

Proceedings of the 2006 OR Society Simulation Workshop
S. Robinson, S. Taylor, S. Brailsford and J. Garnett, eds.

INVESTIGATING DISTRIBUTED SIMULATION WITH COTS SIMULATION PACKAGES: EXPERIENCES WITH SIMUL8 AND THE HLA

Navonil Mustafee

Dr. Simon J E Taylor

Centre for Applied Simulation Modelling
School of Information Systems, Computing and Mathematics
Brunel University
Uxbridge, Middlesex, UB8 3PH, UK.
navonil.mustafee@brunel.ac.uk, simon.taylor.brunel.ac.uk

ABSTRACT:

Commercial-off-the-shelf simulation packages (CSPs) are used widely in industry. Several research groups are currently working towards the creation of distributed simulation with these CSPs. The motivations to do this are various and are largely unproven as there are very few good examples of this kind of distributed simulation in practice. Our goal is therefore to create a distributed simulation environment using CSPs that will allow end users to make their own decisions as to whether this technology will be useful. This paper presents continuing research in creating such an environment using the CSP Simul8 and the High Level Architecture, the IEEE 1516 standard for distributed simulation. The scope of this paper is limited to the CSPI-PDG Type I Interoperability Reference Model.

Keywords: Distributed Simulation, Interoperability Reference Models, HLA, Simul8

1. INTRODUCTION

In this paper, and in the standardisation effort currently in progress [1], we use the term *Commercial-off-the-shelf (COTS) discrete-event simulation packages* (CSPs) to describe commercially available software tools that have been developed to facilitate the practice of discrete-event simulation. Examples of CSPs include: Arena, AutoMod, Flexsim, ProModel, Simul8 and Witness. *Distributed simulation* can be defined as the distribution of the execution of a single run of a simulation program across multiple processors [2]. The current standard to support this is the IEEE 1516 High Level Architecture (HLA).

There are various possible motivations to use distributed simulation with CSPs, or *CSP-based distributed simulation* [3].

- the creation of large models that a single CSP cannot support;
- the speed up of large models;

- integration of discrete-event simulations across virtual organizations, extended enterprises and supply chains;
- the reduction of the cost of model development by enabling the reuse of distributed model components;
- a building block for groupware for simulation; and
- the protection of intellectual property (information hiding in distributed models).

Although there are excellent examples of successful distributed simulations with CSPs [4, 5, 6], a general solution to this problem of heterogeneous integration is illusive. This paper describes progress towards the standardisation of CSP-based distributed simulation by presenting some experiences of linking the CSP Simul8 with the HLA. A case study of how this was used to create a distributed simulation of the UK *National Blood Transfusion Service* is described in a separate paper in these proceedings [7].

The paper is structured as follows. Distributed simulation and the IEEE 1516 High Level Architecture are briefly introduced in section 2. Problems involved in combining distributed simulation and CSP-based simulation to create *CSP-based distributed simulation*, are discussed in section 3. The emerging standards-based solution to these problems and the COTS Simulation Package Interoperability Product Development Group (CSPI-PDG) *Type I Interoperability Reference Model* (IRM) are presented in sections 4 and 5. Section 6 presents our experiences with the implementation of the Type I IRM with the HLA and Simul8. Section 7 concludes the paper with a short discussion of the implications of this approach.

2. THE HIGH LEVEL ARCHITECTURE

The IEEE 1516 standard *The High Level Architecture* (HLA) [8] is a general standard for distributed simulation. This came from the need of the US Department of Defense (DoD) to reduce the cost of training military personnel by

reusing computer simulations linked via a network. In the HLA, a distributed simulation is called a *federation*, and each individual simulator (in our case the combination of a CSP and its model) is referred to as a *federate*. A HLA *Runtime Infrastructure* (RTI) provides facilities to enable federates to interact with one another, as well as to control and manage the simulation. The HLA is composed of four parts: a set of rules, the Object Model Template (OMT), the Federate Interface Specification (FIS), and the Federate Development Process (FEDEP). The rules are a set of ten basic conventions that define the responsibilities of federates and their relationship with the RTI. The FIS is an application interface standard comprising of six groups of services for distributed simulation middleware which defines how federates interact within the federation, and is implemented by an RTI. The OMT provides a common presentation format for HLA federates and consists of a minimum of 14 tables. Each federate defines, in its Simulation Object Model (SOM), the data that it is willing to share (publish) with other federates and the data it requires from other federates (subscribe). The Federation Object Model (FOM) combines the federate SOMs into a single object model for the federation and therefore defines the overall data to be exchanged (published and subscribed) between federates during a simulation execution. The FEDEP defines the recommended practice processes and procedures that should be followed by users of the High Level Architecture (HLA) to develop and execute their federations.

The main observation of all this when considering application development is the flexibility of the HLA. There is no single fixed development path. This is further complicated by the several different RTIs that exist. Each behaves slightly differently and have themselves features that can be beneficially exploited. In the next section we consider what has been done to use the HLA to support distributed simulation of CSPs.

3. CSP-BASED DISTRIBUTED SIMULATION

Although initial work on the use of the HLA to integrate heterogeneous distributed CSPs can be traced back to pioneering work done by Straßburger in the late 1990s [9], this area is still emerging [10].

Research has mainly focussed on technological challenges using combinations of various CSPs and HLA-based and non-HLA based approaches. The use of the HLA and the associated adapter technologies of the MISSION project to support the distributed simulation of manufacturing systems are discussed in [11, 12, 13, 14, 15, 16]. [17, 18] also discuss strategies for HLA use in the same domains. All of these approaches are largely incompatible due to the format of the data exchanged between federates and the protocol used to perform the exchange, and the type of simulation information exchanged.

Why is this so complex? Consider figure 3.1. A distributed simulation (federation) is composed of CSPs and their models (federates) that exchange data (interactions and/or attributes) via an RTI in a time synchronized manner. In this example, two models (federates) of factories, F1 and F2, interact in various ways as denoted by the black double-headed arrow. Each model consists of typical model elements: 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). There are various types of model information that we might share. 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 operators 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

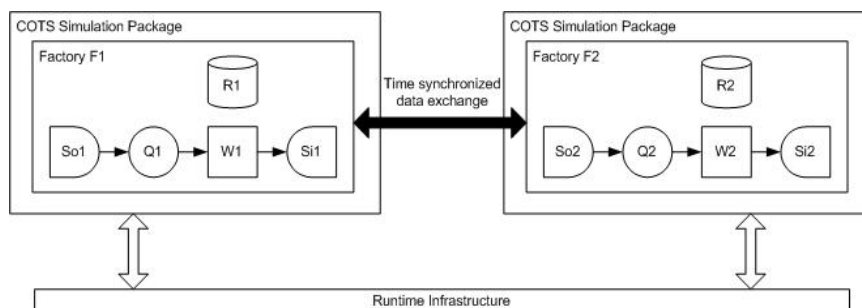


Figure 3.1: The CSP-based Distributed Simulation Problem

F2 must subscribe to and receive that information in the same agreed certain 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 (W1 in our example). The sharing of resources cannot be handled in the same way. For example, when resource (R1) is released or an entity arrives in queue Q1, a CSP executing the simulation of F1 will determine if workstation W1 can start processing an entity. If resources are shared, each time R1 or R2 changes state a timestamped communication protocol is required to inform and update the changes of the shared resource state.

Our problems are therefore these. What are the synchronization demands of data exchanged between federates, how should these be implemented through the RTI, what format should the data take and what relationship should this have to the CSPs and their models? The citations in the previous work section go some way to solving these problems. However, as already noted, these are incompatible. In an attempt to solve this, we now present an emerging standards-based approach.

4. EMERGING STANDARDS AND THE CSPI-PDG

Following on from the observations made in section 3, three comments can be made: not all distributed simulations need all integration approaches; some integration approaches are relatively straightforward and some are extremely complex; and not all integration requirements are known. For example, some distributed simulations only require that entities are passed between models. The problem of entity passing is somewhat simpler than the problem of, for example, synchronous shared state in the case of resource sharing.

As presented in section 2, the HLA well supports the needs of general distributed simulation. However, the specific needs of CSP-based distributed simulation require that the HLA is augmented by additional complementary standards that define how this domain uses the

HLA. These standards are the suite of CSP Interoperability (CSPI) standards being developed under the Simulation Interoperability Standards Organization (SISO) by the CSPI Product Development Group (CSPI-PDG). The suite consists of several *Interoperability Reference Models* (IRMs) that outline different integration needs of CSPI, *Interoperability Frameworks* (IFs) that define the HLA-based solution to each IRM and appropriate *data exchange representations* to specify the data exchanged in an IF [19, 20]. The different interoperability requirements have been encapsulated into (currently) six *Interoperability Reference Models* (IRMs). These are:

- 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

Briefly, the Type I IRM *Asynchronous Entity Passing* deals with the common requirement of transferring entities between simulation models. The Type II IRM *Synchronous Entity Passing* deals with the case where a receiving queue is *bounded*, i.e. in the above example queue Q2 has limited capacity. In this case, the requirement means that the federate containing the sending workstation W1 must, when the processing of an entity is complete, check to determine that there is space in Q2. If there is space available then the entity may be transferred. If there is none the federate must ensure that W1 is *blocked* until space becomes available. The Type III IRM *Shared Resources* deals with the 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. The Type IV IRM *Shared Events* 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.) The Type V IRM *Shared Data Structures* 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

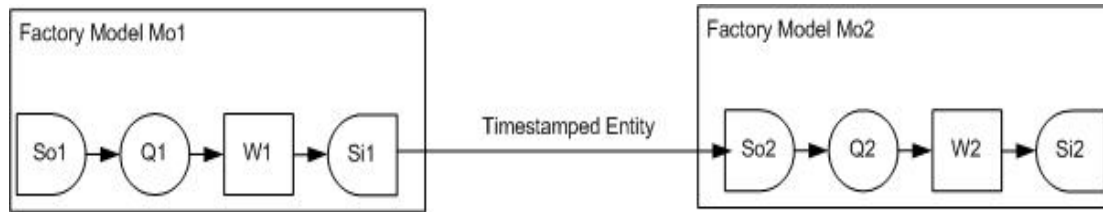


Figure 5.1: The Type I Interoperability Reference Model

shared inventory). Finally, the Type VI IRM *Shared Conveyors* deals with the problem of sharing transportation systems such as conveyor or barges across simulation models (as distinct to the representation of these in Type I IRMs).

We now present the Type I IRM and our approach to its implementation with Simul8.

5. THE TYPE I INTEROPERABILITY REFERENCE MODEL

Figure 5.1 shows the Type I Interoperability Reference Model (Asynchronous Entity Passing). This IRM represents models that interact on the basis of entities; models are linked together so that one model may “pass” an entity to another at a given timestamp. The reason why this is termed “asynchronous” is that there is no *immediate or direct feedback* when an entity is passed. The model elements that have been placed in each model are there to indicate in a simple manner the relationships between models, i.e. the internal structure of a model can be far more complex. Also, it is possible that models could have more than one set of links and that there could be more than two models connected in arbitrary topologies. Again, this IRM is intended to show the simplest relationship between models, one that can be extrapolated to many different scenarios.

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 timestamped entity information in correct order with its own events. An IF solution to this Type I IRM must therefore be able to

- transfer timestamped entity information 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, and

- correctly coordinate this information with the receiving model events being processed by the COTS simulation package.

We now present our experience with the CSP Simul8 and the HLA.

6. CASE STUDY: SIMUL8 AND THE HLA

Simul8 is a CSP that enables users to rapidly construct accurate, flexible and robust simulations using an easy-to-use visual modelling interface [21]. It includes an internal programming language called “Visual Logic” and provides a Windows COM interface that can be used from within any COM-compliant language to “drive” Simul8 [22].

For our Simul8-HLA case study we have developed a CSP Controller Middleware that interacts with both Simul8 Professional Edition and the DMSO RTI 1.3-NG to realize a Simul8-based distributed simulation. The CSP Controller Middleware utilizes the COM interface to access the Simul8 simulation engine and is described in section 6.1. Interaction between the middleware and the RTI is through the services defined in the HLA interface specification as presented in section 6.2. The CSP Controller Middleware comprises of Simul8 Adapter and RTI adapter and the communication between them takes place through well-defined Application Programming Interfaces (APIs) as defined in section 6.3. The differences between this and a previous approach that developed the COTS Simulation Package Handler APIs, introduced in [20], is explained in section 6.4. Our approach is then illustrated in section 6.5 by using three Simul8 federates and one manager federate to form a Simul8 federation.

6.1 Simul8 COM interface

Component Object Model (COM) is a Microsoft technology that allows different software components to communicate with each other by means of interfaces [23]. Simul8 has different levels of COM support for its Standard and Professional Editions. The COM interface in its

Standard Edition allows an application to perform basic operations on Simul8, like opening and closing the model, running the simulation, collecting results, etc. The Professional Edition COM interface, on the other hand, allows more complex operations like creating simulation objects and executing Visual Logic. This more extensive COM interface is required for our research.

6.2 HLA Services

The HLA interface specification organises the communication between federates and the RTI into six different service groups [24]. For our Type I IRM solution with Simul8 and the RTI we require HLA-defined services defined under the groups:

- Federation Management: RTI Calls for creation and deletion of federation; joining and resigning of federates from the federation; and creation and realization of synchronization points.
- Declaration Management: Calls pertaining to publication and subscription of interactions.
- Object Management: Calls that relate to sending and receiving interactions.
- Time Management: RTI calls required to enable time constraint and time regulation and also to advance the federate simulation clock.

The specific RTI calls used in our Simul8-HLA federation can be found in section 6.3, under the discussion on RTI adapter.

6.3 CSP Controller Middleware: Adapters, APIs and protocol

From the preceding discussion it is clear that CSP controller middleware performs two specific tasks; communicates with Simul8 through its COM interface and interacts with RTI using the HLA interface specification. Each of these two tasks is performed by two distinct components of the CSP controller middleware: the *Simul8 adapter* and the *RTI adapter*. The communication between these adapters is via Java Native Interface and Jacob technologies [25, 26].

We hope that our adapter based approach will enable us to reuse existing adapters in some cases. For example, if we needed to integrate Simul8 with a conservative simulation middleware like CMB [19, 20] we would then need to develop a *CMB adapter* but might be able to reuse the existing Simul8 adapter. Similarly, if we wanted to experiment with Witness CSP and

RTI then a *Witness adapter* will be needed but the existing *RTI adapter* might be reused. However, our experience suggests that the development of general purpose adapters may be more difficult than it logically seems.

The CSP controller middleware defines a set of eight APIs that are implemented by either *Simul8 adapter* or *RTI adapter*. The APIs defined by *Simul8 adapter* and invoked by *RTI adapter* are as follows:

OpenSim(modelName, federateName) : Starts Simul8 application in federate *federateName* and loads model *modelName*.

GetNextEventTime(arg) : This method returns the time of the next event scheduled in Simul8 future event list. The argument *arg* is not currently used.

RunSim(time) : Instructs Simul8 to advance simulation until *time*. Also probes Simul8 future event list for external messages to be sent to other federates.

RunSimNoInteraction(time) : Instructs Simul8 to advance simulation until *time*. Does not probe Simul8 future event list for external messages to be sent to other federates. For our producer-consumer topology with no feedback (section 6.5), Simul8 federates A and B use *RunSim(time)* to advance simulation. Federate C uses *RunSimNoInteraction(time)* as there is no loop-back in the model.

Input(time, entity): Introduces entity into Simul8 at current simulation time + *time* (as required by the CSP).

CloseSim() : The invocation of this method closes the model and exits the Simul8 application.

The APIs defined by *RTI adapter* and invoked by *Simul8 adapter* are treated as callbacks (denoted by [†]). The two call back routines are:

Output(time, entity)[†] : Notifies the RTI adapter of an external message to be sent to another federate. Argument *time* represents the time when the Simul8 work station completes processing an *entity*. This method works for our simple producer-consumer topology with no feedbacks. However, if our model were to have a feedback then we would need an additional argument which would specify the federate to which the entity should be passed.

TellSimulationEnd(time)[†] : Informs the RTI adapter that Simul8 has completed simulation till *time*.

The architecture of CSP Controller Middleware is shown in Figure 6.1. The functions performed by the adapters are elaborated below. Please note that CSP Controller Middleware APIs and HLA service calls are enclosed in square brackets [] and curly braces {} respectively.

The Simul8 adapter of CSP Controller Middleware is responsible for the following:

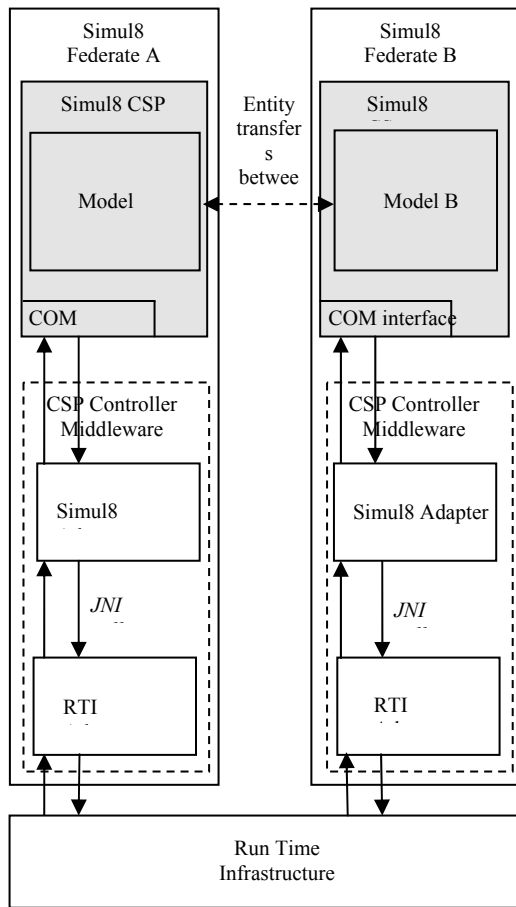


Figure 6.1: CSP Controller Middleware Architecture

1. Starting Simul8 and loading the simulation model [*OpenSim(modelName, federateName)*].
2. Finding the time of the next event by iterating through the event list of the CSP [*GetNextEventTime()*].
3. Advancing simulation after receiving time advance grant from RTI via *RTI adapter* [*RunSim(time), RunSimNoInteraction(time)*].
4. Introducing entities into the Simul8 model [*Input(time, entity)*]. These are external events sent from other Simul8 federates.

5. Informing RTI adapter of external events generated by invoking call back routine [*Output(time, entity)*[†]]. These events are to be forwarded to other Simul8 federates.
6. Informing RTI adapter of the current simulation time by invoking call back routine [*TellSimulationEnd(time)*[†]].
7. Unloading the simulation model and exiting Simul8 [*CloseSim()*].

The RTI adapter of CSP Controller Middleware is responsible for the following:

1. Creating and joining the federation {*createFederationExecution(federationName, fedfile), joinFederationExecution(federateName, federationName, fedamb)*}.
2. Enabling time constraint and time regulation {*enableTimeConstrained(), enableTimeRegulation(currentTime, lookahead)*}.
3. Registering publication and subscription interests {*publishInteractionClass(interactionClassHandle), subscribeInteractionClass(interactionClassHandle)*}.
4. Giving intimation to RTI that Synchronization points have been achieved. The synchronization points are set by the *Manager Federate* to facilitate all Simul8 federates to start simulation at the same time {*synchronizationPointAchieved(labelName)*}.
5. Requesting time grant from the RTI {*nextEventRequest(requestedTime)*}.
6. Informing Simul8 adapter to advance Simul8 time on receiving call back from RTI {*timeAdvanceGrant(newtime)*[†]}.
7. Informing Simul8 adapter to introduce entities on receiving call back from RTI {*receiveInteraction(interactionClass, receivedInteractionSet, time, tag, EventRetractionHandle)*}.
8. Processing call backs received from Simul8 adapter [*SendInteraction(time, entity)*[†], *TellSimulationEnd(time)*[†]].
9. Resigning and destroying the federation {*resignFederationExecution(resignAction), destroyFederationExecution(federationName)*}.

Figure 6.2 shows the protocol between Simul8 adapter, RTI adapter and RTI. The specific COM calls between Simul8 adapter and Simul8 CSP are not shown because these calls are specific to Simul8. The interfaces defined in CSP controller middleware, on the other hand, can be considered as general purpose because different adapters can have different implementations of the same

interfaces. However, as we pointed out earlier, this may be quite difficult to achieve.

The protocol diagram only shows the RTI calls that relate to advancement of federate simulation time and entity passing through interactions. The Simul8 federates use the *nextEventRequest(time)* service call to request advancement of its clock to *time*. The argument *time* is the logical time of the federate's next event in its future event list. The RTI responds to a NER event in one of the following two ways [27].

- The RTI grants time requested by federate in its NER call through *timeAdvanceGrant(time)[†]* callback, or
- The RTI calls back with an external event (*receiveInteraction(params)[†]* in our case) with time stamp which is less than the time requested by the federate in the NER call, and then with *timeAdvanceGrant(time)[†]* carrying the time stamp of the external event.

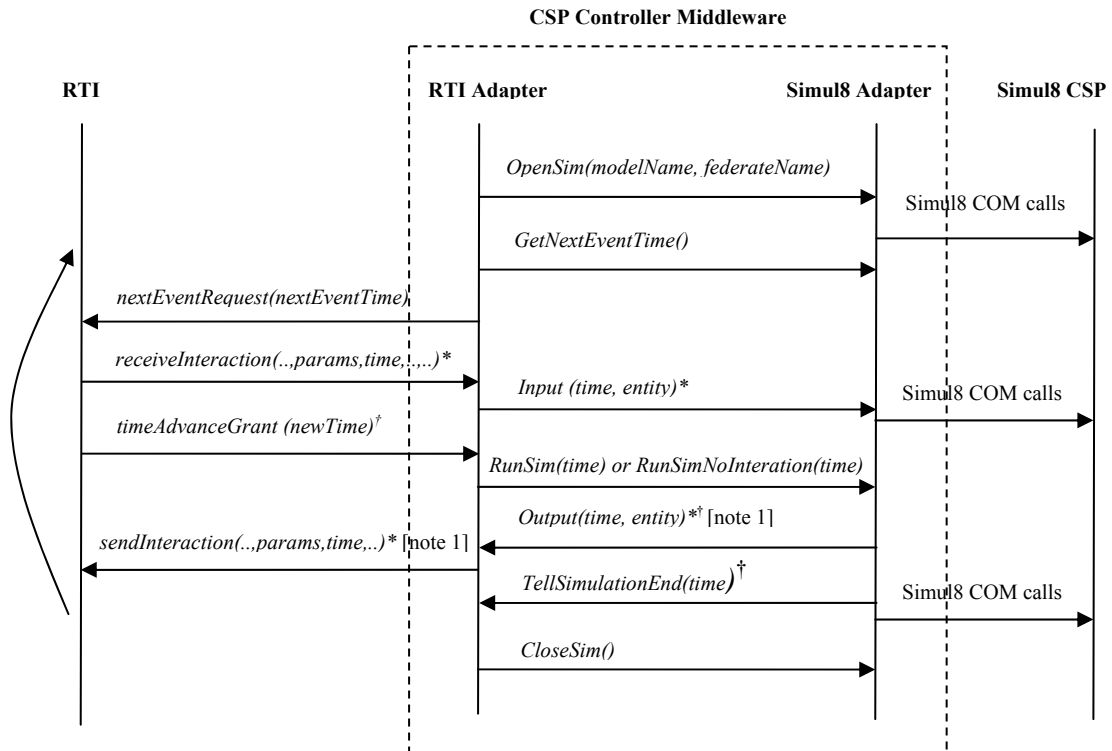
The protocol diagram shows both the responses of RTI.

6.4 Mappings between CSP controller middleware APIs and COTS Simulation Package Handler (CH) APIs

Previous work in this area defined the COTS Simulation Package Handler (CH). CH defined

six APIs for interaction between a COTS Simulation Package Emulator (CSPE) and a distributed simulation middleware [19, 20]. CSPE was designed for solving Type I IRM problems. Four of these CH APIs were for calls made from CH to CSPE: *start()*, *advance(time)*, *advance(time, entity)* and *terminate()*. The call backs received by CH from CSPE were *output()* and *output(time, entity)*. A detailed discussion of these methods can be found in [20].

Our case study presents Simul8-HLA solution to Type I IRM problem and therefore it is only natural for us to consider the CH defined interfaces for CSP-middleware communication. Unfortunately our research has shown that our middleware will not be able to use the same interfaces as CH. Whereas some of the APIs defined by CSP controller middleware can be mapped to the CH APIs, the others differ in terms of argument list and functionality. Two new interfaces have also been introduced in CSP controller middleware to provide additional support required for Simul8-RTI integration. The differences and similarities between the two sets of APIs are outlined below.



[Note 1]: This method is called if *RunSim(time)* has been invoked by RTI Adapter and external events are present in event list..

Figure 6.2: CSP Controller Middleware Protocol Diagram

OpenSim(modelName, federateName) : This is similar to CH *start()* method but has two additional arguments.

GetNextEventTime(arg) : This is a Simul8 specific method which returns the time of next scheduled event in Simul8. It has no equivalent CH method.

RunSim(time) : This method advances Simul8 simulation till *time* and is similar to CH *advance(time)*. However, it additionally reads the future event list of Simul8 to determine whether external events are generated during advance of simulation time.

RunSimNoInteraction(time) : This is a Simul8 specific method. It has no equivalent CH method.

Input(time, entity) : This is a Simul8 specific method and is equal to CH *advance(time, entity)*. Balisford, et. al., [7] discusses how entity attributes are introduced into a Simul8 model.

Output(time, entity)[†] : This method is similar to CH *output(time, entity)*. The Simul8 adapter of CSP controller middleware invokes this method from within *RunSim(time)*.

TellSimulationEnd(time)[†] : This method is similar to CH *output(time)* and is invoked by Simul8 adapter to inform its counterpart that Simul8 has completed simulation till *time*.

CloseSim() : This method is similar to CH *terminate()*.

6.5 Example Simul8 Federation

For our Simul8-RTI case study we have used three *Simul8 federates* and one *Manager Federate* to form a HLA federation. Each *Simul8 federate* models a part of a fictitious manufacturing assembly line consisting of a source, variable number of queues and workstations and a sink.

The three Simul8 federates are arranged in a producer-consumer topology [20] with two producers and one consumer. However to keep our initial experiments simple we have not modelled a feedback from federate C to federates A and B. Figure 6.3 shows the three federate producer-consumer topology along with the representation of model that each federate simulates.

The Manager federate is a special HLA federate that coordinates the execution of the other three

Simul8 federates through registration of synchronization points [27]. Figure 6.4 shows the high level constitution of the Simul8 federation.

For our Simul8-RTI integration experiments we have run the Simul8 federation to arbitrary values in simulation time. The purpose of these experiments was to ensure that the simulation time of the Simul8 federates were synchronized through

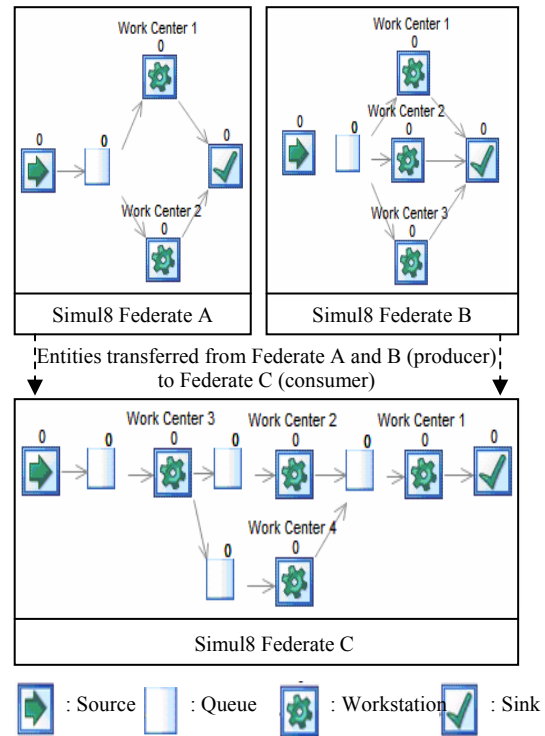


Figure 6.3: Simul8 Federates in Producer-Consumer Topology

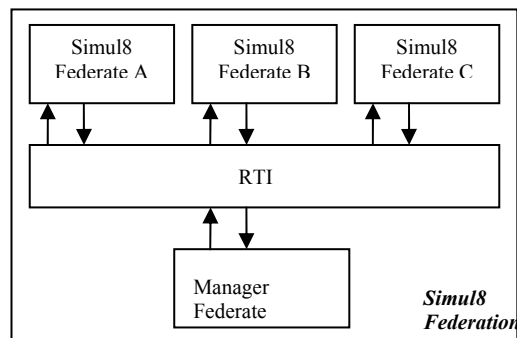


Figure 6.4: Constitution of Simul8 Federation

RTI calls and entity exchanges between federates were properly executed.

7. DISCUSSION AND CONCLUSIONS

Our case study with Simul8 and HLA has highlighted the risks involved in proposing simulation standards based on experimental CSP emulators (like CSPE) and the problems encountered when trying to apply those standards to CSPs, in the absence of active participation of CSP vendors. The fact that we were able to perform a Simul8–HLA distributed simulation at all, is largely due to the support we received from Simul8 Corporation. For the proposed standards to be applied more effectively to existing CSPs would necessitate an even closer co-operation between the research community and the CSP vendors.

This research follows from our earlier work on Type I IRM problems for CSP based distributed simulation. CH introduced in this work defined six APIs for managing communications between CSPE and a distributed simulation middleware. CSPE was designed to solving Type I IRM problems. The CH APIs were loosely based on the Entity Transfer Specification v1.0 (www.cspif.com)

We believed that these CH defined interfaces would accommodate all the communication requirements for distributed simulation between any middleware and CSP for Type I IRM problems. Of course, the implementation of these interfaces would be specific to the CSP and the middleware under consideration. We had an early success when we used RTI and CMB specific implementations of CH API to drive CSPE [20].

In this case study we put the CH APIs to test with a real CSP. Some of the APIs defined by CSP controller middleware were successfully mapped to the CH APIs. And there were some differences also. These differences had arisen because we had no control over Simul8 source code and therefore had to implement the solution with whatever functionality Simul8 exposed. Ideally we would have liked our CSP controller middleware APIs to be *identical* to CH APIs. But in order to realize this we will need to work in close co-cooperation with Simul8.

Our experience has shown the critical role CSP vendors play in distributed simulation research and adoption of standards.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Mark Elder (founder and CEO of SIMUL8 Corporation) and the Simul8 technical support team for their help in the development of the Simul8 adapter.

REFERENCES

- [1] S.J.E. Taylor, S.J. Turner and M.Y.H. Low (2005). The COTS Simulation Interoperability Product Development Group. In Proceedings of the 2005 European Simulation Interoperability Workshop. Simulation Interoperability Standards Organization, Institute for Simulation and Training, Florida, 05E-SIW-056.
- [2] R.M. Fujimoto (2000). Parallel and Distributed Simulation Systems, New York, NY: John Wiley & Sons.
- [3] S.J.E. Taylor, 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), pp.109-122.
- [4] C.A. Boer, A. Verbraeck and H.P.M. Veeke (2002). Distributed simulation of complex systems: Application in container handling. In Proc. EUROSIIW, 02E-SIW-034.
- [5] K. Mertins, M. Rabe and F.W. Jäkel (2000). Neutral Template Libraries for Efficient Distributed Simulation within a Manufacturing System Engineering Platform. In Proc. Winter Simulation Conference, pp. 1549-1557.
- [6] B.P. Gan, M.Y.H. Low, X. Wang, S.J. Turner (2005). Using Manufacturing Process Flow for Time Synchronization in HLA-Based Simulation. In Proc. Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications. IEEE Computer Society. 148-160.
- [7] S. Brailsford, K. Katsaliaki, N. Mustafee and S.J. E Taylor. Modelling Very Large complex Systems using Distributed Simulation: A Pilot Study in a Healthcare Settings, paper submitted to this conference.
- [8] IEEE 1516 (2000). IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA). New York, NY: Institute of Electrical and Electronics Engineers.
- [9] S. Straßburger (2001). Distributed Simulation Based on the High Level Architecture in Civilian Application Domains. Ghent, Belgium: Society for Computer Simulation International, 2001.
- [10] S.J.E. Taylor, B.P. Gan, S. Strassburger and A. Verbraeck (2003). HLA-CSPIF Technical Panel on Distributed Simulation. In Proc. Winter Simulation Conference, pp. 881-887.
- [11] K. Mertins, M. Rabe and F.W. Jäkel (2000). Neutral Template Libraries for Efficient

- Distributed Simulation within a Manufacturing System Engineering Platform. In Proc. Winter Simulation Conference, pp. 1549-1557.
- [12] M. Rabe and F.W. Jäkel (2001). Non military use of HLA within distributed manufacturing scenarios. In Proc. Simulation und Visualisierung, pp. 141-150.
 - [13] M. Rabe and F.W. Jäkel (2003). On standardization requirements for distributed simulation in Production and Logistics. Building the Knowledge Economy, Twente, The Netherlands: IOS Press, pp. 399-406.
 - [14] H. Hibino, Y. Fukuda, Y. Yura, K. Mitsuyuki and K. Kaneda (2002). Manufacturing adapter of distributed simulation systems using HLA. In Proc. Winter Simulation Conference, pp. 1099-1107.
 - [15] C. McLean and F. Riddick (2000). The IMS MISSION architecture for distributed manufacturing simulation. In Proc. Winter Simulation Conference, pp. 1539-1548.
 - [16] R.J. Linn, C.S. Chen, and J.A. Lozan (2002). Development of Distributed Simulation Model for the Transporter Entity in a Supply Chain Process. In Proc. Winter Simulation Conference, pp. 1319-1326.
 - [17] P. Lendermann, B.P. Gan and L.F. McGinnis (2001). Distributed simulation with incorporated APS procedures for high-fidelity supply chain optimization. In Proc. Winter Simulation Conference, pp. 1138-1145.
 - [18] S. Straßburger, G. Schmidgall and S. Haasis (2003). Distributed manufacturing simulation as an enabling technology for the digital factory. Journal of Advanced Manufacturing Systems, vol. 2, no. 1, pp 111-126.
 - [19] N. Mustafee (2003). Performance Evaluation of Interoperability Methods for Distributed Simulation. Masters Thesis. Department of Information Systems and Computing, Brunel University, UK.
 - [20] S.J.E. Taylor, S.J. Turner, N. Mustafee, H. Ahlander and R. Ayani (2005). COTS Distributed Simulation: A Comparison of CMB and HLA Interoperability Approaches to Type I Interoperability Reference Model Problems. SIMULATION. 81, 1, pp. 33-43.
 - [21] K.H. Concannon, K.I. Hunter and J. Tremble (2003). Dynamic scheduling II: SIMUL8-planner simulation-based planning and scheduling. In Proceedings of the 35th Conference on Winter Simulation: Driving innovation, pp. 1488-93.
 - [22] Simul8 Corporation. (2003). Simul8: Manual and Simulation Guide.
 - [23] N.D. Gray, J. Hotchkiss, S. LaForge, A. Shalit and T. Weinberg (1998). Modern languages and Microsoft's component object model, Communications of the ACM, 41(5), pp.55-65.
 - [24] US Department of Defense (1999). High Level Architecture Run-Time Infrastructure RTI 1.3-Next Generation Programmer's Guide.
 - [25] Sun Microsystems Limited. (2003). Java Native Interface. Last viewed on 31st October' 2005.
 - [26] D. Alder (2004). The Jacob Project: A Java-COM Bridge. <http://danadler.com/jacob/>. Last viewed on 31st October' 2005.
 - [27] F. Kuhl, R. Weatherly and J. Dahmann (1999). Creating Computer Simulation Systems, An Introduction to the High Level Architecture. Prentice Hall PTR.

AUTHOR BIOGRAPHIES

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, desktop grids and parallel and distributed simulation modelling.

SIMON J E TAYLOR is the co-founding Editor-in-Chief of the UK Operational Research Society's (ORS) *Journal of Simulation* and this workshop series. He has served as the Chair of the ORS Simulation Study Group since 1996 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.