Proceedings of the 2007 Winter Simulation Conference
S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.

THE SISO CSPI PDG STANDARD FOR COMMERCIAL OFF-THE-SHELF SIMULATION PACKAGE INTEROPERABILITY REFERENCE MODELS

Simon J. E. Taylor
Navonil Mustafee

Centre for Applied Simulation Modelling
School of Inf. Systems, Computing & Mathematics
Brunel University
Uxbridge, Middlesex, UB8 3PH, U.K.

Steffen Strassburger

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

Stephen J. Turner
Malcolm Y. H. Low

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

John Ladbrook

Dunton Engineering Centre
Ford Motor Company
Laindon, Basildon, Essex, U.K.

ABSTRACT

For many years discrete-event simulation has been used to analyze production and logistics problems in manufacturing and defense. Commercial-off-the-shelf Simulation Packages (CSPs), visual interactive modelling environments such as Arena, Anylogic, Flexsim, Simul8, Witness, etc., support the development, experimentation and visualization of simulation models. There have been various attempts to create distributed simulations with these CSPs and their tools, some with the High Level Architecture (HLA). These are complex and it is quite difficult to assess how a set of models/CSP are actually interoperating. As the first in a series of standards aimed at standardizing how the HLA is used to support CSP distributed simulations, the Simulation Interoperability Standards Organization's (SISO) CSP Interoperability Product Development Group (CSPI PDG) has developed and standardized a set of Interoperability Reference Models (IRM) that are intended to clearly identify the interoperability capabilities of CSP distributed simulations.

1 INTRODUCTION

The Simulation Interoperability Standards Organization (SISO) focuses on facilitating simulation interoperability across government and non-government applications worldwide. SISO's interests include methods that support and promote reuse of simulation components; agile, rapid, and efficient development and maintenance of models; as well as integration of models into operational systems or embedding real world systems into virtual environments.

For many years discrete-event simulation has been used to analyze production and logistics problems in commerce, defense, health and many areas of manufacturing. In the early 1980s, visual interactive modelling environments were created that supported the development, experimentation and visualization of simulation models. These Commercial-off-the-shelf (COTS) productivity tools have common facilities including drag and drop model development environments, animation, experimentation support and statistical tools. Examples of these tools include Arena, Anylogic, Flexsim, Simul8 and Witness. Collectively, these tools are referred to as COTS Simulation Packages (CSPs).

There are many problems when a distributed simulation consisting of CSPs and their models is created. A single CSP is used to create a single model. A distributed simulation therefore involves the interoperation of a model and its CSP and other model and their CSPs. There have been many different approaches used to create such distributed simulations (some using the IEEE 1516 High Level Architecture and some not). The differences between approaches and their implementations is extremely difficult to capture. Indeed it is often extremely difficult to assess the capabilities of a particular approach. Further, simulation practitioners that use CSPs tend not to be technically minded and vendors of CSPs often find simulation

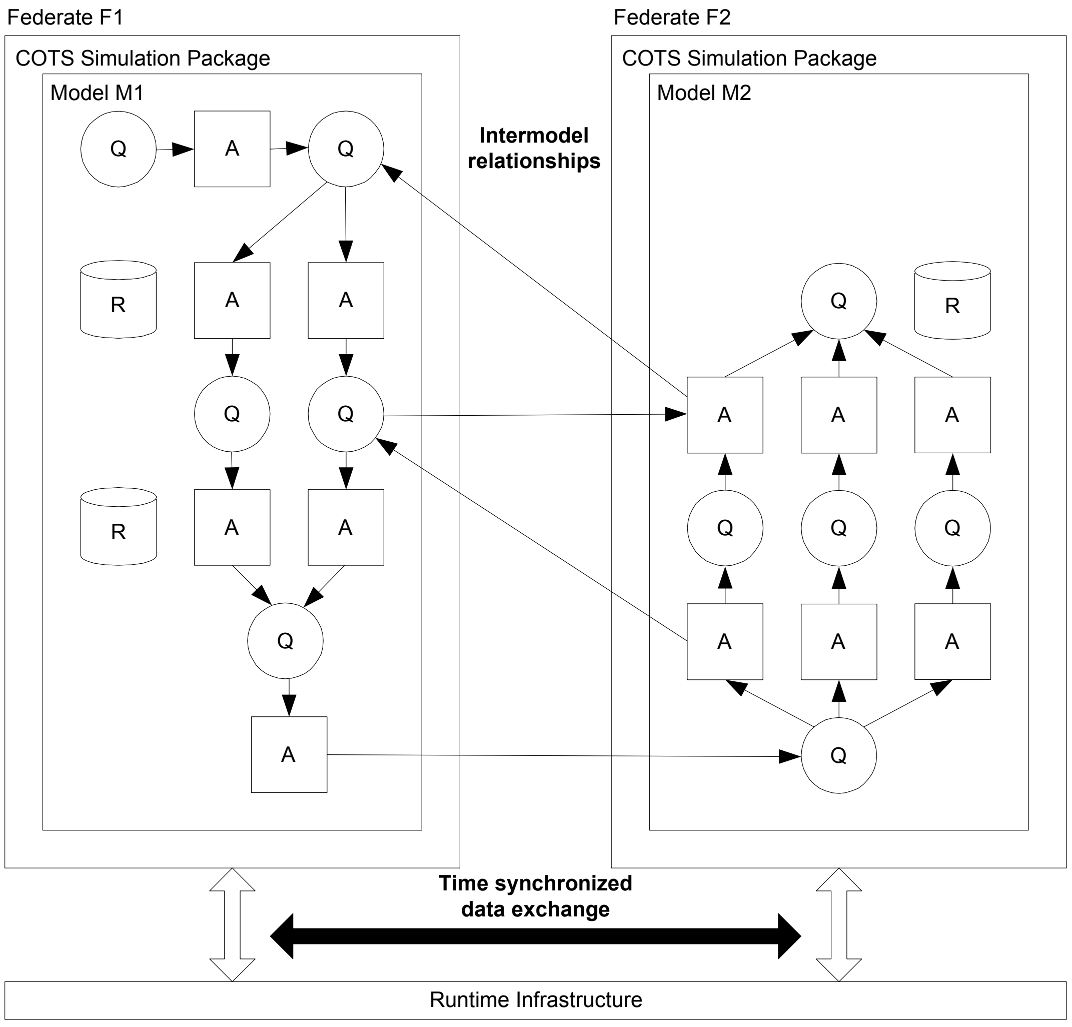

Figure 1: The COTS simulation package interoperability problem.

interoperability techniques inaccessible . It has been the work of SISO's COTS Simulation Package Interoperability Product Development Group (CSPI PDG) to establish common approaches to these problems. As part of this activity a set of "Interoperability Reference Models" have been defined to create a common frame of reference to assess the capabilities of particular approaches and to help practitioners and vendors achieve solutions to complex interoperability problems. This paper reviews the current set of Interoperability Reference Models for CSPs that are being standardized by the CSPI PDG in their *Draft Standard for Commercial-Off-The-Shelf Simulation Package Interoperability Reference Models* (SISO 2007). The wider range of CSPI PDG standards activities can be found in Taylor, et al. (2006).

The paper is structured as follows. Section 2 introduces the problem of CSP interoperability in more detail. Section 3 introduces the Interoperability Reference Models. Sections 4 to 6 define specific Interoperability Reference Models. Section 7 concludes the paper.

2 COTS SIMULATION PACKAGE INTEROPERABILITY

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 kind (such as the end of a shift) or data (such as the current production volume). The question is, how do we implement this distributed simulation?

A distributed simulation or federation is composed of a set of CSPs and their models. Within this document, a CSP will simulate its model using a discrete-event simulation algorithm. Each model/CSP represents a federate normally running on their own computer. In a distributed simulation, each model/CSP federate therefore exchanges data 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/CSP M1 and federate F2 consists of the model/CSP M2. 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 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. It is the purpose of our Interoperability Reference Models to try to simplify this complexity.

3 INTEROPERABILITY REFERENCE MODEL RATIONALE

To reiterate the outline problem stated in the previous section, the complexity of developing, or indeed understanding, a distributed simulation consisting of interoperating CSPs and their models requires some form of simplification to be clear if a distributed simulation correctly implements what it claims to interoperate. *Interoperability Reference Models* (IRMs) are the first deliverable from the CSPI PDG.

An initial set of interoperability problems identified by the CSPI PDG have been divided into a series of problem *types* that are represented by IRMs. An interoperability problem *type* is meant to capture a general class of interoperability problem, while an *IRM* is meant to capture a specific problem within that class. The purpose of an IRM is therefore:

- to clearly *identify* the CSP/model 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 CSP/model interoperability *requirements* of a *proposed* distributed simulation (e.g., the distributed hospital simulation must be compliant with IRMs Type A.1 and C.1).

3.1 Interoperability Reference Model Definition

An IRM is defined as the simplest representation of 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. Where appropriate IRMs will specify time synchronization requirements and will present alternatives where appropriate. IRMs are intended to be composible (i.e. some problems may well consist of several IRMs). Most importantly, IRMs are intended to be understandable by *CSP vendors, simulation users and technology solution providers.*

3.2 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.)

In this standard, we shall adopt a generic set of terms and definitions. These are:

- Model (M): A model represents a real world system and is executed by a CSP.
- Federate (F): A federate consists of a model, its a CSP and interfacing software running a single computer.
- Event (E): An instantaneous state change at a time T. For example, an event E at a time T marks the transition of an entity from a queue to an activity or an activity to a queue or a change to a data structure.
- Time (T): An integer value representing a specific simulation time in a model. As time units are relative, these have no specific units in this standard.
- Entity (e): An entity represents something that is processed, i.e. an entity passes through a series of queues and activities. For example, an entity could represent a patient, an engine part, a task, etc. Entities belong to a class and are differentiated by attributes.
- Queue (Q): A queue of entities. We assume that a queue will have some queuing discipline (FIFO, LIFO, etc.)
- Activity (A): An activity represents some time consuming action in a system with a known duration. The start of an activity is marked by an event and then end of an activity is marked by another event. For example, a workstation in a factory will process an entity for a given time. Typically, an activity will begin if there is an entity in its preceding queue. Optionally, an activity will begin if there is an entity in its preceding queue and a resource available (for example a machine that needs an operator).
- Resource (R): A resource represents something that is needed by an activity to begin. For example, the operator of a machine.
- Data Structure (D): This is similar to a resource but semantically different in terms of modeling and is therefore separately included. For example, a data structure could be an inventory record. In this standard a single variable is considered to be data structure as it has the same interoperability requirements.

Note that all of the above follow different behaviors. These will be addressed as needed in each IRM.

3.3 Interoperability Reference Model Types

There are 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 that are semantically different to resources, for example a bill of materials or a shared inventory.

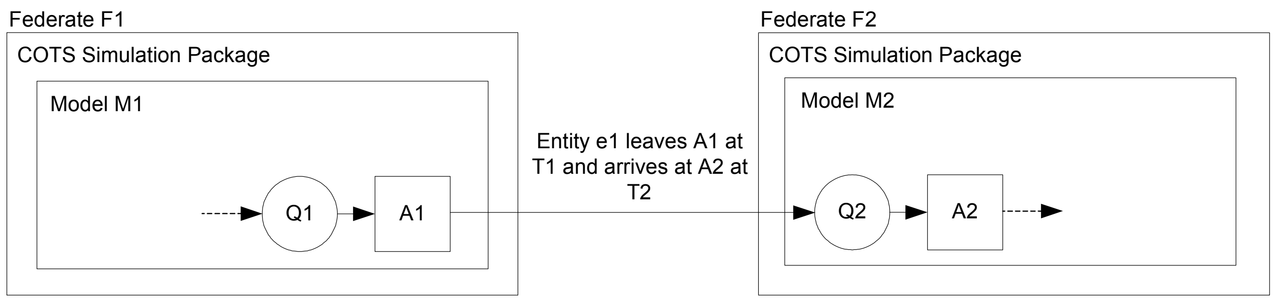

Figure 2: IRM type A.1: General entity transfer.

Note that the previous classification previously appeared as (Taylor et al. 2006):

- 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

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.

Prototype IRMs also carried a UML description. These were removed as in previous CSPI PDG meetings it was felt that they did not add clarity to an IRM.

Note also, and more importantly, that the list of IRMs is not exhaustive. The IRMs are under continuous review by the CSPI PDG. Modifications and new IRMs can be suggested to the CSPI PDG chair and will be reviewed by the PDG for inclusion in subsequent versions of this standard.

3.4 Interoperability Reference Model Use

The Interoperability Reference Models (IRMs) are intended to be used as follows:

- to clearly *identify* the CSP/model 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 CSP/model interoperability *requirements* of a *proposed* distributed simulation (e.g., The distributed hospital simulation must be compliant with IRMs Type A.1 and C.1).

Note that the IRM types and sub-types are intended to be cumulative, i.e. a distributed simulation that correctly transfers entities from one model to a bounded buffer in another model as well as sharing a production schedule (shared data) should be compliant with both IRM Type A.1 General Entity Transfer, IRM Type A.2 Bounded Receiving Element and IRM Type D.1 General Shared Data. Let us now review the different IRMs. The Type D Shared Data Structure has been omitted due to space constraints but is similar to the IRM Type B Shared Resource.

4 INTEROPERABILITY REFERENCE MODEL TYPE A: ENTITY TRANSFER

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

4.1 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

4.2 IRM Type A.1 General Entity Transfer

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

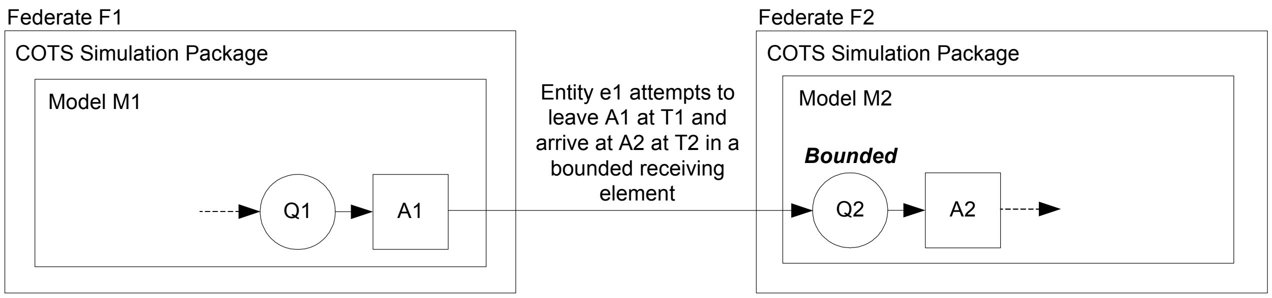

Figure 3: IRM type A.2: Bounded receiving element.

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).

4.2.1 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 in some instances a CSP distributed simulation need only satisfy the condition T1<T2.

4.3 IRM Type A.2 Bounded Receiving Element

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, 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 Bounded Receiving Element. 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).

4.3.1 Definition

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

- If bounded queue Qb is empty, the entity e can leave activity A at T1 and arrive at Qb at T2, or
- If bounded queue Qb is full, the entity e cannot leave activity A at T1; activity A may then block if appropriate and must not accept any more entities.
- When bounded queue Qb becomes not full at T3, entity e must leave A at T3 and arrive at Qb at T4; activity A becomes unblocked and may receive new entities at T3.
- T1=<T2 and T3=<T4.

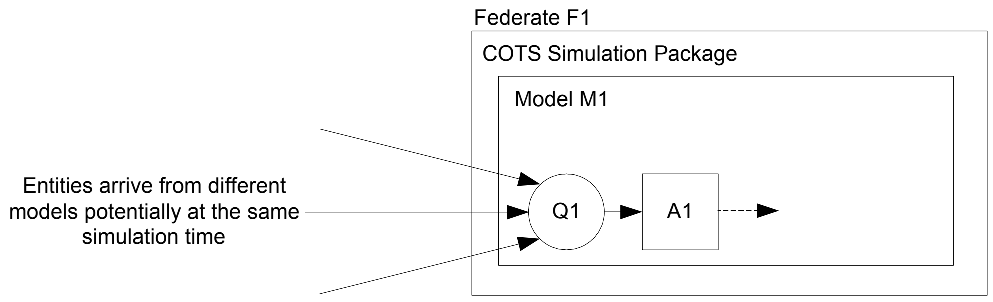
Federate F1
COTS Simulation Package
Model M1
Entities arrive from different models potentially at the same simulation time
Q1
A1

Figure 4: IRM Type A.3 Multiple Input Prioritization

- If activity A is blocked then the simulation of model M1 must continue.

Note:

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

4.4 IRM Type A.3 Multiple Input Prioritization

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 specialised.

- 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).

4.4.1 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 simulation of the real system being simulated.

5 INTEROPERABILITY REFERENCE MODEL TYPE B: SHARED RESOURCE

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 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. There is currently only one IRM Type B Sub-type.

5.1 IRM Type B.1 General Shared Resource

IRM Type B.1 General Shared Resource represents the case, as outlined above, 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.

5.1.1 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 value 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

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. There is currently one IRM Type C sub-type.

6.1 IRM Type C.1 General Shared Events

IRM Type C.1 General Shared Event represents the case 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.

6.1.1 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 its local copies EM1, EM2, etc. will be guaranteed to be executed at the same time T1, and
- 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 CONCLUSIONS

The CSPI PDG is standardizing the way in which COTS simulation packages interoperate via the High Level Architecture. This paper has reviewed the current set of Interoperability Reference Models for CSPs that are being standardized by the CSPI PDG in their *Standard for Commercial-Off-The-Shelf Simulation Package Interoperability Reference Models* (SISO 2007). The wider range of CSPI PDG standards activities can be found in Taylor, et al. (2006). Visit the CSPI PDG's web pages at SISO (www.sisostds.org) for recent updates. The CSPI PDG invites new membership.

REFERENCES

SISO (2007). Draft Standard for Commercial-Off-The-Shelf Simulation Package Interoperability Reference Models. Available via `<http://www.sisostds.org>` [accessed June 11 2007].

Taylor, S.J.E., Strassburger, S. Turner, S.J., Low, M.Y.H., Wang, X. and Ladbrook, J. (2006). Developing Interoperability Standards for Distributed Simulation and COTS Simulation Packages with the CSPI PDG. In *Proceedings of the 2006 Winter Simulation Conference*. Association for Computing Machinery Press, New York, NY. pp. 1101-1110.

AUTHOR BIOGRAPHIES

SIMON J. E. TAYLOR is the co-founding Editor-in-Chief of the UK Operational Research Society's (ORS) *Journal of Simulation* and the Simulation Workshop series. He has served as the Chair of the ORS Simulation Study Group between 1996 to 2006 and was appointed Chair of ACM's Special Interest Group on Simulation (SIGSIM) in 2005. He is also the Founder and Chair of the COTS Simulation Package Interoperability Product Development Group (CSPI-PDG) under the Simulation Interoperability Standards Organization. He is a Senior Lecturer in the Centre for Applied Simulation Modelling in the School of Information Systems, Computing and Mathematics at Brunel His recent work has focused on the development of standards for distributed simulation in industry. His email address is `<simon.taylor@brunel.ac.uk>`.

NAVONIL MUSTAFEE is a research student at the Centre for Applied Simulation Modelling (CASM) in the School of Information Systems, Computing and Mathematics, Brunel University, Uxbridge, United Kingdom. His research interests are in grid computing and parallel and distributed simulation. His email address is `<navonil.mustafee@brunel.ac.uk>` and his Web address is `<http://people.brunel.ac.uk/~cspgnnm>`.

STEFFEN STRASSBURGER is a professor at the Technical University of Ilmenau 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 researcher at the DaimlerChrysler Research Center in Ulm, Germany. He holds a Ph.D. and a Master's degree in Computer Science from the University of Magdeburg, Germany. His research interests include the topics simulation and distributed simulation as well as general interoperability topics within the digital factory context. Mr. Strassburger is also the Vice Chair of SISO's COTS Simulation Package Interoperability Product Development Group. His web page can be found via <www.tu-ilmenau.de/fakww>.

STEPHEN J. TURNER joined Nanyang Technological University (Singapore) in 1999 and is currently an Associate Professor in the School of Computer Engineering and Director of the Parallel and Distributed Computing Centre. Previously, he was a Senior Lecturer in Computer Science at Exeter University (UK). He received his MA in Mathematics and Computer Science from Cambridge University (UK) and his MSc and PhD in Computer Science from Manchester University (UK). His current research interests include: parallel and distributed simulation, distributed virtual environments, grid computing and multiagent systems. His e-mail address is <assjturner@ntu.edu.sg>.

MALCOLM Y. H. LOW is an Assistant Professor in the School of Computer Engineering at the Nanyang Technological University (NTU), Singapore. Prior to this he was with the Singapore Institute of Manufacturing Technology, Singapore (SIMTech). He received his Bachelor and Master of Applied Science in Computer Engineering from NTU in 1997 and 1999 respectively. In 2002, he received his D.Phil. degree in Computer Science from Oxford University. His current research interest is in the application of parallel/distributed simulation, grid computing and agent technology for the modeling, simulation, analysis and optimization of complex systems. His e-mail address is <yhlow@ntu.edu.sg>.

JOHN LADBROOK has worked for Ford Motor Company since 1968 where his current position is Simulation Technical Specialist. In 1998 after 4 years research into modelling breakdowns he gained an M.Phil (Eng.) with the University of Birmingham. In his time at Ford, he has served his apprenticeship, worked in Thames Foundry Quality Control before training to be an Industrial Engineer. Since 1982 he has used and promoted the use of Discrete Event Simulation. In this role he has been responsible for sponsoring many projects with various universities. For the past seven years, he has been Chairman of the Witness Automotive Special Interest Group. His email address is <jladbroo@ford.com>.