Getting started with FOM Modules

Björn Möller
Björn Löfstrand

bjorn.moller@pitch.se
bjorn.lofstrand@pitch.se

Keywords:

HLA Evolved, FEDEP, FOM Modules

ABSTRACT: *One of the most important new features in HLA Evolved is the ability to provide the Federation Object Model (FOM) in the form of several modules. While the standard provides a complete and stringent specification of these concepts, the purpose of this paper is to provide a user-friendly introduction as well as some guidelines for their practical use. A small example FOM is used in the paper to illustrate the principles.*

First of all it is important to understand where FOM modules fit into the FEDEP process and how it further enhances the process of building interoperable and reusable systems. Some general comments about best-practice for FOM modeling are also given.

The following aspects are then covered

- Understanding the predefined HLAstandardMIM module

- Use of standalone versus dependent modules

- Describing different aspects of a domain in different FOM modules

- Developing more general modules that can be reused across federations

- Handling of commonly used data types

- Extending existing modules with subclasses, including guidelines for using "scaffolding" classes

- Strategies for the required Switches table with suggested default values

- Providing FOM modules at Create versus Join time

Finally a list of additional ideas and aspects, that are not covered in the paper, is given for the reader to investigate on his own.

1. Introduction

The High-Level Architecture (HLA) was first developed in the mid 90's and standardized as HLA 1.3 [1] in 1998. It has since gone through two major revisions: IEEE 1516-2000 [2] and, soon to be released, IEEE 1516-2009 [3][4] a.k.a. HLA Evolved. The latest revision contains a large number of new features while maintaining all of the previously available functionality. Some examples of new features are fault tolerance support [5], Web Services support [6] and FOM modules [7]. A previous paper [8] lists all major and most minor technical improvements. Most of the new technical features relate to the HLA Interface Specification and thus the runtime behavior of HLA. FOM Modules, on the other hand, primarily relates to the HLA Object Model Template specification and, more importantly, to how to create information models for interoperability. This paper is dedicated to FOM modules, how they can be used and their relation to the overall process of building interoperable systems.

1.1 About this paper

This is an introductory paper, not a specification. It shows, step-by-step, how to use FOM modules on a basic to intermediate level. Some familiarity with HLA in general and FOM development is highly recommended. We have chosen not to give the full theory and specification upfront but to cover typical FOM module design. At the end of this paper we mention some more advanced topics. The full and

exact specification can be found in the HLA Evolved standard.

1.2 About design

The reader of this paper is most likely interested in getting an introduction to FOM modules with clear and unambiguous design rules. In practice, most of FOM module design, like any engineering, is about striking a balance. When we create a reusable information model, exactly how general and reusable should it be? To what degree shall it be easily adaptable to current simulation systems with all of their quirks and peculiarities? Can we predict all possible future uses of this information model? Shall we build a quick-and-dirty solution or an extremely clean model? To what extent shall we reuse and modify existing models?

There are no final answers to these questions. There are however certain factors that should be taken into consideration, for example:

- What is the expected life-span of the federation and the FOM?
- What user communities are involved, now and in the future? What are their current and expected requirements?
- What budget is available?
- Which subject matter experts are available?
- What is the planned reuse of the FOM?
- What existing efforts can we build upon?

2. FOMs – an Overview

When two or more systems are to interoperate they need to exchange information using a data model, sometimes called an Information Exchange Data Model (IEDM). In HLA this model is called the Federation Object Model (FOM). The FOM describes information that is to be shared between different federates. A FOM contains the following:

- An identification table describing things like the purpose, origin and author of the FOM
- Shared object classes with associated attributes.
- Shared interaction classes with associated parameters.
- Data type definitions for attributes and parameters
- Dimensions used for Data Distribution Management (DDM) filtering
- Time representation used
- Synchronization points used
- Data types for user supplied tags
- Transportation types, in most cases only Reliable and Best Effort
- Notes, typically annotations made during the FOM development
- Runtime switches for the RTI

We will now, step by step, look at some important properties of the FOM

The FOM is based on what other systems need to consume. A common misconception among inexperienced federation builders is that the developer of each system can decide what information they want to send to others. In practice the FOM must be based on what information other systems need to consume from your system. In most cases this information needs to be described in a more generalized form than the internal representation of each system.

The FOM is usually a simplification compared to the internal domain representation of each system. Each system will typically contain a rich internal data model of the reality it seeks to imitate. The information exchange data model focuses on what other systems need. There is no need to expose all the internal details required to produce a high fidelity model of an entity or a process. There are of course several exceptions to this statement. A data logger that has very limited internal representation is an example of such an exception.

Example: a simulation contains an aircraft model which includes a very detailed aerodynamics model. The federation where this simulation is used only looks at the position, speed and type of aircraft. Even if the simulation developer has a strong urge to expose intricate aerodynamics attributes they still should not go into the FOM.

We will now look at how the FOM is key to achieving interoperability and reuse of our simulations.

Using an information bus, such as HLA, with a shared information model, such as the FOM, enables composability. In some interoperability standards, such as web services, the focus is to connect exactly two systems using an information exchange data model that is specific to this pair of systems. If instead an information bus with a shared information exchange data model is used, it is possible to achieve composability. The producers and consumers of data do not need to have detailed knowledge about each other, only about the information being exchanged. Different sets of systems can be combined at different occasions for different purposes. New systems can replace older systems without affecting other systems. The total set of systems can easily grow over time.

A carefully designed FOM facilitates long-term reuse of simulations. If a FOM is designed to meet

the needs of a larger set of federations this means that it will be easier to reuse one system from one federation in another federation. The simulation is already adapted to the FOM. Reusing the FOM means that it is easier to reuse the systems. Note however that it is not enough. For each federation you still need to evaluate each federate, for example if it contains the right fidelity and if it produces and consumes the right information.

The FOM is part of a federation agreement. The federation agreement documents things like who is supposed to produce and consume information, when the data is provided, frequency, accuracy, etc. It also describes how the execution is started and stopped and how the scenario is loaded. It describes expected responses to various events from different federates, and many other things. The FOM gives a good description of the types and format of data that is exchanged but it may prove useless if you don't have access to the full federation agreement. Interoperability doesn't happen by chance and all federations build upon some sort of agreement between federate developers.

3. Some basic best practices for FOMs

There are a lot of best-practices for developing Information Exchange Data Models in general and FOMs in particular. Unfortunately a lot of this is scattered in literature and papers.

As a general advice it is wise to build upon generally accepted information models within your domain, in particular on the attribute or parameter level. There is no need to invent a new coordinate system if most of the participating systems already use lat/long or geocentric coordinates. In many cases there already exists enumerations, for example country codes. Some additional rules of thumb that are useful when developing FOMs are:

Do not use runtime identifiers, such as RTI handles, in the FOM. Imagine the case where one object, like a pilot, references another object, like an aircraft. A first approach may be to use the HLA object handle of the aircraft for this reference. If this data is logged and examined, for example for after action review purposes, this reference will be impossible to resolve and the relationship is lost. Using the "marking" attribute of the aircraft or another domain-oriented name or label is a better choice. It is also important that the federation agreements specifies how this reference is guaranteed to be a unique identifier.

Use records wisely. If several attributes are strongly related and an update of one attribute doesn't really make sense without updating another attribute, then put both of them in a record. A good example is a position that is described using latitude and longitude. If you, on the other hand, put more or less all of the information about an object in a record you will waste bandwidth and loose scalability. In addition to this you will prevent other federates from being able to subscribe only to the information that they really need.

Do not subclass without a good reason. It is easy to confuse the object class hierarchy in the FOM with object oriented languages where different subclasses exhibit different behavior. A better comparison for a FOM would be a corporate data base where data is stored and fetched by different applications. There are really only two main reasons for creating a subclass. The first is interest management where federates want to subscribe to one particular class (for example aircraft) but not another (for example submarine) given a common parent class (like vehicle). The second is that there is a requirement to add one or more attributes or parameters that is relevant to one class but not another.

4. A FEDEP perspective for FOM Module developers

Before developing a FOM it is important to understand the Federation Development and Execution Process (FEDEP) [9]. It provides a framework for developing federations. Federations can be built both from a combination of existing simulation systems, more or less adapted, and new systems. This makes the process slightly different from other processes which assume that everything is built from scratch.

The FEDEP isn't very detailed on a technical level. This is intentional since it is intended to harmonize with, not replace, your current development methodology and software engineering processes.

It is very important to understand the three first steps in FEDEP. They provide key information for your FOM module development. Later steps may provide additional information that can be used to further refine your FOM modules. Figure 1 shows the FEDEP process with some FOM module consideration are:

In step one, *Define Federation Objectives*, the overall goals and constraints of the federation is decided. What is our ambition with the federation? Is it a one-time event or in support of a full-fledge training environment? The objectives of the federation sets the stage for the basic design of the FOM. The level of flexibility, reusability and dynamics in terms of information exchange can often be determined from these initial objectives and metrics. It is important to understand how overall design aspects like reuse affects the way we design a FOM. Are we going to develop a federation that will have a short or long life span? How is this effort related to other efforts and what type of reuse is expected. Is there a requirement, a budget and subject matter experts available for creating more generally reus-

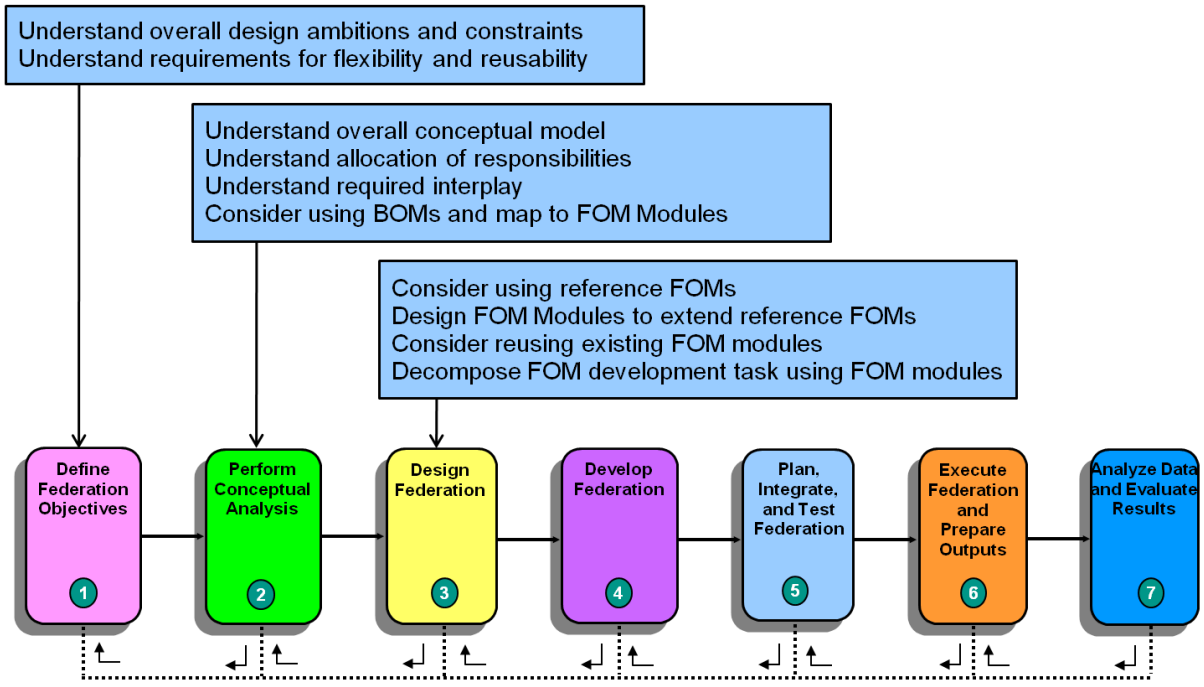

Figure 1: The FEDEP Process and FOM Modules

able FOM modules or should the effort only focus on short-term goals?

In step two, *Perform Conceptual Analysis*, the conceptual model of the simulation is developed based on one or more typical scenarios. Through the conceptual model all relations and interplay between simulated entities must be defined. When distributing the responsibility of modeling all that is defined in the conceptual model these relationships must be upheld. Any interplay between entities modeled in different federates must be manifested as information exchange between federates using constructs defined in the FOM. It is also important to note that the way relationships and interplay is defined in the conceptual model may be captured differently in the FOM depending on how the modeling responsibility is distributed among federates. In this step existing conceptual models and mappings to FOM constructs can be used. A good reuse element in this step are Base Object Models (BOM) [10] [11] that maps conceptual elements to FOM constructs.

In step three, *Design Federation*, the more technical FOM module development work begins. Simulators that are to become federates are selected. The required functionality and modeling responsibility are allocated to federates. It is now time to examine what information that each federate needs to consume from other federates to carry out its work. Information exchange requirements can be derived from the conceptual model, supporting BOMs and the federation specific allocation of modeling responsibilities. The design of the federation must be made carefully to meet all federation requirements. The FOM is defined to support the necessary information exchange between federates.

The use of reference FOMs promote reuse and, to some level, interoperability. When designing a federation it is important to carefully analyze the pros and cons of using an existing FOM. The trade-off between a FOM optimized for a specific federation and the use of a more standard FOM is a key decision point in the development of a federation. Details in the conceptual model specific to the federation are likely not to supported in a standard FOM and therefore require some FOM modifications or a completely different FOM. Although these customized FOMs increases interoperability they also reduce the opportunity of reusing existing federate based on the standard FOM.

In many organizations more and more internal FOM modules are expected to exist. These may be evaluated for potential reuse.

For the development of larger FOMs the task can be decomposed into different aspects of the FOM. Different teams can now take responsibility for the development of different FOM modules. Common concepts, data types in particular, may be shared in a common module.

During the rest of the FEDEP process the need for minor adjustments may arise. However, the major driver for FOM updates will be new requirements for the entire federation, for example if new types of objects are to be modeled or if new sequences of events are to be modeled.

5. General about HLA Evolved FOM Modules

In earlier versions of HLA the FOM was provided as one monolithic FOM. If you look at many real-life FOMs, for example the RPR FOM, you will notice that they are big and contain information in many different areas. HLA Evolved offers the opportunity to provide FOM data as modules. Still, the sum of all FOM modules must result in a valid FOM.

5.1 What to use FOM modules for

A long time ago people would develop software programs as one large piece of code. Time has changed and nowadays we develop in a more modular way, producing libraries, modules, classes and similar building blocks. The main principle for how a solution is divided into modules is usually called "separation of concern" where each building block should have, in some sense, a clear and separate purpose and responsibility. There are many similarities between code modules and FOM modules but also some slight differences. The FOM is mainly a contract between different systems. It describes important aspects of how these systems are to interoperate while the responsibility for carrying out any particular work lies within the systems. One important benefit of most types of modularity is that a solution can be developed, provided and reused in a modular way.

FOM modules can be used for several purposes, for example:

- You can have different working groups develop different parts of a FOM in a more convenient way. You may for example have a radio specialist group develop the "Radio FOM module" while the aircraft specialists develop the "Aircraft FOM module".
- You can put extensions to a reference FOM in a separate FOM module. This will prevent you from getting modified, "non-standard" reference FOMs. More importantly, when you merge federates or federations that use extended reference FOMs, it is easy to inspect what extensions that have been made and possibly to merge them.
- You can achieve extended reuse of some aspects of a FOM. If you want to promote a standardized way to start and stop all of your federations, irrespective of domain, you may put these interactions in a separate FOM module.
- You may also add more FOM modules to an already running federation, thus extending the scope of the FOM.

5.2 Content of FOM modules

Note that a FOM module may contain a subset of all tables, for example only data types. An identification table, describing things like the author and the purpose of the module, is always required for documentation purposes. This table will not be used when several FOM modules are combined.

Unless you are an experienced FOM developer your first FOM modules will typically contain the following HLA Object Model Template (OMT) data:

- An identification table
- Some shared object classes with attributes
- Some shared interaction classes with parameters
- Some data types for attributes and parameters
- Possibly a time representation table if your federation is time managed
- A switches table with runtime switches, as described later.

5.3 Using FOM Modules at runtime

There are two RTI services when you can provide FOM modules: the *Create Federation Execution* call and the *Join Federation Execution* call. One important improvement here is that you can indeed introduce new object and interaction classes into an already running federation. This, obviously, assumes that there is more than one federate that use these new classes.

How and when FOM modules are provided should be documented in your federation agreement. The two most obvious strategies are:

Strategy 1: Each federate supplies all FOM modules that they require upon Join. This is the preferred strategy for flexible federations where you want to be independent of join order. It supports early and late joiners and thus federates rejoining after a fault. The downside is that you need to make sure that all federates have access to the most recent version of the FOM modules that it required.

Strategy 2: One master federate creates and joins before any other federate. It loads all required FOM modules. This requires a highly coordinated startup but allows for a more strict centralized control.

You may also use strategy 2 for commonly used FOM modules and add more specialized FOM modules later.

6. A First Look at a Tiny FOM Module

First of all we will look at a really small FOM module. The module is illustrated in figure 2. It contains only three things:

- An object class called Train.
- An attribute Speed for the train
- A data type, SpeedInt64 for the speed.

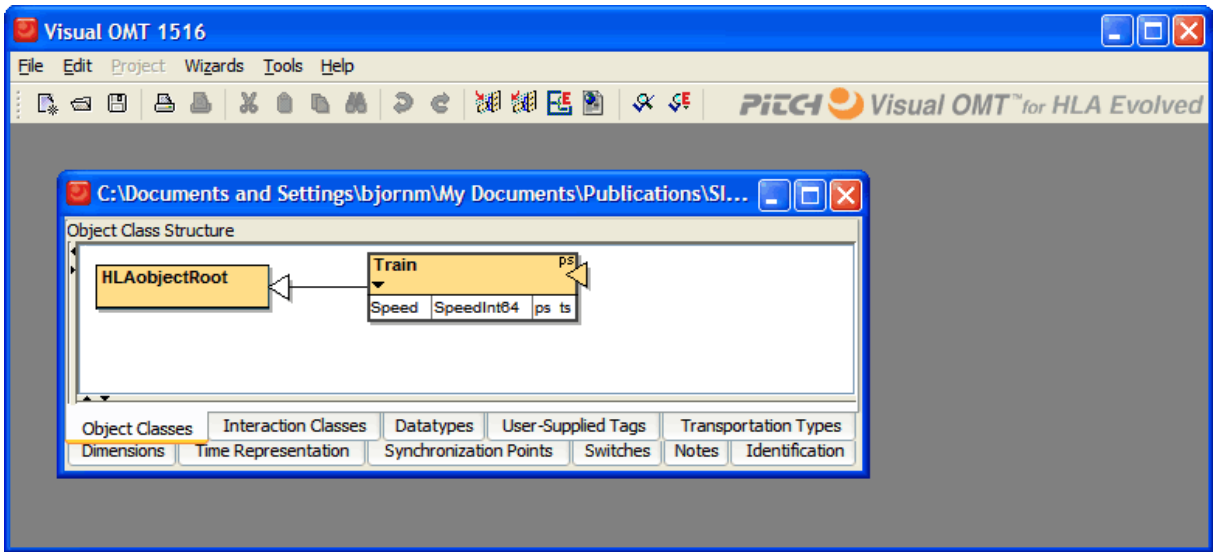

Figure 2: A Tiny FOM Module

In addition to this is contains an identification table. The FOM is shown in XML format in appendix A.

The most interesting thing here is what is not shown in the figure. The Manager.Federate and Manager.Federation class from the MOM isn't included. In fact, all of the MOM and the predefined HLA classes are excluded since they are available in a separate, pre-defined module called HLAstandardMIM, as described later. The class HLAobjectRoot doesn't have any attributes, not even the predefined HLAprivilegeToDeleteObject. The definition of HLAobjectRoot is an empty placeholder, called a *scaffolding* class definition. The purpose is simply to indicate where in the class hierarchy a new class shall be inserted. More about this later.

7. An example of Several FOM modules

We will now look at a sample federation to understand how FOM modules can be used. This is not a real life federation but it is partly inspired by real FOMs such as the RPR FOM [12] and the Dutch Railroad FOM [13]. The purpose of the federation is to simulate a national railway system. It consists of the following federates, as shown in figure 3:

TracksAndSignals provides a representation of the track topology and signals and their state across the nation. It loads the topology from a database.

PassengerTraffic simulates trains that carry passengers.

CargoTraffic simulates cargo trains

SafetyTypeA simulates an older train safety system that is in used in some older parts of the railroad system,

SafetyTypeB simulates a similar but more modern safety system.

TTC simulates the Train Traffic Control center operations.

RailVis is a visualization system that displays the state of the simulation.

Master is a federation execution manager that se-

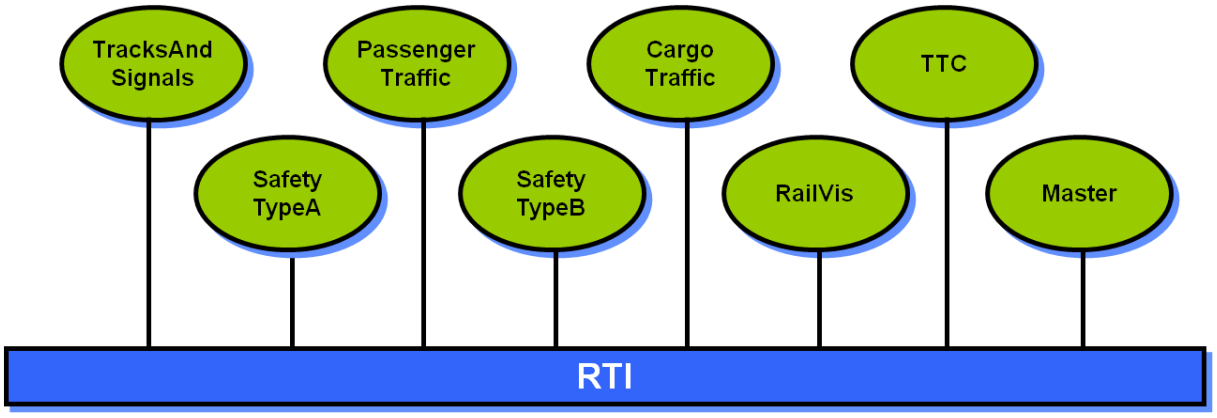

Figure 3: The Sample Railroad Federation

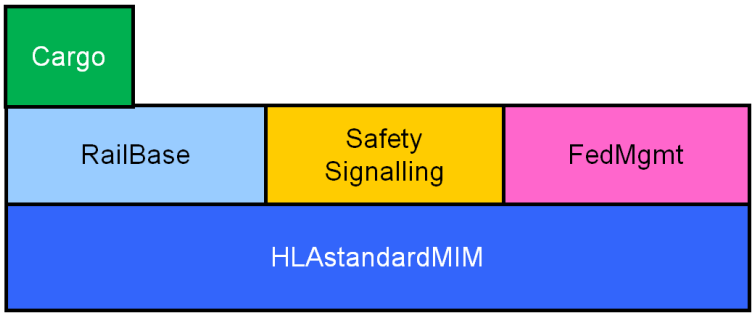

Figure 4: FOM Modules for the Example Federation

lects scenarios, sets the time and starts and stops the federation.

The following FOM modules, as shown in figure 3, are used for this federation:

RailBase contains the basic shared objects that are relevant for all railway simulation such as track segments, signal state, platforms and vehicles such as locomotives and wagons. It also contains the Switches table that the RTI needs as part of the Create Federation Execution call.

Cargo contains a number of specific additions for more advanced cargo transports, in particular heavy or hazardous transports.

SafetySignalling contains a number of more specialized signals that are sent between safety systems, signals and the TTC.

FedMgmt contains interactions that all systems have to obey such as setting scenario time, starting and stopping.

The HLA standard contains a lot of predefined concepts like data types (strings, integers, etc) as well as the roots for the two class hierarchies: Objects and Interactions. These concepts and more are stored in a separate FOM module so we will also need to add the following FOM module to our list:

HLAstandardMIM that contains predefined HLA concepts from the standard. We will build upon this module and use concepts like data types from this module, but we will not modify it in any way.

6.1 Working with object and interaction classes

The RailBase module contains some important classes like TrackSegment, Signal and Train. These three classes build upon the HLAObjectRoot class that resides in the HLAStandardMIM module.

In the Cargo FOM module we want to extend the Train object with the CargoTrain class. This class has a number of advanced properties for example for hazardous trains. The best way to do this is to provide a **scaffolding** (empty) definition for Train with no attributes or other information. We then create the subclass CargoTrain with all of its attributes. See figure 5.

The Cargo module is now said to be **dependent** on the RailBase FOM module. The use of a scaffolding definition has two advantages:

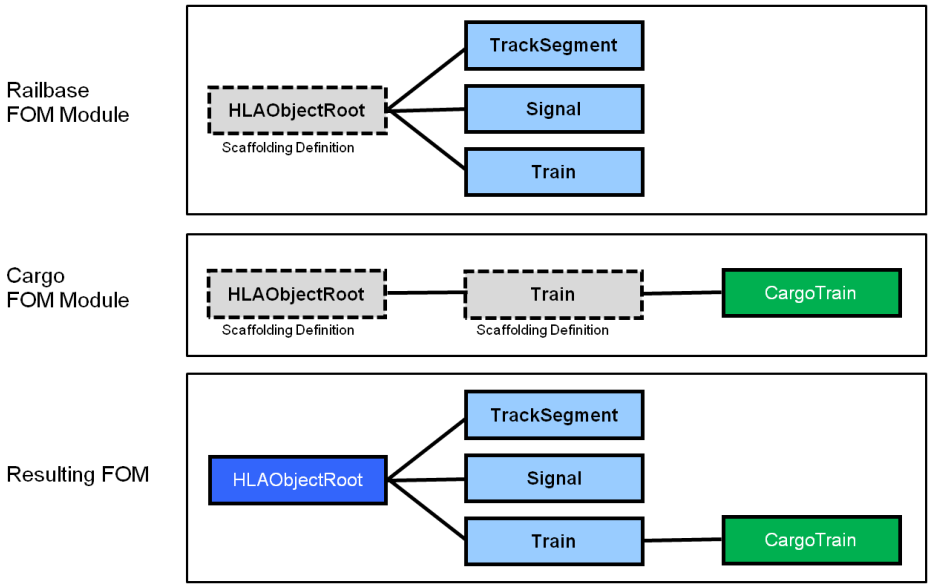

Figure 5: Object Classes in FOM Modules

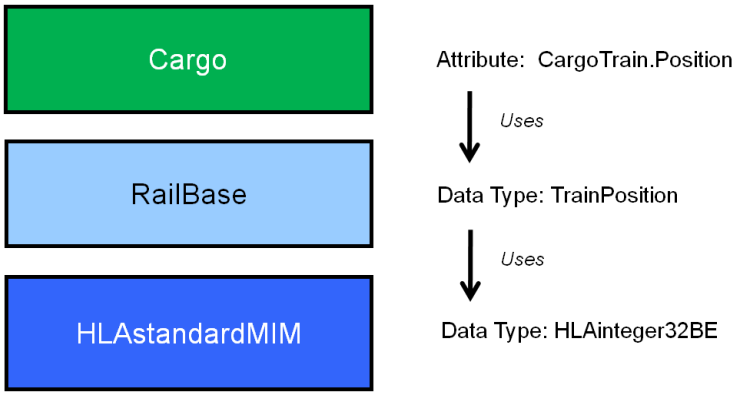

Figure 6: Data Types Dependencies between FOM Modules

1. There is no risk that we will end up with two conflicting definitions of Train
2. The team that is responsible for maintaining the RailBase module can keep refining it without updating the Cargo module.

The RailBase FOM module is a **standalone** module. While it does indeed reference predefined HLA concepts such as the HLAobjectRoot (using a scaffolding definition) it doesn't depend on any user-defined FOM module.

Consider the FedMgmt FOMmodule. This is a module that we may want to reuse across federations. In this case we may want to make it standalone or to have very few dependencies upon other modules.

7.2 Working with data types

No matter if you are simulating trains, signaling systems or train traffic control you will need to reference train IDs, track segments, positions along track segments and lat/long positions. In this example we have decided to put many of these common definitions in the RailBase FOM module. Since the module SafetySignalling uses these data types it will be dependent on the RailBase FOM module. Note that Cargo is dependent on RailBase because of both the Object Classes and the Data Types, as shown in figure 6.

Since we want to make the FedMgmt module standalone we should either avoid using data types from the TrainBase module or we should copy any definitions used into FedMgmt.

In some cases you may also want to put your data types into a completely separate module.

6.3 An introduction to combining FOM modules

The table in figure 7 contains a simplified summary of how FOM modules are combined. It describes data from two hypothetical FOM modules and what the result from merging these would be. Only a subset of the FOM data is used in this example. Note that not all tables need to be present in all FOM modules. Also note that there are some additional constraints that are described later in this paper or in the HLA Evolved specification.

In general the union of the data in all loaded FOM modules is produced. For a few cases the data is required to be equal. This means that the result is independent of load order, assuming that there are no conflicts. If conflicts exists the load operation will fail for all new modules that conflict with the already loaded modules. The entire *Create Federation Execution* or *Join Federation Execution* call will fail if any of the supplied FOM modules introduces a conflict.

8. General Best-practice Principles for FOM Module Development

When you get started with developing FOM modules the following are important things that you need to think of:

Don't skip the Identification table. At some point in time somebody else will probably need to know the origin of your FOM module. This type of data is stored in a well-structured way in the Identification table.

Clearly state the purpose of the FOM module. Why are you developing this module and what is its main purpose? What is the reason for putting this particular set of objects and interaction into a separate module? Is there a particular scope of expected reuse? Is there a particular type of federations that will benefit from it? Or is there a certain group of experts that will be responsible for developing and maintaining it?

Extend reference FOMs using separate modules. If your federation is based on an open or in-house standard FOM module, for example the RPR FOM, do not modify this module. Put all your project-specific extension in a separate FOM module. When your federate is later reused in another federation it is easy to analyze what particular extensions that were used in different projects. If you

Table	Principle	FOM Module A sample data	FOM Module B sample data	Result
Identification	Not combined	Version="2.0"	Version="1.6"	(nothing)
Object classes	Union of class tree	ObjectRoot Vehicle Car	ObjectRoot Vehicle Aircraft	ObjectRoot Vehicle Car Aircraft
Interaction classes	Union of class tree	InteractionRoot RadioMsg	InteractionRoot Explosion	InteractionRoot RadioMsg Explosion
Data types	Union	SpeedInteger AngleFloat64	SpeedInteger CountryEnum	SpeedInteger AngleFloat64 CountryEnum
Dimensions	Union	Country Airline	Country CargoType	Country Airline CargoType
Time Representation	Must be equal if present	HLAinteger64Time	(Not provided)	HLAinteger64Time
Synchronization Points	Union	ReadyToRun	StageB	ReadyToRun StageB
User-supplied tags	Must be equal if present	Update/Reflect= "HLAASCIIString"	Update/Reflect= "HLAASCIIString"	Update/Reflect= "HLAASCIIString"
Transportation Types	Union	LowLatency	Secure	LowLatency Secure
Switches	Must be equal if present	AutoProvide= "Enabled"	AutoProvide= "Enabled"	AutoProvide= "Enabled"
Notes	Union	Note1	Note3	Note1 Note3

Figure 7: An Introduction to Combining FOM Modules

come up with some generally useful extension you may also consider feeding this information back to the reference FOM module development group.

Don't redefine concepts. If you define the same concept, such as a class, a parameter or a data type, in several FOM modules the definitions needs to be identical. If you for example define the data type "Altitude" as a 32 bit little-endian integer, describing the altitude in meters, other modules shall either give exactly the same definition or shall provide no definition at all. The same applies to object classes with attributes, interactions with parameters, switches, time representations, synchronization points, user defined tags and more. A later section of this paper describes what problems you may run into when redefining concepts.

Think twice before repeating definitions. In general it is not a good idea to repeat definitions, such as an object class or a data type, from one FOM module in another module. There are, however, some exceptions. An acceptable reason for repeating concepts is when you have two modules that both need a particular data type and that sometimes will be loaded into the same federation and sometimes used separately. At the same time you introduce a risk that these definitions will deviate over time, making the FOM modules incompatible.

Prefix your Notes with the module name. If you develop a module that will be called "Flight" then consider naming the notes in the notes table "Flight -1", "Flight-2", etc. When several modules are merged for various purposes you will not need to renumber them across the FOM.

Don't forget the Switches table. The RTI requires certain runtime switches when it creates the federation execution. This means that you need to provide a FOM module that contains the Switches table in the Create Federation Execution call. In fact, this is really the only FOM data that is required when creating the federation execution since all the required and predefined concepts in the HLA standard is automatically provided by the RTI in the HLAstandardMIM FOM module. Read more about this module later in the next section of this paper.

Avoid putting the Switches table in a reference FOM. This will prevent federations from using other switches settings without modifying the reference FOM. During federation integration and testing it is useful to modify certain switches for debugging and tuning purposes.

Switch	Suggested default
autoProvide	True
conveyRegionDesignatorSets	False
conveyProducingFederate	False
attributeScopeAdvisory	False
attributeRelevanceAdvisory	False
objectClassRelevanceAdvisory	False
interactionRelevanceAdvisory	False
serviceReporting	False
exceptionReporting	False
delaySubscriptionEvaluation	False
automaticResignAction	CancelThenDeleteThenDivest

Figure 8: Suggested Default Switches

9. The Predefined HLAstandardMIM Module

There are a number of OMT items that always need to be present in a federation. These are contained in the HLAstandardMIM module, where MIM stands for Management and Initialization Module. The most important things are:

The **root classes** for objects and interactions, called HLAobjectRoot and HLAinteractionRoot. You will create your own classes as subclasses of these roots.

The **predefined data types**. Some of them, like HLAUnicodeString, can be used directly for the attributes and parameters that you intend to model. Some others are Basic Data types, like HLAinteger32BE, that cannot be used directly but can instead be used for defining your own data types.

The **Management Object Model** (MOM). These are object and interaction classes that can be used to monitor and control the federates and the federation. These classes are typically not used in more basic federates, but may still be used in simple federations where common federation management tools are used. The most commonly used object class here is the HLAfederate. By subscribing to attributes of this class you will get information about which the other currently joined federates in your federation are. The HLA standard provides an extensive specification of the MOM.

The HLAstandardMIM module is provided automatically for you by the RTI. This means that you don't need to load it when creating the federation execution. The only exception to this is for advanced users that want to extend the MOM.

10. Types of FOM modules

Formally, there are two types of modules:

Standalone modules that builds upon concepts in the HLAstandardMIM only or that don't build upon any other, user-provided module at all.

Dependent modules. These modules are add-on modules that depend on concepts defined in other modules.

The type of module can be indicated in the identification table of the FOM, as can be seen in Appendix A.

There are two additional relationships that are not part of the standard but still useful to understand:

Imagine a module that uses the Start interaction from the RPR FOM. To be useful without the RPR FOM module it repeats this interaction. This FOM module is said to be **compatible** with the RPR FOM module without being dependent. One trivial case when this happens is when modules have nothing in common. The more interesting case is when they contain identical (repeated) definitions of the same concepts (such as data types)

Imagine a FOM module that only contains a Warning interaction. A federate is supposed to send this interaction whenever it detects an aircraft from the opposing side. The use of certain other FOM modules is assumed in the federation agreement. In this case the module is standalone but the federation agreement builds upon several modules. These modules can be considered to be **related**.

11. Switches

The FOM contains a table with runtime switches for the RTI. This table has to be provided in the Create Federation Execution call, or to be more exact: It has to be present in at least one of the FOM modules that are supplied.

11.1 Suggested default values for switches

Here are some suggested default values for these

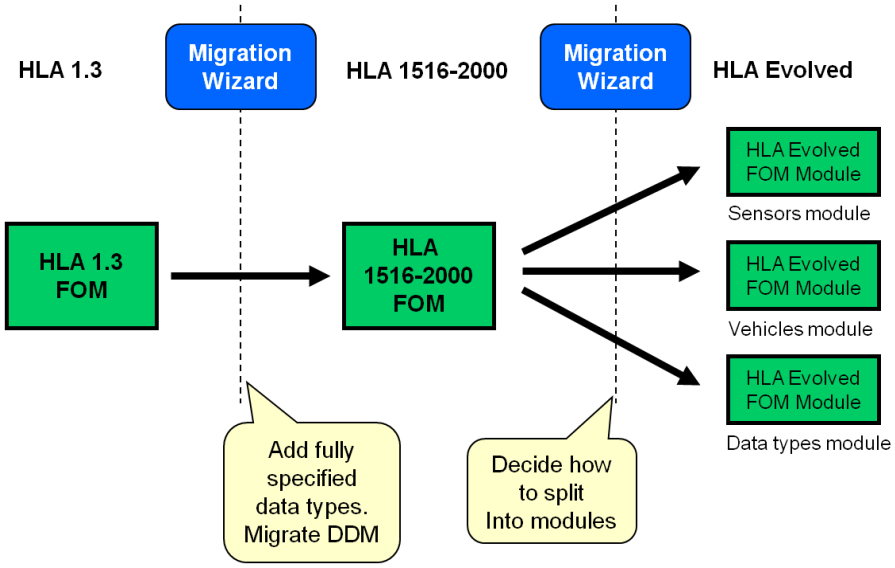

Figure 9: Migrating a FOM to HLA Evolved FOM Modules

switches, see figure 8. Note that these are just suggestions. You need to investigate the design patterns of your federation as well as the service usage of participating federations to find out exactly what's right for you.

The **autoProvide** switch makes the RTI request that attribute updates are provided when a new federate discovers an object instance from another federate. To optimally support late joiners (including federates rejoining after a temporary fault occurred) this switch should be set to true. To fully support late joiners all federates should also be required to respond to the Provide Attribute Values callback.

The **conveyRegionDesignatorSets** switch makes the RTI convey the DDM information for example for attribute updates that are delivered to a federate. This may be useful for debugging. In many cases you may simply want to set this to false.

The **conveyProducingFederate** switch makes the RTI convey the ID of the federate that provided for example an attribute update or an interaction. This is useful for example for logging data from one particular federate. For less advanced federations you may want to set this to false.

There are four **Advisory** switches. They control whether the RTI should send some extra callbacks to a federate indicating whether the data that is produced will actually be used by any other federate. This may be a very powerful base for optimizing more advanced federations. For less advanced federations you may want to set them to false.

There are two **Reporting** switches that may be used to inspect how the RTI is being called by federates and what exceptions that occur. For less advanced federations you may want to set them to false.

The **delaySubscriptionEvaluation** switch enforces a very strict behavior on the RTI. The RTI will need to check whether a certain type of data is still subscribed just before it is delivered from the Local RTI Component (LRC) to the federate code. This may have changed from the moment the data was sent from another, producing federate. For many federations you may want to set this switch to false.

The **automaticResignAction** switch controls the behavior when a federate is lost due to a fault. The RTI will then perform a resign on behalf of the lost federate, using this resign action. To maximum "clean up" after the lost federate the value CancelThenDeleteThenDivest is suggested.

11.2 Strategies for providing the switches

We suggest using one of the following two strategies:

Strategy 1: Provide the switches table as part of the most general FOM module that is provided in the Create Federation Execution call. This is typically the FOM module that provides the base concepts for a domain. This strategy should however be avoided if a reference FOM module is used for basic concepts. Do not put a Switches table in a reference FOM since such a FOM should be read-only and you may want to experiment with the switches for debugging purposes.

Strategy 2: Provide the switches table in a separate FOM module. This is the most general approach but it may be overkill for more basic federations.

12 Errors and Redefinitions

What if you accidentally redefine a concept that already exists in another module? You may encounter one of the following situations:

A) The RTI will refuse to load the FOM module that contains the redefinition of an already loaded concept. An example of this is

when two different modules define different attributes for the same class. This error is encountered when the second of the two conflicting modules are loaded into the Federation Execution. Note that he RTI only looks at a subset of the FOM module data so not all conflicts will be discovered by the RTI.

B) Some federates will crash or behave incorrectly. Imagine if one FOM module defines the data type Altitude as having a 32 bit integer representation and another module defines the same data type as having a 64 bit integer representation. Assuming that different federate developers looked at different modules during the development a crash will most likely occur. The federate that received a 32 bit integer but expected a 64 bit integer will fail to decode it.

C) More or less subtle errors and inconsistencies will occur in the simulation. Imagine if one FOM module defines the data type Altitude and states the units to be meters and another FOM module states the units to be feet. Imagine that different federate developers develop their systems according to these two different definitions. This mistake may be difficult to discover until the simulation output is carefully examined.

D) For a few trivial cases, like misspellings in the semantics field of a data type, nothing bad will happen.

13. Migrating FOMs to HLA Evolved

There are COTS tools readily available that facilitates the FOM migration. This process is shown in Figure 9.

If you currently have an HLA 1.3 FOM you are recommended to first migrate it to the HLA 1516-2000 format. The biggest difference here is if you are using DDM. This has been redesigned and standalone Dimensions have replaced Routing Spaces. During this conversion process you are also highly recommended to add fully specified Data Type descriptions.

The step from HLA 1516-2000 to HLA Evolved is quite simple using a COTS tool. All you will need to do is to specify exactly which tables as well as which object classes and interaction classes that you want to go into which module. The rest of the conversion is automated. While this is technically simple and large FOMs can be migrated in minutes, the reader is still recommended to carefully consider the design recommendations provided in this paper.

14. More Areas to Explore

This paper is intended to give an introduction to FOM modules, not a complete specification. There are a number of areas for the reader to further explore:

A) Study the HLA Evolved specification to understand the exact specification of FOM modules in general and FOM module merging in particular.

B) Study the Base Object Models (BOM) that provides a structure workflow from conceptual models to FOMs and FOM modules.

C) Since FOM modules can be added when a federate joins, it is possible to extend the FOM of already running federation over time. Figure out some new opportunities that this enables.

D) In a large FOM project you may want to put your data types in a separate FOM module. What are the pros and cons of this approach?

E) When you load FOM modules into a federation using the Create or Join call, the RTI will check that the FDD part of the FOM modules can be merged. Study the HLA specification to find out exactly which parts that the RTI checks.

F) Study the HLA Evolved Interface Specification and the Create Federation Execution call in particular to find out how you provide a user-extended HLAstandardMIM.

15. Conclusions

This paper has provided an introduction to the practical usage of FOM modules, including several examples.

FOM Modules is one of the most important additions to the HLA standard in the HLA Evolved version. They enable us to develop and reuse FOMs in a more modular fashion, to separate temporary adaptations from reference FOMs and to split the development work between different teams. In the long run they will also enable us to reuse partial federation agreements and federation design patterns.

References

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

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

[3] IEEE: "IEEE 1516-2009, High Level Architecture (HLA)", www.ieee.org, To be published.

[4] Roy Scrudder, Gary M. Lightner, Robert Lutz, Randy Saunders, Reed Little, Katherine L. Morse, Björn Möller. “Evolving the

High Level Architecture for Modeling and Simulation". Proceedings of the 2005 Interservice/Industry Training, Simulation & Education Conference, Paper No. 2157, National Training Systems Association, December 2005.

[5] Björn Möller, Björn Löfstrand, Mikael Karlsson. "Developing Fault Tolerant Federations using HLA Evolved" Proceedings of 2005 Spring Simulation Interoperability Workshop, 05S-SIW-048, Simulation Interoperability Standards Organization, April 2005.

[6] Björn Möller, Staffan Löf. "A Management Overview of the HLA Evolved Web Service API", Proceedings of 2006 Fall Simulation Interoperability Workshop, 06F-SIW-024, Simulation Interoperability Standards Organization, September 2006.

[7] Björn Möller, Björn Löfstrand, Mikael Karlsson. "An Overview of the HLA Evolved Modular FOMs", Proceedings of 2007 Spring Simulation Interoperability Workshop, 07S-SIW-108, Simulation Interoperability Standards Organization, March 2007.

[8] Björn Möller, Katherine L Morse, Mike Lightner, Reed Little, Robert Lutz. "HLA Evolved – A Summary of Major Technical Improvements", Proceedings of 2008 Spring Simulation Interoperability Workshop, 08F-SIW-064, Simulation Interoperability Standards Organization, September 2008. Joint paper with SAIC, AEgis Technologies, Carnegie Mellon University/Software Engineering Institute and Johns Hopkins University/Advanced Physics Lab.

[9] IEEE: "IEEE 1516.3-2003 IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP)", www.ieee.org

[10] SISO, "BOM Template Specification", SISO-STD-003-2006, SISO, 31 March 2006.

[11] Björn Möller, Paul Gustavson, Bob Lutz, Björn Löfstrand. "Making Your BOMs and FOM Modules Play Together", Proceedings of 2007 Fall Simulation Interoperability Workshop, 07F-SIW-069, Simulation Interoperability Standards Organization, September 2007

[12] SISO, "Real-time Platform Reference Federation Object Model 2.0 ", SISO-STD-001 SISO, draft 17

[13] Fred van Lieshout, Ferdinand Cornelissen, Jan Neuteboom, Björn Möller. "Simulating Rail Traffic Safety Systems using HLA 1516", Proceedings of 2008 Euro Simulation Interoperability Workshop, 08E-SIW-069, Simulation Interoperability Standards Organization, June 2008.

Author Biographies

BJÖRN MÖLLER is the vice president and co-founder of Pitch, the leading supplier of tools for HLA 1516 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. Björn Möller holds an MSc 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 Development Group.

BJÖRN LÖFSTRAND is Manager of Modeling and Simulation Services at Pitch Technologies. He has been involved in the development of several M&S standards and has been working with HLA federation development and tool support since 1996. Björn holds an M.Sc. in Computer Science from Linköping Institute of Technology. Recent work includes developing FOM and Federation Design for NATO Education and Training Network.

Appendix A: The Tiny FOM Module

```
<?xml version="1.0" encoding="UTF-8"?>
<objectModel xmlns="http://www.sisostds.org/schemas/IEEE1516-2009"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.sisostds.org/schemas/IEEE1516-2009
         http://www.sisostds.org/schemas/IEEE1516-DIF-2009.xsd">
  <modelIdentification>
    <name>Small railroad FOM</name>
    <type>FOM</type>
    <version>1.0</version>
    <modificationDate>2009-08-12</modificationDate>
    <securityClassification>Unclassified</securityClassification>
    <purpose>Train training</purpose>
    <applicationDomain>Railroad</applicationDomain>
    <description>Basic train data exchange for analysis applications</description>
    <poc>
      <pocType>Primary author</pocType>
      <pocName>Björn Möller</pocName>
      <pocOrg>Pitch Technologies</pocOrg>
      <pocTelephone>0046 13 13 45 45</pocTelephone>
      <pocEmail>info@pitch.se</pocEmail>
    </poc>
    <reference>
       <type>Standalone</type>
       <identification>NA</identification>
     </reference>
  </modelIdentification>
  <objects>
    <objectClass>
      <name>HLAobjectRoot</name>
      <objectClass>
        <name>Train</name>
        <sharing>PublishSubscribe</sharing>
        <semantics>Generic train</semantics>
        <attribute>
          <name>Speed</name>
          <dataType>SpeedInt64</dataType>
          <updateType>Conditional</updateType>
          <updateCondition>On change</updateCondition>
          <ownership>NoTransfer</ownership>
          <sharing>PublishSubscribe</sharing>
          <transportation>HLAreliable</transportation>
          <order>TimeStamp</order>
          <semantics>Speed of the train</semantics>
        </attribute>
      </objectClass>
    </objectClass>
  </objects>
  <interactions>
    <interactionClass>
      <name>HLAinteractionRoot</name>
    </interactionClass>
  </interactions>
  <dimensions/>
  <tags/>
  <transportations/>
  <dataTypes>
```

```
    <basicDataRepresentations/>
    <simpleDataTypes>
      <simpleData>
        <name>SpeedInt64</name>
        <representation>HLAinteger64BE</representation>
        <units>mph</units>
        <resolution>1</resolution>
        <accuracy>5</accuracy>
        <semantics>Simulated speed</semantics>
      </simpleData>
    </simpleDataTypes>
    <enumeratedDataTypes/>
    <arrayDataTypes/>
    <fixedRecordDataTypes/>
    <variantRecordDataTypes/>
  </dataTypes>
</objectModel>
```