New Object Modeling Opportunities in HLA 4

Björn Möller, Mikael Karlsson
Pitch Technologies
Repslagaregatan 25
S-582 45 Linköping, Sweden
+46 13 470 55 00
bjorn.moller@pitch.se
mikael.karlsson@pitch.se

Keywords:
HLA 4, OMT, RPR FOM, Modular FOM, DDM, Enumerations

ABSTRACT: *One of the main success factors of HLA is that it makes it easy to create, maintain and share federation object models (FOM) for distributed simulations. Such models can be developed and extended for almost any domain, for example defense and security, space, engineering, medicine and transportation. Particular success stories include the Real-time Platform Reference FOM (RPR FOM), NATO Education and Training Network (NETN) FOM and the upcoming Space Reference FOM.*

As experience from developing FOMs have been gathered, HLA has been extended and matured over time. HLA IEEE 1516-2000 introduced well-defined data types and the use of XML syntax. HLA IEEE 1516-2010 introduced modular FOMs. Still, some challenges exist, in particular for making flexible extensions to reference FOMs.

The HLA standard is now open for revision for a new version, nick-named HLA 4. This paper explains some of the proposed new features for object modeling. Several of them are based on requirements from the RPR FOM community but are useful for most FOMs.

One important requirement is to be able to make a more fine-grained extension of an already existing object model using additional FOM modules. There are four cases for this:

- It should be possible to add more attributes to an existing object class, using a new FOM module. The same applies to interaction classes where new parameters need to be added. A typical RPR FOM use case is to add more attributes to the standard Aircraft class.
- It should be possible to specify more Dimensions for DDM filtering to an existing object class or interaction class to meet the requirements of a particular federation. A typical RPR FOM use case is to be able to add Lat/Long based filtering of platforms in a battlefield.
- It should be possible to add more enumerators to an existing enumeration. A typical RPR FOM use case is to be able to add new entity types, that don't exist in the standard SISO-REF-010 enumerations.
- It should be possible to add more variants to an existing variant record data type. A typical RPR FOM use case is to be able to add alternative encodings that describe new data links to the Signal interaction.

It has also been proposed to be able to clearly specify references, for example that the firing entity parameter of a fire interaction refers to a specific platform instance.

This paper explains these new features as well as how they can be used to simplify FOM development and enhance interoperability while maintaining backwards compatibility.

1. Background

The High-Level Architecture (HLA) [1, 2, 3] standard consists of the HLA Rules, the HLA Interface Specification and the HLA Object Model Template. The "HLA Rules" and the "HLA Interface Specification", are less related to any particular application domain whereas the Object Model Template (OMT) provides a generic template that is used for specifying domain-specific information exchange models. The most important parts of the OMT are:

1. Object Classes with Attributes, that are used for modeling classes of shared objects with some degree of persistence, for example aircrafts with position and aircraft type.
2. Interaction Classes with Parameters, that are used to model classes of instantaneous events, for example start signals with a start time.
3. Data Types, that are used to model the exact interpretation and representation of attributes and parameters, for example a position record, containing latitude, longitude and altitude.

The OMT provides a template that is used for two types of object models:

1. The Federation Object Model (FOM), that is used by all federates in a federation as an agreement for the information exchange.
2. The Simulation Object Model (SOM), that is used for specifying the information exchange capabilities of one particular federate.

This paper focuses on the FOM and the process of developing, extending and agreeing on a FOM.

1.1 Modular FOMs

In the first two versions of HLA, HLA 1.3 [1], published 1998, and IEEE 1516-2000 [2], published 2000, FOMs and SOMs were monoliths. In IEEE 1516-2010 [3], modular FOMs were introduced [4]. These allows for FOMs to be developed and provided in a modular format. A FOM can build upon other FOM modules and use and extend them.

Examples of ways to split a FOM into modules is by functional area, like separate FOMs for vehicles, radio, terrain, etc. Another way to split the FOM is by type of FOM data, like generic data types in one module, switches in another.

In retrospect, FOM modules are probably seen as the most important new feature of HLA IEEE 1516-2010. The concept is easy to understand and use by developers since it resembles architectural constructs of programming languages. It makes it easier to develop FOMs as collaborative efforts. It also makes it easy to extend existing FOM modules while keeping the extension separated.

1.2 Reference FOMs

As many FOMs have been developed in a particular domain and/or by particular communities, they have been standardized, within an organization or within a community, for example the Simulation Interoperability Standards Organization (SISO). Some examples of such standardization are:

1. The SISO Real-time Platform Refence FOM ("RPR FOM") [5], for defense and security simulations.
2. The SISO Space Reference FOM [6] for manned and robotic space mission simulation.
3. The NATO Education and Training Network FOM [7] ("NETN FOM").

Reference FOMs provide a number of advantages. They capture a lot of know-how within a domain. A FOM developer will not need to start from scratch but can build upon the experiences from other federation developers. They also enable a priori interoperability. Two federates that have been individually developed are more likely to be interoperable out of the box, which saves time and money.

1.3 Extending reference FOMs

While reference FOMs provide a good starting point, it is very common that they need to be extended with classes, attributes and parameters that are specific to a federation. Before Modular FOMs existed, the only approach was to edit the monolithic reference FOM, effectively creating a project specific FOM, as shown in Figure 1. Further integration events have traditionally meant that different federate developers would bring customized versions of, for example the RPR FOM to the integration event. These would then need to be compared, a non-trivial task, before a new, customized FOM could be agreed upon, and federates updated accordingly.

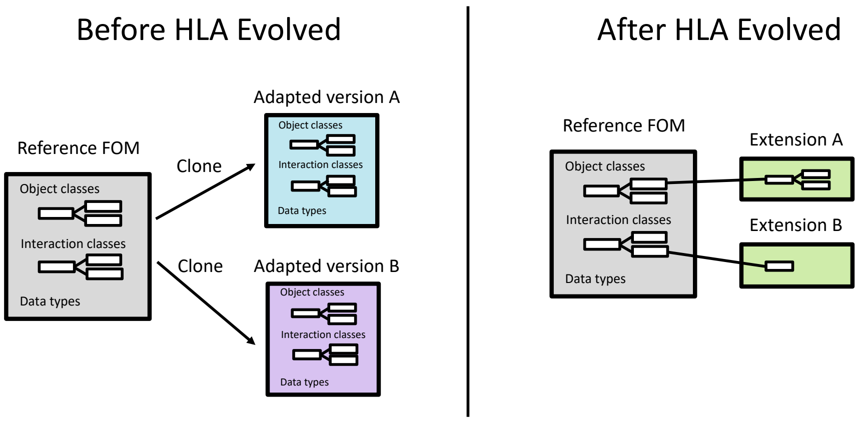

Figure 1: Extending Reference FOMs before and after HLA Evolved

The introduction of Modular FOM solved this problem, to some degree. In many cases, extensions were captured in separate FOM modules that would extend the reference FOM. New, extended subclasses could be added, for example by adding the custom "StealthAircraft" with new attributes to the "Aircraft" class of the RPR FOM. Still projects like the US JLVC FOM [8] and the NATO NETN FOM, have encountered some additional issues that need to be solved. A previous paper [9] has reported on some of the ideas. As the work with the next version of HLA ("HLA 4") progresses, the purpose of this paper is to take a closer look at the actual implementations that have been accepted in the initial comment rounds.

2. Extended composability – Object and Interaction Classes

When Modular FOMs were introduced in HLA Evolved, it was assumed that it would be enough to be able to add Attributes to an object class by creating a subclass in a new module. This subclass would then inherit all attributes from the superclass and additional attributes could be defined. As an example, in the RPR FOM, new attributes could be added to the Aircraft class using a FOM module that defined the subclass SpecialAircraft with additional attributes. In practice this approach has a few issues.

2.1 Legacy federates registering superclass

Consider a legacy federate that registers a number of Aircraft instances. In addition to the legacy federate, we have a number of newer federates that need a number of additional attributes, defined in the SpecialAircraft class, as shown in Figure 2. Since the legacy federate still registers Aircraft instances, the new attributes of SpecialAircraft will never be available, unless you modify the legacy federate. As the legacy federate is used in different federations, it may need to be updated to register different subclasses in different federations.

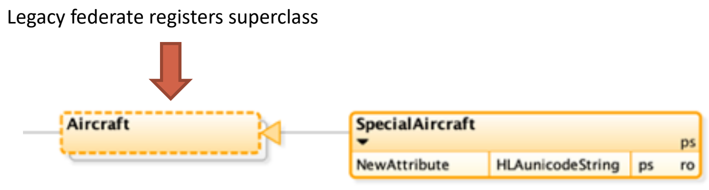

Figure 2: Legacy federate registering superclass

2.2 Add more attributes to non-leaf classes

In many cases it is desirable to define additional attributes for classes, that aren't leaf nodes. It might make sense to add more attributes for example "Mass" to the Platform class, as shown in Figure 3. These are then inherited to inherited to subclasses like Aircraft, GroundVehicle, etc. In this case subclassing is impossible since we need to add attributes to existing classes.

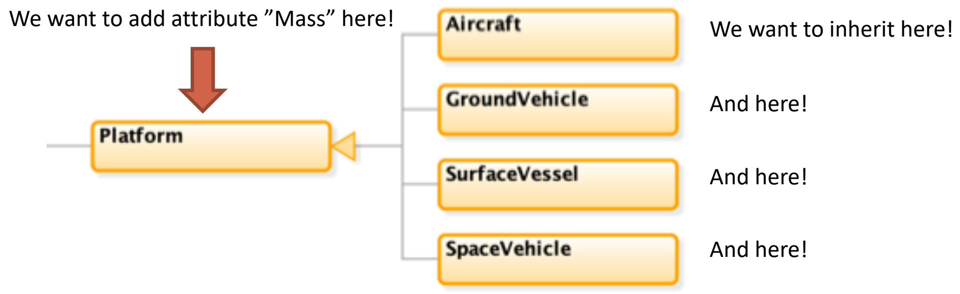

Figure 3: Add attributes to non-leaf class

2.3 Different federate sets need to add different attributes

It is impossible to support different sets of federates that have different needs for additional attributes. One set of federates may need additional attributes defined in the SpecialAircraft subclass, whereas another set of federates may need to additional attributes defined in the StealthAircraft subclass, as shown in Figure 4.

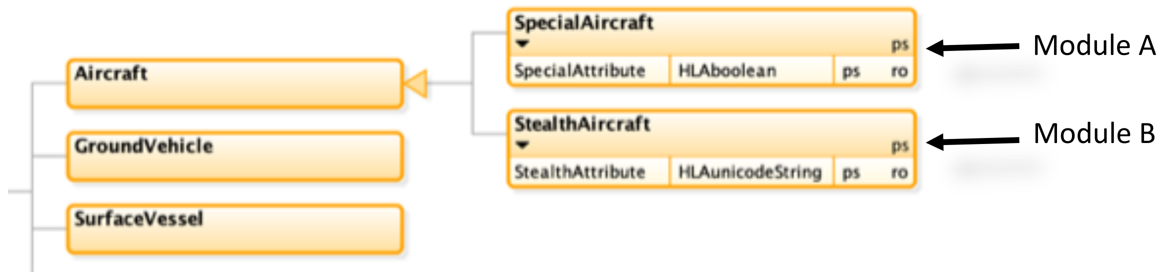

Figure 4: Different federate sets needs to add different attributes'

2.4 Complete definitions and extensions

The solution in HLA 4 is the following:

1. Provide a complete definition of a class and the initial set of attributes. This definition may also be repeated between FOM modules, although this isn't recommended.
2. Provide extensions, i.e. class definitions that only add attributes in one or more modules.

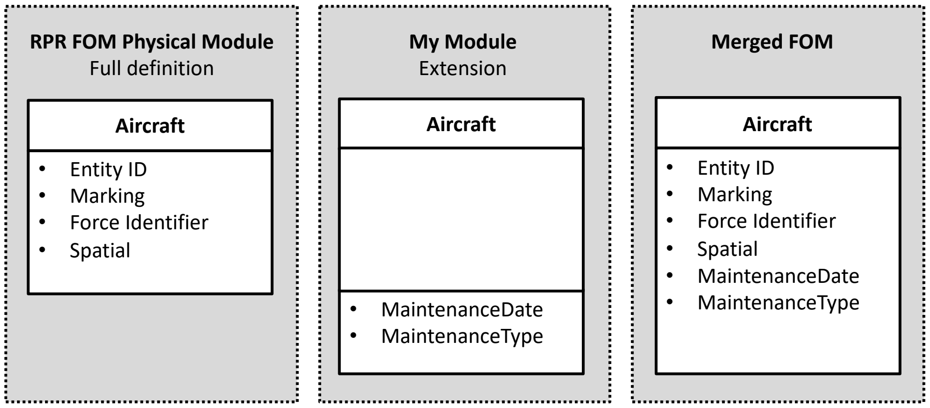

Figure 5: Complete definitions and extensions for object classes

As an example, an extension definition may add the attribute MaintenanceDate to the Aircraft in a RPR FOM federation, as shown in Figure 5.

2.5 Extending interaction classes

The exact same patterns apply to interaction classes. Consider adding parameters to the StopFreeze interaction class and the exact same issues as for object classes are encountered. The solution is exactly the same, by providing extension to the existing interaction class.

Note that a FOM module can (and in many cases will) contain both complete definitions and extensions. As an example, an air refueling federation may be based on the RPR FOM, but also add a FOM module with new fuel-related attributes for the Aircraft class of the RPR FOM as well as new interactions for the refueling process.

2.6 Impact on existing federates

It is also worth noting that adding more attributes to an object class will not have any impact on existing federates since they don't subscribe to these attributes. For interactions, subscription is performed on the entire interaction including all parameters. This means that a federate may encounter new and, to the federate, unknown parameters when receiving an interaction. In this case the federate shall skip any such parameters.

3. Extended Composability – Enumerations and Variant Records

There is also a need to extend the definition of two types of HLA datatypes: enumerations and variant records. The most obvious case is enumerations. An enumerated datatype may contain a commonly used set of enumerators, but a specific federation may have needs for very specific enumerators. In this case there is no possibility to extend the set of enumerators in HLA Evolved since enumerated data types cannot be extended using inheritance, nor can the datatype of an attribute be redefined or extended using a subclass. The only solution is to modify the original definitions of the enumerated datatype, which is undesirable if it is defined in a reference FOM.

As an example, there are a number of highly useful enumerations in the Enumerations FOM Module of the RPR FOM, for example for Object Types, such as type of aircraft. These are based on the ongoing work of the SISO-REF-010 working group. In many federations there is a need to add more enumerators, for example if the purpose of the federation is to analyze a new aircraft is under development.

The solution in HLA 4 is the following:

1. Provide a complete definition of an enumerated datatype and the initial set of enumerators. This definition may also be repeated between FOM modules, although this isn't recommended.
2. Provide extensions, i.e. definitions that only add enumerators to the datatype in one or more modules.

There are some additional considerations for this type of extension:

1. There is a risk that a numeric value of an enumerator conflicts with an already existing enumerator. This can to some degree be handled by providing a recommended range for extended values.
2. Legacy federate may encounter unknown enumerated values, defined by new extension definitions. Exceptions may be avoided by requiring federates to be able to properly process unknown enumerators, but the basic semantic problem remains: the legacy federate has no insight into what the new enumerator actually means.

Another case is the datatype VariantRecord. Using a Variant Record, an attribute or parameter may contain different datatypes, distinguished by different discriminator values. For example, a position attribute may be represented using Lat-Long or using a Geocentric value, with the selected variant specified using a discriminator. A federation may want to extend an existing Variant Record definition with a new variant. As an example, the Signal variant record of the RPR FOM may need to be extended with a new variant. The problem is very similar to enumerations. Currently, there is no way to add new variants using a subclass in a separate module or anything similar.

The solution in HLA 4 is the following:

1. Provide a complete definition of a variant record and the initial set of variants. This definition may also be repeated between FOM modules, although this isn't recommended.
2. Provide extensions, i.e. definitions that only add variants to the datatype in one or more modules.

Adding new variants requires an update of the standard HLA encodings. The encoding used for variant record uses the HLAvariantRecord encoding, which assumes that a federate developer knows about all existing variants when developing a federate, to correctly determine size and padding of the data. If a new variant is introduced after the development, for example a variant with a bigger data type, then the size of the encoded data, the byte alignment and the padding may change. As a result, the code of existing federates will not properly process the data buffer. This is particularly problematic when a variant record datatype is part of a complex data type, such as a record.

The solution is to introduce a new encoding method, the HLAextendableVariantRecord, that explicitly specifies the byte alignment and length of the data for each variant. This enables federates to skip new variants that they have no deeper knowledge about.

4. Making DDM simpler and more powerful

Data Distribution Management (DDM) is a set of HLA services that makes it possible to further specify which information that a federate is interested in receiving. As an example, with DDM it is possible to subscribe only to ground vehicles in a particular geographical area or only to friendly aircrafts. For a more detailed description of how DDM is typically used, see the paper on design patterns for DDM [10].

DDM uses a concept called Dimension. An example of a Dimensions can be ForceID dimension that can take on values 0 to 3 indicating Neutral, Friendly, Opposing and Unknown. We can also split a battlefield into 16 different boxes and assign the number 0 to 15 for each box. By publishing data with associated dimension information, known as regions, and by also subscribing using regions, we can further filter the data.

Currently, all attributes of an object class can have their own dimension. This introduces several issues, as shown in Figure 6.

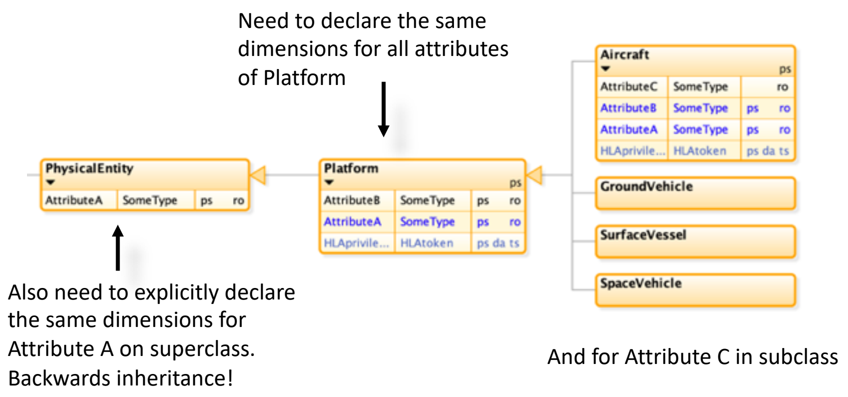
Need to declare the same dimensions for all attributes of Platform
Aircraft
AttributeC SomeType ro
AttributeB SomeType ps ro
AttributeA SomeType ps ro
HLAprivile... HLAtoken ps da ts
PhysicalEntity
AttributeA SomeType ps ro
Platform
AttributeB SomeType ps ro
AttributeA SomeType ps ro
HLAprivile... HLAtoken ps da ts
GroundVehicle
SurfaceVessel
SpaceVehicle
Also need to explicitly declare the same dimensions for Attribute A on superclass. Backwards inheritance!
And for Attribute C in subclass

Figure 6: DDM issues

1. In practice, federation developers want to apply the same DDM pattern to an entire object instance, like an aircraft. Otherwise, data will be received for different attributes of an object instance, based on different criteria. A federate may for example receive position, but not damage state for an aircraft at a given time, or the other way around.
2. If we create subclasses with additional attributes, these will not automatically have the same dimensions as the superclass, so no filtering will take place.
3. In case a federate wants to filter a class using certain dimensions, and the federate has inherited some attributes from a superclass, these dimensions must also be defined on the superclass. This can be seen as "backwards inheritance", a superclass needs to inherit attributes from its subclasses.

The new functionality works as follows, as shown in Figure 7.

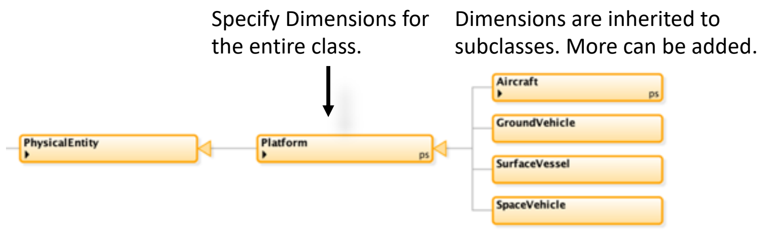
Specify Dimensions for the entire class.
Dimensions are inherited to subclasses. More can be added.
Aircraft
GroundVehicle
SurfaceVessel
SpaceVehicle
PhysicalEntity
Platform

Figure 7: Improved way to specify DDM dimension

1. Dimensions are specified for an object class or an interaction class, making these dimensions available for all inherited and locally defined attributes or parameters.
2. A subclass inherits the dimensions of its superclass.
3. A subclass definition can add more dimensions, making these available for all attributes on the class, including inherited attributes.

Note that the above change only applies to how available dimensions are specified in the FOM/OMT.

5. The New Reference Data Type

It is very common for attributes and parameters to reference other data. A simple example is shown in Figure 9 where an instance of a Car can use the attributes Driver and Passengers to reference instances of Person. Another example is the Fire interaction in the RPR FOM that may have parameters that specify the shooter and the target. A radio object instance may have an attribute that specifies which platform the radio is attached to. Such information can be seen as references between object instances and interactions that occur within the scenario. The problem with the current version of HLA is that these relationships are only specified in documents intended for the human reader, such as the GRIM for the RPR FOM. In some cases, it is also specified in the semantics field of the FOM. It's not possible, for example, for a generic data logger to display a firing event, automatically showing the associated shooter and target. Nor is it possible for a code generator to generate code that automatically delivers the firing event to the target object instance for processing.

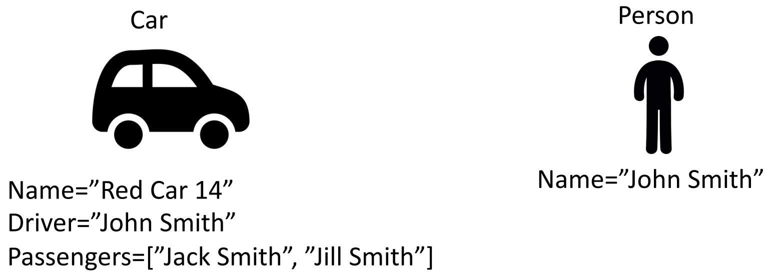

Figure 8: Implicit reference

This has been solved by introducing a new datatype called the reference data type, described in a new datatype table, as shown in Figure 9. Such a data type specifies the object class that is referenced as well as which attribute of the object that shall be used for locating the right instance. As an example, a reference data type could be defined for referring RPR FOM platforms by entity ID. This would then be used as a data type for the shooter and the target of the fire interaction.

Name	**Representation**	**Referenced Class**	**Referenced Attribute**	**Semantics**
PersonReference	HLAUnicodeString	HLAobjectRoot. Person	Name	Reference to an instance of Person
<name>	<data type>	<class name>	<attribute name>	<text>

Figure 9: Referencing object instances

Note that the reference data type only specifies the intent of the data. The HLA standard will not specify any automatic functionality for storing the key values, looking up the data or signaling errors. This is up to the federate to implement.

6. Conclusion

This paper summarizes a number of solutions that have been accepted into HLA 4. They solve a number of issues that have been discovered by projects using the current versions of HLA:

- For object and interaction classes they introduce a robust way to add attributes and parameters, in particular when using reference FOMs.
- For enumerations and variant records, they enable the addition of new values/variants.
- For DDM, they make it easier to specify a coherent set of dimensions across classes and class hierarchies.
- For attributes and parameters that reference object instances, they provide a clear and machine-readable way to specify this.

The authors and several members of the HLA 4 PDG strongly believe that this will make the use of FOMs and reuse and extension of reference FOMs considerably easier and more powerful.

7. 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] 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.

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

[6] SISO: "Space Reference FOM", SISO-STD-XXX, draft 3, www.sisostds.org

[7] Björn Löfstrand: "NATO Education and Training Network Federation Architecture and FOM Design", 12th CAX Forum, September 2017:

[8] 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.

[9] Björn Möller, Andy Bowers, Mikael Karlsson, Björn Löfstrand. "Extended FOM Module Merging Capabilities", Proceedings of 2013 Fall Simulation Interoperability Workshop, 13S-SIW-004, Simulation Interoperability Standards Organization, September 2013.

[10] Björn Möller, Fredrik Antelius, Martin Johansson, Mikael Karlsson. "Building Scalable Distributed Simulations: Design Patterns for HLA DDM", Proceedings of 2016 Fall Simulation Interoperability Workshop, 2016-SIW-003, Simulation Interoperability Standards Organization, September 2016

Author Biographies

BJÖRN MÖLLER is the President and co-founder of Pitch Technologies. He leads the development of Pitch's products. He has more than twenty-five 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 a M.Sc. in Computer Science and Technology after studies at Linköping University, Sweden, and Imperial College, London. He is currently serving as the chairman of the Space FOM Product Development group and the vice chairman of the SISO HLA Evolved Product Development Group. He was recently the chairman of the SISO RPR FOM Product Development Group.

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.