Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

GUIDELINES FOR COMMERCIAL OFF-THE-SHELF SIMULATION PACKAGE INTEROPERABILITY

Simon J. E. Taylor

Centre for Applied Simulation Modelling
School of Information Systems, Computing & Maths
Brunel University
Uxbridge, Middlesex, UB8 3PH, UK

Stephen J. Turner

Parallel & Distributed Computing Centre
School of Computer Engineering
Nanyang Technological University
Singapore 639798, SINGAPORE

Steffen Strassburger

School of Economic Sciences
Technical University of Ilmenau
Helmholtzplatz 3
98693 Ilmenau, GERMANY

ABSTRACT

Commercial-off-the-shelf (COTS) Simulation Packages (CSPs) are widely used visual interactive modeling environments such as Arena™, Anylogic™, Flexsim™, Simul8™, Witness™, etc. CSP Interoperability (or distributed simulation) is a technique that allows a simulation to be executed over several computers or for several simulations running on different computers to run together. This also relates to simulation languages such as SLX™ and GPSS/H™. There have been various attempts to interoperate these CSPs, some with the IEEE 1516 High Level Architecture (HLA). These can be quite complex and it is easy to loose track of exactly what is occurring between interoperating CSPs and their models. This paper introduces a set of Interoperability Reference Models (IRMs), or design patters for CSP Interoperability, that can be used as guidelines to simplify the interoperability process.

1 INTRODUCTION

What are COTS Simulation Packages? Discrete-event simulation has been used to analyze production and logistics problems in many areas such as commerce, defense, health, manufacturing and logistics for many years. The first discrete-event simulation languages appeared in the late 1950s. These evolved during the 1960s and 1970s. With the arrival of the IBM PC, the 1980s saw the rise of visual interactive modeling environments that allowed simulation modelers to visually create and simulate discrete-event models. These have matured into the Commercial-off-the-shelf (COTS) Simulation Packages (CSPs) that are very familiar to simulation modelers today. They include Arena™, Anylogic™, Flexsim™, Simul8™, Witness™, etc. Each has a wide range of functionality including visual model building, simulation run support, animation, optimization and virtual reality. Some have their own dedicated programming language and all are able to be linked to other COTS software (such as Microsoft Excel). Nearly all CSPs only run under Microsoft Windows™. CSPs are typically used by modelers skilled in operations/operational research and management science. Related languages include SLX™ and GPSS/H™.

The simple act of linking together, or interoperating, two or more CSPs and their models, can be extremely complex. This is due to time synchronization requirements and the complexity of distributed simulation algorithms and/or software used to create the link (such as the runtime infrastructures based on the IEEE 1516 High Level Architecture standard (IEEE 2000)) (Fujimoto, 2000). This complexity can often hide the precise nature of what is being shared between these interoperating CSPs. To attempt to simplify this, the Simulation Interoperability Standards Organization's (SISO) COTS Simulation Package Interoperability Product Development Group (CSPI PDG) are developing approaches to the standardization and simplification of CSP interoperability. The first major development by the CSPI PDG is a set of Interoperability Reference Models (IRMs) to help make this simplification possible. First introduced in detail in Taylor, et al. (2006),

these IRMs are effectively design patterns for CSP interoperability. The purpose of this paper is therefore to introduce these IRMs as a set of guidelines for CSP interoperability.

The paper is structured as follows. Section 2 presents a short review of CSP interoperability. Section 3 gives the justification of why IRMs are needed for CSP interoperability. Sections 5-7 presents an overview of the current set of IRMs. Section 8 presents an example of their use. Section 9 introduces the next step in this area and section 10 concludes the paper.

2 COTS SIMULATION PACKAGE INTEROPERABILITY

Consider the following example scenarios for CSP interoperability:

- A supply chain distributes equipment to front line troops. A model is built to represent the different supply centers and transportation links to various battlefronts. Experimentation investigates the reliability of the supply chain under different threat conditions.
- An automotive company is planning to build a new factory. The manufacturing line is modeled and simulated using a CSP. Experimentation investigates how many engines can be produced in one year against different levels of resources (machines, buffers, workers, etc.)
- A regional health authority needs to plan the best way of distributing blood to different hospitals. A model is built using a CSP. Experimentation is carried out to investigate different supply policies against "normal" and emergency supply situations.
- A police authority needs to determine how many officers need to be on patrol and how many need to be in the different police stations that it manages. A model is built and experiments are carried out to investigate staffing against different scenarios (football matches, terrorist attacks, etc.)
- A bank sells different financial products. When a new product is planned, managers need to determine the resource impact against current financial services. Using existing business process models (in BPMN for example), a new model is built and simulated. Experiments investigate different resource levels in different departments.

All the above cases are examples of where CSP interoperability has been used or is planned to be used. What is common is that the models that are being linked together cannot be easily moved. For example, in the automobile case, models are linked to extensive data sources that cannot be easily relocated to other places for reasons of confidentiality or because of linked to fixed data sources (Taylor, et al. 2005). In the case of the health care example, models needed to be interoperated due to the need to share the processing load over several computers. What is common to all these examples are that there is no single approach, i.e. virtually ever approach reported in simulation literature is different and incompatible. Further, vendors and research groups tend not to build on each other's approaches and therefore tend to "build from scratch" each time CSP interoperability is required. This is not intended to be a criticism as there are some quite justifiable reasons for this. The problem is that without a common approach, CSP interoperability will never become the low cost/complexity "plug and play" approach that has been repeatedly called for (Lendermann, et al., 2007). Why is this a complex problem?

Consider the following. The owners of two factories want to find out how many products their factories can manufacture in a year. Both factories have been modeled separately using two CSPs. As shown in figure 1, the (extremely simplistic) factories, modeled as models M1 and M2, are simulated in their own CSPs running on their own separate computers. Queues, activities and resources are represented as Q, A and R respectively. The models interact, in this example, as denoted by the thin arrows connecting the models (possibly the delivery and return of some defective stock). Further, the models might share resources (to reflect a shared set of machinists that can operate various workstations), events of various kinds (such an emergency shutdown) or data (such as the current production volume). The question is, how do we implement this distributed simulation of interoperating CSPs and their models?

A distributed simulation or federation is composed of a set of CSPs and their models. In this paper, a CSP will simulate its model using a discrete-event simulation algorithm. Each model/CSP represents a federate normally running on its own computer. In a distributed simulation, each model/CSP federate therefore exchanges data directly or via a runtime infrastructure (RTI) implemented over a network in a time synchronized manner (as denoted by the thick double-headed arrow). Federate F1 consists of the model M1 and the COTS Simulation Package CSP1 and federate F2 consists of the model M2 and COTS Simulation Package CSP2. In this case federate F1 publishes and sends information to the RTI in an agreed format and time synchronized manner and federate F2 must subscribe to and receive that information in the same agreed format and time synchronized manner, i.e. both federates must agree on a common representation of data and both must use the RTI in a similar way. Further, the "passing" of entities and the sharing of resources require different distributed simulation protocols. In entity passing, the departure of an entity from one model and the arrival of an entity at another can be the same scheduled event in the two models – most distributed simulations represent this as a timestamped event message sent from one federate to another. The

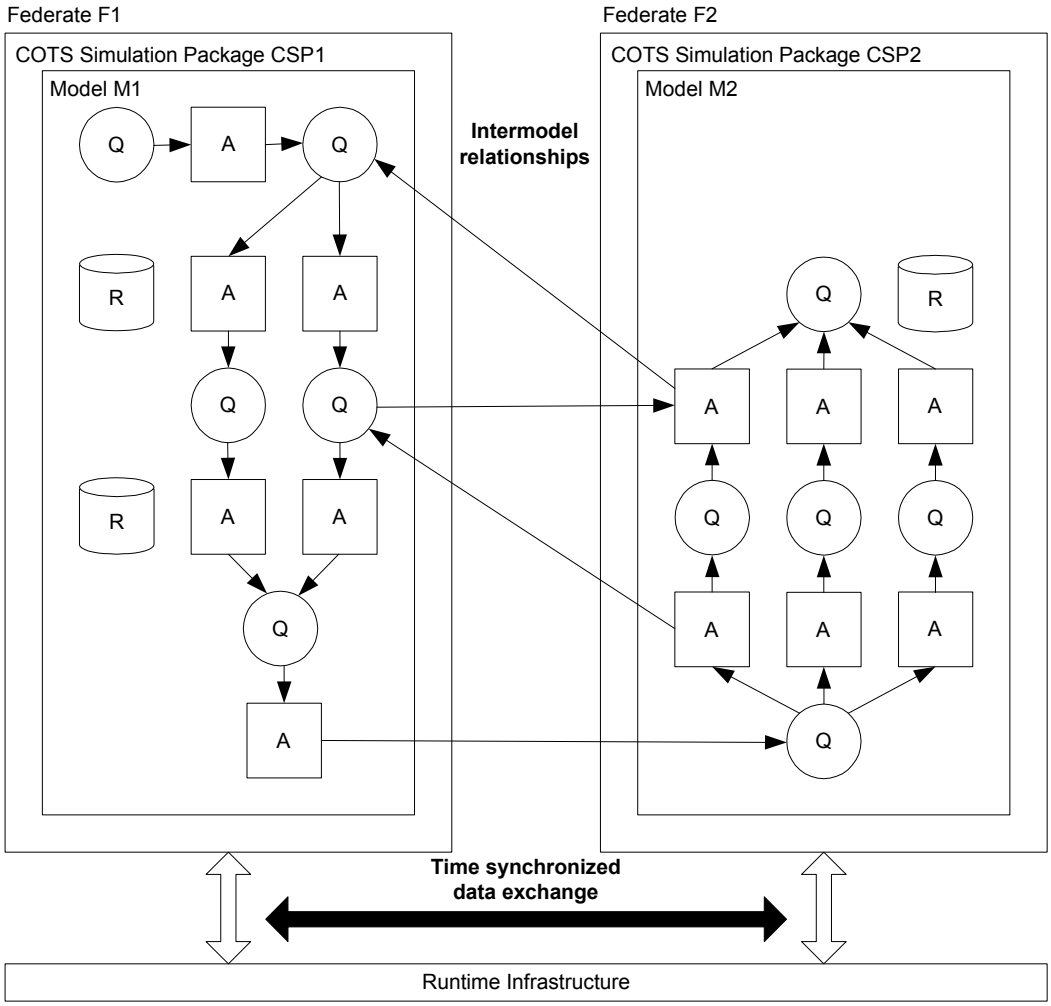

Figure 1: The COTS Simulation Package Interoperability Problem

sharing of resources cannot be handled in the same way. For example, when a resource is released or an entity arrives in a queue, a CSP executing the simulation will determine if a workstation can start processing an entity. If resources are shared, each time an appropriate resource changes state a timestamped communication protocol is required to inform and update the changes of the shared resource state. Further problems arise when we begin to "dig" further into the subtleties of interoperability.

Let us now introduce an approach to simplifying this problem.

3 INTEROPERABILITY REFERENCE MODELS

Different CSPs execute their discrete-event simulation algorithms slightly differently. The approaches to CSP interoperability developed by various researchers and CSP vendors are all different. Indeed the degree of *subtlety* involved in even describing the CSP interoperability problem can lead to long, lengthy discussions where the parties involved typically finish with no definitive understanding of the problems that must be solved. To attempt to solve this, the CSPI PDG has created a standardized set of Interoperability Reference Models or "interoperability design patterns" that attempt to capture these subtleties. These are effectively a set of simulation patterns or templates, that enable modelers, vendors and solution developers to specify the interoperability problems that must be solved. The Interoperability Reference Models (IRMs) are intended to be used as follows:

- to clearly *identify* the model/CSP interoperability *capabilities* of an *existing* distributed simulation, e.g. The distributed supply chain simulation is compliant with IRMs Type A.1, A.2 and B.1.
- to clearly *specify* the model/CSP interoperability *requirements* of a *proposed* distributed simulation, e.g. The distributed hospital simulation must be compliant with IRMs Type A.1 and C.1.

An IRM is defined as the simplest representation of a problem within an identified interoperability problem type.

Each IRM can be subdivided into different subcategories of problem. As IRMs are usually relevant to the boundary between two or more interoperating models, models specified in IRMs will be as simple as possible to "capture" the interoperability problem and to avoid possible confusion. These simulation models are intended to be representative of real model/CSPs but use a set of "common" model elements that can be mapped onto specific CSP elements (see 3.1 Clarification of Terms). Where appropriate, IRMs will specify time synchronization requirements and will present alternatives. IRMs are intended to be cumulative (i.e. some problems may well consist of several IRMs). Most importantly, IRMs are intended to be understandable by *simulation developers, CSP vendors and technology solution providers.*

3.1 Clarification of Terms

As indicated above, an IRM will typically focus on the boundary between interoperating models. To describe an interoperability problem we therefore need to use model elements that are as general as possible. Generally, CSPs using discrete-event simulation model systems that change state at *events*. Rather than providing a set of APIs to directly program discrete-event simulations, these CSPs use a visual interface that allows modelers to build models using a set of objects. These models are typically composed of networks of alternating *queues* and *activities* that represent, for example, a series of buffers and operations composing a manufacturing system. *Entities*, consisting of sets of typed variables termed *attributes*, represent the elements of the manufacturing system undergoing machining. Entities are transformed as they pass through these networks and may enter and exit the model at specific points. Additionally, activities may compete for *resources* that represent, for example, the operators of the machines. To simulate a model a CSP will typically have a simulation executive, an event list, a clock, a simulation state and a number of event routines. The simulation state and event routines are derived from the simulation model. The simulation executive is the main program that (generally) simulates the model by first advancing the simulation clock to the time of the next event and then performing all possible actions at that simulation time. For example, this may change the simulation state (for example ending a machining activity and placing an entity in a queue) and/or schedule new events (for example a new entity arriving in the simulation). This cycle carries on until some terminating condition is met (such as running until a given time or a number of units are made).

A problem is, however, that virtually every CSP has a different variant of the above. CSPs also have widely differing terminology, representation and behavior. For example, without reference to a specific CSP, in one CSP an entity as described above may be termed an *item* and in another *object*. In the first CSP the data types might be limited to integer and string, while in the other the data types might be the same as those in any object-oriented programming language. The same observations are true for the other model elements such as queue, activity and resource. Behavior is also important as the set of rules that govern the behavior of a network of queues and activities subtly differ between CSPs (for example the rules that govern behavior when an entity leaves a machine to go to a buffer). Indeed even the representation of *time* can differ. This is also further complicated by variations in model elements over and above the "basic" set (e.g. entry/exit points, transporters, conveyors, flexible manufacturing cells, robots, etc.)

3.2 Interoperability Reference Model Types

There are currently four different types of IRM. These are:

Type A:	Entity Transfer
Type B:	Shared Resource
Type C:	Shared Event
Type D:	Shared Data Structure

Briefly, IRM Type A Entity Transfer deals with the requirement of transferring entities between simulation models, such as an entity *Part* leaves one model and arrives at the next. IRM Type B Shared Resource refers to sharing of resources across simulation models. For example, a resource R might be common between two models and represents a pool of workers. In this scenario, when a machine in a model attempts to process an entity waiting in its queue it must also have a worker. If a worker is available in R then processing can take place. If not then work must be suspended until one is available. IRM Type C Shared Event deals with the sharing of events across simulation models. For example, when a variable within a model reaches a given threshold value (a quantity of production, an average machine utilization, etc.) it should be able to signal this fact to all models that have an interest in this fact (to throttle down throughput, route materials via a different path, etc.) IRM Type D Shared Data Structure deals with the sharing of variables and data structures across simulation models. Such data structures are semantically different to resources, for example a bill of materials or a common inventory.

Note that the above classification previously appeared as:

Type I:	Asynchronous Entity Passing
Type II:	Synchronous Entity Passing (Bounded Buffer)
Type III:	Shared Resources
Type IV:	Shared Events
Type V:	Shared Data Structures
Type VI:	Shared Conveyor

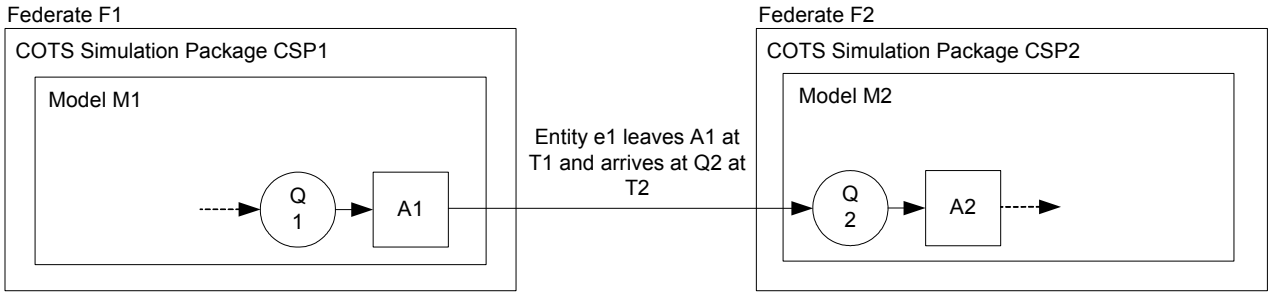

Figure 2: IRM Type A.1: General Entity Transfer

This has been rationalized to the Type A-D classification to "group" IRM problems (essentially new Entity Transfer problems were identified). Note that the "Shared Conveyor" IRM has been deleted as it was felt by the PDG that this would usually be represented as a separate model and therefore fall into the other IRM Types.

4 INTEROPERABILITY REFERENCE MODEL TYPE A: ENTITY TRANSFER

4.1 Overview

IRM Type A Entity Transfer represents interoperability problems that can occur when transferring an entity from one model to another. Figure 2 shows an illustrative example of the problem of Entity Transfer where an entity e1 leaves activity A1 in model M1 at T1 and arrives at queue Q2 in model M2 at T2. For example, if M1 is a car production line and M2 is a paint shop, then this represents the system where a car leaves a finishing activity in M1 at T1 and arrives in a buffer in M2 at T2 to await painting.

Note that the IRM subtypes are intended to be composable, i.e. a distributed simulation that correctly transfers entities from one model to a bounded buffer in another model should be can be compliant with both IRM Type A.1 General Entity Transfer and IRM Type A.2 Bounded Receiving Element.

4.2 Interoperability Reference Model Type A Sub-types

There are currently three IRM Type A Sub-types

- IRM Type A.1 General Entity Transfer
- IRM Type A.2 Bounded Receiving Element
- IRM Type A.3 Multiple Input Prioritization

4.3 IRM Type A.1 General Entity Transfer

4.3.1 Overview

IRM Type A.1 General Entity Transfer represents the case, as described above and shown in figure 2, where an entity e1 leaves activity A1 in model M1 at T1 and arrives at queue Q2 in model M2 at T2 (see above for an example). This IRM is inclusive of cases where

- there are many models and many entity transfers (all transfers are instances of this IRM).

This IRM does not include cases where

- the receiving element is bounded (IRM Type A.2), and
- multiple inputs need to be prioritized (IRM Type A.3).

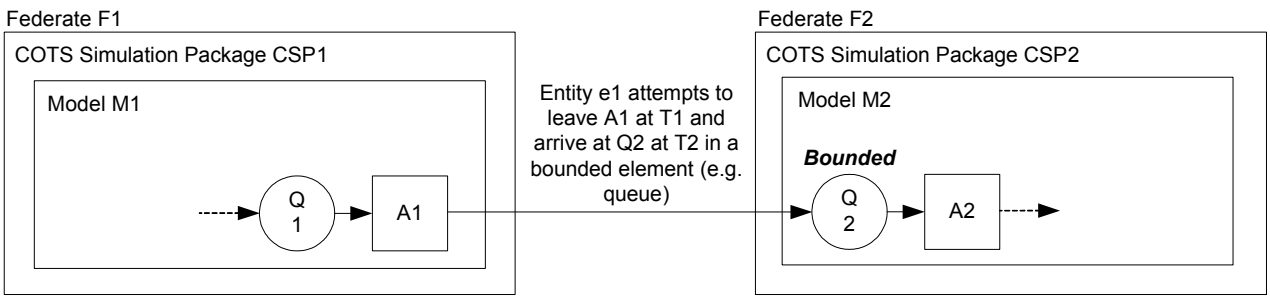

Figure 3: IRM Type A.2: Bounded Receiving Element

4.3.2 Definition

The IRM Type A.1 General Entity Transfer is defined as the transfer of entities from one model to another such that an entity e1 leaves model M1 at T1 from a given place and arrives at model M2 at T2 at a given place and T1 =< T2 or T1<T2. The place of departure and arrival will be a queue, workstation, etc. Note that this inequality must be specified.

4.4 IRM Type A.2 Bounded Receiving Element

4.4.1 Overview

Consider a production line where a machine is just finishing working on a part. If the next element in the production process is a buffer in another model, the part will be transferred from the machine to the buffer. If, however, the next element is ***bounded***, for example a buffer with limited space or another machine (i.e. no buffer space), then a check must be performed to see if there is space or the next machine is free. If there is no space, or the next machine is busy, then to correctly simulate the behavior of the production process, the current machine must hold onto the part and ***block***, i.e. it cannot accept any new parts to process until it becomes unblocked (assuming that the machine can only process one part at a time). The consequences of this are quite subtle. This is the core problem of the IRM Type A.2. Figure 3 shows an illustrative example, where an entity e1 attempts to leave model M1 at T1 from activity A1 and to arrive at model M2 at T2 in ***bounded*** queue Q2. If A1 represents a machine then the following scenario is possible. When A1 finishes work on a part (an entity), it attempts to pass the part to queue Q2. If Q2 has spare capacity, then the part can be transferred. However, if Q2 is full then A1 cannot release its part and must block. Parts in Q1 must now wait for A1 to become free before they can be machined. Further, when Q2 once again has space, A1 must be notified that it can release its part and transfer it to Q2. Finally, it is important to note the fact that if A1 is blocked the rest of model M1 still functions as normal, i.e. a correct solution to this problem must still allow the rest of the model to be simulated (rather than just stopping the simulation of M1 until Q2 has unblocked).

This IRM is therefore inclusive of cases where

- the receiving element (queue, workstation, etc.) is bounded.

This IRM does not include cases where

- multiple inputs need to be prioritized (IRM Type A.3).

A solution to this IRM problem must also

- be able to transfer entities (IRM Type A.1).

4.4.2 Definition

The IRM Type A.2 is defined as the relationship between an element O in a model M1 and a bounded element Ob in a model M2 such that if an entity e is ready to leave element O at T1 and attempts to arrive at bounded element Ob at T2 then:

- If bounded element Ob is empty, the entity e can leave element O at T1 and arrive at Ob at T2, or
- If bounded element Ob is full, the entity e cannot leave element O at T1; element O may then block if appropriate and must not accept any more entities.
- When bounded element Ob becomes not full at T3, entity e must leave O at T3 and arrive at Ob at T4; element O becomes unblocked and may receive new entities at T3.
- T1=<T2 and T3=<T4.
- If element O is blocked then the simulation of model M1 must continue.

Note:

- In some special cases, element O may represent some real world process that may not need to block.
- If T3<T4 then it may be possible for bounded element O to become full again during the interval if other inputs to Ob are allowed.

4.5 IRM Type A.3 Multiple Input Prioritization

4.5.1 Overview

As shown in figure 4, the IRM Type A.3 Multiple Input Prioritization represents the case where a model element such as queue Q1 (or workstation) can receive entities from multiple places. Let us assume that there are two models M2 and M3 which are capable of sending entities to Q1 and that Q1 has a First-In-First-Out (FIFO) queuing discipline. If an entity e1 is sent from M2 at T1 and arrives at Q1 at T2 and an entity e2 is sent from M3 at T3 and arrives at Q1 at T4, then if T2<T4 we would expect the order of entities in Q1 would be e1, e2. A problem arises when both entities arrive at the same time, i.e. when T2=T4. Depending on implementation, the order of entities would either be e1, e2 or e2, e1. In some modeling situations it is possible to specify the *priority* order if such a conflict arises, e.g. it can be specified that model M1 entities will always have a higher priority than model M2 (and therefore require the entity order e1, e2 if T2=T4). Further, it is possible that this priority ordering could be dynamic or specialized.

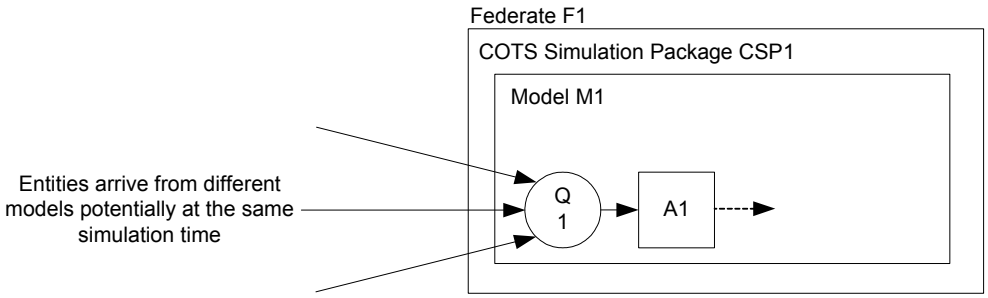

Figure 4: IRM Type A.3 Multiple Input Prioritization

This IRM is therefore inclusive of cases where

- multiple inputs need to be prioritized.

This IRM does not include cases where

- the receiving element is bounded (IRM Type A.2).

A solution to this IRM problem must also

- be able to transfer entities (IRM Type A.1).

4.5.2 Definition

The IRM Type A.3 Multiple Input Prioritization is defined as the preservation of the priority relationship between a set of models that can send entities to a model with receiving queue Q, such that priority ordering is observed if two or more entities arrive at the same time.

Note:

- The priority rules must be specified.
- Priority rules may change during a simulation if required for the real system being simulated.

5 INTEROPERABILITY REFERENCE MODEL TYPE B: SHARED RESOURCE

5.1 Overview

IRM Type B deals with the problem of sharing resources across two or more models in a distributed simulation. A modeler can specify if an activity requires a resource (such as machine operators, doctors, runways, etc.) of a particular type to begin. If an activity does require a resource, when an entity is ready to start that activity, it must therefore be determined if there is a resource available. If there is then the resource is secured by the activity and held until the activity ends. A resource shared by two or more models therefore becomes a problem of maintaining the consistency of the state of that resource in a distributed simulation. Note that this is similar to the problem of shared data. However, in CSPs resources are semantically different to data and we therefore preserve the distinction in this standard.

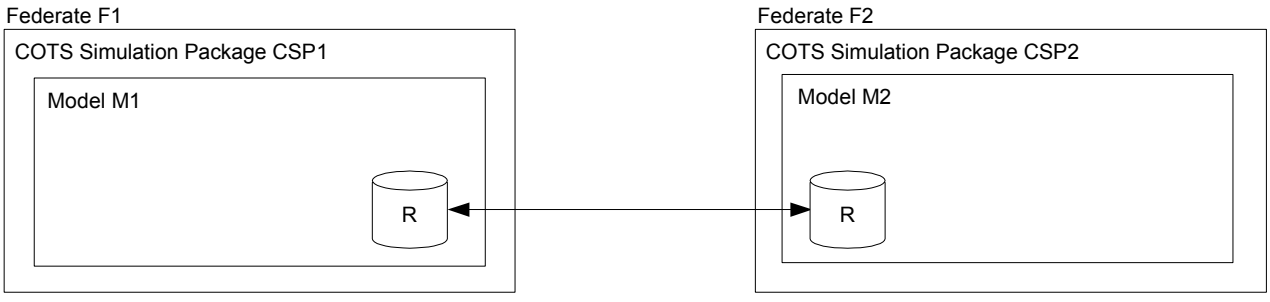

Figure 5: IRM Type B.1: General Shared Resource

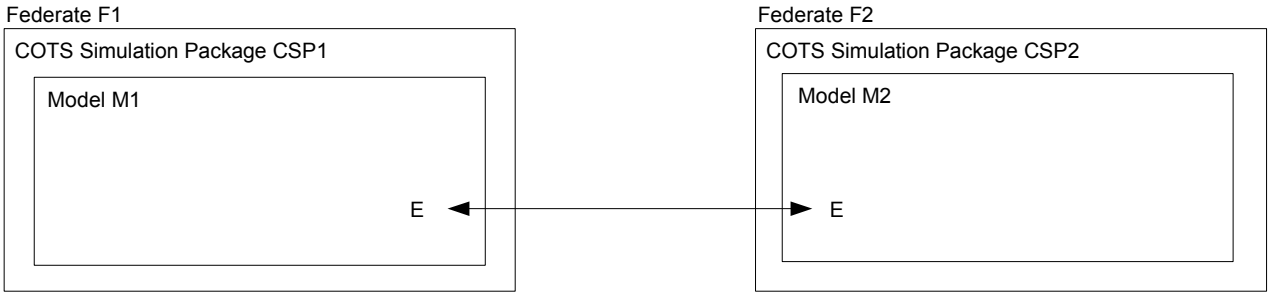

Figure 6: IRM Type C.1: General Shared Event

5.2 Interoperability Reference Model Type B Sub-types

There is currently one IRM Type B Sub-type

- IRM Type B.1 General Shared Resource

5.3 IRM Type B.1 General Shared Resource

5.3.1 Overview

IRM Type B.1 General Shared Resource represents the case, as outlined above and shown in figure 5, where the state of a resource R shared across two or more models must be consistent. In a model M1 that shares resource R with model M2, M1 will have a copy RM1 and M2 will have a copy RM2. When M1 attempts to change the state of RM1 at T1, then it must be guaranteed that the state of RM2 in M2 at T1 will also be the same. Additionally, it must be guaranteed that *both* M1 and M2 can attempt to change their copies of R at the same simulation time as it cannot be guaranteed that this simultaneous behavior will not occur.

5.3.2 Definition

The IRM Type B.1 General Shared Resources is defined as the maintenance of consistency of all copies of a shared resource R such that

- if a model M1 wishes to change its copy of R (RM1) at T1 then the state of all other copies of R will be guaranteed to be the same at T1, and
- if two or more models wish to change their copies of R at the same time T1, then all copies of R will be guaranteed to be the same at T1.

6 INTEROPERABILITY REFERENCE MODEL TYPE C: SHARED EVENT

6.1 Overview

IRM Type C deals with the problem of sharing events (such as an emergency signal, explosion, etc.) across two or more models in a distributed simulation.

6.2 Interoperability Reference Model Type C Sub-types

There is currently one IRM Type C sub-type

- IRM Type C.1 General Shared Event

6.3 IRM Type C.1 General Shared Event

6.3.1 Overview

IRM Type C.1 General Shared Event represents the case, as shown in figure 6, where an event E is shared across two or more models. In a model M1 that shares an event E with model M2 at T1, then we are effectively scheduling two local events EM1 at M1 at T1 and EM2 at M2 at T1. We must therefore guarantee that both copies of the event take place. Care must also be taken to guarantee if two shared events E1 and E2 are instigated at the same time by different models, then both will occur.

6.3.2 Definition

The IRM Type C.1 General Shared Event is defined as the guaranteed execution of all local copies of a shared event E such that

- if a model M1 wishes to schedule a shared event E at T1, then the local copies EM1, EM2, etc. will be guaranteed to be executed at the same time T1, and

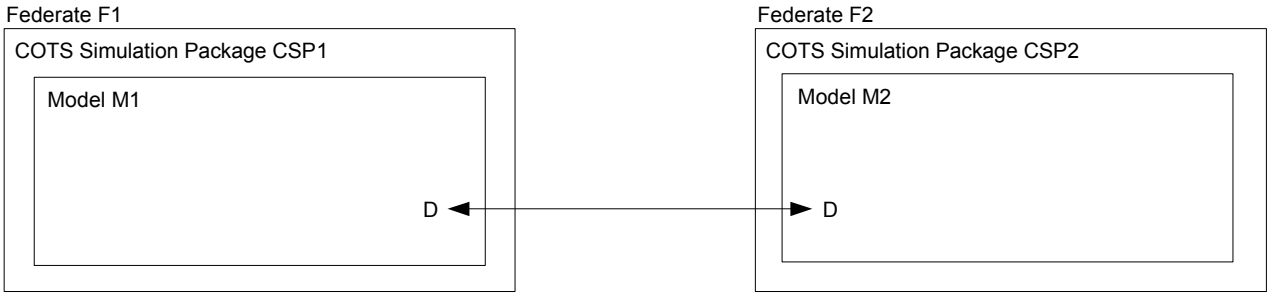

Figure 7: IRM Type D.1: Shared Data

- if two or more models wish to schedule shared events E1, E2, etc. at T1, then all local copies of all shared events will be guaranteed to be executed at the same time T1.

7 INTEROPERABILITY REFERENCE MODEL TYPE D: SHARED DATA STRUCTURE

7.1 Overview

IRM Type D deals with the problem of sharing data across two or more models in a distributed simulation (such as a production schedule, a global variable, etc.) A shared data structure that is shared by two or more models therefore becomes a problem of maintaining the consistency of the state of that data structure in a distributed simulation. Note that this is similar to the problem of shared resources. However, in CSPs resources are semantically different to data and we therefore preserve the distinction in this standard. Note also that we consider the sharing of a single data item such as an integer as being covered by this IRM.

7.2 Interoperability Reference Model Type D Sub-types

There is currently one IRM Type D Sub-type.

- IRM Type D.1 General Shared Data Structure

7.3 IRM Type D.1 General Shared Data Structure

7.3.1 Overview

IRM Type D.1 General Data Structure represents the case, as outlined above and shown in figure 7, where a data structure D shared across two or more models must be consistent. In a model M1 that shares a data structure D with model M2, M1 will have a copy DM1 and M2 will have a copy DM2. When M1 attempts to change the value of DM1 at T1, then it must be guaranteed that the value of DM2 in M2 at T1 will also be the same. Additionally, it must be guaranteed that *both* M1 and M2 can attempt to change their copies of D at the same simulation time as it cannot be guaranteed that this simultaneous behavior will not occur.

7.3.2 Definition

The IRM Type D.1 General Shared Data Structure is defined as the maintenance of consistency of all copies of a shared data structure D such that

- if a model M1 wishes to change its copy of D, DM1 at T1 then the value of all other copies of D will be guaranteed to be the same at T1, and
- if two or more models wish to change their copies of D at the same time T1, then all copies of D will be guaranteed to be the same at T1.

8 EXAMPLE

The UK National Blood Service (NBS) is a public funded body in the UK that is responsible for distributing blood and associated products. The analysis of this health care supply chain is of particular interest as blood donors are in short supply, the shelf-life of blood products is relatively short and blood product ordering policies are potentially complex. The UK NBS is a part of the National Health Service (NHS) Blood and Transplant (NHSBT) organization. The NBS is responsible for collecting blood through voluntary donations, testing the blood for ABO and Rhesus grouping and infectious diseases such as HIV, processing the blood into around 120 different products (of which the main three are Red Blood Cells, plasma and platelets), storing the stockpile and transferring excess stock between different NBS centers, and finally issuing the different blood products to the hospitals as per their needs. The NBS infrastructure consists of 15 Process, Testing and Issuing

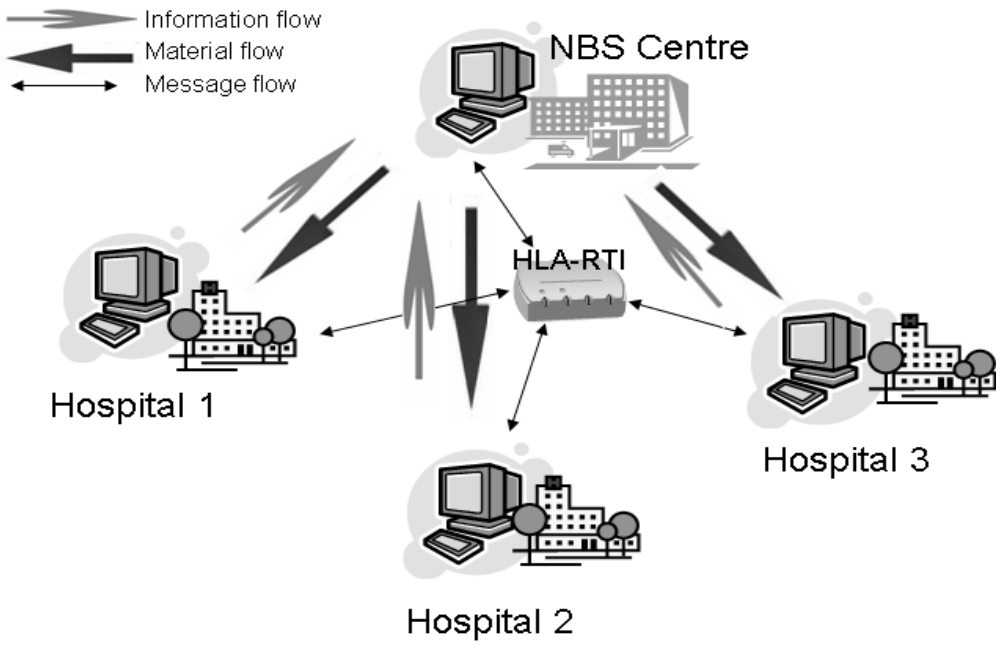

Figure 8: Distributed National Blood Service Distributed Simulation

(PTI) centers which together serve 316 hospitals across England and North Wales.

Blood products are stored in the PTI Centers until they are requested by the hospitals served by that Center. A hospital places an order for blood products when its inventory falls below a predetermined order point, or when rare products not held in stock are requested for particular patients. Hospitals normally receive their orders daily and the blood remains in the hospital bank until it is cross-matched (tested for compatibility) for a named patient. It is then placed in "assigned inventory" for that patient for a fixed time after the operation. If it is not used, it is returned to "unassigned inventory" and can be cross-matched again for another patient. On average a unit will be cross-matched four times before it is used or outdated. In practice, however, only half of the cross-matched blood is actually transfused. The original simulation ran on one PC and is described in (Katsaliaki and Brailsford 2006).

The problem faced by this simulation is speed. Katsaliaki, et al. (2007) developed a distributed simulation that demonstrated that considerably length runtimes on a single computer could be reduced by distributing the simulation over several PCs. How exactly was this distributed simulation designed? Without the use of the IRMs it would be difficult to write down the interoperability requirements in a common "language." With the IRMs this task become quite straightforward. There are no shared resources, events or data structures; the distributed simulation only requires the exchange of entities. There are two types of entity: orders and blood units. There are no bounded buffers in this model and there is no need to preserve queuing discipline when multiple entities arrive simultaneously. There is a travel time between the PTI Centre and hospitals. We can therefore quite clearly and simply state:

The NBS distributed simulation is compliant with IRM Type A.1 such that T1<T2 in all cases.

The IRMs make this simplification possible. Let us now consider our next steps, the development of the "best" interoperability approaches.

9 FURTHER WORK

The next step in the standardization of CSPI interoperability is the comparison of different interoperability approaches for different case studies. The selection of these will be done by the CSPI PDG and an open call for participation in these "interoperability games" will appear in the future and will help determine the "best" interoperability approaches in terms of relevant factors such as appropriateness, implementation complexity and performance. One problem in this is the choice of a runtime infrastructure or algorithm on which to base the comparison. Links to commercial infrastructures are being made. We now present one possible novel infrastructural approach that takes advantage of contemporary developments in Grid Computing that may form part of our interoperability games.

The development of many complex simulation applications requires collaborative effort from researchers with different domain knowledge and expertise, possibly at different locations. These simulation systems often require large amounts of computing resources and data sets which may be geographically distributed. In order to support collaborative model development and to cater for the increasing complexity of such systems, it is necessary to harness

distributed resources over the Internet. The emergence of Grid technologies provides exciting new opportunities for large scale distributed simulation, enabling collaboration and the use of distributed computing resources, while also facilitating access to geographically distributed data sets.

Traditionally, HLA-based distributed simulations are conducted using a vendor-specific RTI software and federates with different RTI versions cannot cooperate with each other. To run a distributed simulation over a WAN, the required software and hardware resource arrangements and security settings must be made before the actual simulation execution. Because of this inflexibility, it is not easy to run HLA-based distributed simulations across administrative domains. To address these inflexibility issues and leverage globally pervasive resources for distributed simulations, the Grid is naturally considered as a solution.

Grid computing was proposed by Foster as flexible, secure and coordinated resource sharing among dynamic collections of individuals, institutions and resources (Foster, Kesselman, and Tuecke 2001). Among the various available Grid middlewares, Globus Toolkit (Globus 2008) is the de facto standard middleware for Grid computing. Its latest version GT4 contains five components, namely Common Runtime, Security, Data Management, Information Services and Execution Management, to facilitate heterogeneous resource sharing.

Three approaches can be defined for HLA-based distributed simulation on the Grid (Pan et al. 2007), namely a Grid-facilitated approach, a Grid-enabled approach and a Grid-oriented approach. In the Grid-facilitated approach, Grid services are defined to facilitate the execution of HLA-based distributed simulations while the actual simulation communications are through a vendor-specific RTI. An example of this approach is the Grid HLA Management System (G-HLAM) proposed by Rycerz (2006) for efficient execution of HLA-based distributed simulations on the Grid.

In the Grid-enabled approach, Grid (or web) service interfaces are provided to enable HLA-based distributed simulations to be conducted in a Grid (or web) environment. A client federate communicates with a federate server using Grid (or web) service communications and the federate server representing the client federate joins an HLA-based distributed simulation using a vendor-specific RTI. An example of this approach is the work done by the XMSF group (Pullen et al. 2005) to integrate simulations with other applications using web services.

In the Grid-oriented approach, the RTI is implemented using Grid services according to the HLA specification. All communications are through Grid service invocations. This approach was raised in Fox's keynote at DSRT 2005 (Fox 2005).

Nanyang Technological University (Singapore) has developed a Service Oriented HLA RTI (SOHR) framework which implements an HLA RTI entirely using Grid services following the Grid-oriented approach. The various Grid services of SOHR cooperate with each other to provide the functionalities of an RTI as services. Federates participate in federations by invoking specific Grid services without the installation of a heavy-weight vendor-specific RTI software at the local site. Since Grid services are used for communications, firewalls can be pierced and distributed simulations across administrative domains can be conveniently conducted on SOHR. Moreover, the various Grid service components can be dynamically deployed, discovered and undeployed on demand. All these features of SOHR enable scalable execution of HLA-based distributed simulations on the Grid.

The SOHR framework contains seven key Grid services, namely the RTI Index Service, the LS (Local Service) and five management services, all of which are implemented based on GT4. The LS (Local Service) is used as a messaging broker of federates and contains multiple LRIs (Local Resource Instances), with one LRI for each federate. A federate communicates with the outside world through its LRI by invoking services and getting callbacks. SOHR maps the six HLA service groups into different modules in its LRI structure and different management services. This enables the inclusion of different algorithms for an HLA service group, which makes SOHR an extensible framework. A decoupled design is chosen between a federate and its LRI, which enables a federate to be run on resource-limited platforms and simplifies federate migration.

The LRC (Local RTI Component) is a federate's local library that implements the HLA service interfaces and does the translation between HLA service interfaces and the corresponding Grid service invocations. Both the HLA 1.3 specification and the IEEE 1516 standard (IEEE 2000) are provided as alternative LRC libraries. Since a standard interface is supported, CSP interoperability solutions that follow the HLA-CSPI standards may readily be supported by SOHR. This allows the possibility of executing federations of CSP models in a Grid environment. Details of the SOHR framework, together with performance results, may be found in Pan et al. (2008).

10 CONCLUSIONS

This tutorial paper has presented the CSPI PDG IRMs for COTS Simulation Package Interoperability. Reference Models. At the time of writing, the Standard is currently undergoing balloting as SISO-STD-006-2007 (DRAFT). As described above, the next main activity of the CSPI PDG is to classify current CSPI approaches via a series of performance tests based on representative case studies. Please see the CSPI PDG discussion group at www.sisostds.org. The CSPI PDG welcomes new members and volunteers. Please email the CSPI PDG chair simon.taylor@brunel.ac.uk for further details.

ACKNOWLEDGEMENTS

The authors would like to thank the CSPI PDG members and the SISO-STD-006-2007 Balloting Group for their enthusiasm and comments in the development of the Interoperability Reference Models and in particularly to the continued support of the CSP vendors.

REFERENCES

IEEE. 2000. IEEE Standard 1516 (HLA Rules), 1516.1 (Interface Specification) and 1516.2 (Object Model Template).

Foster, I., C. Kesselman and S. Tuecke. 2001. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 15, 3:200-222.

Fox, G., A. Ho, S. Pallickara, M. Pierce and W. Wu. 2005. Grids for the GiG and Real Time Simulations. In Proceedings of 9th IEEE International Symposium on Distributed Simulation and Real Time Applications, 129-138.

Fujimoto, R.M. 2000. *Parallel and Distributed Simulation Systems*. John Wiley and Sons Inc. New York NY, USA.

Globus. 2008. Globus Toolkit Version 4. Available via <http://www.globus.org/> [accessed July 3, 2008].

Katsaliaki, K. and Brailsford, S.C. (2007) Using Simulation to Improve the Blood Supply Chain. *Journal of the Operational Research Society*, 58, (2), 219-227.

Katsaliaki, K., Mustafee, N., Taylor, S.J.E. and Brailsford, S. (2007) Comparing conventional and distributed approaches to simulation in a complex supply-chain health system. *Journal of the Operational Research Society*, doi:10.1057/palgrave.jors.2602531.

Lendermann, P., McGinnis, L.F., McLean, C.R., Taylor, S., Heinicke, M. and Strassburger, S. (2007). Panel: Distributed Simulation in Industry – a Real-World Necessity or Ivory Tower Fancy? In *Proceedings of the 2007 Winter Simulation Conference*, ACM Press, New York, NY.pp. 1053-1062

Pan, K., S.J. Turner, W. Cai, and Z. Li. 2007, A Service Oriented HLA RTI on the Grid. In 2007 International Conference on Web Services, 984-992.

Pan, K., S.J. Turner, W. Cai, and Z. Li. 2008. Design and Performance Evaluation of a Service Oriented HLA RTI on the Grid. In Grid Computing: Infrastructure, Service, and Application, ed, by L. Wang, W. Jie, and J. Chen, Taylor and Francis.

Pullen, J.M., R. Brunton, D. Brutzman, D. Drake, M. Hieb, K. L. Morse and A. Tolk. 2005. Using Web Services to Integrate Heterogeneous Simulations in a Grid Environment. Future Generation Computer Systems, 21, 1:97-106.

Rycerz, K. 2006. Grid-Based HLA Simulation Support. PhD thesis, University van Amsterdam and AGH Krakow.

Taylor, S.J.E., Wang, X., Turner, S.J., Low, M.Y.H. 2006. Integrating Heterogeneous Distributed COTS Discrete-Event Simulation Packages: An Emerging Standards-Based Approach. IEEE Transactions on Systems, Man & Cybernetics: Part A, 36, 1, pp. 109-122.

Taylor, S.J.E., Bohli, L., Wang, X., Turner, S.J. and Ladbrook, J. (2005). Investigating Distributed Simulation at the Ford Motor Company. In Proceedings of the Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications. IEEE Computer Society. pp. 139-147.

AUTHOR BIOGRAPHIES

SIMON J E TAYLOR is the Founder and Chair of the COTS Simulation Package Interoperability Product Development Group (CSPI-PDG) under the Simulation Interoperability Standards Organization. He is the co-founding Editor-in-Chief of the UK Operational Research Society's (ORS) *Journal of Simulation* and the Simulation Workshop series. He was Chair of ACM's Special Interest Group on Simulation (SIGSIM) (2005-2008). He is a Reader in the School of Information Systems, Computing and Mathematics at Brunel and has published over 100 articles in modeling and simulation. His recent work has focused on the development of standards for distributed simulation in industry. His email address is <simon.taylor@brunel.ac.uk>.

STEFFEN STRASSBUGER is a professor at the Ilmenau University of Technology in the School of Economic Sciences. In previous positions he was working as head of the "Virtual Development" department at the Fraunhofer Institute in Magdeburg, Germany and as a researcher at the DaimlerChrysler Research Center in Ulm, Germany. He is a member of the editorial board of the *Journal of Simulation*. His research interests include simulation and distributed simulation as well as general interoperability topics within the digital factory context. He is also the Vice Chair of SISO's COTS Simulation Package Interoperability Product Development Group. His email address is <Steffen.Strassburger@tu-ilmenau.de>.

STEPHEN J. TURNER joined Nanyang Technological University (NTU), Singapore, in 1999 and is Director of the Parallel and Distributed Computing Centre in the School of Computer Engineering. Previously, he was a Senior Lecturer in Computer Science at Exeter University (UK). His current research interests include parallel and distributed simulation, distributed virtual environments, grid computing and multi-agent systems.. He is the secretary of the CSPI PDG and a member of the editorial board of the *Journal of Simulation*. His email address is <ass-jturner@ntu.edu.sg>.