The SISO CSPI PDG Standard for COTS Simulation Package Interoperability Reference Models

Dr Simon J E Taylor
School of Information Systems, Computing and Mathematics
Brunel University
Uxbridge, Middx, UB8 3PH, UK
simon.taylor@brunel.ac.uk

Prof Steffen Strassburger
School of Economic Sciences
Ilmenau University of Technology
Helmholtzplatz
98693 Ilmenau, GERMANY
steffen.strassburger@tu-ilmenau.de

Dr Stephen J Turner
Parallel and Distributed Computing Centre
School of Computer Engineering
Nanyang Technological University
Singapore 639798, SINGAPORE
assjturner@ntu.edu.sg

Keywords:
Discrete-event Simulation, Distributed Simulation, COTS Simulation Package, Interoperability, Standards.

ABSTRACT: *Discrete-event simulation is used to analyze production and logistics problems in many areas such as commerce, defense, health, manufacturing and logistics. Commercial-off-the-shelf (COTS) Simulation Packages (CSPs) are black box visual interactive modelling environments that support the development of such simulations. These include CSPs such as Arena™, Anylogic™, Flexsim™, Simul8™, Witness™, etc. 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 model/CSPs are actually interoperating. As the first in a series of standards aimed at standardizing how the HLA is used to support CSP-based 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-based distributed simulations. This paper presents the standard and summarizes the current activities of the PDG.*

1. Introduction

Discrete-event simulation is used to analyze production and logistics problems in many areas such as commerce, defense, health, manufacturing and logistics. 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 modelling environments that allowed simulation modellers 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 modellers today. They include Arena™, Anylogic™, Flexsim™, Simul8™, Witness™, etc. Each has a wide range of functionality including visual model building, simulation run support, animation, optimisation 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 modellers skilled in operations/operational research and management science. Simulation projects in this area are typically used to investigate and analyze real-world problems. Examples include:

- A supply chain distributes equipment to front line troops. A model is built to represent the different supply centres 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.

There are many other examples (see the Proceedings of the Winter Simulation Conference (www.wintersim.org) or the ACM Digital Library (www.acm.org/dl)). What is common to these is that virtually all models are created using a *single CSP*. To interoperate two CSPs together (to execute a single simulation run) is simply not possible without a great deal of very costly "bespoke" effort that requires computing skills out of the scope of typical modellers (i.e. the solution is typically created specifically for a model/CSP will little reuse from previous solutions – most current approaches are "built from scratch"). This is further complicated by the time management required for distributed discrete-event simulation. This means that it is very difficult to distribute large models to share the processing load of the simulation (for example, a real-world automotive simulation can take several hours to run; the blood distribution example took over a day to complete a single run with just four hospitals!). Additionally, if models are difficult to "move" then the lack of a low-cost interoperability solution means that models and their CSPs cannot be simply "linked" together. For example, two or more police authority models cannot be linked together to investigate national security response policies; the business processes of two or more companies cannot be linked together to investigate new financial products; manufacturing models in a supply chain cannot be linked together to investigate the supply of new products to new markets. The lack of standards that specifically address the problem of CSP *interoperability* or CSP-based *distributed simulation* is therefore preventing many new and exciting modelling and simulation problems to be explored.

This is the motivation behind SISO's COTS Simulation Package Interoperability Product Development Group. Dedicated to creating a standardized approach to CSP interoperability, the CSPI PDG has created the first of several standards in this area. We now introduce the CSP Interoperability Problem and our first standard.

2. The CSP Interoperability Problem

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 "language" that attempts to capture these subtleties. This is the thinking behind the CSPI PDG's set of standardized *Interoperability Reference Models*, the "Standard for COTS Simulation Package Interoperability Reference Models", effectively a set of simulation patterns or templates, that will enable modellers, 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

Where is the complexity? Consider the following. As an example, the owners of two factories want to find

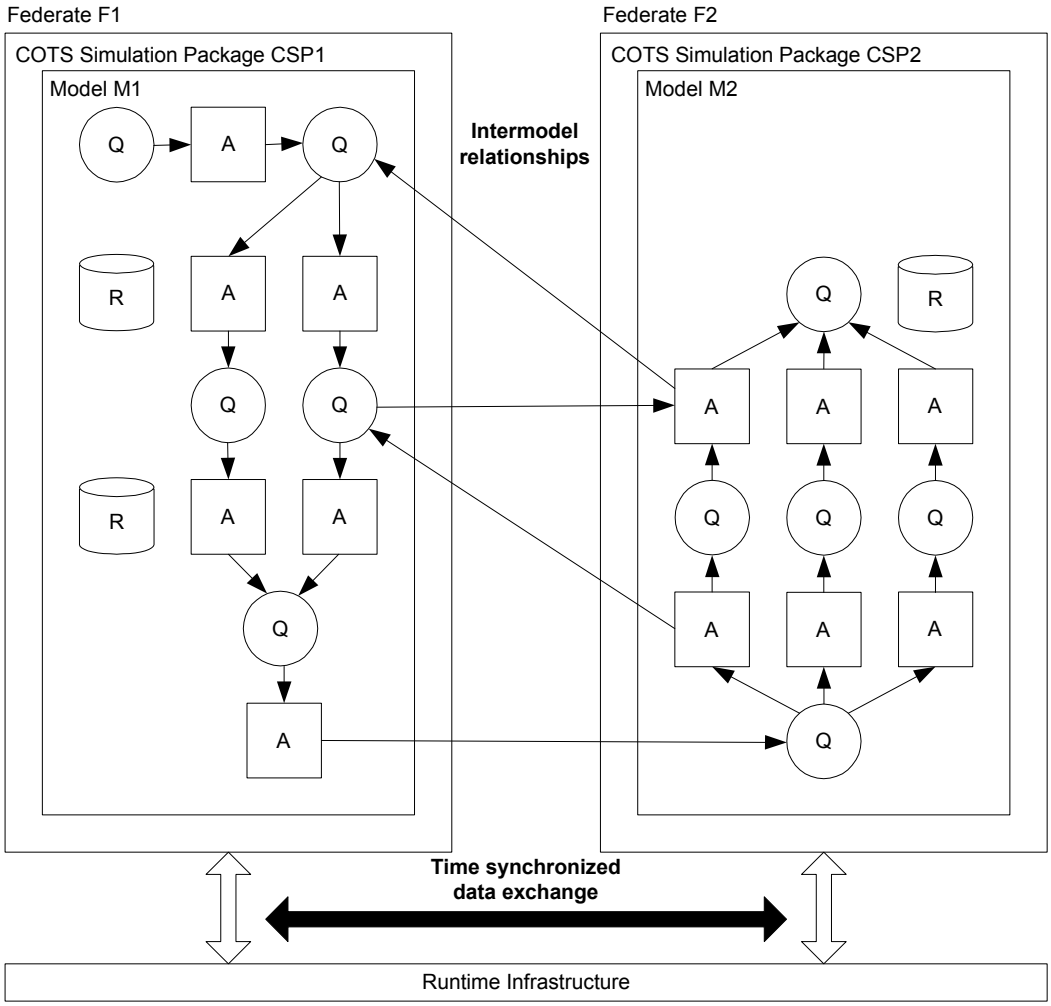


Figure 1: The COTS Simulation Package Interoperability Problem

out how many products their factories can manufacture in a year. Both factories have been modelled separately using two CSPs. As shown in figure 1, the (extremely simplistic) factories, modelled 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 an emergency shutdown) 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. 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 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 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 IRMs to try to simplify this complexity. Let us now describe the current set of IRMs.

3. Interoperability Reference Model Definition

An Interoperability Reference Model (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 modellers 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. Behaviour is also important as the set of rules that govern the behaviour of a network of queues and activities subtly differ between CSPs (for example the rules that govern behaviour 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 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

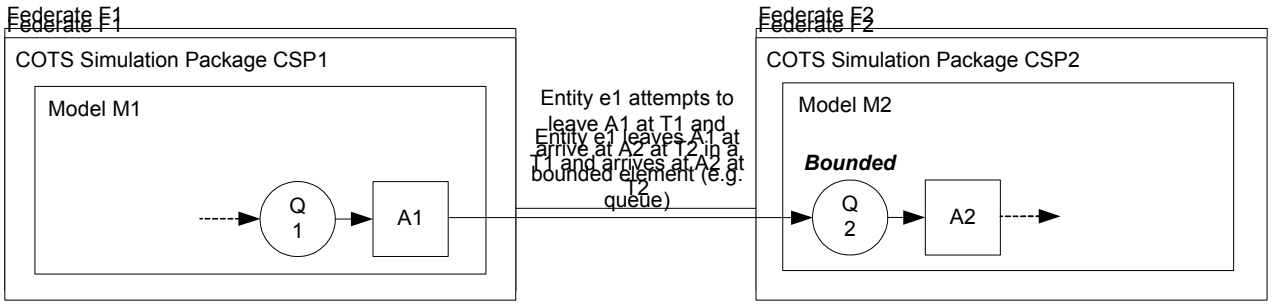


Figure 3: IRM Type A.2: Bounded Receiving Element

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 utilisation, 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

This has been rationalised 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 cumulative, 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).

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

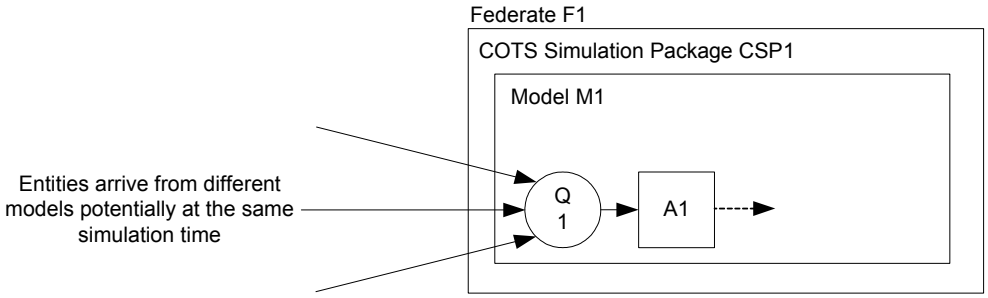


Figure 4: IRM Type A.3 Multiple Input Prioritization

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 modelling 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).

4.5.2 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.

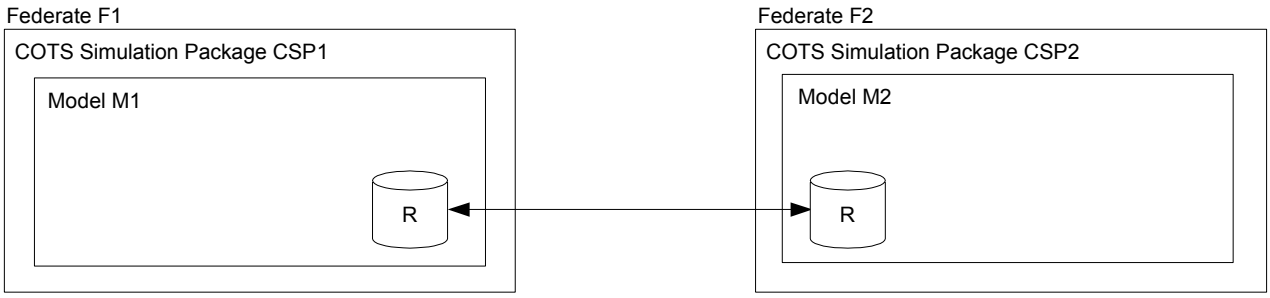


Figure 5: IRM Type B.1: General Shared Resource

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 modeller 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.

5.2 Interoperability Reference Model Type B Sub-types

There is currently one IRM Type B Sub-types

- 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

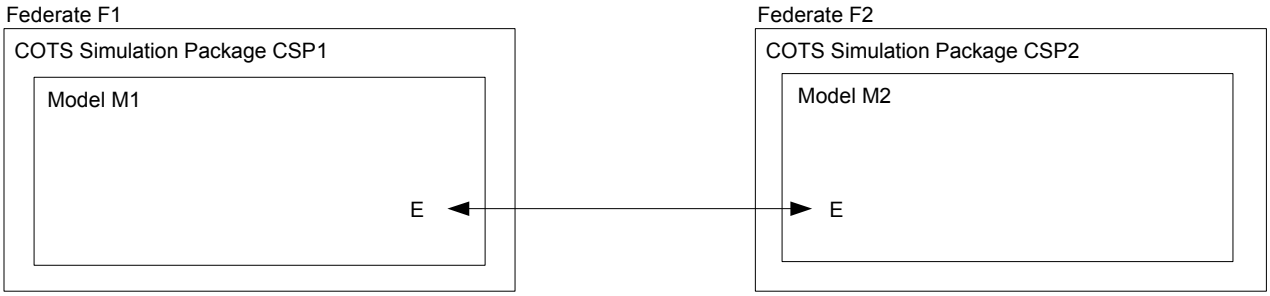


Figure 6: IRM Type C.1: General Shared Event

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
- 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

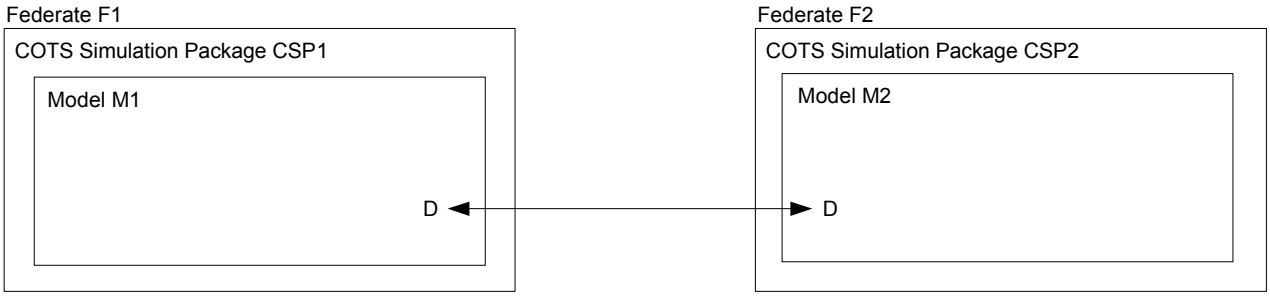
7.3.1 Overview

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



A shared data item D exists at two models M1 and M2. If shared data item D changes at time T1 in model M1 then it must change at T1 in model M2

Figure 7: IRM Type D.1: Shared Data

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. Conclusions

This paper has presented the CSPI PDG Standard for COTS Simulation Package Interoperability Reference Models. At the time of writing, the Standard is currently undergoing balloting as SISO-STD-006-2007 (DRAFT). The next activities of the CSPI PDG is to classify current CSPI approaches and to begin work on a set of Data Exchange Specifications and Interoperability Frameworks to support each IRM. The full standard can be found at the CSPI PDG's site 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.

Acknowledgments

The authors would like to thank the CSPI PDG members for their enthusiasm and comments in the development of this standard.

References

Please see the extensive bibliography in the draft standard available from www.sisostds.org.

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 is the current Chair of ACM's Special Interest Group on Simulation (SIGSIM) (2005+). He is a Senior Lecturer 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 STRASSBURGER 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. 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. His email address is <assjturner@ntu.edu.sg>.