An Overview of the HLA Evolved Modular FOMs

Björn Möller
Björn Löfstrand
Mikael Karlsson
Pitch Technologies
Nygatan 35
SE-582 19 Linköping, Sweden
+46 13 13 45 45
bjorn.moller@pitch.se
bjorn.lofstrand@pitch.se
mikael.karlsson@pitch.se

Keywords:
Interoperability, HLA Evolved, FEDEP, OMT, FOM, Modular FOM, FOM Agility, RTI

ABSTRACT: *The HLA Federation Object Model (FOM) describes the information that is to be exchanged during the execution of a federation. When the federation execution is created, the RTI loads the FOM. This enables the participating federates to refer to the object model, for example when publishing and subscribing to information.*

As in many other cases, a monolithic architecture restricts the speed, flexibility, and accuracy of the development process which in turn affects the resulting FOM. As part of HLA Evolved the concept of FOM modules have been added. The FOM is thus broken down into composable modules that can build upon each other. This allows a subset of federates to specify what data they need to exchange upon joining, in addition to the initially loaded FOM.

The main advantage is the increased flexibility during runtime and, not the least, during the development process. Federation developers can build new modules that extend reference FOMs without modifying them. Reference FOMs can be developed by several smaller communities in different domains or for different local or national extensions. Temporary additions will not result in a multitude of similar FOMs.

It will also now be possible to have long-running or persistent federations in virtual arenas where new capabilities and types of shared information can be added over time.

The modular FOMs are expected to revitalize a long needed development of shared information models. This is an important and necessary next step towards even higher degrees of interoperability that build on top of the High Level Architecture.

1. Introduction

The Federation Object Model (FOM) is a key part of the HLA standard, both for the older DMSO HLA 1.3 version [1] and the newer IEEE 1516-2000 [2] version. A new version of the HLA IEEE 1516 standard, with the working name "HLA Evolved" [3] is under development.

In the world of technology the requirements are always a moving target. As new HLA version evolves, so does the FOM concept.

Important improvements were made already when going from HLA 1.3 to HLA IEEE 1516 in the year of 2000. One example is that XML [4] structuring was introduced to synergize with information description standards and meta-data standards.

As technology evolves, HLA in 2007 needs to meet new requirements, not the least for more dynamic development and deployment of federations These include GIG style development and deployment. In these situations FOMs needs to be loaded dynamically and incrementally, which adds the requirement for FOMs to be structured into modules.

2. To Interoperate You Must Agree

A good way to understand simulation interoperability is to take a look at the Federation Development and Execution Process (FEDEP) [5] (IEEE 1516.3), which is a recommended practice for developing HLA based federations.

To make systems interoperate you first need to determine the objectives (FEDEP step 1). You then define what to model based on scenarios and a detailed conceptual model (FEDEP step 2). Next, you start to design the federation (FEDEP step 3) which includes determining what federates to use and how the actual federation is going to interoperate.

In order to interoperate effectively we need to make a federation agreement. All federate developers need to agree on several things:

- The information to be exchanged. This includes syntax and semantics, resolution in time and space, update conditions, etc.
- Responsibilities of federates for production and consumption of information.
- Synchronization issues like startup, shutdown and check pointing.
- Logical responses and sequences of interactions.
- Common algorithms (like line-of-sight).
- Common information (like synthetic environment).

Several additional things may also need to be agreed upon.

One of the most important parts of the federation agreement is the FOM which is a machine-readable specification of what type of data to exchange. Concepts from the FOM, such as names of shared object classes, will be used both by the participating federates as well as the RTI and possibly other tools.

3. What’s in a FOM?

The exact format of a FOM is described in the HLA Object Model Template. The following is an informal overview of what a FOM contains:

- **Identification**. This is a general description of the FOM – domain, purpose, developer and so on.
- **Object classes and attributes**. Shared object classes are described, for example Vehicle. This is an object class hierarchy similar to what is used in object oriented programming. Subclasses can be introduced, for example an Aircraft that has additional attributes.
- **Interactions and parameters**. These are instantaneous events that are shared between federates. Just like Object classes they are structured into a hierarchy. Note that interactions differ from traditional Remote Procedure Calls (RPC) since they are not necessarily directed to a specific entity and can be resolved to different receiving entities in different receiving federates, for example to a specific aircraft instance or to a universal data logging object instance.
- **DDM information**. This is a description of data to be used for value based filtering, for example nationality, latitude and longitude. The older HLA 1.3 standard provides predefined groupings called routing spaces whereas the HLA IEEE 1516 standard provides a more flexible set of dimensions that can be combined dynamically at runtime.
- **Data types** for attributes, parameters etc. HLA 1.3 provides a list of commonly used simple data types, like float, integer and string. The HLA 1516-2000 standard adds support for a full and unambiguous specification of data types down to the bit level, including complex data types, cardinality and padding rules. A number of predefined data types are also provided.
- **Transportation types** that can be used for attribute updates and interactions. This table contains the predefined data types HLAreliable and HLAbestEffort. It can be extended in case an RTI provides additional transportation types.
- **Update rates**. In HLA Evolved updates for the same object instance attribute can be provided with different update rates to different federates. This table specifies these update rates.
- **Synchronization points**. This table describes global synchronization points that can be used for example for synchronizing start up or scenario execution.
- **User defined tags**. For certain RTI services, like updates, application specific meta-data can be provided. The user defined tags table provides the data types for this data.
- **Time representation**. This table provides the data types for the simulated, logical time, as well as for time intervals.
- **Switches**. This table provides a number of standardized runtime switches that controls the behavior of the RTI.
- **Notes**. These are human-readable comments that may refer to one or more classes, attributes, data types, etc.

There are a few additional things worth mentioning. There is an OMT table called **Lexicon** that describes the semantics of OMT data, for example attributes, parameters, etc. In the XML format as well as in many OMT editors they are integrated into the corresponding data objects instead of being stored separately.

The HLA Evolved OMT also specifies a table called **Conformance Statement** which is mainly used for a Simulation Object Model (SOM) to describe what HLA services that are used.

3.1 The FOM and the FDD

The RTI does not use all of the data from the FOM at runtime. Only a subset, called Federation Document

Data, FDD, is used. The minimum FDD is described in the standard and can be summarized as:

- Object classes with attributes (transportation, order, and available dimensions only)
- Interaction classes with parameters (transportation, order, and available dimensions only)
- Dimensions
- Transportation types
- Update rates (new in HLA Evolved)
- Switches

It is however permissible for an RTI to use a larger set of data from the FOM as FDD. An RTI is however required to be able to operate with the minimum FDD.

Also note that the RTI does *not* use the data types in the FOM.

3.2 What's not in a FOM?

A mistake that is sometimes made about the FOM is the class versus instance description. The FOM contains information about object classes, both scenario classes (aircraft, ground-vehicles, etc) and meta-classes (federate and federation). However, it does *not* contain any information about how and when these will be instantiated. Such a description may be useful when describing scenarios to be executed or deployments to be made. The absence of instances allows a FOM to be reused between different scenarios and deployments.

4. Format and Compliance of a FOM

The HLA 1.3 OMT uses a LISP [6] style format as can be seen in the following example:

```
(Class (ID 1)
  (Name "Vehicle")
  (PSCapabilities PS)
  (Description "Any vehicle")
  (Attribute (Name "Speed")
       (DataType "float")
       (Cardinality "1")
       (Units "mph")
       (Resolution "1")
       (Accuracy "5")
       (AccuracyCondition "Always")
       (UpdateType Periodic)
       (TransferAccept N)
       (UpdateReflect UR)
       (Description "Speed of the vehicle")))
```

Figure 1: Sample HLA 1.3 OMT Data

The FDD is stored in a separate file, called the FED file. The corresponding FDD data may look like this:

```
(class Vehicle
  (attribute Speed best_effort receive ))
```

Figure 2: Sample HLA 1.3 FDD Data

HLA IEEE 1516-2000 introduces the use of XML, which also includes Unicode support. It also merges the FOM and the FDD files into one file and separates out data type information.

```
<objectClass name="Vehicle"
   sharing="PublishSubscribe"
   semantics="Any vehicle">
   <attribute name="Speed"
              dataType="SpeedInteger"
              updateType="Periodic"
              ownership="NoTransfer"
              sharing="PublishSubscribe"
              transportation="HLAbestEffort"
              order="Receive"
              semantics=
              "Speed of the vehicle"/>
</objectClass>

<simpleData name="SpeedInteger"
            representation="HLAinteger32BE"
            units="mph"
            resolution="1"
            accuracy="5"
            semantics=
            "Integer used for speed data"/>
```

Figure 3: Sample HLA 1516-2000 OMT Data

HLA Evolved further develops the XML format to allow for user defined elements in the XML document. It also introduces support for FOM and SOM modules as will be described later.

```
<objectClass>
 <name>Vehicle</name>
 <sharing>PublishSubscribe</sharing>
 <semantics>Any vehicle</semantics>
  <attribute>
   <name>Speed</name>
   <dataType>SpeedInteger</dataType>
   <updateType>Periodic</updateType>
   <ownership>NoTransfer</ownership>
   <sharing>PublishSubscribe</sharing>
   <transportation>HLAbestEffort
                      </transportation>
   <order>Receive</order>
   <semantics>Speed of the vehicle
                      </semantics>
  </attribute>
</objectClass>
-
<simpleData>
 <name>SpeedInteger</name>
 <representation>HLAinteger32BE
                       </representation>
 <units>mph</units>
 <resolution>1</resolution>
 <accuracy>5</accuracy>
 <semantics>Integer used for speed data
                </semantics>
</simpleData>
```

Figure 4: Sample HLA Evolved OMT Data

4.1 FOM Compliance

As the XML format for the FOM was introduced in IEEE 1516-2000 an XML Document Type Definition (DTD) [7] was provided to formally describe the format. XML technologies have since developed and XML Schemas [8] techniques have matured. These allow for a considerably more detailed specification of XML document validation rules. One example of this is referential integrity. Compliance with an XML schema can easily be checked with standard XML software.

For HLA Evolved, three levels of schema based compliance testing have been defined for a FOM:

The DIF Format Schema (IEEE1516-2007-DIF). This schema describes the basic format to be followed. No completeness or consistency is required. It is useful when exchanging FOMs or FOM modules that are under development. Remember that most FOMs are under development for a considerable period of time.

The FDD Schema (IEEE1516-2007-FDD). The FDD is the subset of the FOM that is used for initializing an RTI as described above. If a FOM conforms to the FDD schema it means that it can be used for initializing a federation, that is, it can be loaded into an RTI.

The OMT Compliance Schema (IEEE1516-2007-OMT). This schema imposes additional constraints, for example for referential integrity between non-FDD data.

Note however that the full compliance of a FOM document, as described in the OMT standard, still cannot be verified using XML Schema technologies. One obvious example is that user-provided identifiers may not start with the string “HLA”.

5. Modular FOMs in HLA Evolved

HLA Evolved introduces three important module concepts:

FOM Modules. These are modules that contain FOM information and that follow the OMT DIF Schema. Note that not all tables are required to be present. When FOM Modules are loaded into a federation they shall, when combined, form a valid FOM, similar to the 1516-2000 FOM. A MOM Module is also required in a FOM.

MOM Module. This is a module that contains the MOM information, similar to the one in 1516-2000. It also contains some additional, standardized information such as predefined data types, transportation types, dimensions, etc. There is a standard MOM module that may be provided automatically but it is also possible to provide extended MOM modules.

SOM Modules. In much the same way as FOMs can be built by assembling FOM Modules, SOMs can be built from SOM modules.

A simple example of how FOM modules can be combined is shown here.

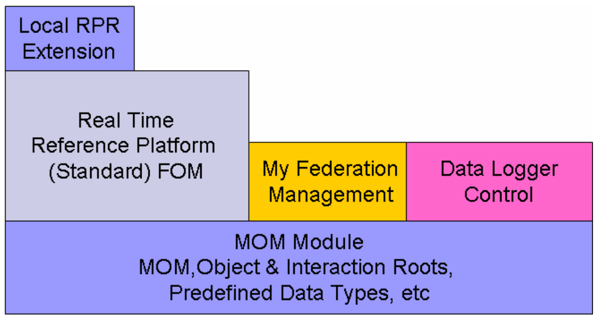

Local RPR Extension
Real Time Reference Platform (Standard) FOM
My Federation Management
Data Logger Control
MOM Module
MOM,Object & Interaction Roots,
Predefined Data Types, etc

Figure 5: FOM Module example

A more detailed example of how classes within FOM modules can build upon each other is shown here:

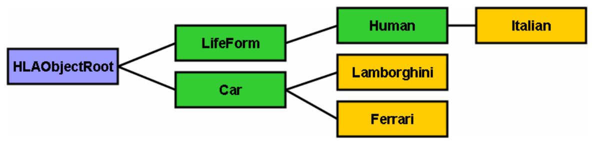


Figure 6: Classes from FOM Modules

The blue HLAobjectRoot comes from the MOM Module. The green entity classes come from one FOM Module. The Italian concepts, here shown in yellow, come from another FOM module.

5.1 Combining FOM Modules

Several FOM Modules can be combined into a new FOM and existing, monolithic FOMs can be decomposed into FOM modules. The following principles for combining the elements of FOM modules are used:

Principle A. Exact equivalency between tables. If the table is present in more than one FOM module then the tables shall be equivalent. Principle A applies to Switches, Time representation and User Supplied Tags.

Example: One FOM module specifies that a 64 bit integer time representation shall be used. Another FOM module specifies that a 64 bit float time representation shall be used. It will not be possible to combine these FOM modules. The underlying reason is that there will be no actual agreement between federates about what logical time representation that shall be used.

If, on the other hand, the other FOM module does not provide a time representation, this means that logical time is not a concern for federates that only support that

FOM module. This may, for example, be federates that monitor the state of the federation and don’t participate in any time managed operations.

Principle B. Produce the union of table elements, duplicates shall be equivalent. If the table is present in more than one FOM module the union of the elements (rows) of the table is produced. If any two elements have the same identifier (name) their content (rows) shall be equivalent. Principle B applies to Dimensions, Data types, Transportation types, Update rates, Synchronization points and Notes.

Example: One FOM module specifies an integer datatype “FuelInt” for fuel level and a fixed record data type “PositionRec” with latitude, longitude and altitude for position. Another FOM module specifies a float data type “SpeedFloat” for speed and the exact same “PositionRec” as in the first FOM. When these FOM modules are combined the resulting FOM will contain all three data types.

If, on the other hand, the second FOM module contains a different definition of the “PositionRec”, for example using geocentric coordinates, it will not be possible to successfully combine these FOM modules. The federation builders will need to go back and agree on how to represent positions or, possibly, provide different representations for different purposes.

Principle C. Produce the union of hierarchical elements, duplicates shall be equivalent or scaffolding (“empty placeholders”). Similar to principle B but this is used for elements in a hierarchy, namely object classes and interaction classes. The hierarchies of different modules are added. Consider the following object class hierarchies:

FOM Module A:

```
HLAobjectRoot
   Vehicle (Name, Speed)
```

FOM Module B:

```
HLAobjectRoot
   Vehicle – scaffolding definition
     AirCraft (Altitude, Airline)
```

Resulting FOM

```
HLAobjectRoot (PrivilegeToDeleteObject)
   Vehicle (Name, Speed)
     AirCraft (Altitude, Airline)
```

Figure 7: Combining Object Classes

If a subclass, for example Aircraft, is going to be added to the superclass Vehicle that is originally defined in FOM module A, it is possible to either *repeat* the full vehicle definition or to provide a *scaffolding* (“empty placeholder”) definition containing just the name. Note that if a class definition is repeated it must contain the exact same set of attributes or parameters. A FOM module cannot add more attributes to an already defined class. This may instead be achieved by creating a subclass. Principle C applies to object classes with attributes as well as to interaction classes with parameters.

Principle D. Table not used. This is a special case for metadata about the FOM module, namely the identification table. There is no way to algorithmically determine the purpose, version or point of contact for a FOM based on the contributing FOM modules. [1]

The complete set of rules for combining FOM Modules is given in the HLA Evolved OMT standard, not in the interface specification. One reason for this is that it is possible to combine FOM Modules without an RTI, for example in an OMT development tool.

5.2 Load operations and module life cycle

At runtime there are two ways to specify that a FOM module shall be loaded:

Create Federation Execution takes a list of FOM modules as an argument. It also takes an optional MOM module argument. If no MOM module is specified the RTI will provide the standard MOM module. This means that, in practice, most federation developers need never worry about providing the MOM in their FOMs any more.

Join Federation Execution also takes a list of FOM Modules. All modules that weren’t already loaded will now be loaded. A possible design pattern is to have federates load all FOM Modules that they depend upon when joining. The FDD data loaded by one federate becomes available to all federates in the federation.

Also note that the above load operations are atomic in the sense that they either succeed with loading all of the specified FOM modules or they fail.

Note that the load operation only really cares about the FDD, not the entire FOM. This means that it shall be possible to successfully combine the FDD portion of the FOM modules. There are no requirements for example for the Notes or Semantics to be equivalent for an attribute that is defined in two different modules. If the FDD portion of the FOM modules cannot be successfully combined due to a conflict between the modules then the whole load operation will fail.

A FOM module will remain available in the federation execution until the Destroy Federation Execution service is successfully executed. The MOM module will of course be available for the entire life span of the Federation Execution.

[1] Other approaches for this has since been suggested.

5.3 Reporting the Current FOM/FDD

The Current FOM is the sum of the currently loaded FOM Modules. When all of the FOM modules that were part of the federation design have been loaded the Current FOM will equal the FOM of the federation.

The RTI is only responsible for combining the FDD part of the FOM Modules. The RTI also provides services for reporting the Current FDD as well as contributing FOM modules. The following MOM functionality is available for this:

For the **HLAfederate** Class:

HLAFOMmoduleDesignatorList is an attribute whose value is the list of identifiers of all FOM modules that were loaded by this federate in *the Join Federation Execution* service call.

For the **HLAfederation** Class:

HLAFOMmoduleDesignatorList is an attribute whose value is the entire list of unique identifiers of all FOM modules that have been loaded into the Federation Execution, either by *Join Federation Execution* service calls or *Create Federation Execution* calls.

HLAMOMmoduleDesignator is an attribute whose value is the identifier of the MOM Module specified in the *Create Federation Execution* service invocation or implicitly used by the RTI.

HLAcurrentFDD is an attribute whose value is the current FDD as an XML Unicode string. This string shall comply to the OMT DIF Schema.

There are also a set of MOM interactions that make it possible to request the actual contents of each FOM and MOM module, namely:

For the Federation:

HLArequestFOMmoduleData requests the contents of a specified FOM module.

HLAreportFOMmoduleData responds with the content of the specified FOM module.

HLArequestMOMmoduleData requests the contents of the MOM module.

HLAreportMOMmoduleData responds with the content of the MOM module.

For each Federate a corresponding set of interactions to request FOM module data is provided, named **HLArequestFOMmoduleData** and **HLAreport-FOMmoduleData**

6. Building FOMs from FOM Modules

There is one additional FOM Module property that needs to be understood before using them. A FOM Module can be *Standalone*, which means that it can be used on its own, without any other FOM module (but a MOM module will always be required). FOMs as we know them in HLA 1.3 and HLA 1516-2000, when converted to HLA Evolved, will become Standalone FOM Modules since they can be used without other FOM Modules. They may of course now be extended by other FOM Modules.

The opposite type of FOM Module is referred to as *Dependent*. A Dependent FOM module contains a reference to a definition in another FOM Module. The OMT format allows for the following types of references:

- **Object Classes**. An Object Class can reference a superclass.
- **Interaction Classes**. An Interaction Class can reference a superclass.
- **Data Types**. Attributes, parameters, time representations, and user defined tags can reference data types.
- **Transportation types**. Attributes and interactions can reference data types.
- **Dimensions**. Attributes and interactions can reference dimensions.
- **Notes**. Anything in a FOM can contain a reference to a note.

Note that a Standalone FOM Module may always contain references to the MOM Module.

6.1 Suggested Block Notation

The following block notation is suggested:

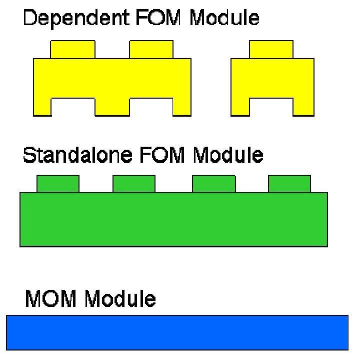
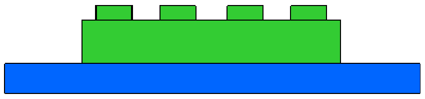


Figure 8: FOM Building Blocks

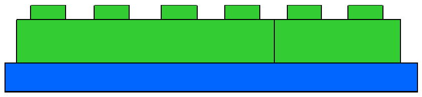
The MOM Module forms a foundation that Standalone FOM Modules can build upon. Dependent FOM Modules may in turn build upon Standalone FOM Modules.

6.2 Allowed Combinations

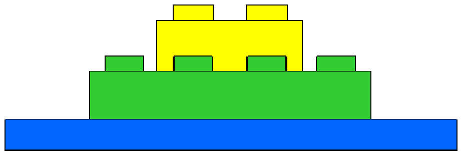
The following pictures illustrate allowed combinations:



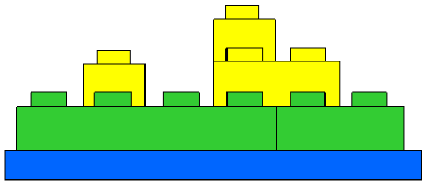
A MOM Module is always required. A Standalone module and a MOM module can be used to build a FOM.



Several Standalone FOM Modules and a MOM Module are also allowed. Remember that no conflicting definitions of concepts may take place.



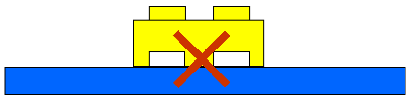
Dependent FOM Modules may be used on top of a standalone FOM Module.



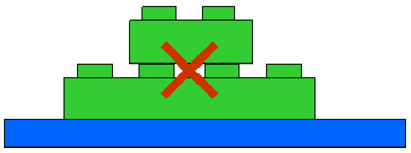
Several dependent FOMs may build upon a standalone FOM module. A dependent FOM module may build upon several standalone or dependent FOMs. A dependent FOM module may build upon a dependent FOM (but there has to be a standalone FOM at the bottom).

6.3 Disallowed combinations

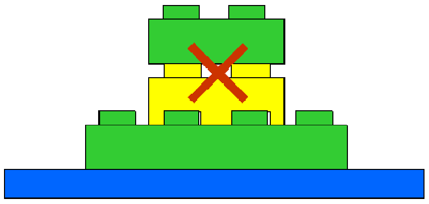
The following combinations are not allowed



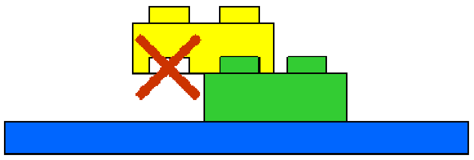
A Dependent FOM Module may not be used without a Standalone FOM module.



A Standalone FOM Module may not build upon another Standalone FOM module.



A Standalone FOM Module may not build upon a Dependent FOM module.



Every FOM concept that is referenced in a Dependent FOM Module must be provided by another (Dependent or Standalone) FOM Module.

The principles above apply to SOM modules as well with the exception that a MOM module is optional for a SOM.

7. Design Issues for Modular FOMs

Three major discussions that took place as part of the Tiger Team work were:

7.1 Repeated Definitions

The first discussion was about the use of scaffolding versus repeated definitions. Consider a FOM Module that build on top of the RPR FOM [9] and just adds a specialized type of aircraft. If no repeated definition of BaseEntitiy, PhysicalEntity, Platform and Aircraft were provided there would be little risk for mistakes causing incorrect redefinitions. It would also be a cleaner design if specific additions were stored in a separate module. On the other hand it is considerably easier to use the extended module on its own if it also contains the full definitions of superclasses. It may also sometimes be difficult to know if a well-known concept, like a commonly used data type, was already provided in some other FOM module. It was thus decided to allow both scaffolding and repeated definitions for convenience.

7.2 Reporting the Current FOM

The second discussion was about Current FOM reporting. Should the RTI be able to provide the full Current FOM or just the Current FDD? The RTI is currently only required to check that the FDD parts can be successfully combined. There is for example no requirement for checking whether the spelling of each note is consistent across modules. If this was required it would mean that the RTI would need to report the Current FOM, including inconsistencies. Some kind of OMT “analysis” format for this would need to be provided.

A viable argument for reporting the full Current FOM is that this kind of Federation Execution insight functionality would be highly desirable by many general-purpose federates. A viable argument against reporting the full Current FOM is that extensive XML processing would need to be incorporated into RTIs, essentially turning it into an OMT analysis tool. This was believed to conflict with common user requirements for robustness and performance. The standards text version that was delivered in January 2007 takes the Current FDD approach.

7.3 Describing Rules for Combining FOM Modules

The third major discussion was about how to describe the rules for combining FOM modules. One approach is to describe the formal relationship between several combined (or superimposed) FOM Modules without considering load order. This is similar to drawing tables on transparencies that are then stacked. Another approach would be to provide an algorithm for step-wise loading a FOM on top of an existing set of loaded FOM modules. In the first approach it is necessary to describe what combinations that are allowed and which ones that aren't. In the second approach it is necessary to describe when to signal what error during the load operation. The standards text version that was delivered in January 2007 takes the earlier approach. A later appendix with a step-wise approach was later proposed.

8. Towards Modular FEDEP

Introducing a new FOM Module into a federation calls for a new FEDEP iteration that needs to start, at the least, at the federation agreement step. At first glance FEDEP may easily be mistaken for a linear, water-fall type of process. A closer look reveals that it is indeed highly iterative which means that such a revisit is already supported. The introduction of a new FOM module was probably a result of some other redesign and a revisit of the federation design was probably necessary anyway.

A considerably larger requirement for a FEDEP revision that may be introduced by modular FOMs is the Modular Federation Agreement. This means that subset of overall federation agreements may need to be "packaged" together with FOM modules.

There are considerable advantages to modular federation agreements. They can be used to capture design patterns as well as well-defined and reusable subsets of federation agreements. One use case is a "federation management" module that contains a FOM module and a corresponding federation agreement.

Note that this should not be confused with Base Object Models (BOMs) [10] that cover a wider range of the simulation development and reuse process but can easily be used for developing FOMs and FOM modules. The relation between Modular FOMs and BOMs will be covered in a later paper.

9. Towards Modular FOM Agility

FOM agility is the ability for a federate to be able to adapt to different FOMs. In practice it may also include the wider ability to adapt to different federation agreements.

Before Modular FOMs were introduced the concept of FOM agility could be viewed as the ability to switch between two particular FOMs in their entirety. With modular FOMs we need to look at the ability to switch between different sets of FOM modules and their corresponding federation agreements. While this topic has not yet been fully explored, four obvious types of Modular FOM Agility can be noted:

1) A federate that can only handle a few, well-defined FOM modules that are known in advance. Best practice when implementing such a federate would be to load the FOM modules that contains the classes that it is going to publish or subscribe to when joining. A typical RPR FOM federate (CGF, viewer, flight simulator, etc) is a good example of this.

2) A federate that, in addition to the above, may optionally use something from an optional FOM module that may or may not be loaded in the federation. Think of a federate that may optionally use additional federation management functionality according to some local federation agreement. It may also be a "chain of command" module (specific to some country) that is to be used together with the standard RPR FOM. Still the specifics of this module would be known when the federate is developed. Best practice for this federate would be load the required FOM when joining and then check for the presence of the additional module.

3) A federate that is FOM agile and can use any FOM but doesn't really use the exchanged FOM data. One example is a data logger that doesn't decode or process the exchanged data in any way. It would need to get the current FDD to subscribe. It would probably not provide any FOM module when joining. It would just use the MOM functionality to get the current FDD.

4) A fully FOM agile federate that, in addition to the above, also decodes and uses the data. This could be an advanced data logger or general object inspector. There are other use cases, like a general standby federate that takes over objects for any crashing federate (using ownership). This type of federate would need to get more of the current FOM, at least data types.

10. Conclusions

A standardized format for Modular FOMs together with RTI Services and Rules has been developed as part of comment round three of HLA Evolved. This provides a new level of flexibility and many new opportunities.

The main advantage is the increased flexibility during runtime and, not the least, during the development process. Federation developers can build new modules that extend reference FOMs without modifying them. Reference FOMs can be developed by several smaller communities in different domains or for different local or national extensions. Temporary additions will not result in a multitude of similar FOMs.

It will also now be possible to have long-running or persistent federations in virtual arenas where new capabilities and types of shared information can be added over time.

12. Acknowledgements

The authors would like to acknowledge the Modular FOM Tiger Team that participated in extensive discussion. The tiger team consisted of: Martin Tapp, Bob Lutz, Reed Little, Doug Wood, Randy Saunders, Graham Shanks, Mikael Karlsson, Len Granowetter, Richard Andrade, Andreas Tolk, Annette Wilson, Ben Watrous, Björn Löfstrand, Dannie Cutts, John Fay, Jerry Szulinsky, John Schloman, Katherine Morse, Mike Lightner, Paul Gustavson, Roger Wuerfel, Tram Chase and Björn Möller (Tiger Team lead).

References

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

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

[3] Roy Scrudder et. al. "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.

[4] "Extensible Markup Language", http://www.w3.org/XML/

[5] "Federation Development and Execution Process (FEDEP)", IEEE 1516.3, www.ieee.org, 2003

[6] John McCarthy: "Recursive Functions of Symbolic Expressions and Their Computation by Machine (Part I)", Communications of the ACM, April 1960

[7] "Document Type Declaration", http://www.w3.org/TR/2000/REC-xml-20001006

[8] "XML Schema", http://www.w3.org/XML/Schema

[9] SISO: "Real-time Platform Reference Federation Object Model (RPR-FOM)", Version 2.0d17, based on SISO-STD-001.1-1999, www.sisostds.org, September 2003

[10] SISO: "Base Object Model (BOM) Template Specification", SISO-STD-003-2006, www.sisostds.org, May 2006

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.

BJÖRN LÖFSTRAND is Manager of Modeling and Simulation Services at Pitch Technologies. He holds an M.Sc. in Computer Science from Linköping Institute of Technology and has been working with HLA federation development and tool support since 1996. Recent work includes developing design patterns for HLA based simulation in the future Swedish Networked Based Defense.

MIKAEL KARLSSON is the 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.