Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

A COTS SIMULATION PACKAGE EMULATOR (CSPE) FOR INVESTIGATING COTS SIMULATION PACKAGE INTEROPERABILITY

Xiaoguang Wang
Stephen John Turner

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

Simon J. E. Taylor

Centre for Applied Simulation Modeling
School of Information Systems,
Computing and Mathematics
Brunel University, Uxbridge
UB8 3PH, UK

Malcolm Yoke Hean Low
Boon Ping Gan

D-SIMLAB Programme
Singapore Institute of Manufacturing Technology
71 Nanyang Drive
638075 SINGAPORE

ABSTRACT

Commercial Off-The-Shelf (COTS) Simulation Packages (CSPs) are widely used to facilitate the creation of simulation models using some kind of visual interactive interface. CSPs have "evolved" over the years and are well used to support modeling demands in different market niches and domains. However, in terms of distributed simulation, there is almost a complete lack of support for interoperability. The advent of the High Level Architecture (HLA) standard makes it possible to connect distributed model components together. The model components can be developed using specific CSPs best suited to the application area. In this paper, a CSP Emulator (CSPE) is proposed to investigate the interfaces between the CSPs and the HLA Runtime Infrastructure. In addition to support for standalone models as provided by current CSPs, the CSPE has some new features needed for building distributed models. An evaluation is conducted between CSPE and Simul8, one of the popular CSPs.

1 INTRODUCTION

In the last few years, Commercial Off-The-Shelf Simulation Packages (CSPs) have gained popularity in a range of diverse areas such as commerce, health care, manufacturing, military, supply chains, civil and maritime transportation. A CSP provides an easy, efficient and reliable way to build a discrete event simulation model using some kind of visual interactive modeling interface. Many commercial and governmental organizations are now relying on CSPs rather than developing and maintaining their own programs. Examples of CSPs include: ProModel, Arena, AutoMod, Simul8, Extend and Witness.

Meanwhile, distributed simulation has become increasingly important as a way of supporting reuse and interoperability. Other reasons, such as scalability and group working also motivate the use of distributed simulation. It is desirable to develop a distributed simulation based on CSPs by customizing the various simulation packages to suit local requirements, and gluing the components together. Most of the CSPs have some degree of extensibility through VB, COM or Excel etc. which allow the model to interact with the external environment. However, this still introduces time consuming work in developing middleware to link multiple simulation components built using appropriate CSPs (possibly from different companies, even in dispersed locations). In addition, the lack of expertise in distributed simulation (e.g. time synchronization algorithm) may also be a barrier to the development of such middleware. There is a pressing need for a common standard for interoperability of CSPs.

The IEEE 1516 standard *The High Level Architecture* (HLA) was developed by the U.S. Department of Defense (DoD) to facilitate interoperability and reusability (IEEE 1516 2000). The standard provides a common technical framework for the interoperability of simulation models. It comprises four components: a set of HLA rules, the interface specification, the object model template (OMT) and the federation development process (FEDEP). In HLA terms, a federate is a simulation model while a federation is a collection of federates that form the entire simulation. The responsibilities of federates and the federation are described by the rules. The interface specification, implemented by the Runtime Infrastructure (RTI), defines how federates interact with one another. Each federate defines the objects and interactions that are shared in its simulation object model (SOM) using the object model template. The FEDEP is a generalized process for building and executing HLA federations.

Although there are examples of successful distributed simulations with CSPs based on the HLA, a general solution to this problem of heterogeneous integration is missing (Wang et al. 2004a). In this paper we describe a generic interface between the CSP and the HLA RTI which can then be tailored to specific CSPs. By integrating with the HLA RTI through the interface, the CSPs allow the modelers to design their model components in a "plug & play" way (involving only process modeling but no intervention due to the needs of interoperability).

Since currently CSPs are heterogeneous in terms of their properties and extensibility, different CSPs have different degrees of capabilities for their external interfaces. This makes it extremely difficult to find a general approach for the integration. To investigate this problem, we designed a CSP Emulator (CSPE) with a simple simulation engine. The CSPE is intended to emulate the functionality and interface to a CSP and can be used to investigate and to compare various interoperability approaches. Based on the CSPE, the requirements for integration of CSPs and the HLA were investigated and interfaces are proposed for the Type I Interoperability Reference Model (IRM) specified by the CSPI-PDG (COTS Simulation Package Interoperability Product Development Group) (Taylor, Turner and Low 2005). Additionally, the evaluation of the CSPE is also conducted using the Type I IRM.

The rest of the paper is organized as follows. Section 2 describes related work in integrating CSPs with the HLA. In section 3 we propose our generic architecture as well as the interface for the integration of CSPs. Section 4 presents our CSP Emulator (CSPE) to investigate the interfaces between the CSPs and the HLA RTI. Some new features in CSPE to support distributed simulation are described. The CSPE is evaluated in section 5 by conducting a comparison between our CSPE and the CSP, Simul8 (Simul8 Corporation 2005) on the basis of a simple bicycle manufacturing system. For CSPE, both a standalone model and distributed simulation are created for the system. Finally, section 6 concludes the paper and outlines further work in this area.

2 RELATED WORK

2.1 Requirements for Integration of CSPs with the HLA

In (Straβburger et al. 1998), the requirements for integration of CSPs with the HLA are analyzed from two aspects: requirements resulting from being part of a distributed simulation and requirements resulting from a certain programming paradigm. Being part of a distributed simulation indicates that the CSP should provide an interface to connect to other systems or programs. This interface also requires a common standard for data representation and exchange as well as a synchronization mechanism between different simulation components. For the programming paradigm, on the other hand, special consideration is needed because of the ambassador paradigm of the HLA (Kuhl, Weatherly, and Dahmann 1999). The CSP should be able to communicate with the HLA RTI through the RTIAmbassador and implement the corresponding FederateAmbassador.

Based on the analysis of requirements, a number of HLA interfaces for CSPs have been developed. These interfaces make the CSPs HLA compatible, allowing them to be integrated into a distributed simulation environment. The interfaces can be classified as either explicit or implicit from the modeler's point of view. While the explicit approach needs the modeler to enhance the model with HLA functionality, the implicit approach means all HLA functionality is hidden from the modeler since the CSP and its underlying software handle all the HLA synchronization and communication. For example, while an implicit approach could be used for Simplex3 (Straβburger 2001) because its source code is available, an explicit approach is easier for SLX (Straβburger et al. 1998) since SLX does not provide the functionality to support the implicit approach. Obviously, the implicit approach makes it easier for the modeler to link simulation models together. We propose a generic architecture which can be adopted for various CSPs using an implicit approach from the user's point of view.

A CSP needs new features to enable a model built using the package to join a distributed simulation. Some work has been done in this area to suggest new features to be added to Simul8 and CSPs in general to provide interoperability functionality (Ryde and Taylor 2003). Similarly new features are also introduced in our CSPE. Our CSPE extends the CSPE in (Taylor et al. 2005a), which was first introduced to create a pipeline model for benchmarking purpose. The extensions include increased functionality and a more flexible user interface for building general models.

2.2 CSPI–PDG Interoperability Reference Models

In 2004, the CSPI-PDG (Commercial-Off-The-Shelf Simulation Package Interoperability Product Development Group) was approved by the Simulation Interoperability Standards Organization (SISO) (Taylor, Turner and Low 2005). Previously known as the HLA CSPI Forum (HLA-CSPIF 2004), it is dedicated to creating a standardized approach to support the interoperation of discrete event models created in CSPs using the IEEE 1516 High Level Architecture.

The Interoperability Reference Models (IRMs) are one set of products produced by the CSPI-PDG. The aim of the IRMs is to categorize the integration problem into different requirements, thereby providing an easy way to create solutions for each specific integration problem. The following are the six IRMs currently identified by the CSPI-PDG:

- 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.

The Type I IRM *Asynchronous Entity Passing* deals with the common requirement of transferring entities between simulation models. In the Type II IRM *Synchronous Entity Passing*, the input model can transfer entities only when it makes sure that the destination side is not blocked (workstation) or not full (queue) in the receiving model. The other four types of IRM deal with the sharing of resources, events, data structures and transportation tools across simulation models. In this paper, the interface for the Type I IRM model is investigated.

3 INTEGRATION OF A CSP WITH THE HLA

3.1 The Generic Architecture

Based on the previous requirements analysis, we propose a generic architecture with the incorporation of a DSManager library and extended RTI as shown in Figure 1. The DSManager provides a generic interface consisting of a set of functions to be invoked by the CSP or CSP Emulator (see Section 4) when necessary. The C++ / Java based RTI is wrapped by "normal" C functions, that can easily be integrated with most of the current CSPs written in C, C++, Java or VB. Another important feature of the DSManager is to try to hide the HLA concept from both the CSP and the modeler. It is difficult to match model information represented in the CSPs to the object/interaction concept in the HLA standard. In addition, the terminology between different CSPs differs as there is no internationally recognized naming convention. The interface adopts a generic approach based on the concept of entity transfer, and will be proposed as a standard by the CSPI-PDG in the future.

In the architecture, the DSManager also interacts with the extended RTI which is developed using a middleware approach (Gan et al. 2003). In this extended RTI, known as RTI+, appropriate synchronization algorithms are designed in order to improve the simulation performance and relieve the user from the burden of time management. Examples are a shared state manager (Gan et al. 2003) for conservative synchronization and a rollback controller (Wang et al. 2004b) for optimistic synchronization. The RTI+ library is composed of the RTIAmbassador+, the FederateAmbassador+ and the original RTI library. It provides all the services in the RTIAmbassador as well as some new or modified/extended services and filters the callbacks through the FederateAmbassador+ before passing them to the FederateAmbassador. It is important that all necessary implementation in the RTI+ is transparent to the DSManager, CSP and the modeler. The RTI+ library is linked with the DSManager instead of the RTI library.

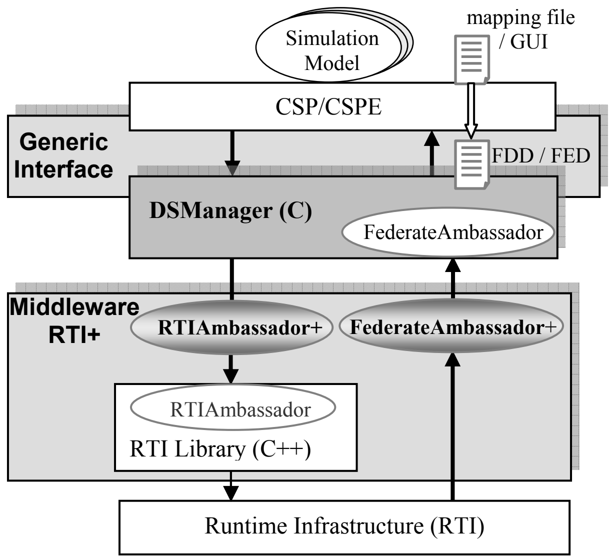


Figure 1: A Generic Architecture for Integration of a CSP with the HLA

The implementation of the generic architecture is based on the cooperation between the model, the CSP, the DSManager and the RTI+. Although the DSManager and the RTI+ are responsible for interacting with other simulation components within the distributed simulation, the model and the CSP have to fulfill some requirements to be part of the distributed simulation (as discussed in Section 4).

3.2 The Generic Interface

The DSManager provides an interface consisting of a set of functions to be invoked by the CSP when a distributed simulation is created. Through the interface, the DSManager invokes necessary calls to the RTIAmbassador+ on behalf of the CSP and transfers the information received from the FederateAmbassador+ to the CSP. Figure 2 shows the basic communication protocol between the CSP, DSManager and RTI+ for CSPI-PDG Type I IRM.

In the initialization phase, the CSP registers its model as part of a distributed simulation via *registerDS*. Receiving such information, the DSManager invokes the corresponding *createFederationExecution* and *joinFederationExecution* in the RTIAmbassador+. One of the component models must be a controller in a distributed simulation. It is responsible for synchronizing the start of the distributed simulation after all the component models have joined the federation. For the controller model in the distributed simulation, *registerController* is also invoked. Since the component model needs to interact with other component models, it should have some input and/or output information. In the generic interface, such information is represented using the concept of entity. The functions *registerInEntity* and *registerOutEntity* are provided for the CSP to declare an exchanged entity. For each exchanged entity, the modeler should give the entity name and the name of the external model with which the entity is exchanged. This information is passed to the RTIAmbassador+ by calling *subscribeInteractionClass* and *publishInteractionClass*. It should be noted that an interaction class rather than an object class is used to transfer an entity in the interface. This decision is based on the work in (Turner et al. 2004).

During the simulation execution, each model needs to advance time to progress the whole distributed simulation. There are various approaches to time management when using the HLA RTI to support distributed simulation (Fujimoto 1998). The approach described here is based on *NextEventRequest* (conservative synchronization). When the CSP wishes to advance to the time of its next event, it issues an *advanceTime* request to the DSManager. The DSManager invokes the corresponding RTI+ service *nextEventRequest*. The response from the RTI+ is zero or more 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 this time may be less than the time initially requested by the CSP (i.e. if entities arrive before the time of the original next event the new time of the 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 number of entities received *nEntityReceived* (if any). If *nEntityReceived* is larger than zero, the CSP will retrieve all the entities via *receiveEntity* and *getAttributeValue* (if the entity has some attributes). In this way, the CSP advances its local simulation time and continues execution. Conversely, if any entities leave the simulation model, the CSP will send them to the DSManager using *setAttributeValue* and *transferEntity* as many times as appropriate. The DSManager will translate these into interactions and then forward these to the RTI+ by invoking *sendInteraction*. This procedure continues until some terminating condition, such as the simulation end time, is met.

The CSP informs the DSManager that the terminating condition is met via *terminateDS*. This causes the DSManager to invoke the RTI+ service *resignFederationExecution* to leave the distributed simulation. In addition, *destroyFederationExecution* is called by the DSManager of the controller model to destroy the distributed simulation.

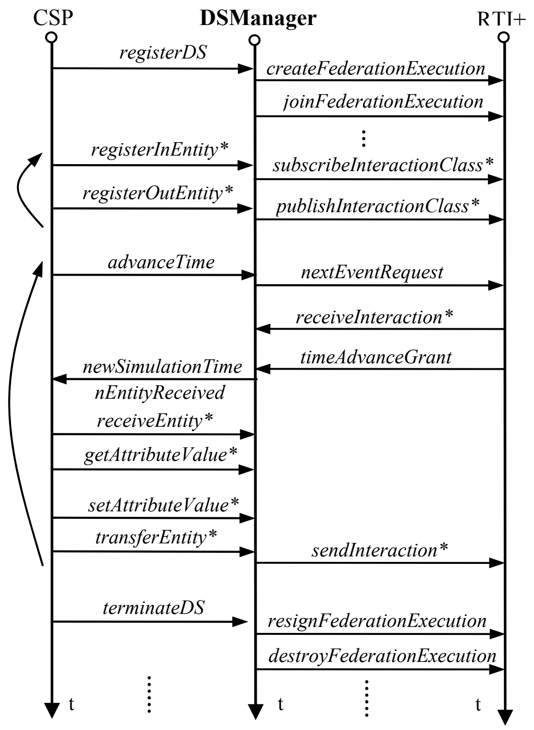


Figure 2: Interaction among CSP, DSManager and RTI+

Besides the functions mentioned above, there are some other functions supporting the distributed simulation. For example, for benchmarking purposes a lookahead value can be set and the synchronization approach to be used can be declared. It may be possible for this part of the work to be transparent to the modeler based on model analysis by the CSP (Taylor et al. 2005b).

For CSPI-PDG Type II IRMs, additional functions must be provided for the CSP to update or check the status of external models. Before sending entities to a destination model, the input model must make sure that the destination side is not blocked (workstation) or not full (queue) in the receiving model. The status information is converted into an interaction by the DSManager and sent/received through the RTI+. This part of the interface is under implementation as part of our research work.

4 STRUCTURE OF THE CSPE

A challenge arises from the implementation of the proposed generic architecture. Due to the heterogeneous properties and various degrees of extensibility, it is extremely difficult to find a general approach for the integration of CSPs with the HLA. The CSP Emulator (CSPE) is proposed to emulate the functionality and interface to a CSP and can be used to investigate and to benchmark alternative interoperability solutions. The CSPE supports the creation of a standalone model in the same way as CSPs, and additionally it has some new features used by the modeler to build a component model as part of a distributed simulation.

4.1 Features for Supporting Distributed Simulation

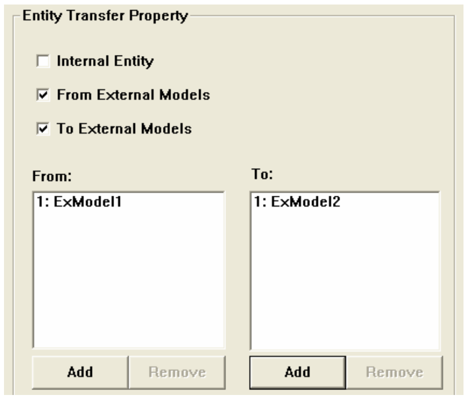
A model built using the CSPE may become part of a distributed simulation by providing some necessary information to the CSPE (as discussed in Section 4.1.1). It is achieved through the new features introduced in the CSPE for supporting distributed simulation. These suggested external and internal features provide an example of how current CSPs may be enhanced with interoperability functionality.

4.1.1 External Features

The CSPE enables the modeler to design interoperating models through a GUI or a specified file. In the CSPE, an entity is processed and passes through some simulation objects. There are four basic types of simulation objects: entry point, queue (bin or storage), workstation (machine) and exit point. The modeler needs to define the attributes and property of each entity type, and assign and link necessary simulation objects to process each entity. To be part of a distributed simulation, some additional menu options are provided in the CSPE.

Figure 3 shows part of the menu for entity definition. The entity can be generated in the local model or received from some external models, and can remain in the local model or be transferred to some external models. If the modeler chooses that the entity is exchanged with external models, the name of the source models or destination models should be input in the list.

Figure 4 shows part of the menu for entry definition. Choosing an external entry means the entity comes from an external model. It should be noted that each external entry here is only for one type of entity from one source model. The modeler can use other external entries if there are entities from other source models.

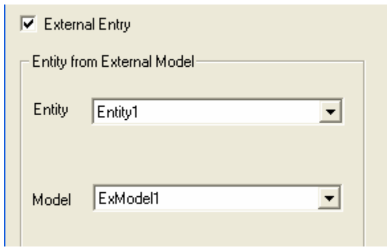

Entity Transfer Property

☐ Internal Entity
☑ From External Models
☑ To External Models

From: 1: ExModel1

To: 1: ExModel2

Figure 3: External Entity


☑ External Entry

Entity from External Model

Entity: Entity1

Model: ExModel1

Figure 4: External Entry

Figure 5 shows part of the menu for exit definition. Similarly to external entry, choosing an external exit means the entity finishing its life cycle in the local model will be transferred to an external model. It is possible there are more than one entity that need to be sent to the same destination model at the same time (one kind of simultaneous events). Some tie-breaking rule is needed to schedule the ordering of these simultaneous events. In CSPE, the approach used is to assign a different priority to each external exit. The higher the priority (lower value), the earlier (in real time) the entity will be sent out.

Figure 6 shows part of the menu for the general definition of the model. A model built using CSPE can be a standalone model or part of a distributed simulation. The modeler needs to choose one and only one of the component models as a controller model. The controller model is in charge of managing the creation and termination of the distributed simulation. The number of component models in the distributed simulation is only needed by the controller model. It is used for the initialization phase of the simulation execution. Each component model also should give the name of the distributed simulation, and the name of the FED Configuration File which is used to supply the RTI with all necessary federation execution details during the creation of a new federation (DoD 1998). The menu also shows the names of all external models interoperating with local model.

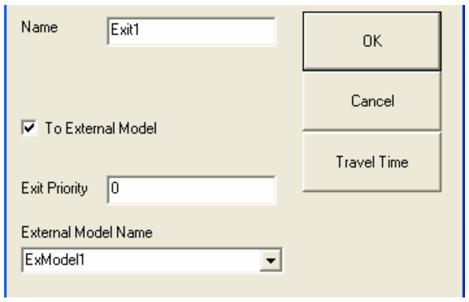


Figure 5: External Exit

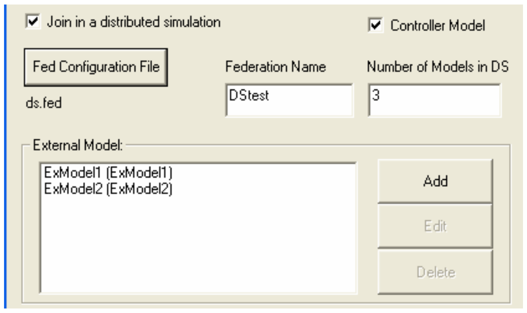


Figure 6: Component Model in a Distributed Simulation

By providing the above additional information in a straightforward way from the modeler's point of view, the model built using the CSPE can join a distributed simulation. The modeler need not worry about the details of transferring entities among the models and is able to build the models easily and efficiently.

4.1.2 Internal Features

In addition to managing the execution of the local model, the CSPE is responsible for interacting with the DSManager. It includes forwarding necessary information describing the distributed simulation to the DSManager, transferring entities from external exit points, and receiving entities from the DSManager and passing them to the corresponding external entry points.

For CSPI-PDG Type II IRMs, the CSPE has extra tasks. For example, it needs to update the status of the external entry points in the local model when they are changed, or check the status of external exit points related with the destination model as appropriate. More work is also necessary for handling simultaneous events received from different external models. The CSPE may also need to provide some tie-breaking rule (e.g. priority information) to the DSManager to achieve simulation consistency over repeated executions.

4.2 Three Phase Approach

The CSPs require some kind of simulation engine (control program or executive) to manage the activities of the entities and trace the life history of each entity. The three phase approach used for modeling a discrete event simulation was suggested by Tocher (Tocher 1963) and discussed in detail in (Pidd 1998). It combines the simplicity of the activity approach with the efficient execution of the event approach. Our CSPE adopts the three-phase approach to demonstrate our architecture to support interoperability.

In a three phase simulation, there are two different types of system events (activities). One type is Bound (B) event, which can be scheduled in advance, bound to happen at some time. The other type is Conditional (C) event, which cannot be scheduled to occur at some time in advance. They depend on some condition, i.e. the states of system resources or availability of the entities.

A three phase simulation has three phases, called A, B and C phase. These three phases are executed continually in a cycle as the simulation proceeds. The phases are as follows:

- **A phase (time scan)** During this phase, the simulation executive determines when the next event is due to happen and then advances the simulation time to this next event time.
- **B phase (B events execution)** During this phase, the simulation executive will process all the B events which are due to execute at the current simulation time. After processing these events, some other new B events or C events may be generated. In addition, these B events may change some conditions (e.g. the states of workstations) and lead to some C events to be executed at the current simulation time.
- **C phase (C events scan and execution)** During this phase, the executive will scan all the C events and execute those C events whose conditions have been satisfied at the current simulation time. Processing C events may also change some states which may satisfy the conditions of other C events. This is repeated until no more C events can be executed in this phase.

In our CPSE, the three phases are executed with operations on two event lists as follows:

- **Bound (B) event list** The list of B events, referred to as the B event list, is ordered by the time stamp of the B event which indicates when the event will happen. For simultaneous events with the same time stamp, they are ordered by some other elements, such as, the priority of the event or the priority of the entities themselves.
- **Conditional (C) event list** The list of C events, referred to as the C event list, is also ordered by some rules, such as the time when the entity

starts waiting for the service, the priority of the event or the priority of the entities themselves.

To avoid spending much time on the C phase, we try to keep the C event list as short as possible. For example, in a manufacturing simulation system, there are usually some entities in a queue, waiting to be processed in a workstation. It is usual to schedule this kind of activity as a C event waiting for the condition that the workstation becomes idle. The entity information is associated with the corresponding C event. However, in some situations where the entity arrival interval is less than the processing time of the workstation, the queue will keep increasing and in turn the C event list will become quite long, resulting in a time consuming C phase. An alternative way is not to produce a C event for this kind of activity. Instead the entity information is associated with the queue where the entity is stored. If any workstation becomes idle, all of its input queues will be scanned to find possible entities waiting to be processed by the workstation. In this method, however, some unnecessary time will be spent in checking the queues without entities in them. Our implementation tries to exploit the advantages of the previous two methods. That is, one and only one such C event is scheduled for each queue with entities in it. The information about each entity in the queue is also kept in the corresponding queue object. In this way, less time is spent on scanning C events compared to the first method. Moreover, it is also more efficient than the second method since queues without entities will not be scanned when the workstations become idle.

5 EVALUATION

In order to evaluate the correctness of the CSPE, we compare simulation results between the CSPE and Simul8. Simul8 is one of the popular discrete event CSPs, and is used in almost every industry for a wide variety of applications. Since Simul8 currently cannot support distributed simulation, only a standalone model is built using it. For CSPE, however, both a standalone and distributed simulation are created. The experiments were carried out using a bicycle manufacturing system, which for the distributed simulation is a CSPI-PDG Type I IRM.

Figure 7 shows a distributed and deterministic simulation for the bicycle manufacturing system. It consists of three main parts: a wheel production line (WPL), a frame production line (FPL), and a bicycle assembly line (BAL) that assembles two wheels to one frame to produce a bicycle. The BAL checks wheels for faults and can return them to the WPL for re-machining (an example of valid feedback for Type I IRMs). To achieve a deterministic model for evaluation, the *Circulate* routing-out rule is used here at workstation W3a. This means that the first entity will go to the first destination (exit point Ex3b), the second work item to the second (queue Q3b) and so on. Frames have no such feedback.

To describe part of the simulation, raw materials for the WPL arrive every 20 minutes at entry point En1a and wait for processing in Q1a. When machine W1a becomes free, raw materials are taken from queue Q1a, processed into wheels and released. This activity takes a fixed time of 20 minutes. We assume that the entry point, the queue and the machine are adjacent. The newly created wheels then take 100 minutes travel time to be transferred to the BAL's entry point En3a. The rest of the distributed simulation can be described in a similar manner with the various times to perform actions shown on the models. In this deterministic model all distributions are fixed. A corresponding standalone and deterministic model is shown in Figure 8, where the simulation process is the same as the distributed one except that all the process is completed in one combined model named Bicycle Manufacturing System (BMS).

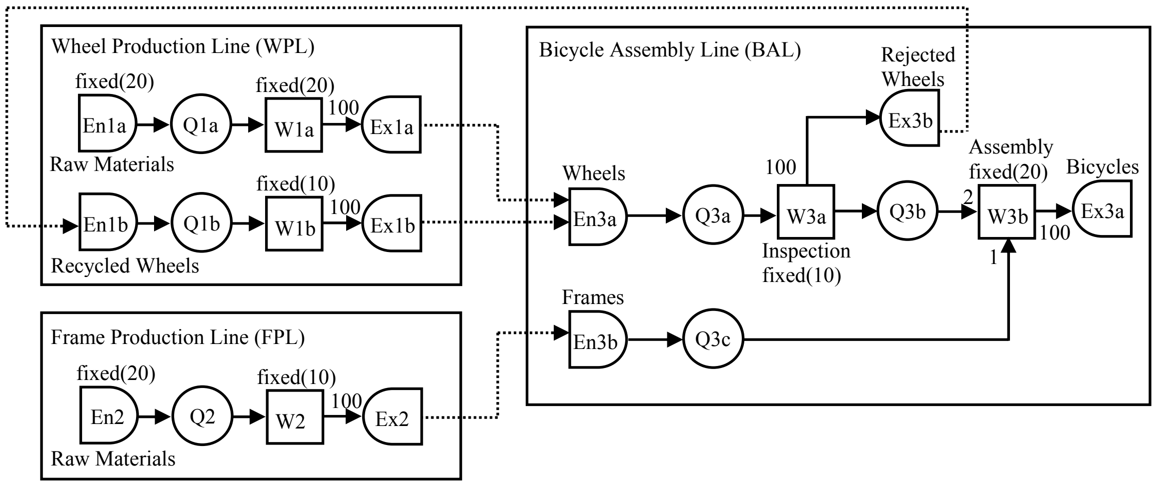


Figure 7: The Bicycle Manufacturing System (Distributed & Deterministic)

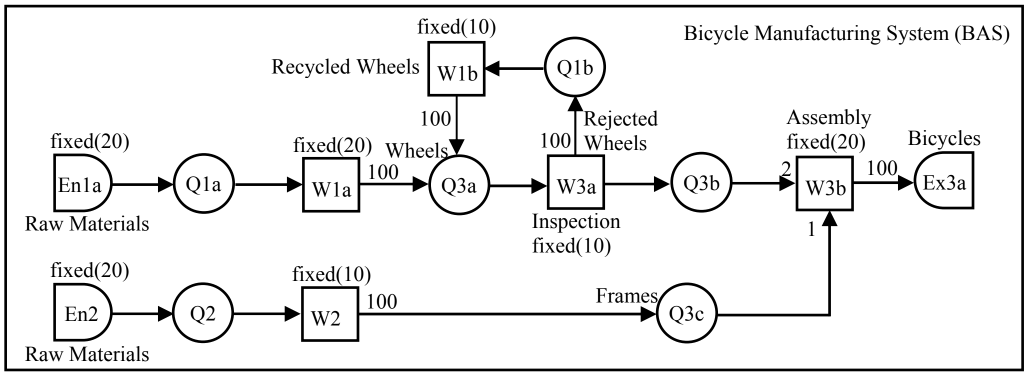


Figure 8: The Bicycle Manufacturing System (Standalone & Deterministic)

The experiments for the distributed simulation were run on four DELL 2.8GHz P4 1GB memory computers connected via a 1Gbps network. One computer was used to run the rtiexec (DMSO RTI1.3NG-V6), and the other three for three separate components models (WPL, FPL and BAL models respectively). The experiments for the standalone model were run on one such computer.

Table 1 shows the experimental results for simulating the system for 100,000 simulation time in Simul8, CSPE(SA) (standalone model) and CSPE(DS) (distributed simulation). The final throughput of the system as well as the statistics for each simulation object are identical for all three cases, showing the correctness of the CSPE and successful interoperability of CSPE models.

Table 1: Experiment Results for Deterministic Model

		Simul8	**CSPE(SA)**	**CSPE(DS)**
En1a	Arrival Entities	5000	5000	5000
En2		5000	5000	5000
En1a	Refused Entities	0	0	0
En2		0	0	0
Q1a	Total Entered Entities	5000	5000	5000
Q1b		4978	4978	4978
Q2		5000	5000	5000
Q3a		9966	9966	9966
Q3b		4982	4982	4982
Q3c		4994	4994	4994
Q1a	Queue Length at End Time	0	0	0
Q1b		0	0	0
Q2		0	0	0
Q3a		0	0	0
Q3b		0	0	0
Q3c		2503	2503	2503
W1a	Completed Entities	4999	4999	4999
W1b		4977	4977	4977
W2		4999	4999	4999
W3a		9965	9965	9965
W3b		2490	2490	2490
W1a	Status at End Time	busy	busy	busy
W1b		busy	busy	busy
W2		busy	busy	busy
W3a		busy	busy	busy
W3b		busy	busy	busy
Ex1	Completed Entities	2488	2488	2488

Table 2: Experiment Results for Stochastic Model

		Simul8	**CSPE(SA)**	**CSPE(DS)**
En1a	Arrival Entities	5000	5000	5000
En2		5000	5000	5000
En1a	Refused Entities	0	0	0
En2		0	0	0
Q1a	Total Entered Entities	5000	5000	5000
Q1b		4976	4894	4894
Q2		5000	5000	5000
Q3a		9947	9869	9869
Q3b		4920	4969	4969
Q3c		4994	4994	4994
Q1a	Queue Length at End Time	16	15	15
Q1b		0	0	0
Q2		0	0	0
Q3a		47	1	1
Q3b		0	0	0
Q3c		2534	2509	2509
W1a	Completed Entities	4983	4984	4984
W1b		4975	4894	4894
W2		4999	4999	4999
W3a		9899	9867	9867
W3b		2459	2484	2484
W1a	Status at End Time	busy	busy	busy
W1b		busy	idle	idle
W2		busy	busy	busy
W3a		busy	busy	busy
W3b		busy	idle	idle
Ex1	Completed Entities	2456	2481	2481

By introducing some probability distributions into the bicycle manufacturing system, another set of experiments is carried out for stochastic models. Instead of a fixed distribution, a normal distribution is used for the processing time in all workstations. For instance, the processing time of W1a is changed from *Fixed* (20) to *Normal* (20, 5) and that of W1b is changed from *Fixed*(10) to *Normal* (10, 2.5). Moreover, the routing-out rule for W3a is changed from *Circulate* to *Percent* (50%, 50%), which also introduces some stochastic property into the model. It is due to the fact that the destination is decided randomly based on the specified percentage going to each destination. As before, both a standalone model and distributed simulation are created for the CSPE. Table 2 shows the experimental results for the stochastic model in all three cases. As with the deterministic model, CSPE(SA) and CSPE(DS) generate identical results. The results between Simul8 and CSPE are also almost identical, showing the correctness of the CSPE. The minor differences between the CSPE and Simul8 are mainly due to different ways of generating random numbers. We can see that the percentage of the entities to be sent to different destinations by W3a is nearly (50%, 50%) for all three cases.

From the above experiments, we find the CSPE can support both standalone and distributed simulations, and generate correct simulation results. It is also interesting to compare the performance between the standalone and distributed simulations, which is one of the motivations for distributed simulation. A comparison conducted based on the same Bicycle Manufacturing System is given in (Taylor et al. 2005b).

6 CONCLUSIONS AND FUTURE WORK

The CSP Emulator (CSPE) is developed to investigate the requirements for integration of CSPs and the HLA. By invoking the interface provided by the DSManager, a middleware in the generic architecture, the CSPE can allow the modeler to build and link component models in a distributed simulation. The DSManager takes the responsibility of interacting with the HLA RTI, which is almost transparent to the modeler and the CSPs. The interface between the CSPE and the DSManager has been defined for CSPI-PDG Type I IRM. To evaluate the CSPE, some experiments were conducted to compare the simulation results between the CSPE and a typical CSP, Simul8. For the CSPE, both a standalone model and distributed simulation are created for the same system. The results are analyzed, showing the correctness of the CSPE. Based on the work of the integration of the CSPE with the HLA, the generic interface has also been applied for another CSP, Autosched AP, to support the interoperability of distributed simulation models (Gan et al. 2005).

Some work is still required in this area. We will develop the interface for CSPI-PDG Type II IRM and extend the CSPE to support this type of model. To improve the performance of distributed simulation, it is necessary to exploit lookahead. It is desirable for the CSPE to compute the lookahead value automatically based on the scenario of the model, without interference of the modeler. The lookahead value plays an important role in performance improvement for conservative synchronization. In those situations where lookahead value is difficult to exploit, it is worthwhile to include optimistic synchronization into the interface for the interoperation of CSPs.

ACKNOWLEDGMENTS

This project is part of a pilot project funded by the Agency for Science, Technology and Research (A-STAR) Thematic Strategic Research Program under the theme "Industrial IT - Integrated Manufacturing and Service Systems".

REFERENCES

DoD 1998. High Level Architecture Federation Execution Data (FED) File Specification – RTI 1.3 Version 3. Department of Defense. 31 July.

Fujimoto, R.M. 1998. Time Management in the High Level Architecture, *Simulation*, Dec., 71(6), pp. 388-400.

Gan, B.P., M.Y.H. Low, J.H. Wei, X.G. Wang, S.J. Turner and W.T. Cai. 2003. Synchronization and Management of Shared State in HLA-Based Distributed Simulation, *Proc. 2003 Winter Simulation Conference*, New Orleans, USA, Dec. 7-10, pp. 847-854.

Gan, B.P., M.Y.H. Low, P. Lendermann, S.J. Turner, X.G. Wang, and S.J.E. Taylor. 2005. Interoperating Autosched AP using the High Level Architecture, Submitted to *Proc. 2005 Winter Simulation Conference*, Orlando, USA, Dec. 4-7.

HLA-CSPIF 2004. CSPI-PDG Product Nomination, www.cspif.com, [viewed November 22, 2004.]

IEEE 1516 2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA), New York, NY: Institute of Electrical and Electronics Engineers.

Kuhl, F., R. Weatherly and J. Dahmann. 1999. Creating Computer Simulation Systems: An Introduction to the High Level Architecture, *Prentice Hall PTR*.

Pidd, M. 1998. Computer Simulation in Management Science, *Wiley*, 4th edition.

Ryde, M.D. and S.J.E. Taylor. 2003. Issues Using COTS Simulation Software Packages for the Interoperation of Models, *Proc. 2003 Winter Simulation Conference*, New Orleans, Louisiana. Dec. 7-10, pp. 772-777.

Simul8 Corporation 2005. Avalable via www.simul8.com.

Straβburger, S., T.Schulze, U. Klein and J.O. Henriksen. 1998. Internet-based Simulation using Off-the-Shelf Simulation Tools and HLA, *Proc. 1998 Winter Simulation Conference*, Washington D.C. December 13-16, pp. 1669-1676.

Straβburger, S. 2001. Distributed Simulation Based on the High Level Architecture in Civilian Application Domains, PhD Dissertation, University of Magdeburg, Germany, April.

Taylor, S.J.E., S.J. Turner and M.Y.H. Low. 2005. The COTS Simulation Interoperability Product Development Group. *Proc. 2005 European Interoperability Workshop*. Simulation Interoperability Standards Organization, Institute for Simulation and Training, Florida, 05E-SIW-056.

Taylor, S.J.E., S.J. Turner, N. Mustafee, H. Ahlander and R. Ayani. 2005a. COTS Distributed Simulation: A Comparison of CMB and HLA Interoperability Approaches to Type I Interoperability Reference Model Problems. *SIMULATION*. 81, 1, pp. 33-43.

Taylor, S.J.E., X.G. Wang, S.J. Turner and M.Y.H. Low. 2005b. Integrating Heterogeneous Distributed COTS Discrete-Event Simulation Packages: An Emerging Standards-Based Approach, Submitted to *IEEE Transactions on Systems, Man and Cybernetics*.

Tocher K.D. 1963. The Art of Simulation. *English Universities Press*, London.

Turner, S.J., Y.Y. Yap, S.J.E. Taylor, M.Y.H. Low. 2004. Comparing High Level Architecture Entity Transfer Mechanisms, Technical Report, Nanyang Technological University.

Wang, X.G., S.J. Turner, M.Y.H. Low and B.P. Gan. 2004a. A Generic Architecture for the Integration of COTS Packages with the HLA, *UK Operational Research Society Simulation Workshop*, Birmingham, UK, Mar. 23-24, pp. 225-233.

Wang, X.G., S.J. Turner, M.Y.H. Low and B.P. Gan. 2004b. Optimistic Synchronization in HLA Based Distributed Simulation, *Proc. 18th Workshop on Parallel and Distributed Simulation*, IEEE Computer Society, Kufstein, Austria, May 16-19, pp. 225-233.

AUTHOR BIOGRAPHIES

XIAOGUANG 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 form 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.

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.

SIMON TAYLOR is a Senior Lecturer in the Department of Information Systems and Computing and is a member of the Centre for Applied Simulation Modeling, both at Brunel University, UK. He is also a Visiting Associate Professor at the Parallel and Distributed Computing Centre at Nanyang Technological University in Singapore. He is also Information Director of ACM SIGSIM, ACM SIGSIM PADS Liaison Officer and Chair of the Simulation Study Group of the UK Operational Research Society. He is a steering committee member of PADS and general co-chair of the UK Simulation Workshop Series. His main research interests are distributed simulation and applications of ICT to simulation modeling. His email address is simon.taylor@brunel.ac.uk.

MALCOLM YOKE HEAN LOW is a Research Fellow with the Production and Logistics Planning Group at the Singapore Institute of Manufacturing Technology. He received his doctorate from Oxford University in 2002. His research interests are in the areas of adaptive tuning and load-balancing for parallel and distributed simulation systems, and the application of multi-agent technology in supply chain logistics coordination. His e-mail address is yhlow@SIMTech.a-star.edu.sg.

BOON PING GAN is a Research Engineer with the D-SIMLAB Programme at the Singapore Institute of Manufacturing Technology. The focus of his research is on the application of distributed simulation technology for supply chain simulation. He received a Bachelor of Applied Science in Computer Engineering and a Master of Applied Science from Nanyang Technological University of Singapore in 1995 and 1998, respectively. His research interests are parallel and distributed simulation, parallel programs scheduling, and application of genetic algorithms. His email address is bpgan@SIMTech.a-star.edu.sg.