

DEVELOPING INTEROPERABILITY STANDARDS FOR DISTRIBUTED SIMULATION AND COTS SIMULATION PACKAGES WITH THE CSPI PDG

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
Malcolm Y.H. Low
Xiaoguang Wang

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

Steffen Strassburger

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

John Ladbrook

Dunton Engineering Centre
Ford Motor Company
Laindon, Basildon, Essex, UK

ABSTRACT

For many years discrete-event simulation has been used to analyze production and logistics problems in manufacturing and defense. In the early 1980s, visual interactive modelling environments were created that supported the development, experimentation and visualization of simulation models. Today these environments are termed Commercial-off-the-shelf Simulation Packages (CSPs). With the advent of distributed simulation and, later, the High Level Architecture, the possibility existed to link together these CSPs and their models to simulate larger problems within enterprises (e.g. multiple production lines) and across supply chains. However, the problem of standardizing the use of the HLA and its constituent parts in this domain exists. The solution of this problem is the work of the CSP Interoperability Product Development Group (CSPI PDG). The purpose of this paper is to introduce the CSPI PDG and to review the suite of standards proposed by the group and current progress.

1 INTRODUCTION

Annually at the Winter Simulation Conference there are frequent examples of where discrete-event simulation has been used to analyze production and logistics problems in manufacturing and defense. Many of these use software tools developed to support the process of simulation, are based on visual interactive modelling environments devel-

oped in the early 1980s and support model development, experimentation and visualization. Today, these environments are sometimes termed Commercial-off-the-shelf Simulation Packages (CSPs). With the advent of distributed simulation and, later, the IEEE 1516 High Level Architecture (HLA), the possibility existed to link together, or *interoperate*, these CSPs and their models to simulate larger problems within enterprises (e.g. multiple production lines) and across supply chains. However, for two or more CSP/models to interact across a communications network, there must be some agreement as to the form of interaction that takes place between them. In distributed computing, the need to standardize the form of interaction over a network is generally accepted and has given rise to standards that support the Internet and the World Wide Web (respectively, RFCs and Recommendations). It seems natural therefore to develop communication standards to support interoperability between CSPs. Further, as the HLA has been put forward as the *general* interoperability/communication standard for distributed simulation, it also seems natural to develop a standard based on the HLA that will therefore be potentially compatible with other HLA work. Combined work towards this began in 2002 with the formation of the CSP Interoperability Forum (CSPIF). However, in order for standards to be recognized in a community, representation has to be made to a recognized standards authority (e.g. the W3C or IETF). The Simulation Interoperability Standards Organization (SISO) has authority for the HLA and other simulation interopera-

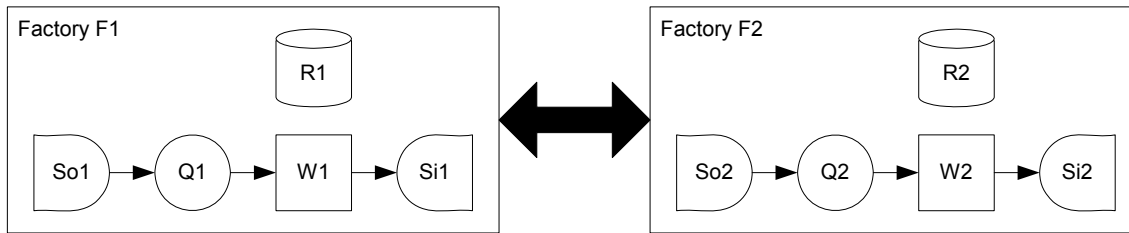


Figure 1: A Simple Distributed Simulation

bility standards. In 2004 the CSPIF formally applied to SISO and was granted responsibility for the development of CSP interoperability standards. The result of this was the formation of the CSP Interoperability Product Development Group (CSPI PDG) (www.csipi-pdg.org). Note that the term “product” is used in the same sense as RFC or recommendation. The purpose of this paper is to introduce the CSPI PDG and to report on the current status of standards proposed by the group. The paper also assumes some familiarity with the HLA.

The paper is structured as follows. Section 2 introduces the problem of CSP interoperability in more detail. Section 3 presents the aim and objectives of the CSPI PDG. Sections 4 to 6 introduce elements of the proposed standards. Section 7 concludes the paper.

2 COTS SIMULATION PACKAGE INTEROPERABILITY

Consider a federation composed of CSPs/model federates that exchange data via a runtime infrastructure (RTI) implemented over a network in a time synchronized manner (figure 1). Two factories, F1 and F2, generically interact as denoted by the black double-headed arrow. Each model consists of an arrival source So_i , a queue Q_i , a workstation W_i , a resource R_i , and an exit sink Si_i (where i is the factory identifier). Different types of information might be exchanged. For example, entities might be passed between models (i.e. the two factories are linked together – entities leave F1 at Si_1 and arrive in F2 at So_2) and the resources R_1 and R_2 might be shared to reflect a shared set of machinists that can operate workstations W_1 and W_2 . If this was the case, factory F1 must publish and send information to the RTI in an agreed format and time synchronized manner and factory 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 at a sink and the arrival of an entity at a source is effectively the

same scheduled event in the two models – most distributed simulations represent this as a timestamped event message sent from one federate to another (with the timestamp typically equal to the time that the entity finished processing in the last workstation (W_1 in our example) or with travel time. The sharing of resources cannot be handled in the same way. For example, when resource (R_1) is released or an entity arrives in queue Q_1 , a CSP executing the simulation of F1 will determine if workstation W_1 can start processing an entity. If resources are shared, each time R_1 or R_2 changes state a timestamped communication protocol is required to inform and update the changes of the shared resource state.

This outlines our interoperability problem. First, what information can be exchanged between CSP/model federates? Secondly, of these, what are their synchronization requirements and in what common format should the data be exchanged? Finally, given that there is no standard on which these CSPs are based, how can we accommodate changes in interoperability requirements? It is the answers to these questions that define the CSPI PDG range of proposed standards. The next section outlines the aim and objectives of the CSPI PDG.

3 THE CSPI PDG

The CSPIF was created in 2002 in an attempt to unify research and developments in the interoperation of CSPs. The main outcome of the CSPIF was the reduction of the interoperability problem into different interoperability requirements. For example, in the last section it was briefly outlined that the interoperability requirements of entity transfer and resource sharing were different. In discussions over two years, this was further expanded into specific interoperability requirements. It was further identified that as new features were continually added to CSPs, it was likely that this list of interoperability requirements will grow further. However, only by a successful combination of end users, CSP vendors and researchers, could these interoperability requirements be solved successfully. The concept of several interoperability reference models (IRMs) was identified. These were meant to be under-

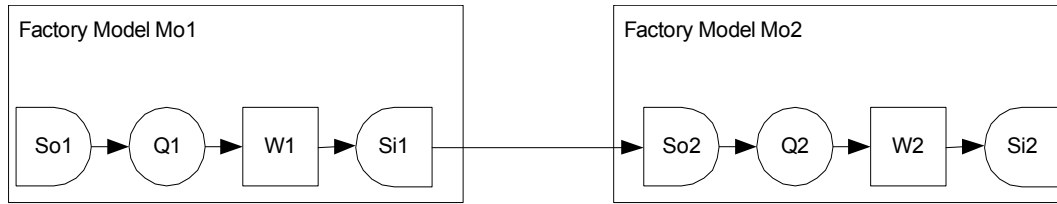


Figure 2: Type I Interoperability Reference Model

standable by CSP user and technology provider alike, and form the basis of the interoperability solution (in terms of data exchange representation and mechanism). Finally the forum identified the need for benchmarks to compare different solutions. After being officially recognized by SISO, the CSPIF became the CSPI PDG. The aim of the CSPI PDG is to create a standardized approach to CSP interoperability using the IEEE 1516 *High Level Architecture*.

But what of the questions posed in section 2? The IRMs are intended to reduce the requirements of information exchange into solvable problems and so answer the first question and produce our first objective. To answer the second question produces the need for three objectives: standard data representations, standard exchange mechanisms, and a set of interoperability frameworks that combine these as needed by each IRM. The final question requires an open, extensible standard and the final objective. The objectives of the CSPI-PDG are therefore to:

- Create standard reference model(s) that can be used to communicate concepts and problems between researchers, users, and vendors in support of the CSPI-PDG aim.
- Develop standard data exchange representations.
- Develop standard data exchange mechanisms.
- Develop standard interoperability frameworks.
- Create an open, extensible standard.

Each of the above objectives of the standards development process will now be discussed as they outline the CSPI-PDG proposed range of standards.

4 INTEROPERABILITY REFERENCE MODELS

Interoperability Reference Models (IRMs) are the first deliverable from the CSPI PDG. It is difficult without hindsight to determine what information needs to be exchanged between CSPs and their models. Over one year and five workshops the CSPIF discussed what the needs of CSP interoperation would be. What the forum came to realize quite quickly was that there were some major obstacles to overcome and major problems to be solved. For example, many of CSPs are difficult to interface to – each has its

own interfacing needs (time/data) and has its own particular data representation. Additionally, some interoperability problems, such as shared resources, are non-trivial. To make progress, the interoperability problems were divided into a series of problem *types* that were represented by IRMs. A problem *type* is meant to capture a general class of interoperability problem, while an *interoperability reference model* is meant to *promote common understanding between CSP vendors, simulation users and technology solution providers*. These two concepts are key as (i) trying to solve everything at once would mean zero progress and (ii) distributed simulation is complex – a common frame of reference promotes communication between key stakeholders. Additionally, the existence of “*types*” means that interoperability solutions to identified problems can be first *standardized* and then *certified*. Let us now review the currently identified IRMs.

4.1 Type I Interoperability Reference Model (Asynchronous Entity Passing)

The *Type I IRM* represents models that interact on the basis of entities, i.e. models are linked together so that one model may pass an entity to another. The reason why this is referred to “asynchronous” is that there is no *immediate or direct feedback* when an entity is passed. Note that some COTS simulation packages do not use entities. “Entity” in this context is used to mean the representation of some physical or logical system element that undergoes some set of instantaneous state changes (events). “Object” or “transaction” are two terms used in a similar context. Additionally, various comments have been made on the semantics of passing an entity from one model to another. For purposes of this discussion, “entity passing” is intended to mean “the passing of timestamped information relating to the representation of an entity transferred from one model to another.”

Let us consider the creation of such a simple distributed simulation consisting of two factory models, Mo1 and Mo2, shown in figure 2. Both are identical with one exception. The sink Si1 in Mo1 and the source So2 in Mo2 are now logically linked together. The combined models represent the combined factory as parts finishing machin-

ing in W1 leave the model Mo1 at sink Si1 and enter the model Mo2 at So2 and are placed in queue Q2 to await machining in W2. Note that the original form of this reference model replaced the sink and the source with a direct link between W1 and Q2. This was modified to a logical link between Si1 and So2 as it encourages transparency, i.e. the minimum amount of technological intervention (model alteration) in the development of a distributed simulation solution.

In terms of minimum technological support of the logical link between the two models, all that is required is the transmission of timestamped entity information between model Mo1 and model Mo2 in such a way that model Mo2 receives the entity information in correct order with its own events. This is the reason why this IRM has been termed “asynchronous” – there is no synchronous message exchange needed to transfer the entity information between the two models (as will be required in the Type II IRM).

Specifically, in terms of a distributed simulation, any solution to this Type I IRM must be able to

- Transfer timestamped entity messages from one model to another via a timestamped message or such.
- Allow a model to correctly receive timestamped entity messages from one or more models.
- Correctly coordinate this information with the receiving model events being processed by the COTS simulation package.

Note that this discussion assumes that any additional model detail added as a result of validating the logical link between two models can be rooted in the Type I IRM as long as it can be reduced to entity information being sent *asynchronously* between the two models in the manner described. A UML version of this IRM is available from the CSPI PDG website.

4.2 Type II Interoperability Reference Model (Synchronous Entity Passing)

The Type I IRM effectively dealt with sending timestamped entity information between models with no *immediate* feedback. The Type II IRM deals with this case. Consider the problem of a *bounded queue*. This represents the case where there is limited physical space in the system being represented. In the case where model has a bounded queue that is full, then if a workstation that precedes the queue finishes work on a work item the workstation must keep that work item as there is nowhere to place it – the workstation is unable to accept any new work. When the queue has space then the workstation can pass on the work item and accept new work.

Figure 3 shows version 2 of the Type II IRM (with sink/source added after version 1). In this example, queue Q2 is a bounded queue. When W1 in Mo1 finishes work on a part entity, the logical link between sink So1 and source Si2 dictates that the part entity must be passed to Q2. However, if Q2 is full then W1 must retain that part entity until Q2 has space (when W2 accepts its next part entity). In terms of a technological solution, the transfer of timestamped entity information between the models must be done *synchronously*, i.e. whenever an entity is transferred between the models some kind of synchronous exchange of information and agreement must take place before the entity is transferred. The implications of this are that

- When a distributed simulation solution seeks to pass timestamped entity information from one model to another model with a bounded queue, and the bounded queue to which the entity will be received is full, the distributed simulation solution must ensure that the sending model blocks the appropriate workstation (the entity is still in the workstation) and vice versa. This is an immediate synchronous interaction derived from an initial asynchronous timestamped entity message.
- When the bounded queue in the receiving model has space, the distributed simulation solution must ensure that the sending model unblocks the appropriate workstation and the receiving model places the entity blocking the workstation in the bounded queue. This is an asynchronous timestamped message representing the unblocking of the workstation (and therefore must be processed as any other timestamped message).

Technologically, many variants of the above are possible (such as when the entity is actually “sent”). However, the same variant must be used by both packages. Figure 4 shows an example of this. Model 1 and Model 2 infer the models, the COTS simulation packages, and the distributed simulation technology. The first message exchange represents the case where the bounded queue has space. The second is representative of the other case where the bounded queue is full and later empties. A UML version of this is available from the CSPI PDG website.

4.3 Type III Interoperability Reference Model (Shared Resources)

Figure 5 shows the case where two models share information on the basis of resources. In this simple scenario, resource R is common between the two factory models and represents a pool of workers. When either workstation wishes to process an entity waiting in their queues, in this scenario they must also have a worker. If a worker is

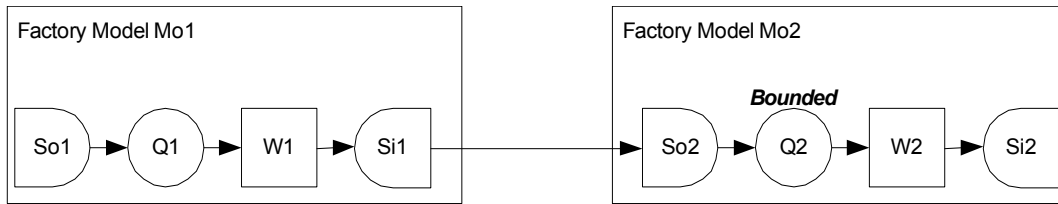


Figure 3: Type II Interoperability Reference Model showing Bounded Queue

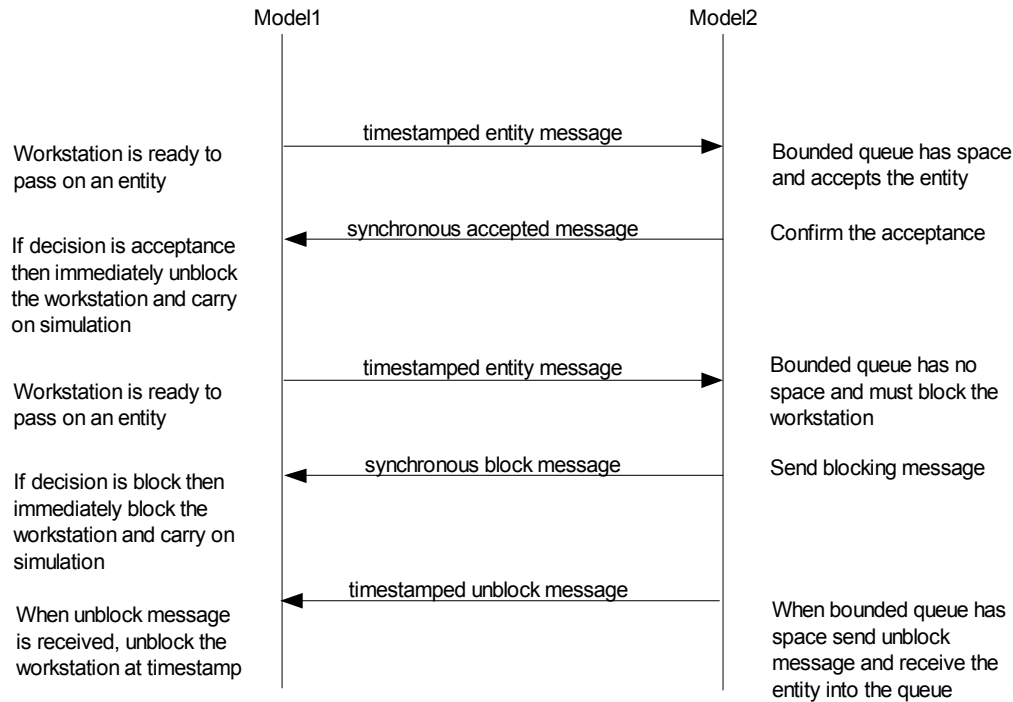


Figure 4: Example Protocol for Type II Interoperability Reference Model

available in R then processing can take place. If not then work must be suspended until one is available.

There have been some suggestions that a simple solution to this problem is to convert the resources to some kind of entity that is passed between the two models. This is not at all satisfactory as some models use resources that are effectively workers which have various skills and shift patterns, i.e. if the modeler wishes to model system elements as resources then they should not be limited by distributed simulation.

A distributed simulation solution should therefore be able to make sure that the contents of R are consistent between the two models.

4.4 Type IV Interoperability Reference Model (Shared Event)

Figure 6 shows the case where two models share an event E. This is intended to be a shared “signal” between the two models. For example when model Mo2 reaches a given threshold value (a quantity of production) it should be able to signal this fact to all models that have an interest in this fact.

4.5 Type V Interoperability Reference Model (Shared Data Structure)

Figure 7 shows a data structure D that is shared between two models. This is intended to represent any data that needs to be shared between the models. The implication of

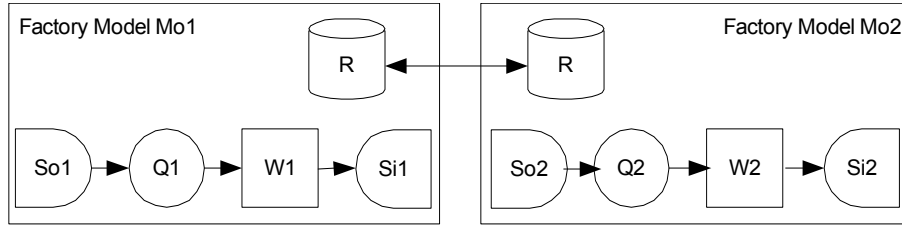


Figure 5: Type III Reference Model (Shared Resources)

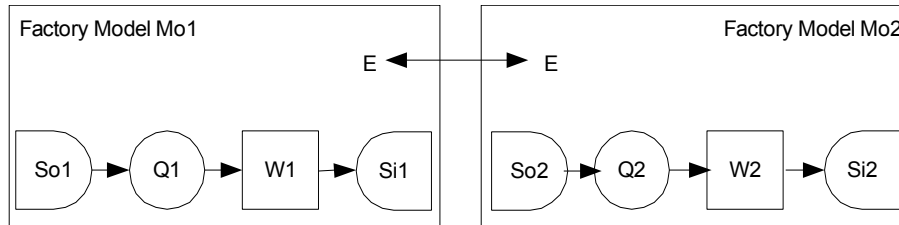


Figure 6: Type IV Reference Model (Shared Event)

any distributed simulation solution to this problem is that any update to one copy of D has to be synchronized with all copies of D.

4.6 Type VI Interoperability Reference Model (Shared Conveyor)

Figure 8 shows two models sharing a conveyor, i.e. a form of transport that requires physical representation in a model. The implication of this IRM is that the conveyor is shared between, for example, the production units in a fac-

tory production line. Various views have been made on this IRM that range from it being a necessity to it being superfluous. As the IRMs are intended to be targets for standard solutions derived from problems identified by the end user community, this modelling problem remains as an IRM as was indeed an end user who identified this as such.

5 DATA EXCHANGE SPECIFICATION

Each IRM represents the exchange of a particular type of data. The data exchange specifications being developed by

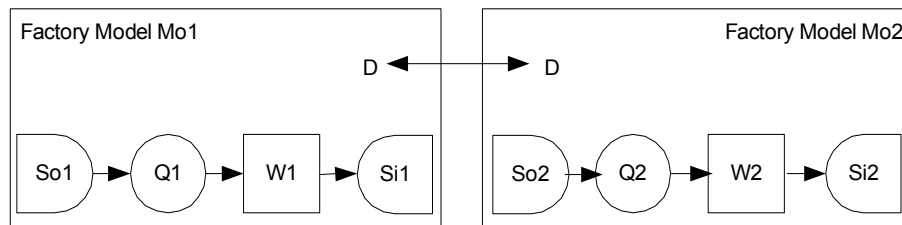


Figure 7: Type V Reference Model (Shared Data Structure)

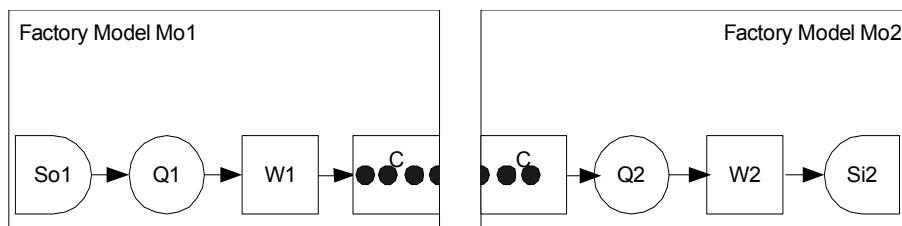


Figure 8: Type VI Reference Model (Shared Conveyor)

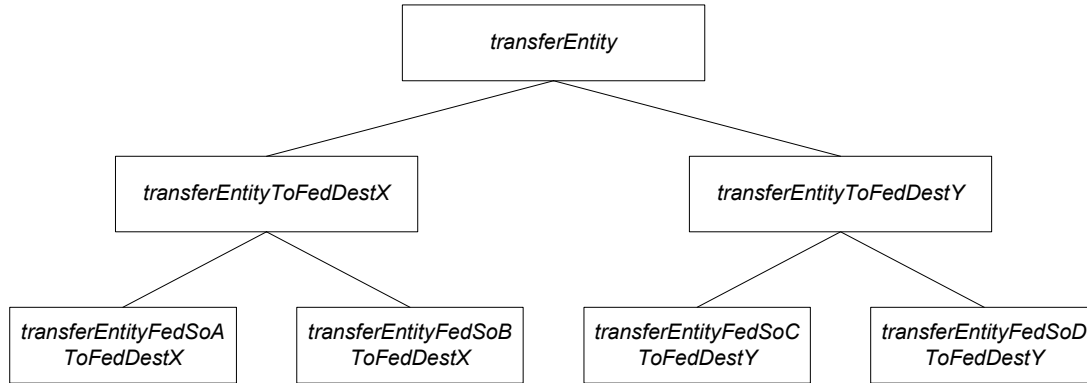


Figure 9: Interaction Class Hierarchy

CSP PDG are intended to specify how this data should be formatted in as general a manner as possible in the HLA's Object Model Template (OMT) notation (in its current IEEE 1516-2000 form). The current focus, the Entity Transfer Specification (ETS), deals with the representation of entities in Type I and Type II IRMs. The reason for this is that both IRMs deal with the transfer of entities between CSPs. The difference between the IRMs is that Type II requires additional synchronization to deal with the bounded buffer problem. The following discussion is based on the current version of this emerging standard Version 1.1.1. We shall define a *source* model as being the model from which a timestamped entity leaves and a *destination* model as the model at which the timestamped entity arrives. These are necessary as there may be different possible routings between models (as defined by the model, not the RTI and Interoperability Framework) and there must be enough information for this to be conveyed for this model routing to be accomplished. Models may also have multiple sources and it is important that there must be some way of indicating at which source point an entity enters a model. In this version of the ETS, we assume there is only one receiving point in the destination model for a specific entity type from a specific source model, i.e. for different entity types there are different single receiving points. Version 2 of this data exchange specification will address multiple entry/exit combinations. We shall define *time* as being the time when an entity leaves a source model and instantaneously arrives at the destination model (i.e. an event has occurred at *time* marking the departure of an entity from one subsystem to instantly arrive at another).

In terms of *entity representation*, as we are concerned with the transfer of a timestamped entity from a model in one federate to a model in another, our focus is a common data exchange format of the entity that has been prepared for transfer. We will assume that there is some translation mechanism between the heterogeneous CSPs to convert to and from our ETS representation via the appropriate Interoperability Framework (IF). We shall also assume that

time has been converted into the same units and resolution in both models. As with most distributed systems, the representation of an item must be marshaled (flat) so that it can be sent as a stream of bytes. We shall therefore represent a mapped entity as a *name* and zero or more *attributes*. The form and type of the attributes are the result of the entity-entity mapping between the heterogeneous CSPs and their models.

An entity is therefore defined as $entity = \{entityName, attributes^*\}$, e.g. $widgetEntity = \{widgetEntity, 24, "Acme"\}$, which represents a widget entity with attributes of (integer) 24 and (string) Acme.

When a CSP determines that an entity has left its model, the CSP must be able to deliver the following information to the IF: $output(entity, time, source, destination)$.

Similarly, when the IF is ready to pass an entity to the CSP, indicating that an entity has arrived, the IF must be able to deliver the following information: $input(entity, time, source)$ where:

- *entity* is the name of the entity *entityName* and zero or many attributes,
- *time* is the time at which the entity left the model,
- *source* is the name of the sending model, and
- *destination* is the name of the destination model.

On output the *source* and *destination* are used by the Interoperability Framework to select the appropriate transfer mechanism. On input, the CSP uses *source* to determine the appropriate entry point in a model (i.e. where the entity has been transferred from). *time* is used to perform CSP time synchronization.

In this specification HLA *interactions* are used to represent the passing of an entity from one model to another at the RTI level. Figure 9 shows the ETS interaction class hierarchy. Features of this are:

- *transferEntity* – the superclass. This allows a federate to conveniently subscribe to all instances of entity transfer (for purposes of monitoring, visualization, etc.)
- *transferEntityToFedDest* – a single subclass per receiving federate where *FedDest* is the name or abbreviation of the receiving federate’s model. It exists for the convenience of the *FedDest* federate to subscribe to all instances of *transferEntity* bound to the destination federate without explicit naming.
- *transferEntityFedSoToFedDest* – subclasses for each entity transfer relation where *FedSo* is the name or abbreviation of the sending federate’s model. It allows the source federate to send a timestamped interaction that represents the transfer of an entity from source to destination at a given time.

Note that in the above for an actual implementation *FedSo* and *FedDest* are replaced by the source and destination federate names and *Entity* is replaced by the name of the entity as appropriate. In a federate’s SOM (Simulation Object Model) or the federation FOM (Federation Object Model) the following tables are used in our exchange format.

- Interaction Class Table: This contains the interaction classes used to transfer the entities. These will be the interaction superclass *transferEntity*, its interaction subclasses *transferEntityToFedDest*, and their interaction subclasses *transferEntityFedSoToFedDest*.
- Parameter Table: Each *transferEntityFedSoToFedDest* interaction will have a named parameter *Entity* with a named datatype *EntityType*. In the table, unless otherwise stated, *Available Dimensions* shall be NA (as data distribution management is not used, *Transportation* shall be HLAReliable (TCP semantics) and Order shall be timestamp, i.e. messages must arrive in timestamp order.
- Datatype Table: A *fixed record datatype* table shall exist to represent the named *EntityType* and will consist of *entityName*, *source*, *destination*, and *attributes*. The datatypes of *entityName*, *source* and *destination* will be of type *HLAASCIIstring*. The type of the attributes will be defined using the HLA datatype types as appropriate to best represent the type of the attribute.

The above is enough to define the representation of an entity transferred from one model to another via the RTI. The translation of the datatype of this representation and

the internal type representation of the CSP must be performed by the IF according to the requirements of the CSP.

The interaction classes are meant to be used in the following way in an IF. During initialization, a federate will:

- Indicate that it is capable of sending entities to various destination federates by publishing all *transferEntityFedSoToFedDest* interactions, and
- Indicate that it is capable of receiving entities from any other federate by subscribing to all *transferEntityToFedDest* interactions.

During runtime, when the CSP sends the message equivalent to *output(entity, time, source, destination)*. The IF will use *destination* to select the appropriate interaction class to use. It will then parameterize an interaction instance with the details supplied in the output message details. When the RTI passes an interaction instance to the IF, the IF will use the instance’s details to pass the entity to the CSP in some input message with *source* to indicate which model the entity has arrived from.

6 INTEROPERABILITY FRAMEWORKS

Figure 10 shows the current CSPI PDG reference interoperability framework (IF) for the Type I IRM. The IF consists of a set of functions (see Table 1) that allows a CSP to create/join a distributed simulation, and also to send entities to other CSPs as well as receive them. The functions are arranged into 4 groups: initialization, simulation execution, termination and supporting services.

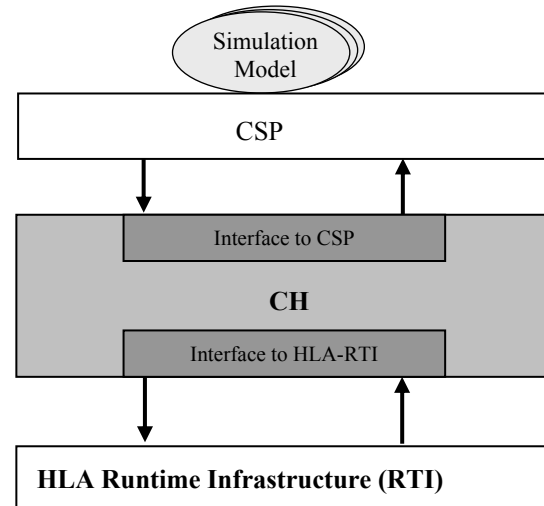


Figure 10: Reference Interoperability Framework

As part of the IF, the CSP Handler (CH) provides an interface consisting of a set of functions to be invoked by the CSP when needed. Through the interface, the CH invokes the necessary calls to the RTI ambassador on behalf

of the CSP and transfers the information received from the federate ambassador to the CSP.

Table 1: Functions in the Interoperability Framework

Group	Name
Initialization	setSynMethod()
	setLookahead()
	registerController()
	registerModel()
	registerInEntity()
	registerOutEntity()
Simulation Execution	advanceTime()
	setAttributeValue()
	transferEntity()
	receiveEntity()
	getAttributeValue()
	getExEntryStatus()
	setExEntryStatus()
Termination	terminateDistributedSimulation()
Supporting Services	getInEntityID()
	getOutEntityID()
	getInEntityName()
	getOutEntityName()

Fig. 11 shows the reference interoperability framework protocol under investigation. This consists of a CSP sending n entities through the CH and RTI to other CSPs using the IF functions such as *transferEntity* and *receiveEntity*. These two functions correspond to the ETS *output* and *input* to pass entities between CSPs.

There are various different approaches to time management using a HLA RTI to support distributed discrete-event simulation. The approach described here is based around *NextEventRequest* (others are currently under investigation as part of the work developing the Type I IF). When the CSP wishes to advance to the time of its next event, it issues an *advanceTime* request to the CH. The CH invokes the corresponding RTI service *nextEventRequest*. The response from the RTI is zero or many ETS interactions received via *receiveInteraction* and a new simulation time granted via *timeAdvanceGrant*. The interactions represent the arrival of entities at the time granted by *timeAdvanceGrant* and may be less than the time initially requested by the CSP (i.e. entities arrive before the time of the original next event – the new time of next event is that of the arriving entities). If no interactions appear, the time granted is exactly the requested time. Either way, this grant time is returned to the CSP with the entities received (if any) via *input(entity, time, source)*, the CSP advances its local simulation time and continues execution. If, as a consequence of this, any entities leave the simulation model, the CSP will send to CH as many *output(entity, time, source, destination)* as appropriate. CH will translate these into ETS interactions and then forward these to the RTI by invoking *sendInteraction*. This continues until some terminating condition is met.

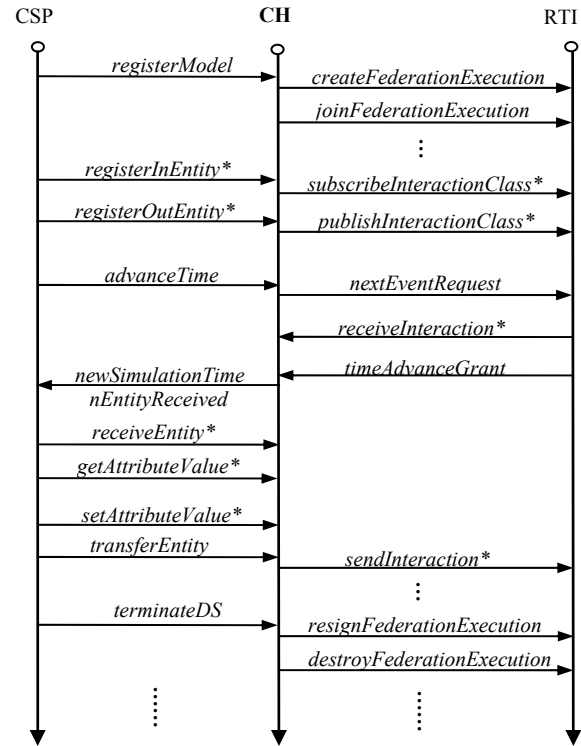


Figure 11: Reference Interoperability Framework Protocol

7 CONCLUSIONS

This paper has presented an overview of the work being done by the CSPI PDG to standardize the way in which COTS simulation packages interoperate via the High Level Architecture. There has been substantial progress and reference implementations exist with Autosched AP, Simul8 and Witness. At the time of writing we are waiting for the new version of the HLA to be published in order to update the PDG's work in compliance with the new version of the standard. Taylor, et al. (2006) further develops issues discussed in this paper and Wang, et al. (2006) discusses issues concerning the development of a solution to the Type II IRM. Space prevents listing current papers associated with this work. Instead visit the CSPI PDG's web pages at www.csapi-pdg.org and at SISO (www.sisostds.org). The CSPI PDG invites new membership.

REFERENCES

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

Wang, X., S. J. Turner, M. Y. H. Low, and S. J. E. Taylor, 2006. COTS Simulation Package (CSP) Interoperability – A Solution to Synchronous Entity Passing. In *Proceedings of the Twentieth ACM/IEEE.SCS Workshop on Principles of Advanced and Distributed Simulation*. 201-210. IEEE Computer Society.

AUTHOR BIOGRAPHIES

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 e-mail address is <jladbroom@ford.com>.

MALCOLM YOKE HEAN 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>.

STEFFEN STRASSBURGER is head of the department “Virtual Development” at the Fraunhofer Institute for Factory Operation and Automation in Magdeburg, Germany. He was previously working as researcher at the Daimler-Chrysler Research Center in Ulm, Germany, where he was responsible for research topics in the Digital Factory and Digital Engineering context, esp. in the area of simulation integration and distributed simulation. He holds a Ph.D. and a Master’s degree in Computer Science from the Otto-von-Guericke University in Magdeburg, Germany. His international experience includes a one-year stay at the University of Wisconsin, Stevens Point and a stay at the Georgia Institute of Technology, Atlanta. He actively participates in several international conferences. The main research interests of his department include virtual reality solutions for product and process development, the combination of virtual reality and discrete event simulation, distributed and web-based simulation, and middleware tech-

nologies like the High Level Architecture, CORBA, and Web Services. His e-mail address is <strassburger@iff.fraunhofer.de>.

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 University and a visiting Associate Professor at Nanyang Technological University. His recent work has focused on the development of standards for distributed simulation in industry. His e-mail address is <simon.taylor@brunel.ac.uk>.

STEPHEN JOHN 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>.

XIAO GUANG WANG is currently a Ph.D. student at School of Computer Engineering (SCE), Nanyang Technological University, Singapore. She received her B.Sc in Computer Science from Nanjing University of Aeronautics and Astronautics, China in 1997. Her research interests lie in Distributed Simulation and the High Level Architecture. Her e-mail address is <xgwang@pmail.ntu.edu.sg>.