COTS Simulation Package (CSP) Interoperability – A Solution to Synchronous Entity Passing

Xiaoguang Wang Stephen John Turner
School of Computer Engineering, Nanyang Technological University
Singapore 639798
xgwang@pmail.ntu.edu.sg assjturne@ntu.edu.sg

Simon J E Taylor
School of Information Systems, Computing and Maths, Brunel University
Uxbridge, UB8 3PH UK
Simon.Taylor@brunel.ac.uk

Abstract

In this paper we examine Commercial-Off-The-Shelf (COTS) Simulation Package (CSP) interoperability for one type of distributed simulation problem, namely synchronous entity passing. Synchronous entity passing is also referred to as the bounded buffer interoperability reference model. It deals with the case where for entities passed between models the receiving queue is bounded or the receiving workstation has limited capacity. This means the sending model must check the status of the receiving model before it can send entities. Correspondingly, the receiving model should update the status information dynamically when it changes. Similar to the work done on asynchronous entity passing, the High Level Architecture is chosen as the underlying standard to support reuse and interoperability. To simplify the integration of the CSP and the HLA, a middleware layer called DSManager is provided. Some new problems generated for synchronous entity passing are discussed and solutions are proposed together with a description of their implementation. Two sets of experiments are conducted to evaluate the solutions using a CSP Emulator (CSPE) which supports both standalone and distributed simulation.

1. Introduction

Commercial-Off-The-Shelf (COTS) Simulation Package (CSP) interoperability aims to enable distributed simulation by linking multiple simulation components built using appropriate CSPs (possibly from different companies, even in geographically dispersed locations). A CSP supports the creation of a discrete event simulation model using some kind of visual interactive modeling interface. Examples of CSPs include: Simul8, Witness, Arena and ProModel. The advent of the High Level Architecture [1] makes it possible to link together these CSPs. The HLA standard was originally developed by the U.S. Department of Defense (DoD) and later adopted as an IEEE standard to facilitate interoperability and reusability. It provides a common technical framework for the interoperability of simulation models.

In 2005, the CSPI-PDG (COTS Simulation Package Interoperability - Product Development Group) [2] was approved by the Simulation Interoperability Standards Organization (SISO). Previously known as the HLA CSPI Forum, 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. There are six IRMs currently identified by the CSPI-PDG.

Based on previous work [3] to successfully link some of the CSPs with the HLA, a generic architecture [4] was proposed for CSP interoperability using middleware named DSManager that adopts an implicit approach from the modeler's point of view. While the explicit approach needs the modeler (those who develop the model using the CSP) 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 middleware handle all the HLA synchronization and

Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06)
0-7695-2587-3/06 $20.00 © 2006 IEEE

communication. Obviously, the implicit approach makes it easier for the modeler to link simulation models together. In this way, the modeler only needs to focus on designing the model components without intervention due to the need of interoperability.

However, currently CSPs are heterogeneous in terms of their properties and extensibility, and 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 solve this problem, a CSP Emulator (CSPE) [5] was designed to emulate the functionality and interface to a CSP and this can be used to investigate and compare various interoperability approaches. Based on the CSPE, the requirements for the integration of CSPs and the HLA were investigated and interfaces were proposed for asynchronous entity passing, the Type I Interoperability Reference Model (IRM) [6].

While asynchronous entity passing focuses on the general problem of entity representation specification, synchronous entity passing (CSPI-PDG Type II IRM) represents another more complicated type of model. In the Type II IRM, the sending model may transfer entities into a bounded queue or a workstation with limited capacity in the receiving model. Thus, entities can be transferred only when the sending model is sure that the destination side is not full (queue) or blocked (workstation). This introduces a synchronous feature into the model, which can be solved by exchanging status information dynamically between the models.

In addition, another problem arises from the existence of inter-model simultaneous events. For example, entities of the same type from different models may need to be sent into the same bounded queue in the receiving model. If there is only space for one entity available, only one model can successfully transfer one such entity and all other such entities need to wait for new space to be available. Different orderings of these inter-model simultaneous events may generate dramatically different simulation results. Usually, the tie is broken by allowing the modelers to specify different priorities for each entry point through which the entity will be transferred into the local model. However, it is possible one entry point may have multiple priorities and the priority may be changed dynamically due to some simulation activities. This requires the priority information to be updated and exchanged at run time. In many simulation systems, the priority is represented by adding a hidden field to the simulation time. In this paper, we state the new problems introduced by synchronous entity passing and describe the solutions and their implementation using the DSManager and two hidden fields appended to the simulation time. To verify the solutions, several sets of experiments are conducted using some typical Type II IRMs.

The rest of this paper is organized as follows: Section 2 discusses related work in CSP interoperability as well as the simultaneous events problem. The special problems of synchronous entity passing are stated in Section 3 and solutions are provided in Section 4. Section 5 describes some issues in implementing the solutions. To evaluate the proposed solutions, several sets of experiments are conducted and discussed in section 6. Conclusions and future work are presented in Section 7.

2. Related work

2.1. CSPI Emulator (CSPE) and DSManager

As one part of the suite of CSPI-PDG standards, the CSP Emulator (CSPE) is intended to emulate the functionality and interface to a CSP. It supports the creation of a standalone model or a model component that is part of a distributed simulation. Based on the CSPE, various interoperability approaches can be investigated and compared. Another benefit of the CSPE is to provide a suggestion how current CSPs may add HLA capability to support distributed simulation.

The CSP or CSPE integrates with the HLA through a generic interface called DSManager. The interface consists of a set of functions to be invoked by the CSP or CSPE when necessary. The C++ / Java based HLA RunTime Infrastructure (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 model. 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.

Based on the CSPE, the requirements for integration of CSPs and the HLA were investigated and interfaces were proposed for the Type I IRM. In this paper, with new features added into the DSManager and the CSPE, the Type II IRM synchronous entity passing is investigated and the solutions are evaluated.

2.2. Simultaneous events problem

In a discrete event simulation, the events are timestamped and executed in increasing order to ensure causality. It is possible two or more events are scheduled at exactly the same simulation time, or at a slightly different simulation time but below the level of the machine precision. These events are considered as simultaneous events. Different orderings of the simultaneous events may generate different simulation results, which may conflict with the requirement of repeatable execution of the simulation programs. Repeatability means the execution of the simulation should produce exactly the same results on each execution when using the same initial state and external inputs.

Much work has been proposed to solve this problem [7]. Usually the solution is to execute these events in an arbitrary order unless the modeler explicitly specifies some tie-breaking technique, for example, FIFO (first-in, first-out), LIFO (last-in, first-out), or dependency order. Some tie-breaking mechanisms can be implemented by extending the timestamp to include additional, lower-precision bits that are hidden from the application program [8]. With different values to these bits, the simulation engine can ensure no two events in the simulation contain exactly the same timestamp. The values could be assigned based on the specified tie-breaking techniques to satisfy the simulation modeler's requirements.

3. Problems of synchronous entity passing

The Type II IRM synchronous entity passing deals with the case where a receiving queue is *bounded* or the receiving workstation has *limited* capacity. An example is shown in Figure 3.1, where the distributed simulation (federation) is composed of two factory models (federates), M_1 and M_2, interacting in the way denoted by the arrows. Each model consists of an entry point En_i, a queue Q_i, a workstation W_i, a resource R_i, and an exit point Ex_i (where i is the model identifier). After being processed in W_1, entities need to be sent periodically via Ex_1 and then entry point En_2 into a bounded queue Q_2 (or a workstation with limited capacity) in M_2. It indicates the requirement that M_1 containing the sending workstation W_1 must, when the processing of an entity is complete, check to determine that there is space in Q_2. If there is space available then the entity may be transferred. Otherwise M_1 must ensure that W_1 is blocked until space becomes available. In this paper, we call the entry point designed to receive entities from external models 'external entry point' and give it the abbreviation of 'EEP'. On the sending side, the EEP in the receiving model is referred to as the remote EEP, and on the receiving side the EEP is referred to as the local EEP.

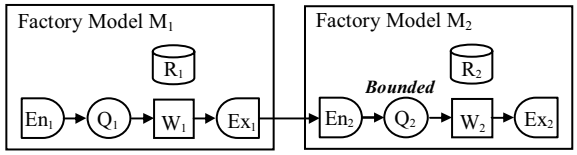

Figure 3.1: Synchronous entity passing

3.1. Status information

As discussed above, it is essential to update the status information of a bounded queue or a workstation with limited capacity in synchronous entity passing. Different from simply updating the status information in a standalone simulation, the status information should also be transferred between the models. In Figure 3.1, when Q_2 becomes full, a message with a small increment to the current simulation time is sent back to M_1, which causes M_1 to block. The small increment is added because the status event is dependent on the entity sending event from M_1. At some later simulation time, when the entity is processed in workstation W_2, M_2 clears a slot in Q_2 and sends another message with a small increment to the current simulation time to M_1, which allows new entities to be transferred.

Due to the complexity of distributed simulation scenarios, the status information may not be updated in time to external models. One case is for inter-model simultaneous events. For example, two models may want to send entities to the same remote EEP at the same simulation time. In the situation where there is only space available to receive one entity, the status of the remote EEP cannot be shown as idle for both sending models. Another case is for passing more than one entity with the same simulation time to the same remote EEP from the same sending model. Suppose two entities from M_1 are waiting to be transferred into M_2 via En_2. After receiving the first entity at time t, it is possible En_2 becomes blocked and the status information of 'blocked' will be transferred to the DSManager in M_1 a short time later at $t+\delta$ (δ is the small increment due to the dependency order). However, M_1 is trying to send the second entity at t since the new status information can only be received at $t+\delta$. Therefore, even though the receiving model has already updated the status of En_2 as 'blocked' or 'idle' based on local information, the status of the remote EEP may be uncertain for the DSManager in the sending model. The possible status of a remote EEP specified by the DSManager in the sending model can be summarized as follows.

0: idle and it is safe for the sending model to send an entity
1: blocked
-1: uncertain since there are possibly some other entities sent from other models to this remote EEP at the same simulation time
-2: uncertain since the entity just sent from the local model may cause the remote EEP to be blocked

To avoid the need for the CSP to handle the uncertain status information (the status of a remote EEP known by the CSP is only 'idle' or 'blocked'), the DSManager should update the status automatically and forward 'blocked' to the sending model when the status is uncertain ('-1' or '-2'). After the DSManager is sure it is safe to send an entity from the sending model, 'idle' will be returned instead.

3.2. Inter-model simultaneous events

In a standalone simulation, it is relatively easy to order the simultaneous events in the local event list based on some tie-breaking mechanisms. In distributed simulation, however, there may exist some simultaneous events transferred between different model components. In the example discussed in section 3.1, two models may want to send entities to the same remote EEP at exactly the same simulation time. These simultaneous events are generated in different models but interleave with each other, referred to as inter-model simultaneous events.

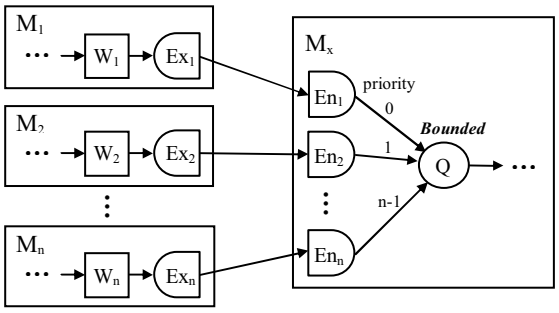

Figure 3.2: Inter-model simultaneous events to external entry point with single priority

Usually, the modeler will assign a priority to order the entities from different sending models. For those cases where no priority is explicitly specified, the DSManager will order them in an arbitrary order. Here our discussion is based on the assumption that the priority is already assigned for each local EEP.

In figure 3.2, Model M_i (i = 1, … , n) generates entities in workstation W_i, and sends them periodically via Ex_i and En_i to a bounded queue Q in Model M_x. In M_x, each local EEP En_i is assigned a different priority for accessing Q. It is possible two or more inter-model simultaneous events exist to transfer entities to Q. In a standalone simulation these entities can be ordered in the event list waiting to be processed. In distributed simulation the entities from each sending model can be transferred to M_x only when the sending model makes sure there is space available in Q and no entities from other sending models with higher priority need to be transferred to the same queue.

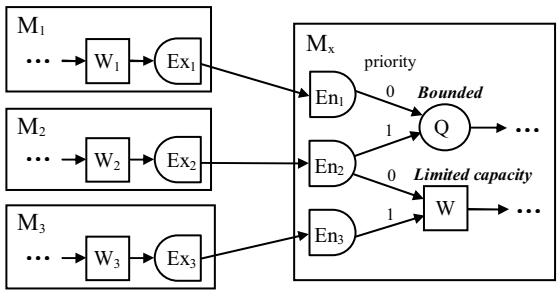

Figure 3.3: Inter-model simultaneous events to external entry point with multiple priorities

Another case is shown in Figure 3.3. The entities may be scheduled with different priorities into multiple queues or workstations via the same local EEP. That means each local EEP is associated with multiple priorities. Therefore, in addition to status information, the priority information should also be updated dynamically based on the simulation activities.

4. Solutions to synchronous entity passing

To address the new problems introduced by synchronous entity passing, solutions are proposed including extending the DSManager and introducing two hidden fields in the timestamp representation.

4.1. Extension to DSManager

Our solution to CSP interoperability is based on a generic interface and associated middleware named DSManager which wraps the HLA synchronization and communication, and provides a set of functions for entity transfer. It provides the necessary functionality used by the CSP simulation engine to support distributed simulation during the whole simulation life cycle.

As discussed in section 3.1, the status information in the Type II IRM is transferred with the timestamp of the current simulation time plus a small increment, considered as a NZL (near zero lookahead) message [9]. Lookahead represents a guarantee from a federate (model) that it will not generate any external message with a timestamp smaller than its current time plus the value of the lookahead. It is critical for conservative synchronization to achieve better performance. In the

Type II IRM, however, the lookahead value has to be set to near zero due to the status information. The DSManager will collect information from the model and automatically set the lookahead value. The CSP needs to tell the interface whether each local EEP is restricted or not. Here 'restricted' means the local EEP may be blocked as it is linked to a bounded queue or a workstation with limited capacity. If any one of the local EEPs is restricted, the DSManager has to set the lookahead as near zero. Otherwise, a larger lookahead may be adopted based on the scenario of the model itself.

Other new functions need be provided to allow the model to update and check status information. In the model which will receive entities from an external model, it must set the status of the local EEP each time it changes. When necessary, it also associates the priority information with the status since it is possible the local EEP has different priorities when it sends entities to different queues or workstations. The local DSManager will transfer such information to the DSManager for the sending model. Before the sending model transfers entities, it will invoke the necessary function to check the status of the appropriate remote EEP. As discussed in section 3.1, to hide the complicated implementation details from the CSP and the model, the status returned by the DSManager is only idle or blocked.

4.2. Hidden fields in timestamp representation

4.2.1. Purposes of hidden fields. Hidden fields in the timestamp can be used to solve the problems of simultaneous events. In the DSManager designed for the Type II IRM, we utilize hidden fields for three purposes.

The first purpose of the hidden fields is to represent the small increment to the simulation time for status information. Different CSPs may have different time units and machine precisions in simulation execution. It is difficult to select a suitable value as the smallest time increment. By appending a hidden field of integer type to the simulation time, it can ensure the small increment will not conflict with the timestamp of any event scheduled by the model since the hidden field is transparent to the model layer.

Another purpose of the hidden fields is to contain priority information to order the inter-model simultaneous events. The lower the priority, the larger the value of the hidden field. In this way, the events with higher priority will be associated with a smaller timestamp and will be processed earlier.

The third purpose of the hidden fields is especially for the case when the sending model needs to send more than one entity to the same remote EEP simultaneously, as discussed in section 3.1. These simultaneous events should be ordered using a hidden field in logical time.

4.2.2. Two hidden fields for synchronous entity passing. There are two hidden fields appended to the simulation time to support synchronous entity passing: one is priority for priority value to order inter-model simultaneous events (status of '-1'), the other is age used to order those simultaneous events sent from the same source model to the same remote EEP (status of '-2'). The small increment to the simulation time for status information is also contained in the second hidden field age. Thus, the logical time is defined as (t, priority, age) where t is the simulation time shown to the model. Importantly, the first hidden field priority has precedence (assigned to more significant bits) over the second hidden field age (more sensitive). Even for the entities with the same type sent to a remote EEP with a specific priority, it is also possible to schedule simultaneous events with different values of age (the first entity sent is with age 0, the second one is with age 1, and so on). It is easier to use two hidden fields to represent the precedence relationship instead of one hidden field.

To ensure the status information is updated as soon as possible, the small increment of simulation time is added to the second hidden field, which is more sensitive than the first one. The value sent for the small increment is less than the value increased each time the model needs to send another entity to the same remote EEP. Here, we represent each age a (a is a non-negative integer 0, 1, 2, …) as $10*a$ (0, 10, 20, …) and use 5 (any value between 1 to 9 is acceptable) as the small increment in *age* for status information. Consequently, the near zero lookahead discussed previously is also set as 5 in the second hidden field since it is the smallest increment for the logical time.

Let us illustrate the hidden fields using the case in Figure 3.2. Suppose the status of Q at time t is idle. Only M_1 can directly transfer an entity to Q because the corresponding remote EEP En_1 has the priority of 0. For each other sending model M_i (i = 2, 3, …, n) that wants to transfer the a_j^{th} (a_j = 0, 1, …) entity at time t, the DSManager sets *priority* as p_i (p_i = 1, 2, ... , n-1) and age as $10*a_j$, and tries to advance time to (t, p_i, $10*a_j$). Only when the granted time is equal to the requested time and no status information of 'blocked' is received during the time advancement, is the 'idle' status returned to the model by the DSManager for M_i. If the a_j^{th} entity sent from M_i causes Q to be blocked, the new status information will be sent to all sending models at time (t, p_i, $10*a_j+5$), which stops M_i sending other entities and meanwhile allows the models, including M_{i+1} to M_n, to receive the 'blocked' signal

before their requested time is granted. In this way, the entities from the sending models can be sent in the correct order as specified by the priority of the corresponding remote EEP.

5. Implementation issues

The proposed solutions are implemented in the DSManager middleware as well as the logical simulation time defined by the IEEE HLA standard.

5.1. DSManager

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. The basic communication protocol between the CSP, DSManager and RTI for CSPI-PDG Type I IRM is described in [6]. Here we only discuss the new features in the interface to the CSP for Type II IRM.

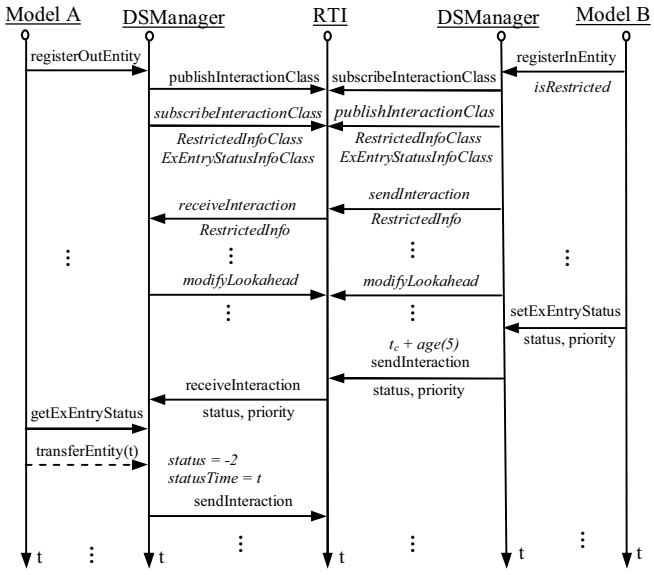

Figure 5.1 New features of interface for Type II IRM

In Figure 5.1, we use model A and model B to demonstrate the sending and receiving models respectively. Suppose model A transfers entities to a bounded queue or a workstation with limited capacity in model B. On each side, there is a DSManager used to communicate with the HLA RTI on behalf of the model.

In the initialization phase, model A and model B need to register the entity which is exchanged via *registerOutEntity* and *registerInEntity*. It should be noted that the 'isRestricted' information is also provided by each local EEP in model B. If any local EEP is restricted, the DSManager in Model B will call *modifyLookahead* to modify the lookahead value to near zero. Correspondingly, the lookahead in Model A should also be set to near zero by the DSManager in Model A. This information can be forwarded to the DSManager in model A by invoking *sendInteraction*. Before that, the DSManager on each side needs to declare the interest to send or receive such information by calling *publishInteractionClass* and *subscribeInteractionClass*. Additionally, the DSManager also automatically declares the interest to send or receive the status information as well as the priority for each EEP.

During the simulation execution, if the status of a local EEP is changed due to the simulation activities, model B will inform the DSManager by calling *setExEntryStatus* with the new status ('idle' or 'blocked') and current priority. Instead of increasing the time at the model level, the hidden field *age* is increased by the small increment which is transparent to the model. Then the DSManager will transfer the information to model A via *sendInteraction*.

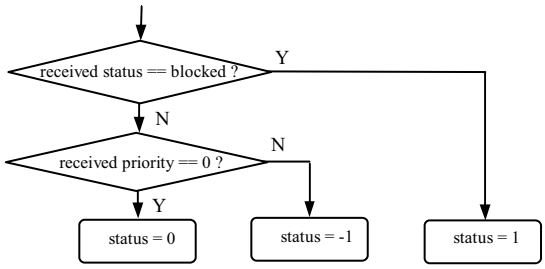

Figure 5.2: Receive ExEntryStatus procedure

In model A, the DSManager will set the status based on the received information (as shown in Figure 5.2). If the status is idle while the priority value is larger than 0, it is possible entities may be transferred to a remote EEP with a higher priority which also shares the queue or workstation with the remote EEP for this entity. In this case, the status is uncertain and has be set as '-1'. Before transferring an entity to model B, the CSP needs to check the status of the corresponding remote EEP in model B using *getExEntryStatus*. The DSManager will return 'idle' or 'blocked' after considering the simulation activities in the local model in addition to the status and priority information received from model B (as shown in Figure 5.3). After transferring an entity via *transferEntity* to model B, the DSManager in model A will locally change the status of the corresponding

remote EEP to '-2' since the entity may cause the remote EEP in model B to be blocked.

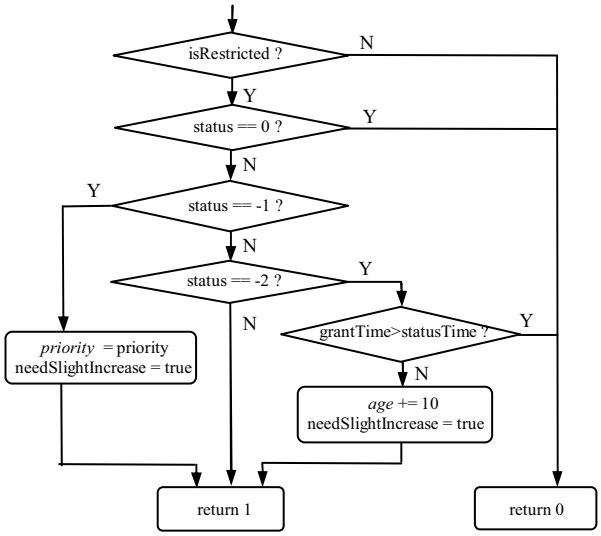

Figure 5.3: getExEntryStatus procedure

As we know, each model needs to advance time to progress the whole distributed simulation. Specifically, in the Type II IRM, the requested time forwarded to the RTI is possibly associated with a slight increase represented by the hidden fields. This may slow down the simulation if the hidden fields are added for each time request. The variable '*needSlightIncrease'* is used to identify whether it is necessary to set the hidden fields in the next request to advance time. Figure 5.3 shows that it is only set to 'true' when the hidden field needs to be appended. Another variable '*statusTime'* gives the time when the new status is updated. After sending the entity to the external model, the status is set as '-2' and the *statusTime* is updated to the current logical time. However, if the current granted time is larger than *statusTime* and the status is still '-2', this means there is no new status information of 'blocked' received from the external model. In this case, 'idle' is returned to the model. Otherwise, 'blocked' is returned since the status is still uncertain, and the hidden field age should be increased enough to see whether there is new status information received in the next request to advance time.

Figure 5.4 shows the general procedure for time advancement. Each model advances time by invoking *advanceTime* to the DSManager. In the procedure, the hidden fields may be added to the requested time (requestedTime) provided by the CSP and passed to the RTI (by calling *setHiddenField* method). After a safe time is granted from the RTI, the DSManager will clear the hidden fields and update the uncertain status information based on the granted time and received status information (if any) during the time advancement. If the granted time is equal to requested time and the status is still '-1' or '-2', the status is updated to '0' since that means no new status information of 'blocked' is received before the requested time during the time advancement. Finally, the simulation time without hidden fields (by calling *getTime* method) is returned to the model since the hidden fields are transparent to the model.

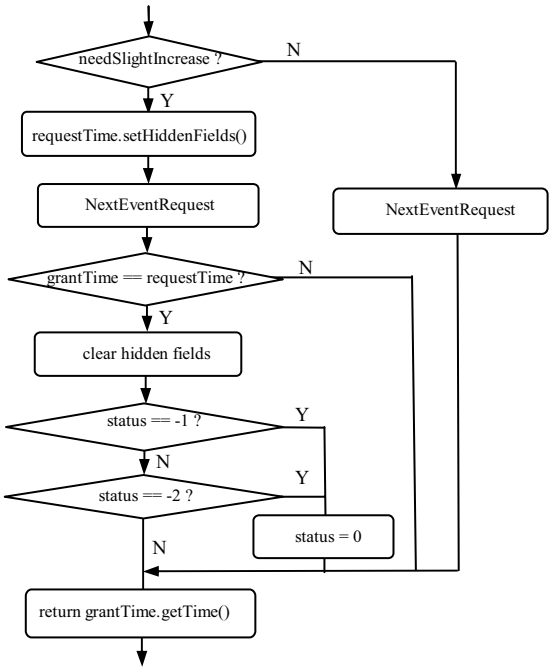

Figure 5.4: advanceTime procedure

5.2. FedTime in the RTI implementation

In the HLA standard, logical time is defined as an abstract class which allows the user to implement a version of this class for their own purposes. This provides the possibility to add the hidden fields to the *FedTime* in the RTI implementation. To extend *FedTime* with the new attributes of '*priority*' and '*age*', some supported functions are provided for operation and comparison between logical time values. For instance, for comparison using the '>' operator, suppose there are two timestamps: T_1 (t_1, $priority_1$, age_1) and T_2 (t_2, $priority_2$, age_2). If t_1 is larger than t_2, the result is 'true'; else if t_1 is equal to the t_2, the result is 'true' when $priority_1$ is also larger than $priority_2$; else if t_1 is equal to t_2 and $priority_1$ is equal to $priority_2$, the result is 'true' only when age_1 is also larger than age_2. Modifications also need to be made to the *encode* and *decode* functions in the *FedTime* class to include and exchange the hidden fields via the network.

In our implementation, the *FedTime* class provided by DMSO RTI1.3NG-V6 [10] was extended and the new generated library *libFedTime* was linked to the DSManager.

6. Experiments

Some experiments are designed to test the proposed solutions for Type II IRM synchronous entity passing. The experiments are conducted using the CSPE which is linked with the DSManager for Type II IRM. To ensure the simulation results are correct, we choose Simul8 [11], one of the popular discrete event CSPs, to run a standalone simulation for the same simulation model.

6.1. Normal synchronous entity passing model

Figure 6.1 shows a distributed and deterministic simulation for the bicycle manufacturing system [5]. 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). To achieve a deterministic model for evaluation, the *Circulate* routing-out rule is used here at workstation W_{3a}. This means that the first entity will go to the first destination (exit point Ex_{3b}), the second work item to the second (queue Q_{3b}) and so on. A corresponding standalone and deterministic model is also created, 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). To demonstrate the Type II IRM, the maximum length of all the queues in the model is set as 1, 10 and 100 separately for three sets of experiments.

Moreover, another set of experiments is carried out for stochastic models by introducing some probability distributions into the system. 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 the routing-out rule for W_{3a} is changed from *Circulate* to *Percent* (25%, 75%), 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.

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 component models (WPL, FPL and BAL models respectively). The experiments for the standalone model were run on one of these computers.

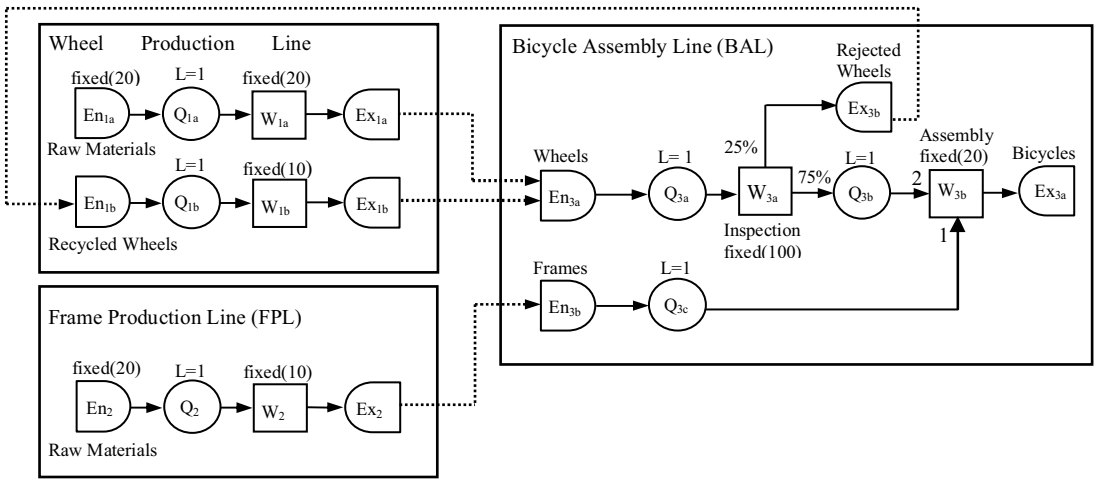

Figure 6.1: The Type II Bicycle Manufacturing System (distributed & deterministic)

Table 6.1: Experimental results for distributed and standalone simulation on CSPE and Simul8 (L=1)

		Deterministic			Stochastic		
		Simul8	CSPE(SA)	CSPE(DS)	Simul8	CSPE(SA)	CSPE(DS)
En_{1a}	Arrival Entities	5000	5000	5000	5000	5000	5000
En_2		5000	5000	5000	5000	5000	5000
En_{1a}	Refused Entities	4496	4496	4496	4843	4822	4822
En_2		4747	4747	4747	4921	4910	4910
Q_{1a}	Total Entered Entities	504	504	504	157	178	178
Q_{1b}		500	500	500	47	64	64
Q_2		253	253	253	79	90	90
Q_{3a}		1001	1001	1001	200	238	238
Q_{3b}		499	499	499	151	172	172
Q_{3c}		251	251	251	77	88	88
Q_{1a}	Queue Length at End Time	1	1	1	1	1	1
Q_{1b}		0	0	0	1	1	1
Q_2		1	1	1	1	1	1
Q_{3a}		1	1	1	1	1	1
Q_{3b}		0	0	0	0	0	0
Q_{3c}		1	1	1	1	1	1
W_{1a}	Completed Entities	503	503	503	156	177	177
W_{1b}		500	500	500	46	63	63
W_2		252	252	252	78	89	89
W_{3a}		999	999	999	199	237	237
W_{3b}		249	249	249	75	86	86
W_{1a}	Status at End Time	busy	busy	busy	busy	busy	busy
W_{1b}		busy	busy	busy	busy	busy	busy
W_2		busy	busy	busy	busy	busy	busy
W_{3a}		busy	busy	busy	busy	busy	busy
W_{3b}		busy	busy	busy	busy	busy	busy
Ex_1	Completed Entities	249	249	249	75	86	86

Table 6.1 shows the experimental results for simulating the system for 100,000 time units in Simul8, CSPE(SA) (standalone model) and CSPE(DS) (distributed simulation) with a maximum queue length of 1. The final throughput of the system as well as the statistics for each simulation object are identical for all three cases when the deterministic model is used, showing the correctness of the CSPE and successful interoperability of Type II IRMs. As for the stochastic 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. With a queue length of 10 and 100, similar experimental results were generated (not shown here).

These results show that the CSPE integrated with the DSManager for Type II IRM can generate correct simulation statistics, indicating the status information is successfully transferred between the models. It is also interesting to investigate the overhead introduced by the new features in the DSManager in situations where the EEP is not restricted. We carried out another set of experiments using a Type I IRM, the same BMS except all the queues are unbounded. The experimental results were compared between the CSPE with DSManager for Type I IRM and the CSPE with DSManager for Type II IRM. We found the simulation results were identical and only around 2 more seconds were spent in execution time using the Type II DSManager, 36.56 seconds as compared to 34.38 seconds using the Type I DSManager. It is not a large overhead and optimization will be applied to the DSManager in the future.

6.2. External entry point with multiple priorities

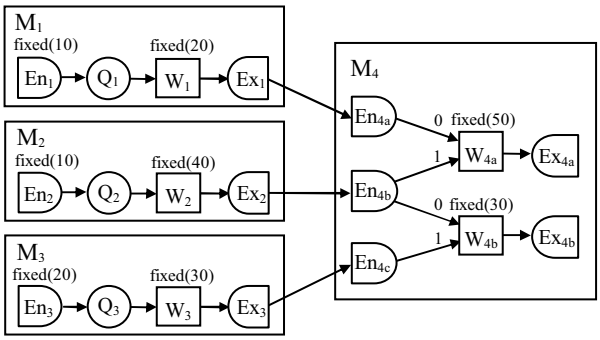

Figure 6.3: Type II IRM with external entry points having multiple priorities

Table 6.2: Experimental Results for Type II IRM with external entry points having multiple priorities

		Simul8	CSPE(SA)	CSPE(DS)
En_1	Arrival Entities	100	100	100
En_2		100	100	100
En_3		50	50	50
En_1	Refused Entities	0	0	0
En_2		0	0	0
En_3		0	0	0
Q_1	Total Entered Entities	100	100	100
Q_2		100	100	100
Q_3		50	50	50
Q_1	Queue Length at End Time	79	79	79
Q_2		83	83	83
Q_3		33	33	33
W_1	Completed Entities	21	21	21
W_2		17	17	17
W_3		16	16	16
W_{4a}		19	19	19
W_{4b}		31	31	31
W_1	Status at End Time	busy	busy	busy
W_2		busy	busy	busy
W_3		busy	busy	busy
W_{4a}		busy	busy	busy
W_{4b}		busy	busy	busy
Ex_{4a}	Completed Entities	19	19	19
Ex_{4b}		31	31	31

To test the Type II IRM with EEPs having multiple priorities, we create another distributed simulation consisting of 4 models M_1, M_2, M_3 and M_4. M_1, M_2 and M_3 transfer entities to two workstations with fixed capacity in M_4 via three EEPs. As is shown in Figure

6.3, En_{4b} has lower priority than En_{4a} for W_{4a}, but has higher priority than En_{4c} for W_{4b}. So it is possible the priority of En_{4b} may be changed dynamically when an entity is passed to a different workstation.

Table 6.2 shows the distributed simulation produces identical results to the standalone simulation. This proves that priority as well as status information is correctly transferred between different models. Also the inter-model simultaneous events are processed in the correct order when updating the priority dynamically.

From the above two sets of experiments, we found the CSPE integrated with the new DSManager can run both a normal Type II IRM with bounded queue and those special models with EEPs having multiple priorities. Furthermore, the new DSManager designed for Type II IRM can also be applied for Type I IRM without too much overhead. In this way, the model only needs to inform the DSManager whether each local EEP is restricted (linked with a bounded queue or a workstation with limited capacity) or not, without identifying the type of the model itself.

7. Conclusions and future work

This paper investigates the integration of CSPs for CSPI-PDG Type II IRM synchronous entity passing. We describe solutions to the new problems introduced by status information transfer and inter-model simultaneous events. The implementation was achieved by adding new features into the DSManager and extending the HLA RTI logical time with two hidden fields. Importantly, all the complicated details are transparent to the CSPs and the modelers. This allows the modelers to design their model components in a "plug & play" manner without worrying about interoperability. Several sets of experiments were conducted for Type II IRM. The simulation results were compared between standalone and distributed simulation using the CSPE, as well as standalone simulation using Simul8, showing the correctness of proposed solutions. It was also observed that the DSManager designed for Type II IRM using the modified DMSO RTI1.3NG-V6 logical time can also be applied for Type I IRM, without introducing too much additional overhead.

Future work is necessary in this area. Synchronous entity passing leads to the situation of near zero lookahead, which is the main constraint to performance in applying conservative synchronization in distributed simulation. It is worthwhile to see how optimistic synchronization could improve the performance. By integrating a rollback controller [12] into the DSManger, the modelers and CSPs can be released from the burden of the complex rollback procedure. More work can also be done to investigate CSP interoperability for other types of IRMs. Each IRM type categorizes a particular problem and we hope the DSManager could provide a generic interface to the CSP for other types of IRMs.

References

[1] IEEE P 1516, "Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)", April 2002.

[2] CSPI-PDG, www.cspi-pdg.org, viewed 28th Oct., 2005.

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

[4] X.G. Wang, S.J. Turner, M.Y.H. Low and B.P. Gan, "A Generic Architecture for the Integration of COTS Packages with the HLA", *UK Operational Research Society Simulation Workshop*, Birmingham, UK, Mar. 23-24, 2004, pp. 224-233.

[5] X.G. Wang, S.J. Turner, S.J.E. Taylor, M.Y.H. Low and B.P. Gan, "A COTS Simulation Package Emulator (CSPE) for Investigating COTS Simulation Package Interoperability", *Proc. 2005 Winter Simulation Conference*, Florida, Dec. 4-7, 2005, pp. 402-411.

[6] S.J.E. Taylor, X.G. Wang, S.J. Turner and M.Y.H. Low, "Integrating Heterogeneous Distributed COTS Discrete-Event-Simulation Package: An Emerging Standards-Based Approach", *IEEE Transactions on Systems, Man and Cybernetics*, Jan. 2006, Vol. 36, No. 1, pp. 109-122.

[7] F. Wieland, "The Threshold of Event Simultaneity", *Transactions of the Society for Computer Simulation International*, 1999, Vol. 16, No. 1, pp. 23-31.

[8] R.M. Fujimoto, "Parallel and Distributed Simulation Systems", *Wiley Interscience*, January 2000.

[9] F. Wieland, "Parallel Simulation for Aviation Applications", *Proc. 1998 Winter Simulation Conference*, Washington DC, Dec. 13-16, 1998, pp. 1191-1198.

[10] Defense Modeling and Simulation Office (DMSO), "High Level Architecture RTI 1.3NG Programmer's Guide, Version 5", February 2002.

[11]. Simul8, www.simul8.com, viewed on 28th Oct., 2005.

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