitch.se
bjorn.lofstrand@pitch.se

Keywords:
HLA Evolved, RPR FOM, FOM modules, DDM, Enumerations

ABSTRACT: *The HLA Evolved specification describes a number of FOM Module merging principles whereby classes, data types, etc. from different modules can be merged into the FOM of a federation. Based upon practical experiences with FOM modules, this paper proposes three additional capabilities for the next HLA version. They are particularly useful, and in some cases necessary, when reusing and extending standardized FOMs, for example the RPR FOM. They can also be designed so that backwards compatibility is maintained.*

The first capability is to enable a module to add attributes to an already existing object class. This enables federations to extend object classes of a reference FOM without modifying the reference FOM.

The second capability is to enable a module to add DDM dimensions to an already existing attribute or interaction class. This enables federations to use DDM filtering for concepts described in a reference FOM. An even more powerful approach would be to not require dimensions to be explicitly specified in the FOM for attributes and interactions classes, which would increase flexibility.

The third capability is a development time rather than runtime FOM merging capability. An enumerated data type in a FOM should be able to reference and include a separately stored list of enumerated values. This would simplify the usage of enumerations in the RPR FOM, where several enumerations, with a large number of values, are shared with the DIS standard.

These three capabilities are presented in detail for discussion and possibly inclusion in the next version of HLA.

1. Introduction

This paper proposes three updates to the High Level Architecture (HLA) standard. It contains background, rationale, analysis, and some discussion. The updates are mainly related to the HLA Object Model Template but a few Runtime Infrastructure (RTI) services from the HLA Interface Specification are also involved.

The overall purpose of the updates is to add more flexibility based on practical experiences. These updates are mainly focused on long-term reuse. They are based on practical experiences in several projects. An example from work done by the United States Joint Staff J7 Deputy Director Joint Environment (DDJE) is presented and similar experiences also exist in NATO project work.

1.1 Object Model Template, FOM and SOM

The HLA Object Model Template (OMT) is a template that is used for describing object classes with attributes, interactions with parameters, data types, and several other things. It is used to describe two things in HLA:

The **Federation Object Model** (FOM), which is the information exchange model that is used by a federation at runtime.

The **Simulation Object Model** (SOM) that is used to describe the capabilities of a federate. The SOM is typically used when judging the suitability of a federate for a particular purpose. Not all of the information in the SOM may actually be exchanged by the federate in a particular federation.

In this paper, we will mainly talk about FOMs but most of the discussion also applies to SOMs.

Every federation has a FOM. In many cases it builds upon some standardized FOM, like the Realtime Platform Reference FOM (RPR FOM). This considerably improves the interoperability and potential for reuse.

1.2 The evolution of the HLA Object Model Template format

As the HLA standard has evolved, so has the Object Model template format.

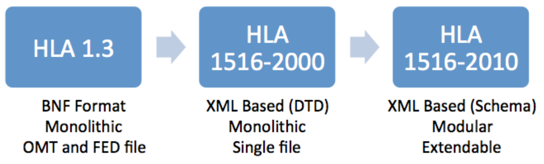


Figure 1: Evolution of the HLA OMT format

In the HLA 1.3 standard (1996-1998) the FOM data was provided as a monolith. A Backus-Naur Form (BNF) format was used. There were actually two types of files: the regular OMT file and the FED file, which was used to initialize a federation execution. To keep these two files synchronized, a popular solution was to use the OMDT format (not officially part of the HLA standard) to store all of the information in one single file which was then used to generate the OMT file and the FED file.

In HLA 1516-2000, all of the information was moved into one file. The HLA standard was harmonized with XML, at that time an emerging standard. The XML Document Type Definition (DTD) was used to specify the format.

In HLA 1516-2010, FOM modules were introduced. It now became possible to separate different concerns into different FOM modules. Development, maintenance, and reuse could now be done in a modular and composable way. Other improvements included the use of standardized XML schemas for verification of format, syntax, and consistency of FOMs and SOMs. Yet another feature was the possibility to add custom XML tags to the format.

2. FOM Merging in HLA Evolved

We will now look at FOM modules in the HLA Evolved (HLA 1516-2010) standard. The basic idea is that different FOM data can be kept in different FOM modules, for example a Vehicle module, a Radio module, and a Federation Management module. This allows for modular development, maintenance, and reuse of FOM data.

2.1 FOM merging using simple union

Let's look at the most commonly used FOM data and how they are merged: Object classes with parameters, interactions with parameters, data types, and dimensions. The basic principle is the union operation. Here is an example with data types (Figure 2).

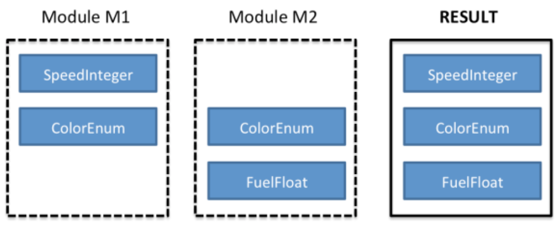


Figure 2: Merging data types from two modules

Module M1 defines the data types SpeedInteger and ColorEnum. Module M2 defines the data types ColorEnum and FuelFloat. When these modules are merged, the result is the union, i.e. SpeedInteger, ColorEnum, and FuelFloat, if and only if the two definitions of ColorEnum are equal. If the definitions of ColorEnum differ then the operation shall fail.

The same principle as for data types applies to dimensions.

2.2 FOM merging using union of trees

For object and interaction classes, the union still applies but using a tree structure. The simplest case is shown in Figure 3.

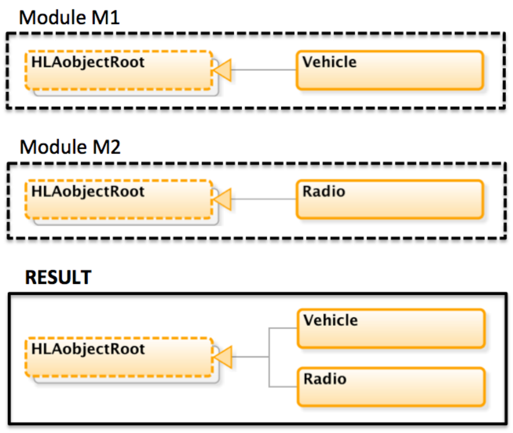


Figure 3: Merging two sibling classes

Module M1 defines the object class HLAobjectRoot.Vehicle. Module M2 defines the object class HLAobjectRoot.Radio. When these modules are merged, the result is the union, i.e. both the Vehicle and Radio classes as subclasses to HLAobjectRoot.

Another example is subclassing using FOM Modules as shown in Figure 4.

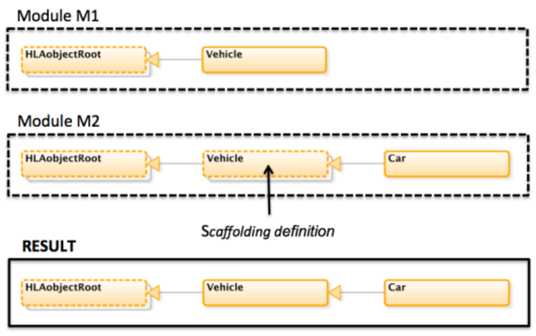


Figure 4: Merging subclasses

Module M1 defines the class HLAobjectRoot.Vehicle. Module M2 defines HLAobjectRoot.Vehicle.Car. When combined, the result is a hierarchy with the Car class as the subclass of the Vehicle class. Module M2 may either provide a "**scaffolding**" (empty) definition of Vehicle or **repeat** the definition from M1. The recommended approach is to use scaffolding definitions to avoid having to maintain the same class in two or more different modules.

Note that for repeated object or interaction classes to merge successfully, they are required to have the same properties, including the same set of attributes/parameters.

2.3 When are FOM Modules merged?

Earlier HLA versions only support one monolithic FOM for a federation execution. In HLA Evolved, there are two possibilities:

A number of FOM modules can be provided when the federation execution is created, using the Create Federation Execution service. These modules are merged and, if and only if the merge is successful, the Federation Execution is created and initialized with that FOM data. In practice, at least one FOM module has to be provided since some FOM module has to provide the Switches table, used to configure certain RTI features. Roots for the object and interaction class hierarchies as well as the Management Object Model (MOM) are provided in a FOM module called the Management Initialization Module (MIM), which is provided automatically by the RTI (although this can be overridden with a custom MIM).

Additional FOM modules can be provided by a federate that joins the federation by calling the Join Federation Execution service. These modules are merged with each other and the FOM data in the federation. If and only if the merge is successful, the federate is joined to the federation.

In most projects, there is also a third situation when FOM modules are merged, namely during FOM development using FOM editing software.

3. Extended Semantics for Merging Classes

There is one limitation with the current FOM module merging that has become obvious in practical use. It is not possible to use an additional FOM module to add attributes to an already defined object class, without introducing a subclass. The same applies to parameters and interaction classes. Consider the following example: We have a FOM module M1 that defines the Car object class with the attributes Name, Color, Position and Speed. We now have some additional federates that want to add the attributes FuelType and FuelLevel. This can be done by creating a specialized FOM module M2 that subclasses the class Car with the new object class SpecialCar, as shown in Figure 5.

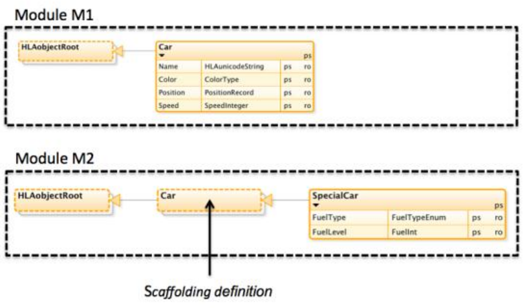


Figure 5: Adding more attributes using a subclass

This solution has a major drawback. If an existing federate registers Car instances it will use the original Car class, not the SpecialCar. We will need to update existing federates which may not be possible.

Another option would be to simply modify the original module M1. This presents no problem unless M1 is a reference FOM or a FOM that is standardized in some other way. The result is that we branch a standard FOM into numerous customized versions, forfeiting the advantage of standardized FOMs.

The first proposal in this paper is to allow new modules to add new attributes to existing object classes and new parameters to existing interaction classes. The current requirement in the standard is that repeated definitions of classes must be equal. The new requirements would be that

a) when a definition of a class is repeated, the merging process shall take the union of the attributes/parameters of the classes.

b) If a definition for the same named attribute/parameter for a given class exists in several modules then they are required to be equal.

The proposed, extended semantics is shown in Figure 6.

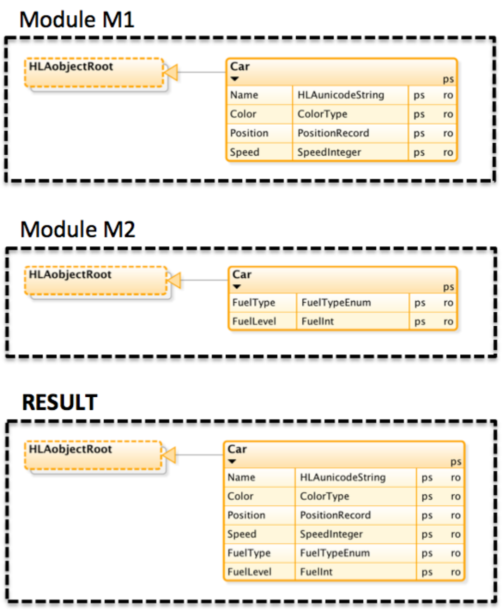


Figure 6: Adding more attributes to a class using a module

3.1 Analysis of technical implications

The proposed update would not create a problem for existing federates, FOMs, and federation agreements that use the HLA Evolved semantics. The new semantics are a superset of the current semantics and an existing federation would work as before.

In RTI implementations there are a few challenges, in addition to being able to implement the new merging rules. Consider the case when a new federate joins and loads a FOM module that adds more attributes to an existing object class. What implication does this have for already registered object instances?

One could argue that only new object instances get the new attribute. This would create confusion since some instances of the same class have more attributes than others. Federates that are aware of the new attributes would need to take this into account in their logic. In our example we could not rely on all Cars having a FuelLevel. In some cases this could result in instances of the same object class having different behaviors depending on at what time they were registered.

One could also argue that existing object instances would get the new attribute automatically. But what is the status of that attribute? It would probably be unowned since the registering federate probably only knows about (and publishes) the definitions in the original FOM module. On the RTI implementation side there are also issues with adding data to object instances that have already been discovered by several federates.

One approach that solves the above problem is to only allow for FOM modules that add attributes to existing classes in the Create Federation Execution service call (i.e. not in the Join service call). The drawback is that this creates different FOM merging rules depending on if the FOMs are provided during Create Federation Execution or Join Federation Execution service call.

3.2 Temporary workaround

HLA users may want to achieve the above functionality today, in particular if they want to use reference FOMs, such as the RPR FOM. The simplest way to do this is at development time. The following procedure is then suggested:

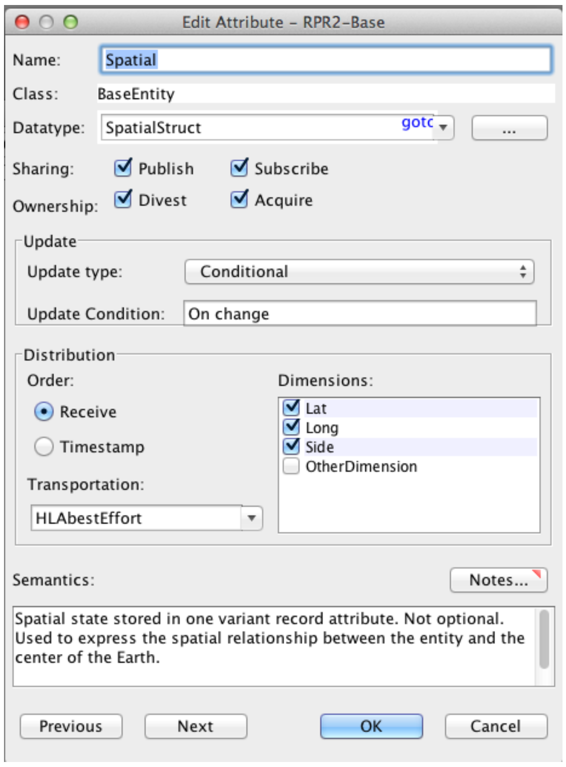
1. The FOM development is based on a reference FOM in the HLA 1516-2010 format.
2. Custom attributes, for example for the RPR FOM “platform” class, are added in a separate module.
3. These are merged, manually or using a tool with the proposed semantics.
4. The result is a monolithic, extended RPR FOM that is used to initialize the federation.

The same method can be used for adding parameters to interaction classes.

4. Extended Semantics for Merging Dimensions

The HLA Data Distribution Management (DDM) is used to reduce the incoming (subscribed) information to a federate. Subscriptions can thus be based not only on classes and attributes (such as “aircraft.marking”) but also on other dimensions, such as a geographic grid overlaid on a battlefield or which side the aircraft belongs to.

Figure 7 shows how the DDM dimensions Lat, Long and Side are added to the spatial attribute in the RPR FOM.



Name: Spatial
Class: BaseEntity
Datatype: SpatialStruct
Sharing: ✓ Publish ✓ Subscribe
Ownership: ✓ Divest ✓ Acquire
Update
Update type: Conditional
Update Condition: On change
Distribution
Order: [● Receive] ○ Timestamp
Transportation: HLAbestEffort
Dimensions: ✓ Lat, ✓ Long, ✓ Side, ☐ OtherDimension
Semantics: Spatial state stored in one variant record attribute. Not optional. Used to express the spatial relationship between the entity and the center of the Earth.

Figure 7: Spatial attribute with dimensions added

It is not possible to add more dimensions to an already defined attribute or interaction without updating the original FOM module. There is one additional complication compared to attributes. It is not possible to add more dimensions to an attribute using subclassing. This means that there is currently no way to add dimensions to a reference FOM without modifying it.

Our proposal is to allow a new FOM module to add new dimensions to an existing attribute or interaction class. This should result in the attribute or interaction having the union of all dimensions specified in all FOM modules.

4.1 Analysis of technical implementation

The proposed update would not create a problem for existing federates, FOMs, and federation agreements that use the HLA Evolved semantics. The new semantics are a superset of the current semantics and an existing federation would work as before. One minor issue is that if the Convey Region Designator Sets switch is enabled, regions using unexpected dimensions may be conveyed to federates that are unaware of all FOM modules that are currently loaded.

In RTI implementations there are fewer challenges compared with the attribute case. The main issue would be to implement the new merging rules.

4.2 A bolder proposal

It can also be argued that it is cumbersome to specify numerous dimensions for numerous attributes and interaction in a FOM. When these dimensions are used at runtime, the RTI uses the DDM information of the subscription and matches it against the DDM information of the attribute update or interaction, resulting in a binary yes/no decision. There is little or no connection with the class, attribute, interaction, or encoded value during this operation. For many RTI implementations it would make no difference to avoid specifying which dimensions that are used for a particular attribute or interaction. This is more a documentation that is of interest to the federate developer. A bolder proposal would be as follows:

1. Remove the Dimensions property of attributes and interactions.
2. Allow any dimension in the FOM to be used when subscribing to and sending interactions.
3. Allow any dimension in the FOM to be used when subscribing to and updating attributes.

5. JLVC: A FOM Merging Use Case

The United States Joint Staff J7 Deputy Director Joint Environment (DDJE) sponsors the development and use of the Joint Live, Virtual, Constructive (JLVC) Federation to support Joint Force and Coalition training. JLVC is a RPR FOMv2d17-based federation, but the JLVC FOM has diverged significantly from the RPR FOM during the past ten years as new training requirements necessitated FOM additions, and evolving data exchange agreements allowed removal of existing RPR FOM transactions. JLVC federation engineers realized the extent of the JLVC FOM divergence from the RPR FOM as they studied migrating JLVC from the HLA v1.3 specification to the IEEE 1516-2010 standard in 2010. They sought to take advantage of the modularity feature of the new specification while complying with its requirements. JS J7 experiences with the new standard, and lessons learned in merging FOMs, include an experiment in 2011 developing a “mixed-mode” HLA federation, in which a federate that used the HLA 1516-2010 API was added to an existing HLA 1.3-based federation [1]. In addition, beginning in 2012, JS J7 started migrating the JLVC to the current HLA 1516-2010 specification.

5.1 Extending the RPR FOM

In both the experiment and the JLVC migration, JLVC engineers sought to use the RPR FOMv2d17 in its entirety as a reference FOM and extend it as necessary to account for the additions implemented in JLVC over the last ten years. JLVC engineers therefore removed “non-RPR” attributes and parameters from RPR FOM classes in the JLVC FOM and built new JLVC classes comprised of the removed attributes or parameters. These new JLVC classes were then subclassed to the RPR FOMv2d17 classes from which they were removed. For example, JLVC_Aircraft extends the RPR2 object class "Aircraft" with attributes for Callsign, IntelState, and so on. The new JLVC FOM consisting of the extension classes was then merged with the RPR FOM to form a single Federation Description Document (FDD), adhering to the 1516-2010 specification by being expressed in XML and in using standard datatypes. At this point, federation engineers encountered the problem with dimensions addressed in section 4. JLVC uses DDM and while dimensions were easily added to the attributes and parameters in the JLVC extensions, the extension classes in the merged FDD include inherited attributes/parameters from RPR2 classes which did not have dimensions. In order to have a useful FDD, dimensions were added to the RPR FOM modules. This was necessary, but it violates the IEEE 1516.2-2010 rules in section 7 and Annex C for merging FOMs.

5.2 Object class name issues

JLVC Engineers encountered another issue during the "mixed-mode" federation experiment, that of misaligned attributes and unrecognized names. As previously reported in [6], the RTI used in the experiment required that the 1516-2010 federate initiate the federation creation and that data exchanges with the 1.3 federate were based on the 1516-2010 FOM file. In most cases, the class name of the leaf class in the 1516-2010 FOM was not recognized by the 1.3 federate because it was an extension class name, e.g. JLVC_Name, not used in the 1.3 FOM. Some classes in the 1516-2010 FOM kept the 1.3 FOM name, for example "BaseEntity.PhysicalEntity.Platform.GroundVehicle.SPArtillery", because all of the JLVC-specific attributes could be added to the "SPArtillery" class in the 1516-2010 FOM since "SPArtillery" is not a RPR FOM class name. Although in this particular class the two FOMs had the same attributes, to include inherited attributes, the order of the attributes was different in the two FOMs and that misalignment also precluded data exchange. JS J7 experience therefore supports the proposal recommended in section 3 for allowing new modules to add new attributes to existing object classes and new parameters to existing interaction classes; this would solve both the name and attribute alignment issues.

5.3 Road ahead

In 2013, JS J7 intends on decomposing the new JLVC FDD into separate modules. These will maintain the class relationships expressed in the current FDD, but enable subsequent development, maintenance, and reuse based only on relevant modules. JS J7 has drafted these modules based on the RPR FOM modules which have been proposed to SISO as candidates for RPRv3.

6. Thoughts on Enumerations

A look at the RPR standard reveals that there are many enumerations. These can roughly be divided into two types of enumerations:

1. Enumerations that don't change much over time. In many cases they have a fairly limited set of enumerators (values).
2. Enumerations that are frequently updated, usually with additional enumerators. They usually have a large number of enumerators. Sometimes they are maintained as a separate project, possibly in coordination with other standards.

The second type of enumerations are less suitable for maintenance as part of a reference FOM due to the different update cycles and possibly the need to update several lists in parallel. It is also inconvenient to store a large number of enumerated values as part of a FOM.

So how should enumerated data types that are frequently updated be maintained? There are at least two possible approaches.

Put the enumerated data type in a **separate FOM module**. Develop software (like applications or XML transformations) that creates this FOM module from a list of enumerators specified in some other format. This mechanism is non-intrusive and can be used for the current standard. The drawback is that there is some additional information, like the Identification table, that should be included.

Use the **Xinclude** feature of XML that allows for inclusion of one XML document or document fragment into another. The exact format of the fragment needs to be specified for this mechanism to be useful.

Both of the above approaches would require some type of configuration management in order to be useful in practice. This question needs to be further analyzed and discussed in the HLA Evolved product support group.

The RPR FOM Drafting Group has encountered the above issue and has chosen to put enumerations in separate FOM Modules for the modular HLA 1516-2010 version of the RPR FOM 2.0. This brings up a new interesting topic: not only do we need to have reference FOMs that can be extended in a wellcontrolled way but we also need to manage how certain parts of a reference FOM can be replaced.

7. Conclusion

This paper proposes three extensions to the FOM merging of HLA 1516-2010. The two major extensions are:

Allow the addition of new attributes to an existing object class using a new FOM module.

Allow the addition of new dimensions to an existing attribute or interaction class using a new FOM module.

One of the main drivers for these extensions is the use of reference FOMs that should stay unmodified. A practical use case is presented to support this.

The impact of the above updates is minimal for existing federations and federates since a superset to the existing functionality is proposed. The bigger impact is on the RTI implementation side.

The third extension relates to the maintenance of enumerated data types with a large number of enumerations. Some thoughts have been presented.

These proposals and the technical analysis should be further discussed with the SISO HLA product support group. If accepted, some additional work is needed to produce accurate standards texts.

References

[1] “High Level Architecture Version 1.3”, DMSO, www.dmso.mil, April 1998

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

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

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

[5] Björn Möller, Björn Löfstrand. “Getting started with FOM Modules”, Proceedings of 2009 Fall Simulation Interoperability Workshop, 09F-SIW-082, Simulation Interoperability Standards Organization, September 2009.

[6] Andy Bowers, Brian C Gregg, John Tufarolo. “FOM Modularity and Mixed-Mode Federation Design and Development: Challenges and Observations”, Proceedings of 2011 Fall Simulation Interoperability Workshop, 11F-SIW-029, Simulation Interoperability Standards Organization, September 2011.

Author Biographies

BJÖRN MÖLLER is the vice president and co-founder of Pitch Technologies, the leading supplier of tools for HLA Evolved, 1516-2000 and HLA 1.3. 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. He is currently serving as the vice chairman of the SISO HLA Evolved Product Support Group.

ANDY BOWERS is a Lead Simulation Modeling Engineer in the MITRE Corporation's Modeling & Simulation Engineering Department. He is a retired United States Army officer and has more than 15 years of experience in simulation and federation design and development for military training. He has supported several NATO Modeling and Simulation Group (NMSG) efforts and currently supports the Joint Staff J7's Joint Live, Virtual, Constructive (JLVC) Training Federation System Engineering Team. His interests include multi-resolution modeling and simulation interoperability.

MIKAEL KARLSSON is the Infrastructure Chief Architect at Pitch overseeing the world's first certified HLA IEEE 1516 RTI as well as the first certified commercial RTI for HLA 1.3. He has more than ten years of experience of developing simulation infrastructures based on HLA as well as earlier standards. He also serves on several HLA standards and working groups. He studied Computer Science at Linköping University, Sweden.

BJÖRN LÖFSTRAND is the Services and Training Manager at Pitch Technologies. He has more than 15 years of experience in distributed simulation architecture and design. He has been supporting several simulation standardization activities within SISO including HLA, DSEEP and BOM. He is also supporting several NATO Modeling and Simulation Group (NMSG) activities and is currently leading the technical activities in MSG-106 for defining the NATO Education and Training Network federation design. Mr Lofstrand has a M.Sc. in Computer Science from the Technical University of Linköping in Sweden.