Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

A FEDERATION OBJECT COORDINATOR FOR SIMULATION BASED CONTROL AND ANALYSIS

Seungyub Lee
Sreeram Ramakrishnan
Richard A. Wysk

Department of Industrial and Manufacturing Engineering
Pennsylvania State University
University Park, PA 16802, U.S.A.

ABSTRACT

This paper presents an architecture and a design for a Federation Object Coordinator (FOC) for simulation based control and analysis. This research focuses on developing a methodology for implementing a distributed simulation control mechanism which can be adopted to virtual manufacturing or virtual enterprises. In this method, distributed fast or real time simulation models interact with low level controllers and among themselves to actively control a system. The timing and coordination requirements of the simulation models to interact with the MRP systems and control systems as well as the interaction among the distributed simulation models are discussed in this paper.

1 INTRODUCTION

As the size and complexity of simulation systems increases, "conventional" sequential simulation fails to deliver the necessary performance (time and model fidelity become issues). In large interactive simulations, a distributed implementation can help decrease the simulation model and case representation issues. Distributed simulation have been used for numerous reasons: which include the faster execution of models, ability to better represent a system physically distributed rather than localized to a single machine or processor and increase in model fidelity. An architecture for distributed simulation-based control and analysis are discussed in this paper. In this modeling methodology, an FOC plays a critical role such as time management and coordination among federates (or simulation models).

2 SYNCHRONIZATION METHODS

2.1 Time Increment in Discrete Event Simulation

The two phases of a Discrete Event Simulation (DES) model's run - Entity Movement Phase (EMP) and the Clock Update Phase (CUP) has been discussed in Schriber and Brunner (2000). In the CUP, two principal timing methods are used: ***variable Δt method*** and ***fixed Δt method***. These methods have been discussed in Law and Kelton (2000). It needs to be noted that the performance of these methods also depends upon the level of interaction among the different models.

2.2 Synchronization Techniques

Methods to coordinate simulation models can be classified into synchronous and asynchronous methods. In the former, the required coordination is achieved using ***exact synchronous mechanism*** or ***rollback mechanism***. In the exact synchronous mechanism, models do not process current events simultaneously; only one model is active. The synchronization is accomplished by maintaining a "*master event calendar*" with the next event for each of the distributed components (Misra 1986, Boukerche and Tropper 1995). In asynchronous method, every simulation model runs independently, giving the potential for maximum parallelism (Ghosh and Lee 2000). It can be achieved using ***conservative*** or ***optimistic*** approaches (Chandy and Misra 1979, Jefferson 1985, Misra 1986, Lin and Fishwick 1996, Fujii et al. 1999).

3 OVERVIEW OF METHODOLOGY

Figure 1 outlines the framework of the proposed system. The overview of the system consists of an FOC which coordinates different simulation models, distributed simulations, controllers in units and the MRP/DB system. The control of the federation of simulations shown in Figure 1 is accomplished by using active status for the slowest running simulation while others are in inactive status. In this research, the FOC uses the ***exact synchronous variable Δt mechanism***. In this case, an FOC designated as a so-called "*master event calendar*" allocates inter-process events from this calendar to all simulation models. The FOC also resynchronizes all simulations at the end of every activity.

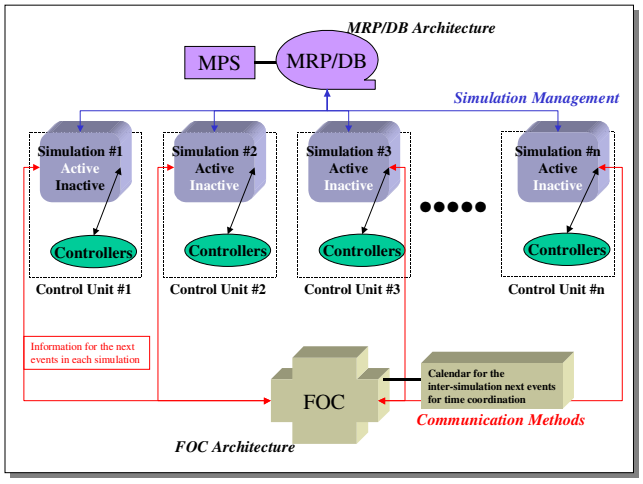

Figure 1: Overview of the Proposed System

With this mechanism, simulation models do not process current events at the same time; one model is active while the other models are inactive or delayed. An active model is a synchronized model that advances to the time of its next future event and completes the events on its current event list. The synchronization is accomplished by maintaining a "*master event calendar*" embedded in the FOC.

The two principal time increment methods for simulation federations are variable and fixed Δt for time advance. The two mechanisms can be implemented in the system along with an FOC; however, all distributed simulation models update their state at the same time by variable Δt since time increment in a simulation depends on time of the next event. On the contrary, if a fixed Δt method is used in the system, all models update their states at fixed time increment, and a small Δt results in a large number of update points to synchronize the simulation time to the global simulation time set by a simulation step size of an FOC.

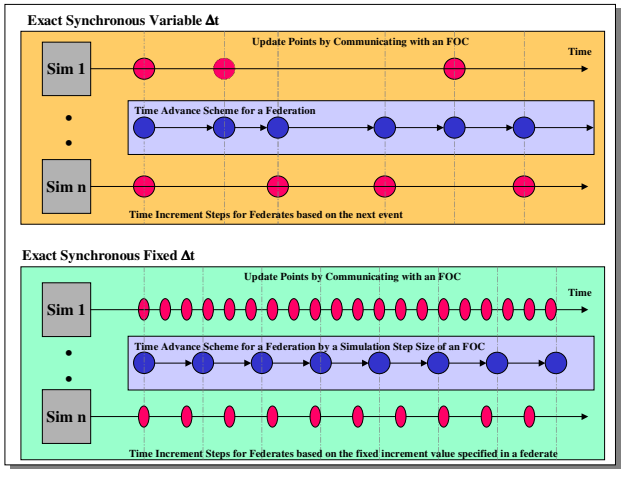

Figure 2: Time Updates for Two Synchronization Methods Applicable to the System

Figure 2 compares time increment steps and update points in the systems using these two methods. If large steps are taken, then constant step back occurs (moving backward in time).

The main objectives for this research are:

- An architecture for an FOC and software system to coordinate federates or simulations: Under the distributed simulation modeling environment, it is necessary to have an architecture of the system, which can govern and coordinate federates within the system. There are several alternatives to construct this mechanism. This mechanism can be categorized by its developer, platform or architecture, the synchronization technique or communication protocol. Possible alternatives are as follows: HLA/RTI, Common Object Request Broker Architecture (CORBA), Remote Method Invocation (RMI) and Distributed Manufacturing Simulation (DMS) (Buss and Jackson 1998, McLean and Riddick 2000, Allen and Garlan 1997).
- Management for distributed simulation models within the federation: The methodology for synchronization discussed here includes information about the time of the next event and the time for the inter-process events from a "master event calendar". It is necessary to develop a mechanism to determine the time of the events in simulation models and exchange the information via an FOC. Consequently, both a time management mechanism (to find out information regarding to time of the events using simulation's variables and functions relative to its calendar) and communication mechanism to exchange information via an FOC are required for a variable Δt FOC.
- An architecture and management for the DB/MRP system: Several possible software packages such as Microsoft Access, ERP packages (Oracle ERP systems or SAP R/3) or commercial large database systems such as DB2 RDBMs can be used to generate input data such as master production schedule (MPS), Bill of Material (BOM) and so on. In this research, details for DB/MRP architecture are not discussed.

4 DISTRIBUTED SIMULATIONS

In a distributed simulation-based control system, many simulation models can be introduced to analyze and control physical systems instead of a single simulation model. They act as decision makers for each control unit and perform inter-processing communication between them via an FOC. The interactions among the models, modeling architecture of distributed simulation models coordinated by an FOC have been discussed. For the implementation, models

in Arena 4.0™ is used to obtain MRP/DB information and made to interact with a master control object to execute tasks though a messaging system.

Distributed simulation models are first run in the fast mode to detect system failures or other performance problems as an analytical tool. Even though distributed simulation is applied to model relatively small manufacturing systems, the benefits to model and analyze the systems using distributed simulation can be more prominent for large applications such as remote virtual factories or supply chain management. First, fast-mode distributed simulations are run to detect system problems and check fulfillment of tasks while communicating each other via an FOC. After debugging and modifying the overall models to satisfy all the requirements, simulated system events can be progressively replaced by plugging in actual physical equipments to develop a fully integrated control system (Wu and Wysk 1989, Son et al. 1999).

Each distributed simulation model obtains MRP/DB information using an SQL connection to an interactive Open Database Connectivity (ODBC) compliant database system. Two simulation models (for example) run while exchanging messages using an Ethernet communication link to two master controllers in each control unit via two routers. In this case, two controllers perform the execution functions and keeps track of the status of each low level controller within each control unit. The master controllers and simulations exchange messages. Once the master controllers confirm completion of tasks within a unit, then it sends a similar message to the simulation, and the simulation knows that the current task was completed. This procedure of communication between each master controller and simulation model for controlling a unit is performed in parallel via a router while communication among simulation models is done separately via an FOC. The control architecture for a case of two distributed simulations appears in Figure 3.

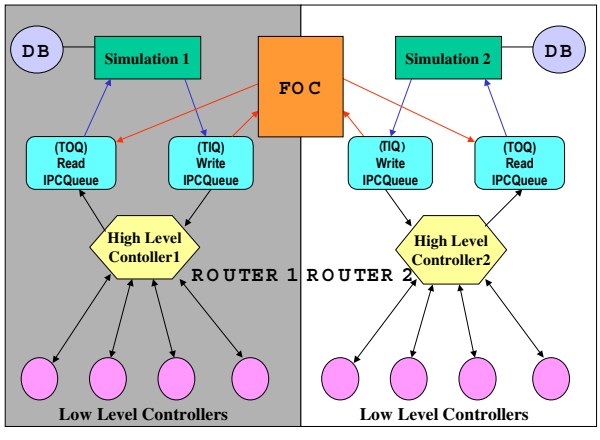

Figure 3: Control Architecture for Two Distributed Simulations

As soon as a simulation model starts, it firstly tries to make connections to a router to communicate with a master controller, an FOC to send and receive synchronization information and MRP/DB through ODBC. When an entity is created in the model, the entity invoke DB/MRP to obtain necessary information. Every entity created has attributes to store the information needed to process transactions and to store the information about time of the next event, time of the review event (physical halt for synchronization among distributed simulation models). The names and numbers for the attributes in the simulation are listed in Table 1. Arena simulation supports either Visual Basic Application (VBA) or DLL user-defined code for application integration. User-written code for database interaction for process plans and master production schedule, automatic control, communication with controllers, coordination with an and FOC can be linked using VBA or DLL files. Here, a DLL file generated using Microsoft Visual C++ files and header files provided by Systems Modeling Corporation is used. C++ file structure and functions of the DLL file are shown in Figure 4.

Table 1: Attributes Used in the Simulation Models

Attributes	Attribute name	Function
100 – 199	Transaction Parameters	Executing transactions in controller
200	Next Time	The time of next event in the current simulation
201	Review Time	The time of review event sent back from an FOC for synchronization purpose

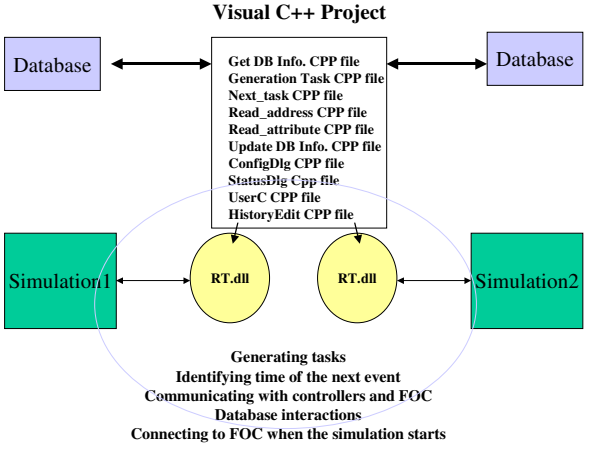

Figure 4: Architecture of the DLL File

5 A FEDERATION OBJECT COORDINATOR

5.1 Architecture of an FOC

The FOC discussed in this paper has been implemented in the form of a TCP/IP socket program. The primary purpose of the FOC is to facilitate the communication among the simulation models. While a router is used to exchange messages related to physical tasks between a simulation

and a system controller, the FOC exchanges a message related to time of the next event from each simulation model. In addition, the FOC executes and sorts the messages from simulation models in order to construct a master event calendar. A schematic representation is depicted in Figure 5.

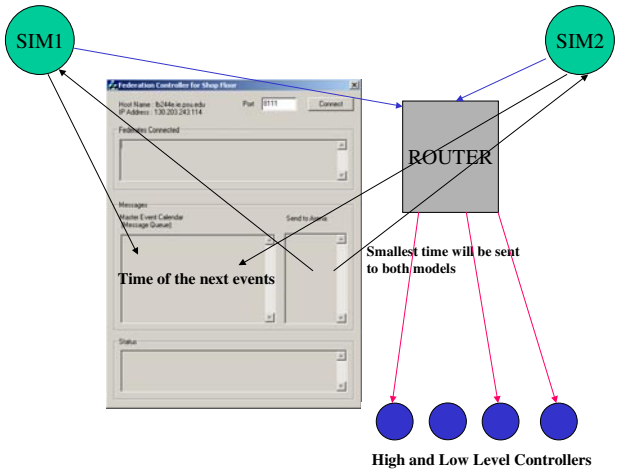

Figure 5: Communication between Simulation Models and an FOC and a Router for Two Simulations

In most methods, synchronization of the simulations is usually achieved by messaging (Lin and Fishwick 1996). In this architecture, seven key functions have been defined for the FOC. The `SocketListen( )` function is used to monitor any incoming signals which requires establishing a connection to a specific port. The connection of the simulation model to the FOC using the `OnConnect( )` function. The `AcceptSimulation( )` function in an FOC application verifies if an acceptable connection has been made with the relevant simulation models. The `ReceiveMessage( )` function waits for messages from simulation models connected to the FOC. Upon receiving messages (next event times) from all the connected models, the FOC sorts the event times to create a "master event calendar" (in increasing order of time of the next event after consolidating events from all simulations). This sorted result are stored in a buffer. A `SendMessage( )` function is invoked to send information regarding the next scheduled event (top ranked value in a buffer or master event calendar) to each simulation model. Finally, the `GetTnext( )` function handles message strings from each simulation model and deciphers them to help the FOC in selecting the next event.

5.2 Synchronization Scheme for an FOC

Exact synchronous variable Δt mechanism is used for this implementation since this approach is expected to faster than other approaches and is typically limited only by the computing or communication resources. Figure 6 describes basic procedures of the synchronization mechanism. As shown in Figure 6, time increment and update in each simulation depends on time of the next event. Therefore, it is necessary to develop a method to obtain information about time of the next event in each simulation for the proposed synchronization mechanism. When it is time to execute the next event, the top record is removed from the calendar and the information in this record is used to execute the appropriate logic. In addition, the current value of time in the simulation is simply held in a variable called the simulation clock and it is stored in Arena system variable "TNOW". During initialization of the simulation, and then after executing each event, the event calendar's top record is taken off the calendar. The simulation clock advances to the time of the next event, and the information in the removed event record is used to execute the event at that instant of simulated time (Kelton et al. 1998, Pegden et al. 1995).

Simulation Model 1		**Simulation Model 2**	
Time	**Event**	**Time**	**Event**
10	Event 1 Compare time of the next event	10	Waiting
	Waiting		Event 2 Compare time of the next event
		15	Event 3 Compare time of the next event
20	Event 4 Compare time of the next event		Waiting
25	Event 5 Compare time of the next event		
	Waiting	30	Event 6 Compare time of the next event
35	Waiting	35	Event 7 Compare time of the next event

Global Event Calendar

Figure 6: Synchronous Distributed Simulation

A simulation model should start in process mode environment and control of the entity is passed to "`cevent ( )`" user-coded C function when an entity arrives at the EVENT block. If a function to generate time of the next event should be developed in user coded C++ file, those EVENT blocks in process mode can be used to connect the process model to event scheduling mode. Figure 7 shows Arena block diagrams for sending time information to a FOC and delaying for a "dummy event".

Once the FOC gets the messages about the time of the next event, it returns the smallest value (of time of next events) to each simulation model. When a simulation gets the response from an FOC in a same delay block, it reads

and processes this message. After receiving messages from the FOC, the simulation restores the returned value into another attribute, creates a review event and updates the attribute again. The entity will proceed to the next delay block and it is delayed for "the smallest time among the next events – TNOW".

Event execution logic in event scheduling and process mode associated with time of the next event is depicted in Figure 8. Finally, a synchronization mechanism along with messages and creation of dummy events is proposed as ***an exact synchronous variable Δt mechanism with message packets***.

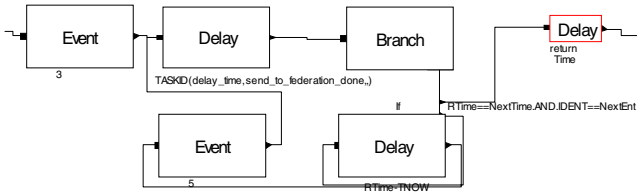

Figure 7: Arena Process Block Diagram

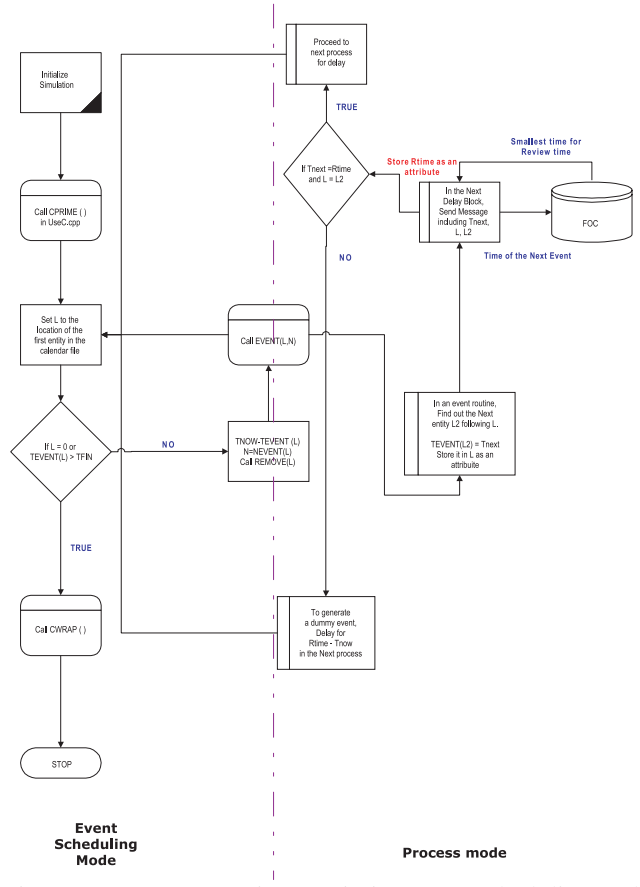

Figure 8: Event Execution Logic in Event Scheduling and Process Mode

Figure 9 describes a basic procedure of the proposed synchronization mechanism between an FOC and simulation models. Since synchronization mechanism will use inter-process communication for message parsing, message structure and executing messages can be critical issues for its performance.

While (Simulation is not over)

Wait until message queue (master event calendar) contains messages (= the number of simulation models) which include time of next event in each simulation

Remove and send smallest time (Ts) stamped message M from its calendar back to each simulation model.

In each simulation model
If
its simulation clock (TNOW) = Time stamp of M
Successfully process M and proceed to the next event
Else
Generate ***a review (Dummy) event*** and delay simulation time for Ts.
Send another time stamped message to master event calendar

Figure 9: Exact Synchronous Variable Δt Mechanism with Message Packets

6 COMMUNICATION ARCHITECTURE

6.1 Communicaion Scheme

Based on schemes for inter-process communication, the test-bed system can be implemented by TCP/IP socket program embedded in each object member. There are three elements in the inter-process communication in the system. Figure 10 represents communication architecture of the system, message flows and physical communication channels. From the above figure, simulation models, controller via a router and the FOC have similar communication processes to interact each other (Figure 11).

6.2 Messaging Scheme for Objects in the System

In the view of a simulation model as a task generator, it needs to send all necessary communication to a master controller. On the other hand, in the view of a simulation model as a federate which will coordinate with other federates via the FOC, all of messages to send out information of time from a federate to other federates is handled by the FOC. Since synchronization mechanism uses inter- process communication for message parsing, message structure and methods used for executing messages is critical point for system performance. The messages generated by a simulation model are defined in "Message" or "Task" elements in Arena 4.0 (Table 2).

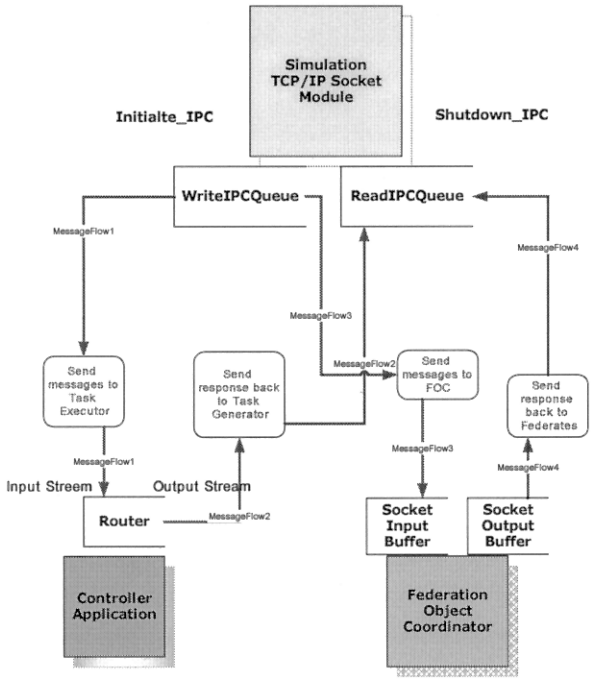

Figure 10: Communication Architecture

Communication Process in Simulations, FOC and Controllers

1. The Communication Socket is created (Initialization)
2. Listen and wait for a connection with the simulation models
3. Read the incoming messages from simulations in the socket's Input buffer
4. Parse the message into a format in order to be executed in the program
   (String find store a section of string found
   Execute functions using those intermediate strings)
5. Parse the application's response into a response message
6. Write the response message to the output buffer

Figure 11: Communication Procedure

Table 2: Message Schema in Simulation Models

Messages	Description	Parameters
Send_from_simulation_to_controller	Process the tasks or transactions in a specified class of objects in the system	IDENT(Current active entity), Object name, Task Action
Send_to_federation	Determine the smallest time of the next event among simulation models then send it back to models	IDENT, NextTime, TNOW

7 FUNCTIONAL ARCHITECTURE FOR THE PROPOSED SYSTEM

7.1 Information Flow among Object Members in the System

Information and message flow can be modeled in a sequence diagram, one of interaction diagram in Unified Modeling Language (UML) (Booch et al. 1999). Figure 12 represents a sequence diagram which contains for the proposed system. A related paper discusses the use of fast-mode models and real time models to control a system such a manufacturing shop floor or supply chain interactions. (Ramakrishnan and Wysk 2002). In addition, since fast-mode and real time mode simulation models share the same attributes and operations (Figure 13), they can be generalized into one simulation class. The five different classes in this architecture - simulation, an FOC, a controller, DB/MRP system, and a router is shown in Figure 13.

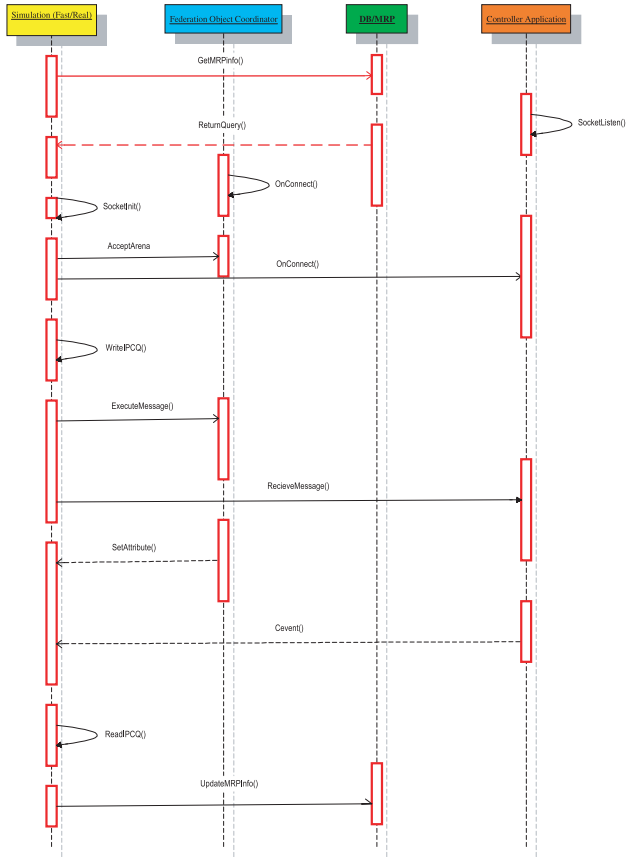

Figure 12: A "Sequence Diagram" for the Proposed System

8 IMPLEMENTATION ISSUES

The Arena 4.0™ simulation software used in the implementation is designed for building computer models that accurately represent real world applications. In order to exchange

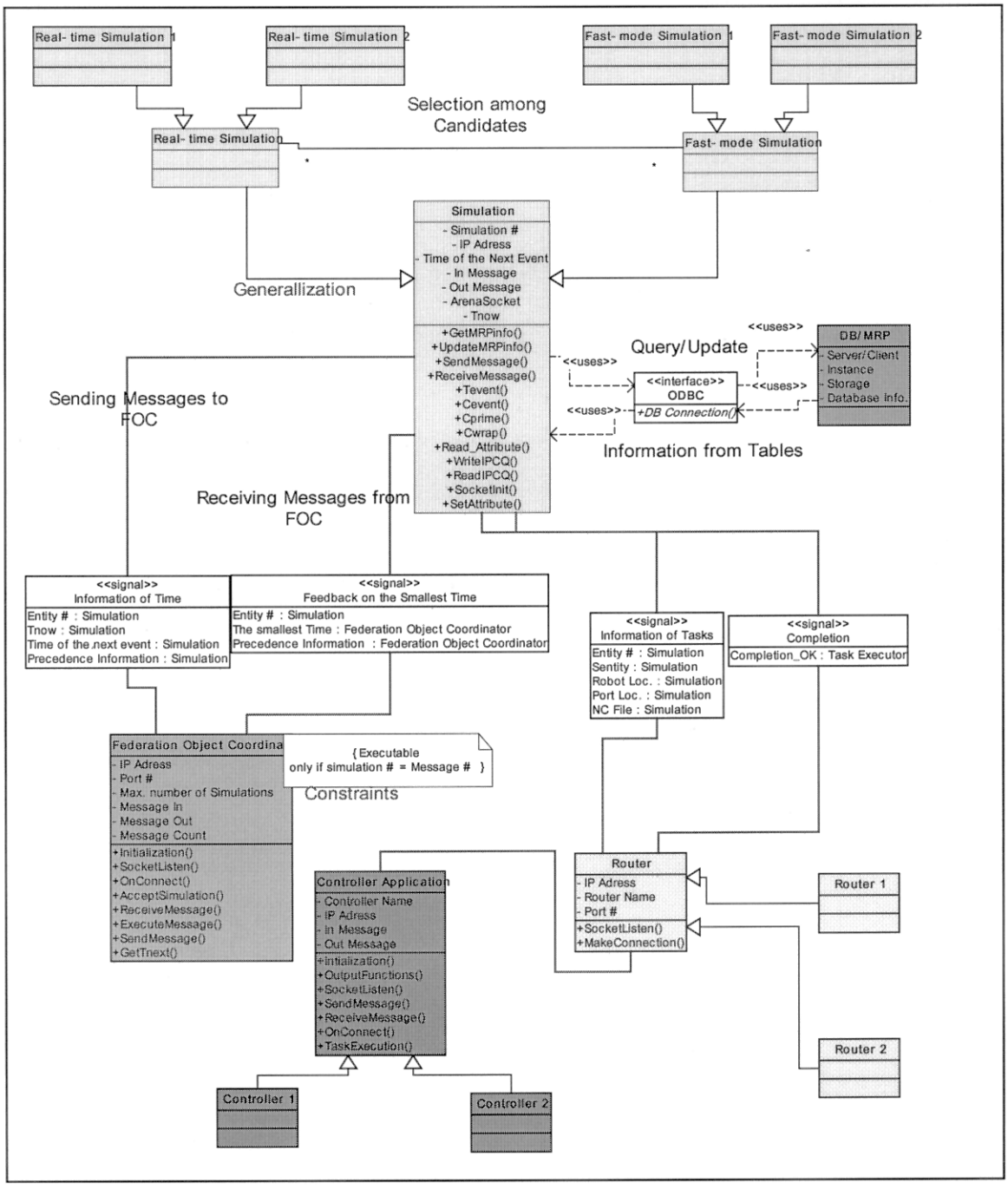

Figure 13: A "Class Diagram" for the Proposed System

messages among objects (simulation models, controllers via a router and an FOC in this case) in the system, it is required to have a real-time package which is capable of sending messages to third-party controller software.

In this research, Arena RT (real-time) package is used to analyze and control the entire system which is divided into two control units along with two high-level controllers as shown in Figure 3. Basically, the simulation model reads the order entries and a corresponding master production schedule from the IBM's DB2 database and sends the required messages to each high-level controller. After sending the message, the simulation model waits for the verification message from Big-E informing that the command has been executed properly and the system is ready for the next message. Consequently, Arena simulation sends a sequence of messages throughout the processing of a shop order which direct the part inside the physical system. Upon the completion of an order, the simulation model updates the order en-

try table in the database verifying that the order has been completed and ready for shipment. However, Arena does not support a command to halt the simulation. If such a command existed, it would be possible to implement synchronization of simulation time easily by simply halting one model. Moreover, two models cannot run simultaneously on a single computer. Therefore, in order to implement the system including several distributed simulation models, it is necessary to run distributed simulation models on physically distributed computers.

As mentioned earlier, the same number of high-level controllers as the number of distributed simulation models is required to control the system and execute tasks generated by each simulation model. In each distributed control unit, low-level controllers for physical equipments communicate with their supervisor. Consequently, the same number of routers with the number of high-level controllers is required in each computer to connect all objects such as a simulation, high-level controller and low-level controllers.

The speed and traffic of a network and performance of computers considerably affect overall system's performance since the proposed system depends on communication process and message parsing for synchronization among simulation models. Those factors should be also considered prior to implementation. Finally, it can be stated that the purpose of implementation using the proposed methodology is to show the effectiveness of the proposed system and its ability to make the overall system work.

9 CONCLUSIONS

In this paper, a methodology to synchronize distributed simulations has been presented. An architecture for implementing the methodology using Arena 4.0 as the example DES has also been discussed. The basic idea was to present the design and implementation of an FOC and a messaging scheme that synchronizes distributed simulations. This architecture can be flexibly applied to various systems such as shop floor control systems and supply chain management. The functional architecture depicted by object dependencies for the system and information flow among objects in the system using UML diagrams can aid the implementation in any DES software. The proposed architecture was also implemented in some test-bed systems. From the results, it was experienced that simulations with an FOC performed more efficiently and flexibly than fixed Δt method and implementation was done more easily than HLA/RTI or other architectures. The tests will be discussed in a later paper.

REFERENCES

Allen, R., and D. Garlan. 1997. Formal Modeling and Analysis of the HLA RTI. In *Proceedings of the 1997 Winter Simulation Conference Interoperability Workshop*, 1-9.

Booch, G., J. Rumbaugh, and I. Jacobson. 1999. *The Unified Modeling Language User Guide*. New York: Addison-Wesley.

Boukerche, A., and C. Tropper. 1995. SGTNE: Semi-Global Time of the Next Event Algorithm. In *Proceedings of the IEEE*, 68-77.

Buss, A., and L. Jackson. 1998. Distributed Simulation Modeling: A Comparison of HLA, CORBA, and RMI. In *Proceedings of the 1998 Winter Simulation Conference*, 819-825.

Chandy, K. M., and J. Misra. 1979. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering* 5 (5): 440-452.

Fujii, S., A. Ogita, Y. Kidani, and T. Kaihara. 1999. Synchronization Mechanisms for Integration of Distributed Manufacturing Simulation Systems. *Simulation* 72 (3): 187-197.

Ghosh, S., and T. S. Lee. 2000. *Modeling and Asynchronous Distributed Simulation: Analyzing Complex Systems*. New York: IEEE Press.

Jefferson, D. 1985. Virtual Time. *ACM Transactions on Programming Languages* 7 (3): 403-425.

Kelton, D. W., R. P. Sadowski, and D. A. Sadowski. 1998. *Simulation with ARENA*. Boston, Massachusetts: McGraw Hill.

Law, A. M., and W. D. Kelton. 2000. *Simulation Modeling and Analysis*. 3d ed. New York, NY: McGraw-Hill.

Lin, Y., and P. A. Fishwick. 1996. Asynchronous Parallel Discrete Event Simulation. *IEEE Transcations on Systems, Man and Cybernetics-Part A: Systems and Humans* 26 (4): 397-412.

McLean, C., and F. Riddick. 2000. The IMS Mission Architecture for Distributed Manufacturing Simulation. In *proceedings of the 2000 Winter Simulation Conference*, 1539-1548.

Misra, J. 1986. Distributed Discrete-Event Simulation. *Computing Surveys* 18 (1): 39-65.

Pegden, D. C., R. E. Shannon, and R. P. Sadowski. 1995. *Introduction to Simulation using SIMAN*, 2d ed. New York, NY: McGraw Hill.

Ramakrishnan, S., and R. A. Wysk. 2002. A Real Time Simulation-based Control Architecture for Supply chain Interactions, Industrial Engineering Research Conference (on CD-ROM) Available at <http://fie.engrng.pitt.edu/iie2002/proceedings/ierc/papers/2030.pdf> [accessed July 18, 2002]

Schriber, T. J., and D. T. Brunner. 2000. Inside Discrete-Event Simulation Software: How It Works and Why It Matters. In *Proceedings of the 2000 Winter Simulation Conference*, 90-100.

Son, Y. 2000. *A Simulation-based Shop Floor Control*. Ph.D. Dissertation. Department of Industrial and Manufacturing Engineering, Pennsylvania State University.

Son, Y., H. Rodriguez-Rivera, and R. A. Wysk. 1999. A Multi-pass Simulation-based, Real-time Scheduling and Shop Floor Control System. *Transactions, The quarterly Journal of the Society for Computer Simulation International* 16 (4): 159-172.

Wu, S. D., and R. A. Wysk. 1989. An application of discrete-event simulation to on-line control and scheduling in flexible manufacturing. *International Journal of Production Research* 27 (9): 1603-1623.

AUTHOR BIOGRAPHIES

SEUNGYUB LEE is a doctoral candidate in the Department of Industrial and Manufacturing Engineering at Pennsylvania State University, University Park. He received his B.S. degree in Industrial Systems Engineering from Yonsei University, Seoul, Korea and his M.S. in Industrial and Manufacturing Engineering from Pennsylvania State University. His research interests are distributed simulation, simulation-based control, web-based technology in virtual environments and computer integrated manufacturing. His email address is <sxl287@psu.edu>.

SREERAM RAMAKRISHNAN is a doctoral candidate in the Department of Industrial and Manufacturing Engineering at Pennsylvania State University, University Park. He received his B.Tech (Mechanical) from College of Engineering, Trivandrum, India and his M.S. (Industrial Engineering) from S.U.N.Y., Binghamton where he won the department Award for Academic Excellence. Upon graduation, he will be an Assistant Professor in the Engineering Management department at University of Missouri – Rolla. His research interests are simulation-based control and supply chain management. His email address is <sxr270@psu.edu>.

RICHARD A. WYSK is the Leonhard Chair in Engineering and a Professor of Industrial Engineering at Pennsylvania State University, University Park. Dr. Wysk has coauthored six books including *Computer-Aided Manufacturing*, with T.C. Chang and H.P. Wang -- the 1991 IIE Book of the Year and the 1991 SME Eugene Merchant Book of the Year. He has also published more than a hundred and fifty technical papers in the open-literature in journals including the *Transactions of ASME*, the *Transactions of IEEE* and the *IIE Transactions*. He is an Associate Editor and/or a member of the Editorial Board for five technical journals. Dr. Wysk is an IIE Fellow, an SME Fellow, a member of Sigma Xi, and a member of Alpha Pi Mu and Tau Beta Pi. He is the recipient of the IIE Region III Award for Excellence, the SME Outstanding Young Manufacturing Engineer Award and the IIE David F. Baker Distinguished Research Award. He has held engineering positions with General Electric and Caterpillar Tractor Company. He received his Ph.D. from Purdue University in 1977. He has also served on the faculties of Virginia Polytechnic Institute and State University and Texas A&M University where he held the Royce Wisenbaker Chair in Innovation. His email and web addresses are rwysk@psu.edu and <http://www.engr.psu.edu/cim/wysk.html>