Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

THE ROAD TO COTS-INTEROPERABILITY: FROM GENERIC HLA-INTERFACES TOWARDS PLUG-AND-PLAY CAPABILITIES

Steffen Strassburger

Fraunhofer Institute for Factory
Operation and Automation IFF
Sandtorstrasse 22
39106 Magdeburg, GERMANY

ABSTRACT

Interoperability between commercial-off-the-shelf (COTS) simulation packages (CSPs) is a topic which has been discussed for many years without a solution. With the advent of the High Level Architecture for Modeling and Simulation (HLA) for the first time a real industry standard has been made available which promises interoperability for a wide range of simulation systems and applications. Successful attempts to integrate HLA interfaces into different simulation packages have been made in the past. However, these interfaces typically place a significant overhead on the simulation developer. Also, as often a generic HLA interface is provided, different HLA interfaces for different simulation packages are not necessarily interoperable per se, as there are different possible ways to use HLA for the same task. This article addresses these issues and discusses interoperability solutions based on and beyond of HLA. It further investigates and comments the interoperability reference solutions put forward by SISO's COTS Simulation Package Interoperability Product Development Group.

1 MOTIVATION

After the initial definition of the HLA it soon became obvious that its applicability was not limited to military simulation applications. However, it was also obvious to all researchers involved in that area, that there would be major differences between both military and non-military simulation communities. One major difference is in the way how simulations are developed. While in the military community simulations are often developed in languages such as C++ or Java, in the civilian simulation community, the use of commercial-off-the-shelf simulation packages (CSPs) (e.g. Arena, Extend, GPSS/H, Pro Model, Simul8, SLX, etc.) is commonplace. These simulation tools satisfy the need to develop models rapidly and cost-effectively.

These different approaches for developing simulations also apply to HLA. While it is rather straightforward to directly access and use HLA in a simulation developed in C++, alternative means had to be developed for allowing HLA usage from a CSP (Strassburger 2001).

Initial solutions have integrated HLA into these simulation packages by providing a simulation system specific HLA interface for the individual systems. Some of these interfaces (e.g., the HLA interface for SLX) were generic in the sense of being usable almost as flexibly as the native HLA application programming interface (API). Others were limited to a specific HLA use patterns.

Section 2 of this paper discusses some of these solutions in further detail and explains the differences and problems associated with them.

As the general HLA-enablement of simulation packages has been proven possible, it could be assumed that the solution to the interoperability issue of CSPs is that easy: each vendor simply needed to provide their system with a generic HLA interface.

Reality, however, is more difficult (Taylor et al. 2002). The following facts indicate that more standardization efforts are needed to provide true plug-and-play-like and easy-to-use interoperability between different COTS simulation packages:

1. HLA is a quite complex standard. A generic HLA interface provides a developer with all means to solve his/her interoperability problem, however, it also leaves the burden of doing it on his/her side. Also, COTS vendors tend to be reluctant to support HLA because of its complexity.
2. Approaches to hide the HLA complexity from the user exist, however, they often introduce proprietary protocols on top of HLA resulting in non-interoperable solutions.
3. Different use patterns exist in HLA for accomplishing the same task. A simple matter like entity passing from one model to another can be solved in different ways, e.g., using HLA interactions or

HLA object instances plus ownership management. The results are non-interoperable solutions.

4. Insufficiencies in the HLA specification (e.g. non-time managed services) can lead to the usage of proprietary protocols as workarounds.

The remainder of this paper is structured as follows. Section 2 introduces previous solutions for HLA-enabling COTS simulation packages and discusses their advantages and disadvantages. Section 3 introduces the initiative of SISO's COTS Simulation Package Interoperability Product Development Group (CSPI PDG). Section 4 outlines in a case study how the interoperability reference solutions suggested by the CSPI PDG can be implemented based on an existing generic HLA interface. Based on the experience gained in these efforts, section 5 makes recommendations for further standardization efforts. Section 6 provides a summary and gives directions for future research.

2 REVIEW OF EXISTING HLA INTERFACES FOR COTS SIMULATION PACKAGES

HLA enabling a simulation package imposes the fulfillment of two types of requirements:

- Requirements derived from the HLA Interface Specification and the resulting programming paradigm, and
- Requirements derived from being part of a distributed simulation in general (Strassburger et al. 1998). Both categories will first be discussed in general. In the following, the properties of different solutions for HLA-enabling SLX, Simplex3, QUEST and IGRIP will be discussed.

The HLA Federate Interface Specification (IEEE 1516.1-2000) defines a two-part interface, which federates are required to use for communicating with the Runtime Infrastructure (RTI). This interface is based on an ambassador paradigm. A federate communicates with the RTI using its RTI ambassador. Conversely, the RTI communicates with a federate via the federate's ambassador. From the federate programmer's point of view these ambassadors are C++ or Java objects and the communication between the participants is performed by calling methods of these objects.

For COTS simulation packages this means that they must be able to create an instance of the RTI ambassador and they must implement a federate ambassador for receiving callbacks. Neither of these tasks can be typically done inside the COTS simulation package.

Besides conformance with the HLA programming paradigm, being part of a distributed simulation requires COTS simulation packages to be able to synchronize their local simulation clocks with other simulations and to meaningfully exchange data with them.

2.1 HLA Interface for SLX

The HLA interface for SLX (Henriksen 1997) uses the dynamic link library (DLL) interface of SLX to access a wrapper library providing access to the HLA functionality (Strassburger et al. 1998). The wrapper library implements the HLA ambassador objects discussed previously.

The wrapper furthermore converts the two directional calling convention of HLA into an SLX-usable form. All outgoing communication (time advance requests, attribute updates, interaction messages) can be triggered by the SLX model using the DLL-provided functions (e.g., RTI_NextEventRequest, RTI_UpdateAttributeValues, RTI_SendInteraction). These functions are provided in close affinity to the most important functions of the original HLA API, but with SLX specific argument lists.

All incoming communication is received by the federate ambassador inside the wrapper library via callback functions. The federate ambassador internally stores all received information and provides it to the SLX model at the appropriate time. Most commonly, this will happen during a synchronization request issued from the SLX model.

The most interesting feature offered by the SLX wrapper library in that context is that attribute updates and interactions are delivered *directly* to the associated SLX object instances. In this way an attribute update for a remote object instance directly modifies the SLX proxy instance of that object. This is possible due to a special feature of SLX which allows a DLL to dynamically determine the memory layout of complex data objects at *runtime*.

Other interfaces (e.g., the HLA interface for QUEST and IGRIP, see section 2.3) have to use a query feature for explicitly retrieving events received by the federate ambassador and then making the changes from within the simulation system.

Another highlight of the HLA interface for SLX is its good integration with SLX. This allows for a good simplification of the sometimes very complex HLA API, while still maintaining the highest degree of flexibility which only a generic HLA interface can offer. Let us consider an example: For generating an attribute update for any HLA object instance, the following native HLA function calls are needed:

1. RTI::AttributeSetFactory::create;
2. RTI::RTIambassador::getObjectClassHandle
3. RTI::RTIambassador::getAttributeHandle
4. RTI::AttributeHandleValuePairSet::add
5. RTI::RTIambassador::updateAttributeValues

From SLX, only a single function call must be issued:

```
procedure RTI_UpdateAttributeValues
    (int Object_ID,
    string(*) AttributeList,
    double TimeStamp) returning int dll="slxrti13ng";
```

The SLX model only has to pass the object ID to the wrapper library and the name of the attributes it wishes to update. The data encoding is performed entirely transparently inside the function implementation.

With these mechanisms, the HLA interface for SLX provides a very good balance between being a very flexible and generic HLA interface and maintaining a high degree of user friendliness. The drawback of this flexibility, of course, is that usage of the HLA functionality must be explicitly integrated into the simulation model.

The HLA interface for SLX in its native form can not be considered as a true plug-and-play interface. This is not a disadvantage introduced by the way the interface is implemented, rather, it is caused by a lack of further standardization of top of HLA (compare section 3).

Section 4 will discuss how the SLX statement concept can be used to add plug-and-play capabilities based on the existing generic interface and additional standardization.

2.2 HLA Interface for Simplex3

In contrast to the HLA interface for SLX, the HLA interface for Simplex3 uses an implicit approach to integrate HLA functionality into the simulation system. "Implicit" indicates that HLA functionality is not explicitly called from the simulation model, rather, the simulation system itself generates the appropriate RTI function calls whenever necessary. This was achieved by integrating the distributed simulation functionality with the Simplex3 modeling paradigm. As the source code of the simulation system was available for this work, the HLA ambassador object could be directly integrated into its code (Lantzsch et al. 1999), (Strassburger 2001).

Simplex3 is a simulation system for discrete, continuous, and combined simulation models (Eschenbacher 2006). Simplex allows the creation of hierarchical models, in which different subcomponents can be assembled to a new component, which again can be part of a component from a higher level.

Each component can be compiled and executed as an individual simulation for itself or in conjunction with other components. This characteristic has formed the basis for the development of the HLA interface, since each component already has defined interfaces with other components. Therefore the HLA interface basically replaces the internal connection of components by a mechanisms allowing an arbitrary hierarchical collection of Simplex components to interact with "components" from different simulation systems (in HLA terms federates) using the HLA.

The connectivity between components is defined within a mapping component which indicates which internal state variables translate into which object/interaction classes of the HLA federation object model.

From the Simplex components, the following component types can be mapped to HLA:

1. Basic Components are the basic building blocks of a Simplex model. They define the dynamic behavior of a certain part of the model. The best translation into the HLA world view is to treat a basic component as an HLA object class. The user can define which state variables are mapped onto which HLA object attributes.
2. Mobile Components are responsible for modeling moving entities within the Simplex language. They contain the same constructs as basic components except dynamic behavior definitions. Mobile components can only exist within locations, which need to be defined before using mobile components within a model. For passing entities between components, a mechanism using HLA interactions has been implemented. Once a mobile component leaves the Simplex modeling world, an interaction is generated containing the relevant state variables as parameters as well as the target location.

The HLA interface for Simplex3 leaves it up to the user to define the mapping of basic and mobile components to HLA object classes and interactions. Some care has to be taken, since each basic component only exists once in the model. Therefore it is not advisable to define a connection between an HLA object class with multiple instances and one basic component. Rather, these should be mapped with mobile components, as these can be generated dynamically.

A major advantage of the HLA interface for Simplex3 is its user-friendliness: As the user does not have to place HLA calls into his code, the usage comes close to a plug-and-play manner.

However, there are also some drawbacks: Due to the integration of the HLA functionality with the simulation kernel attribute updates and interaction messages are always sent for a single attribute instances or interaction parameters only, i.e., changes which belong together are not sent together. The reason for that is simple: whenever a state variable which is mapped to HLA changes, the change has to be sent. Bundling cannot be easily implemented, as it would change the Simplex modeling paradigm.

In summary, the HLA interface for Simplex 3 is a good example for an attempt of hiding the HLA complexity from the user. The HLA interface for Simplex3 leaves the path of being a fully generic and flexible HLA interface and by doing this it gains user friendliness.

For creating distributed simulations which only consist of Simplex3 models, the interface provides true plug-and-play capabilities. For the connection with other federates (e.g., implemented in SLX with its HLA interface), the plug-and-play capabilities diminish, as potential other federates have to take into account some of the implementation details used within the HLA interface for Simplex3.

With that, the HLA interface for Simplex is a good example to show the necessity for further standardization on top of HLA. If all COTS simulation packages were to implement the same use pattern of HLA, plug-and-play capabilities can become possible.

2.3 HLA Interfaces for QUEST and IGRIP

The HLA interfaces for QUEST and IGRIP apply a combination of both (explicit and implicit) approaches introduced in the previous sections. HLA functionality is provided for both systems using a DLL interface and a wrapper library called *universal federate adapter* (UFA). (Strassburger et al. 2003). The UFA wrapper library provides the callable HLA functions to both systems in a form accessible to them. Inside the simulation systems, the HLA functions are provided to the modeler in special HLA building blocks which are ready to use. Examples for these blocks are the HLA controller, the HLA source, and the HLA sink.

The HLA controller encapsulates the synchronization of the simulation clock with other federates and the external event retrieval and dispatching. The HLA source and sink blocks allow the passing and reception of entities between models.

Model developers can chose between two ways of using HLA. They can simply use the predefined building blocks and place them into their models without worrying about HLA details. This comes close to a plug-and-play, since the modeler only needs to specify the connection between each HLA sink and source pair in different models. Alternatively, they can use the HLA functions provided by UFA directly in their own SCL and BCL logics. SCL and BCL are the simulation languages which are used in QUEST and IGRIP for defining customized behavior (called "logics").

The first approach provides simple plug-and-play like capabilities but is limited to passing entities between models. The UFA function provided for this is RTI_TRANSFER_INSTANCE. It takes the sink name and the instance ID as parameters. Internally, the UFA implements a proprietary protocol for passing the entities (Strassburger et al. 2002). The basic principle is that each HLA source has a parameter telling it which HLA sink it is receiving entities from. Entities are modeled as HLA object instances with attributes. Passing of entities is implemented using HLA ownership management. The sender-receiver connection is communicated using the *userDefinedTag* of these services. In addition, a time stamped interaction has to be used for making the entity transfer time managed. This is necessary because ownership management services are not time-managed – a lack of the HLA interface specification (compare with Section 1).

Like in the Simplex3 approach, the UFA for QUEST and IGRIP introduces a proprietary protocol of how to use HLA for a certain purpose (here: entity passing). This has allowed the creating of a plug-and-play capable interface, but at the same time it limits its applicability to all simulation systems using the same approach.

3 CSPI PDG

As outlined in the previous section, there have been various attempts in the past to interoperate models and the COTS simulation packages in which they have been developed. For the reasons stated above these approaches are not fully compatible. In essence, there is no real standard "use" pattern for the High Level Architecture within the context of the focused application area. As an attempt to create such a use pattern and to unify research and development activities in this area the High Level Architecture – COTS Simulation Package Interoperability Forum (HLA-CSPIF) has been created (Taylor 2003), (Taylor et al. 2003a). This has now become SISO's COTS Simulation Package Interoperability Product Development Group (CSPI PDG) (note that "product" is used in SISO to denote standards).

The ultimate goal of the Forum is to create standards through SISO that will facilitate the interoperation of COTS simulation packages and thus make available to users of such packages the benefits of distributed simulation in a user-friendly and plug-and-play like manner. As a first activity, CSPI-PDG has created standard interoperability reference models that can be used to communicate concepts and problems between researchers, users, and vendors in support of the CSPI-PDG aims.

The following lists the interoperabilily reference models (IRMs) that have currently been identified. For an in-depth discussion of these IRMs please refer to (Taylor 2003).

- Type I Interoperability Reference Model - Asynchronous entity passing.
- Type II Interoperability Reference Model - Synchronous entity passing.
- Type III Interoperability Reference Model - Shared resources.
- Type IV Interoperability Reference Model - Shared events.
- Type V Interoperability Reference Model - Shared data structure.
- Type VI Interoperability Reference Model - Shared conveyor.

The remainder of this article is mainly concerned with the type I IRM with an outlook on the challenges of type II solutions.

The *Type I IRM* represents models that interact on the basis of entities, i.e. models are linked together so that one model may pass an entity to another. The term "Entity" here refers to the dynamically created elements that move through a simulation. They may be called differently in different simulation system (e.g., transaction in GPSS/H, widget in QUEST, "movable elements" in eM-Plant, mobile components in Simplex3, etc.). The IRM is labelled "asynchronous" as there is no *direct feedback* when an entity is passed (Figure 1).

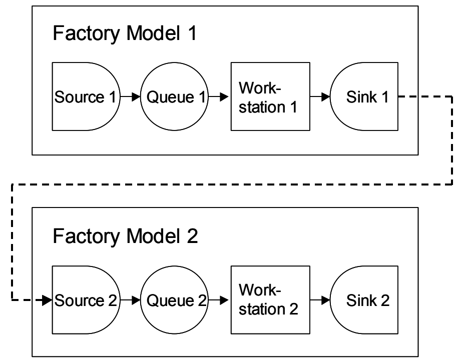

Figure 1: Type I IRM for Asynchronous Entity Passing

The *Type II IRM* extends the Type I IRM by introducing the need for immediate feedback to the entity passing. This is required in cases where the receiving simulation may be in a state unable to accept the entity being passed, e.g., because of a bounded queue (Figure). In this case the entity transfer cannot succeed. One may argue that this is an academic problem, since models could in almost any case be partitioned to prevent this from happening. However, this IRM is an example for an entire class of problems requiring immediate communication between simulations. Actually, all IRMs above the Type II RM require this basic functionality. Therefore a solution to the Type II RM could be the basis for standardizing solutions to the other RMs as well.

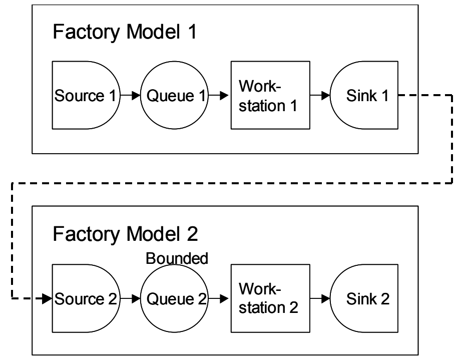

Figure 2: Type II IRM for Synchronous Entity Passing

Besides these reference models, the CSPI-PDG has put forward a proposal for an Entity Transfer Specification Standard as the basis for solving the Type I IRM (Taylor et al. 2004).

In a nutshell, the ETS version 1.1.1 suggests the usage of HLA interactions for transferring entities between two models. The sender-receiver relationship is represented by building a special hierarchy of interaction classes and subclasses. As the root class, an interaction class called "TransferEntity" is specified. It has sub-classes which correspond to all potential recipient models, e.g. "TransferEntityToFactoryModel1, TransferEntityToFactoryModel2, etc.). These subclasses are meant to build a unique class which a certain federate would have to subscribe. In our simple example, Factory Model 2 would subscribe to "TransferEntityToFactoryModel2" and could be sure that it received information about all potential subclasses of this class. The practical feasibility of this is commented on later.

Further to this subclass identifying the target federate, individual interaction classes are introduced for each connection between a sending model and a target model. In our example, there would be a single subclass to "TransferEntityToFactoryModel2" called "TransferEntityFactoryModel1ToFactoryModel2". This interaction class would be published by the sending model (Factory Model 1).

For transmitting the state of an transferred entity, the interaction classes have a single parameter named "Entity". The type of this parameter is a complex datatype (record) identifying the name of the entity, identifiers for the source and destination, and any simulation dependent parameter. The source and destination tags operate on the federate identification level, i.e., they do not identify any sink/source inside a model. As this article cannot reproduce the entire ETS the reader is referred to (Taylor et al. 2003b) and (Taylor et al. 2004) for further details. For specific comments on this specification please refer to section 5 of this article.

The following section investigates if and how a simulation package with a generic HLA interface can adapt to the so-defined standard and what benefits towards plug-and-play interoperability it yields.

4 CASE STUDY: FROM GENERIC HLA INTERFACES TO PLUG-AND-PLAY CAPABILITIES

This case study has investigated, if and how the proposed entity transfer specification (ETS) can be implemented on top of the existing generic HLA interface for SLX and which benefits it yields.

For performing these tasks, the SLX statement concept has been used. SLX statements allow the extension of the SLX language itself. Statements can be compared with marcos known from other languages, however SLX statements extend the macro concept by allowing the execution of SLX code at compile time. In this way, a very flexible way of extending the SLX language is provided. For implementing the ETS specification two new statements ("Federate" and "Transfer Entity") have been defined inside SLX as follows:

```
statement Federate #FederateName
    joining Federation #Federation
    using FedFile #FedFile
    receiving entities [=  {<#Recipient yes>| <#Recipient no>}]
                  [@from  {#SourceFederateName} ,...]
    [sending entities @to {#TargetFederateName} ,...]
    [lookahead = #lookahead];
```

The federate statement has to be used for specifying certain definitions like the federate name and the federation name. Most importantly, the federate can specify if it expects to receive entities from other federates, and if it intends to send entities to other federates. In the latter case, multiple target federates can be specified.

In our example, factory models 1 and 2 would use the following federate statements:

```
    Federate FactoryModel1
        joining Federation FactorySimulation
        using FedFile Factory.fed
        receiving entities = no
        sending entities to FactoryModel2;

    Federate FactoryModel2
        joining Federation FactorySimulation
        using FedFile Factory.fed
        receiving entities from FactoryModel1;
```

The main benefit of the federate statement is the hiding of all HLA details in its inside. From the simple argument list shown above the federate statement generates all necessary HLA calls completely transparently to the user. The statement automatically constructs the correct interaction class names which it has to publish and subscribe according to the ETS. It further on generates the code required for joining and synchronizing the model with other federates. Please note that in the above example the optional parameter "lookahead" is not specified. In that case, the statement always assumes the safe option of "zero lookehead". Otherwise the specified lookahead value will be used. Finally, the statement also contains code for receiving incoming entities and dispatching them to the right location. This will be explained later.

For transferring entities to external federates, the following statement is defined:

```
statement Transfer [Entity] [#EntityPtr]
        To #TargetFederateName
        [transferTime =  #timeValue]
```

This statement internally generates the correct transfer interaction which it has to send according to the ETS. The optional parameter "transferTime" specifies the duration of the transfer. If no value is specified, the transfer at the current simulation time. The "Transfer" statement internally performs runtime checking about potential user errors, which could result from specifying a transfer time which is smaller than the lookahead specified in the "Federate" statement. In our example from above Federate 1 would use the "Transfer" statement as follows:

```
Transfer Entity me To FactoryModel2;
```

The transfer statement is capable of transferring any user defined entity. The user can specify any number of attributes for the entity. This is achieved by encapsulating the complex record data type into a single parameter of the transfer interaction. This certainly also has drawbacks (compare section 5), but in its current form it allows for a very user friendly implementation. For our example factory models, the entity class is defined as follows:

```
class Entity {
    string(256)   EntityName;       //parameter according to ETS
    string(256)   Source;           //parameter according to ETS
    string(256)   Destination;      //parameter according to ETS
    int           size;             //model specific parameter
    double        production_time;  //model specific parameter

    procedure dispatch_me   {
            //Add code here to dispatch externally received
            //entities within your model
        }
}
```

It can be noted that this interaction has a special method called "dispatch_me". This method only has to be implemented in the receiving federates. The method will be called from the synchronization algorithm, whenever a new entity is received for the local federate. The method has to make sure that the entity is dispatched to the right source or entry point in the model. According to the ETS it can use the parameters source and destination for that. This code section has to be implemented by the user as it highly depends on his choice of modelling the factory inside SLX.

While the "Federate" statement has to be inserted as the first statement to be executed, the "Transfer" statement can be placed anywhere you want to transfer entities to a different model. Typically it will occur before a "terminate" statement removing an entity from the local model.

Summarizing the case study it can be noted that by using the SLX statement concept and the entity transfer specification it becomes extraordinarily easy to interoperate SLX models with other models using the ETS. The effort for implementing a federate in SLX is reduced to introducing two statements into the code of an SLX model plus implementing a dispatch procedure for externally received entities. The latter can be typically done in a single line of code. In terms of plug-and-play capabilities, this solution is as close as it can get to this objective in a simulation *language*. In the best case, the effort for networking a SLX model is changing three lines of code.

5 STANDARDIZATION RECOMMENDATIONS

On the positive side, the experiences from the case study prove that the entity transfer standard can be implemented on top of existing generic HLA interfaces. No changes of the wrapper library implementing the HLA interface of SLX have been necessary. Using encapsulation mechanisms like the SLX statement concept can yield tremendous additional simplifications for the user.

On the other hand, during the work on the case study it was also observed that the current ETS specification has some significant drawbacks:

1. Source and destination designators both operate on the federate level, i.e., they specify the sending and the receiving federate. This specification lacks a possibility to differentiate between multiple connections between any two federates. Assume two sinks in factory model 1 and two entry points in factory model 2. The ETS in its current form would not allow the target federate to dispatch entities to the right entry point, since all it knew was that they were coming from factory model 1. *The author suggests that the ETS is modified to allow the specification of both entry and exit points as additional parameters of the transferred entity.*
2. The ETS suggests a hierarchy of transfer interactions. The main intention is to have one single superclass which federates can subscribe to in order to receive all entities transferred to them. The idea behind that is tempting. However, since the ETS specifies the actual data to be transmitted as interaction parameters at the leaf level of the lowest subclass, a federate subscribing to the superclass will never receive the values transmitted in the interaction parameter. This simply cannot work since parameters are inherited from the superclass to its subclass, but not vice versa. *The author suggests that the hierarchical structure of the transfer interactions be removed. Source and destination can be easier communicated as direct parameters of the transfer interaction.*
3. The specification of user defined attributes is placed into a complex record datatype. Although the motivation for this is obvious, it introduces new room for interoperability challenges as all participating federates have to be able to interpret all of the attributes. A definition of user defined attributes as individual parameters of the transfer interaction would allow federates to subscribe to only those parameters they are able to handle.
4. The ETS leaves some room for misinterpretation in its definition of "Entity" and the "EntityType". In the template of the interaction parameter table the transfer interaction has a single parameter called "Entity" with the datatype "EntityType". EntityType itself is defined in the fixed record datatype table. It is defined there with mandatory fields "EntityName", "Source" and "Destination". Only from the given example of how to use this template it becomes clear that actually the *name of the parameter* has to be changed according to the name of the entity. This introduces the requirement of changing FOMs whenever a new entity type is talked about. This is an unnecessary disadvantage, as the definition of the entity type could as well be put as a mandatory parameter either into the fixed record for the EntityType or it could directly become an interaction parameter itself.

It can be noted that different solutions to comments 3 and 4 exist. The author suggests that the fewest possible changes to the FOM be required of the user. It would be advisable to define more properties directly as individual parameters of the transfer interaction.

A better use of the hierarchy inside the interaction table might be to actually diversify the transmitted entities in terms of its parameters. A superclass of "TransferEntity" could only have the significant parameters like source, destination, entity type, and entity name. Subclasses could inherit from that class and add user defined attributes to it.

In addition to the current efforts made by the CSPI PDG it might be useful to discuss and agree on reference federation object models for certain application domains. The could, in the simplest case, specify which attributes entity types in a certain modeling domain should have.

6 SUMMARY AND FUTURE WORK

This paper has reviewed the innovations towards COTS simulation package interoperability that have become available through HLA. It has analyzed different solutions for HLA enabling simulation systems. It can be noted that the general feasibility of HLA enabling such systems has been proven successfully in the past. A simulation system with a generic HLA interface is capable of interoperating with other federates in all ways offered by HLA.

However, this paper also has outlined the reasons why generic HLA capabilities are not sufficient for achieving true plug-and-play interoperability between different COTS simulation packages. The main reason is in the effort needed for the user to incorporate HLA functionality into his/her model. Only by encapsulating HLA functionality into a user friendly form, the next step towards plug-and-play capabilities can be made.

The paper has further introduced and reviewed the CSPI-PDG initiative with its main objective of creating standardized HLA use patters for COTS simulation package interoperability. These standardized use patterns can help to encapsulate HLA functionalities in a user-friendly form. They also prevent some of the pitfalls that can happen when applying a complex standard like HLA.

The first proposal for such a standardized HLA use pattern, namely the entity transfer specification as a solution to the Type I Interoperability Reference Models has been commented and suggestions for its approval have been made.

Future work will include the discussion of the next revision of the ETS and its extension for Type II Reference Models. The latter imposes specific new challenges on the community, as its solutions require zero lookahead in any case. Additional problems arise from the need of dealing correctly with simultaneous events (e.g. two federates trying to pass an entity to a queue at the same time which only has a remaining capacity of one). Another interesting area of future work will be to apply the encapsulation attempts made with a simulation language like SLX to building block based tools like Extend.

REFERENCES

Eschenbacher, P. 2006. Simplex3 – The Universal Simulation System. Available at <http://www.simplex3.net>.

Henriksen, J.O. 1997. An introduction to SLX™. In *Proceedings of the 1997 Winter Simulation Conference*, ed. Andradóttir, S., K. Healy, D. Withers, and B. Nelson, 559-566. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

IEEE 1516.1-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) Federate Interface Specification.

Lantzsch, G., S. Strassburger, C. Urban. 1999. HLA-basierte Kopplung der Simulationssysteme Simplex III und SLX. In: *Proceedings Simulation und Visualisierung '99*, eds. O.Deussen, V. Hinz, P. Lorenz. Magdeburg, March 4.-5. 1999, SCS International, pp. 153-166.

Strassburger, S., T. Schulze, U. Klein and J. O. Henriksen. 1998. Internet-based Simulation using off-the-shelf Simulation Tools and HLA. In: *Proceedings of the 1998 Winter Simulation Conference*, 1669-1676. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Strassburger, S. 2001. *Distributed Simulation Based on the High Level Architecture in Civilian Application Domains.* Ghent: Society for Computer Simulation International, ISBN 1-56555-218-0.

Strassburger, S., A. Hamm, G. Schmidgall, and S. Haasis. 2002. Using HLA Ownership Management in Distributed Material Flow Simulations. In *Proceedings of the 2002 European Simulation Interoperability Workshop*. June 2002. London, UK.

Strassburger, S., G. Schmidgall, and S. Haasis. 2003. Distributed Manufacturing Simulation as an Enabling Technology for the Digital Factory. In *Journal of Advanced Manufacturing Systems (JAMS)*. 2(1), 111-126.

Taylor, S., A. Bruzzone, R. Fujimoto, B. Gan, S. Strassburger, and R. Paul. 2002. Distributed Simulation and Industry: Potentials and Pitfalls. In *Proceedings of the 2002 Winter Simulation Conference*, 688-694. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Taylor, S. 2003. HLA-CSPIF. The High Level Architecture COTS Simulation Package Interoperability Forum. In: *Proceedings of the 2003 Fall Simulation Interoperability Workshop*. September 14 – 19, 2003. Orlando, USA.

Taylor, S., B. P. Gan, S. Strassburger, and A. Verbraeck. 2003a. HLA-CSPIF Technical Panel on Distributed Simulation. In *Proceedings of the 2003 Winter Simulation Conference*, 881-887. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Taylor, S. et al. 2003b. *HLA-CSPIF Discussion Document Entity Transfer Specification*. Version 1.1.1. Available at <http://people.brunel.ac.uk/~csstsjt/HLA-CSPIF/ets1_1_1.doc>

Taylor, S., S. Turner, and M. Low. 2004. A Proposal for an Entity Transfer Specification Standard for COTS Simulation Package Interoperation. In *Proceedings of the 2004 European Simulation Interoperability Workshop*. June 28 - July 1, 2004. Edinburgh, Scotland.

AUTHOR BIOGRAPHIES

STEFFEN STRASSBURGER is head of the "Virtual Development" department at the Fraunhofer Institute for Factory Operation and Automation in Magdeburg, Germany. He was previously working as researcher at the DaimlerChrysler Research Center in Ulm, Germany, where he was responsible for research topics in the Digital Factory and Digital Engineering context. He holds a Ph.D. and a Master's degree in Computer Science from the Otto-von-Guericke University in Magdeburg, Germany. His international experience includes a one-year stay at the University of Wisconsin, Stevens Point and a stay at the Georgia Institute of Technology, Atlanta. The main research topics of his department include virtual reality solutions for product and process development, the combination of virtual reality and discrete event simulation, and distributed and web-based simulation. He is also the Vice Chair of SISO's COTS Simulation Package Interoperability Product Development Group.