Interoperating COTS Simulation Modelling Packages: A Call for the Standardisation of Entity Representation in the High Level Architecture Object Model Template

Simon J E Taylor
Centre for Applied Simulation Modelling
Department of Information Systems and Computing
Brunel University, Uxbridge, Middx, UB8 2DD, UK.
simon.taylor@brunel.ac.uk
www.brunel.ac.uk/~csstsjt

KEYWORDS

Discrete Event Simulation, Simulation Modelling, COTS Simulation Modelling, Distributed Simulation, High Level Architecture, Object Model Template.

ABSTRACT

The High Level Architecture is the defacto standard for distributed simulation. Although widely used in defence-related real-time, virtual distributed simulations, the High Level Architecture has yet to make a significant impact in other simulation application areas that use COTS simulation modelling packages. There are many possible reasons for this. In an attempt to make progress in the development of distributed COTS simulation modelling applications, this paper discusses the use of the OMT in this area. This paper argues that the *entity* is the basis of information exchange between distributed models and that there is a the need for the standardisation of entity representation in the HLA Object Model Template. The paper attempts to do this by presenting a simple distributed simulation and demonstrates how entities can be represented in the Object Model Template as attributes and as interactions. The paper concludes that while neither representation is ideal, both can be used to represent entities and that standardisation is urgently needed.

INTRODUCTION

In the year 2000 the High Level Architecture (HLA) became the defacto standard for distributed simulation. It was developed to provide a common architecture for distributed modelling and simulation. Geographically remote models (federates) interact and share information via software to give the appearance of a single model (federation). The HLA is composed of three parts: a Framework and Rules (IEEE 2000a), a Federate Interface Specification (IEEE 2000b), and an Object Model Template (OMT) Specification (IEEE 2000c). The Framework and Rules define the general principles of the HLA, the Federate Interface Specification defines the standard services and interfaces of software (called the *runtime infrastructure* (RTI)) used facilitate communication between federates, and the OMT defines the format and syntax for recording information in HLA object models. The HLA is widely used in defence to support the interoperation of real-time, virtual simulations. So far, however, despite attempts made by several researchers, the HLA has yet to make any significant impact in other simulation modelling areas such as business, health, manufacturing, and transportation.

There are many possible reasons for this lack of impact outside of the defence sector. Taylor, *et al.* (2002) discuss some of these. However, one possible reason is that many simulation practitioners outside of the defence sector tend to use commercial-off-the-shelf (COTS) simulation packages (such as Arena, Extend, Simul8, Taylor, or Witness for example – see www.groupsim.com for links to the home pages of these packages) rather than object-oriented, or at least object-based, software engineered code. It might be argued that there is some difficulty in matching the requirements of COTS simulation packages to the HLA. To contribute to this discussion, this paper considers how the Object Model Template can be used to represent model information that must be shared between models running in distributed COTS simulation packages. The paper is structured as follows. Section 2 discusses COTS Simulation Modelling. Section 3 introduces a simple case study. Section 4 discusses how the OMT might be used to represent the shared model information of the simple case study. Finally Section 5 concludes the paper with some reflections on this representation and identifies the need for standardisation.

COTS SIMULATION MODELLING

COTS Simulation Modelling refers to the methods and tools employed by simulation practitioners who use commercial-off-the-shelf simulation packages to support their role. COTS simulation packages are used mostly for model building, experimentation, and animation/visualisation (reporting). We concern ourselves with only those packages based upon some variant of the discrete event simulation paradigm, i.e. models change state at discrete points in time by scheduled or conditional events and typically represent entities (documents, patients, parts, trains, etc.) passing through networks of queues and workstations (work queuing at a desk in an office, patients waiting to see a doctor, parts buffered for machining, trains waiting at a station, etc.) Generally, each package typically has a range of basic model elements (queue, workstation, resource, source, sink, etc.) that are used to build a model via a drag and drop visual interface. Each model element can be modified as is required, either by a menu system or by a package programming language, to better represent the system being studied (for example the queuing logic of a queue or the

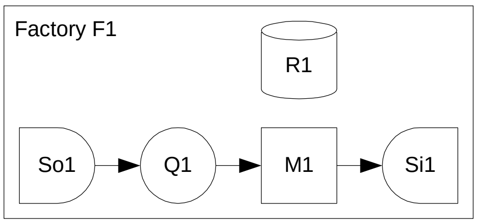

Figure 1: Simple Factory Model

appearance of a resource). Different entities passing through a model can be identified. If necessary individual entity differences can be identified by adding entity attributes. In addition to model building, the visual interface of a package will also support animation for model debugging and demonstration purposes. Statistical experimentation facilities are also usually provided by the package for model experimentation and reporting. Terminology between packages differs as there is no internationally recognised naming convention.

For purposes of orientation, figure 1 shows a simple factory model that might be built in a COTS simulation package to study the mean throughput for a variable product mix. The model consists of an arrival source So1, a queue Q1, a machine M1, a resource R1, and an exit sink Si1. The entities passing through this factory are generically known as *Parts*. Parts have three attributes: Type, Entry time, and Exit time. *Type* specifies the type of the part (which for the sake of argument are designated by different suppliers General Electric, PurpleTC, and Brunel) and *Entry/Exit time* are used for analysis purposes to determine, for example, the mean time it took a type of Part to go though the factory. Parts (entities) arrive in the factory at the source So1 according to an arrival time distribution and are placed in the queue Q1. Parts wait in Q1 until the machine M1 is free and not undergoing repair. If M1 is free, a part is loaded and processed according to a processing time/part type distribution. When processing is finished, the part exits the factory via the sink Si1 and, if there are parts waiting in Q1 and the machine does not breakdown, another part is processed. If the machine does breakdown a repairman must be available. If one is, then the machine will be repaired according to a machine repair distribution. The repairman is modelled by resource R1 (and for purposes of this simple model will always be available as there is no competition!) Various details such as arrival time distribution, processing time distribution, machine breakdown distribution, etc. are detailed by using menus associated with each model element. Note that most COTS packages save such a model as a simple text file that records the relevant details of each model element – it is not saved as an executable nor as source code (as the simulation/execution of a model is performed by the package).

A SIMPLE DISTRIBUTED SIMULATION

In order to discuss the options for entity representation that the OMT can provide us, let us consider the simple distributed simulation of two factories, F1 and F2, shown in figure 2. Both are identical to the simple factory model described above with one exception. The sink Si1 in F1 and the source So2 in F2 are replaced by a direct link agreed with the stakeholders of both factories (i.e. the link is a consequence of simulation methodology and not as a result of technological intervention). The combined models represent the combined factory as parts finishing machining in M1 are transferred directly to queue Q2 to await machining in M2.

In terms of HLA distributed simulation, each model and the COTS simulation package in which it runs is a federate. The federation therefore consists of the two federates (F1 and F2) that run on different machines connected by a network and distributed simulation software (an RTI). Focusing on the information requirements of the two federates (and not on RTI issues such as object ownership/transfer, distributed management, time management, distributed experimentation control, and RTI-COTS simulation package integration, etc.) all that must be represented in this case is the transfer of individual part entities and their attributes between federate F1 and federate F2. In other words some common representation of the entity *Part* and the three attributes *Type*, *Entry time* and *Exit time* must be agreed. Let us now examine the options available to us in the HLA OMT.

REPRESENTING ENTITIES WITH THE OBJECT MODEL TEMPLATE

To understand how entities can be represented in the HLA by the OMT, let us first consider some of the concepts of the OMT Specification (IEEE 2000c). The HLA requires that federations and individual federates be represented by an object model that identifies the data that is to be exchanged during the execution of a federation/federate. The

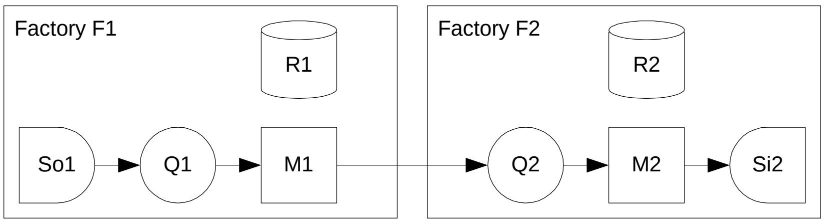

Figure 2: A Simple Distributed Simulation

description of this data is the purpose of the OMT. An HLA object model can be used to represent the information exchange characteristics of a federate in an HLA Simulation Object Model (SOM), or a federation in an HLA Federation Object Model (FOM). The SOM specifies the information that a federate can generate and the information that the federate needs. The FOM specifies the entire information exchange of a federation (and therefore the federates).

In the HLA, object models are represented as classes (with each particular member referred to as an instance). Each class has a set of named data characteristics called attributes (which are semantically different to entity attributes) and a set of interaction classes that represent explicit actions taken by a federate that may have some effect or impact on another federate (*ibid*, p.7). Federates essentially share information indirectly via HLA services (the runtime infrastructure software) by updating attribute values or by sending interactions (an explicit action taken by a federate that may have some effect or impact on another federate). There are fourteen tables in the OMT that must be included in a SOM or FOM (irrespective of a table being empty). For purposes of this discussion we shall restrict ourselves to the following tables:

- Object Class Structure Table.
  This records the namespace of all federate or federation object classes.
- Interaction Class Structure Table
  This records the namespaces of all federate or federation interaction classes.
- Attribute Table
  This specifies features of object attributes in a federate or federation.
- Parameter Table
  This specifies features of interaction classes parameters in a federate or federation.
- Datatype Tables
  These are various tables used to specify the representation of data in the object models.

As we will see, this gives us two principle opportunities to represent the entities that are to be exchanged between our two federates: as published attribute values or as published interactions..

Exchanging Entities as Attribute Values

If we are to represent the exchange of entities as published attribute values, we can represent entities in the OMT using four tables. These are:

- Object Class Structure table
- Attribute table
- Fixed Record Datatype table
- Enumerated Datatype table

The Object Class Structure table declares the classes of the object model and the Attribute Table declares the specific named characteristics of each class that are to be accessible by another federate. If we consider federate F1, we might argue that as the model it represents needs to exchange entities with federate F2, then as each federate consists of a factory model we might declare a factory model as being a class with the publicly accessible attribute Part. In this case our federate F1 would repeatedly publish Part attributes which would be received (via the RTI) by the federate F2. Table 1 shows a possible Object Class Structure table for our federate F1, table 2 shows its Attribute table, and table 3 shows the Fixed Record Datatype table, and table 4 shows the Enumerated Datatype table. In table 1 the HLAobjectRoot is the root class for all object models – Factory F1 is therefore a subclass of this. N (Neither) and P (Publish) denote that the federate is incapable of publishing/subscribing to any attributes of the object class and that the federate is capable of publishing at least one attribute of the object class respectively. This denotes that the class FactoryF1 is capable of publishing at least one attribute which is detailed in Table 2. Table 2 shows that the class FactoryF1 publishes the single attribute Part which is of type PartEntity. Table 3 describes the detail of the PartEntity datatype as a fixed record datatype. Finally Table 4 enumerates the possible values of PartType. (Note that a complete discussion of the details of these tables is outside the scope of this paper. Also note that the HLAobjectRoot has been omitted in the Attribute table for clarity).

Exchanging Entities as Interactions

An alternative to representing an entity as a published attribute is to represent an entity as an interaction between one federate and another. If we do this, we might use the following tables of the OMT.

- Interaction Class Structure table
- Parameter table
- Fixed Record Datatype table
- Enumerated Datatype table

The Interaction Class table declares the interactions of the object model, the interactions that the federate is capable of sending or receiving to/from another federate. Taking this view, we might argue that federate F1 interacts periodically with federate F2 by notifying F2 of the arrival of a new Part entity. If we choose to do this, then we may represent this interaction with the Interaction Class table shown in table 5, the Parameter table of table 6, the Fixed Record Datatype table of table 3, and the Enumerate Datatype table of table 4. In table 5 the HLAinteractionRoot is the root class for all interactions– ProducePart is therefore a subclass of this. Similarly to attribute values, N (Neither) and P (Publish) denote that the federate is incapable of publishing/subscribing any instances of the interaction class and that the federate is capable of publishing at least one instance of the interaction class respectively. This therefore denotes that the interaction ProducePart capable of being published by federate F1 at lease once. Each interaction has a set of parameters. In this case the interaction ProducePart has a single interaction *Part* as shown in table 5. *Part* has the same datatype details as the attribute *Part* which are detailed in table 3 and 4. (Again note that a complete discussion of the details of these tables is outside the scope of this paper. Also note that the HLAinteractionRoot has been omitted in the Parameter table for clarity).

Table 1: Object Class Structure Table for Federate F1

HLAobjectRoot(N)	FactoryF1(P)

Table 2: Attribute Table for Federate F1

Object	**Attribute**	**Datatype**	**Update Type**	**Update Condition**	**D/A**	**P/S**	**Available Dimensions**	**Transp-ortation**	**Order**
FactoryF1	Part	PartEntity	Condi-tional	When Ready	D	P	NA	HLAReliable	Time-stamp

Table 3: Fixed Record Datatype Table for Federate F1

Record name	**Field**			**Encoding**	**Semantics**
	Name	**Type**	**Semantics**		
PartEntity	Type	PartEntityType	Type of PartEntity	HLAfixedRecord	A Part manufactured by Factory F1
	EntryTime	HLAinteger32BE	Entry Time of Part in factory		
	ExitTime	HLAinteger32BE	Exit Time of Part from Factory		

Table 4: Enumerated Datatype Table for Federate F1

Name	**Representation**	**Enumerator**	**Values**	**Semantics**
PartEntityType	HLAinteger32BE	GeneralElectric	0	Possible types of Part produced by factory F1
		PurpleTC	1	
		Brunel	2	

Table 5: Interaction Class Structure Table for Federate F1

HLAinteractionRoot(N)	ProducePart(P)

Table 6: Parameter Table for Federate F1

Interaction	**Parameter**	**Datatype**	**Available Dimensions**	**Transportation**	**Order**
ProducePart	Part	PartEntityType	NA	HLAreliable	Timestamp

CONCLUSION

To conclude this paper I ask the question, which representation is best? Amongst the many options available, the options for representation that this paper has identified are:

- An entity is represented as an attribute of a class model, with entity attributes represented as datatypes.
- An entity is represented as a parameter of an interaction, with entity attributes represented as datatypes.

The answer to this question is difficult. The two options as presented are good as the cohesion between the entity and its entity attributes is preserved as a datatype of an HLA attribute or interaction. The problem is that neither really fully capture the semantics of an entity passed between models. An entity is neither an attribute of a model nor a interaction between models – it is a construct in its own right.

Irrespective of an ideal “fit”, the problem of entity representation in the HLA must be solved for any chance of HLA distributed simulation making any significant impact in industry. The standardisation of entity representation is a key element to this. The integration of the HLA with various COTS simulation packages is a non-trivial matter. However, as the major unit of information exchange between most models is the *entity*, there is an urgent need to agree how the OMT must be used to represent entities. It is hoped that this paper will foster discussion on this matter.

REFERENCES

IEEE 2000a. *IEEE Standard for Modelling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules*. IEEE Std 1516-2000. IEEE Computer Society, New York, NY.

IEEE 2000b. *IEEE Standard for Modelling and Simulation (M&S) High Level Architecture (HLA) –Federate Interface Specification*. IEEE Std 1516.1-2000. IEEE Computer Society, New York, NY.

IEEE 2000c. *IEEE Standard for Modelling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template (OMT) Specification*. IEEE Std 1516.2-2000. IEEE Computer Society, New York, NY.

Taylor, S.J.E, Bruzzone, A., Fujimoto, R., Gan, B.P., Straßburger, S., and Paul, R.J. 2002. "Distributed Simulation and Industry: Potentials and Pitfalls." In *Proceedings of the 2002 Winter Simulation Conference*, E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds. *To Appear*. Association for Computing Machinery. New York, N.Y.

AUTHOR BIOGRAPHY

SIMON J.E. TAYLOR is the Chair of the Simulation Study Group of the UK Operational Research Society and the collaborative simulation modelling forum, the GROUPSIM Network (www.groupsim.com). He is a Senior Lecturer in the Department of Information Systems and Computing and is a member of the Centre for Applied Simulation Modelling, both at Brunel University, UK. With Dr Gary Tan of the School of Computing, National University of Singapore he is joint leader of the UK(EPSRC)/Singapore(DSTR)-funded BRUNUSIM distributed simulation research programme. He has an undergraduate degree in Industrial Studies (Sheffield Hallam), a M.Sc. in Computing Studies (Sheffield Hallam) and a Ph.D. in Parallel and Distributed Simulation (Leeds Metropolitan). His main research interest is collaborative simulation modelling. He is also a member of the London-based Purple Theatre Company.