Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

DISTRIBUTED SIMULATION: AN ENABLING TECHNOLOGY FOR THE EVALUATION OF VIRTUAL ENTERPRISES

Jayendran Venkateswaran
Mohammed Yaseen Kalachikan Jafferali
Young-Jun Son

Department of Systems and Industrial Engineering
The University of Arizona
Tucson, AZ 85721-0020, U.S.A.

ABSTRACT

This paper presents an application distributed simulation to the evaluation of virtual enterprises. Each company or candidate can use a simulation of its facilities to determine if it has the capability to perform its individual function in the virtual enterprise. Then, these simulations can be integrated into a distributed simulation of the complete enterprise, and used to predict the viability and profitability of the proposed product collaboration. In this paper, a prototype distributed simulation for such a purpose is presented. First, information flows as well as material flows among members in a virtual enterprise are identified using IDEF∅, a formal function modeling method. Sequences of the identified functions are then presented using the finite state automata formalism. These interactions are then implemented for a commercial simulation package. Finally, a distributed simulation composed of three individual simulations is successfully tested across platforms over both the internet and the local area network.

1 INTRODUCTION

Today's manufacturing industries face the challenge of responding more rapidly and efficiently to the changing markets driven by customized products. The agile manufacturing paradigm has been proposed to solve this problem. Agile manufacturing is a technology that allows a firm to achieve flexibility and rapid responsiveness to the changing market and customers needs, by enabling the firm to quickly respond to customers' requirements and design, prototype, manufacture, test and deliver a high-quality product to the market in the least time possible (Cheng et al., 1998). In this paradigm, manufacturers must emphasize not only quality, productivity and reduced cost, but also the ability to react quickly and effectively to changes in markets, production technology, and computer and information technology.

One way in which manufacturing industries can take advantage of their agility is to form virtual enterprises. Virtual enterprises are ephemeral organizations in which several companies collaborate to produce a single product or product line. Participating in virtual enterprises allows an agile company to use its knowledge, resources, and particular manufacturing expertise to take advantage of business opportunities that are on a larger scale than the company could handle alone. Here, the knowledge and expertise may include business and engineering activities throughout the product's life cycle, such as product design, process planning, production costing, scheduling of shop activities, shop floor control, quality control, sales, marketing, resource maintenance, product disposal, etc. This virtual enterprise is accomplished without making a long-term commitment to the other partners of the virtual enterprise or to the new business area.

The scope of the virtual enterprise discussed in this paper is relatively narrow in terms of the business and engineering activities considered for the collaboration. The virtual enterprise in this paper is somewhat similar to a supply chain system, synchronizing business processes in each member company by using shared information within and among firms. The specification of a virtual enterprise includes physical transactions of manufacturing and the transportation system, as well as informational transactions of planning, data processing, manufacturing management transactions, and negotiation among member firms in the virtual enterprise. Such enterprise integration requires sophisticated process specifications for business activities and a well-defined information communication structure.

To facilitate the creation of virtual enterprises, potential partners must be able to quickly evaluate whether it will be profitable for them to participate in a proposed enterprise. Simulation technology in general, and distributed simulation technology in particular, can facilitate the evaluation process. Each partner can use a simulation of its facilities to determine if it has the capability to perform its individual function in the virtual enterprise. Then, these simulations can be integrated into a distributed simulation of the complete enterprise, and used to predict the viability and profitability of the proposed product collaboration. The use of distributed simulation technology allows each potential partner to hide any proprietary information in the implementation of the individual simulation, but still to provide enough information to evaluate the virtual enterprise as a whole.

2 SCENARIO OF A VIRTUAL ENTERPRISE

This section presents a prototype virtual enterprise for manufacturing a few final products. The configurations for different virtual enterprises, depending on production requirements and the characteristics of potential collaborating companies. The prototype virtual enterprise considered in this paper is composed of three different players: two component suppliers and one final assembly plant (see Figure 1). The prototype shown in Figure 1 is a simplified version of the full model that is being developed by the authors. In addition to component suppliers and assembly plants, the full model includes a head-quarters, warehouses, distributors, retailers, and transportation systems.

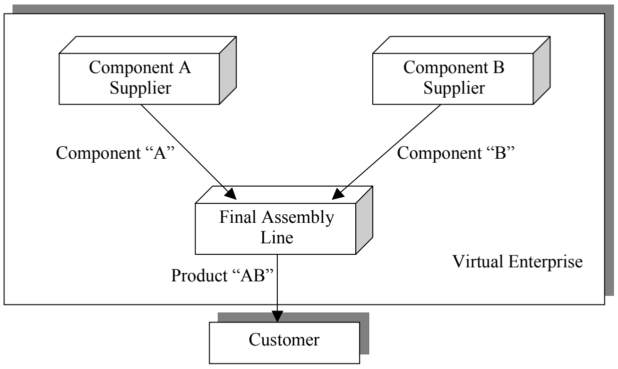

Figure 1: Prototype Virtual Enterprise

The enterprise shown in Figure 1 produces one type of product, which is made up of two different components. The components are assembled at the assembly plant. There are two suppliers supplying the two different components to the assembly plant. This virtual enterprise is a pull system. The assembly line maintains a buffer stock of both of the components. When either of the components falls below the prescribed threshold level, a purchase order is issued to that supplier. The supplier, on receiving the order, releases the required quantity to the assembly line. The suppliers continuously maintain a minimum level of stock. This is to ensure that the assembly line always gets its requirements immediately. The interactions between the assembly line and the suppliers are initiated by the assembly line only. A 'handshake' agreement is performed to open and close a transaction. More detailed interactions among the components will be described in Sections 3 and 4.

3 IDEF∅ FUNCTIONAL MODELING

Figure 1 showed the members of the virtual enterprise and the material flows among them. In this section, information flows as well as material flows among virtual enterprise members are identified using the IDEF∅, a formal function modeling method. The IDEF∅ method has been used for modeling the functions of an organization or a system and the relationships between those functions (Mayer 1992). The function of the prototype virtual enterprise system is represented in Figure 2. Two external components interacting with the virtual enterprise are raw material suppliers and customers (or other companies).

The function of a virtual enterprise shown in Figure 2 is composed of two sub-functions as shown in Figure 3. The function A1, final assembly plant, interacts both internally with the members of the virtual enterprise and externally with other parties that do not belong to the virtual enterprise. The function A2, component suppliers, interacts only internally with the final assembly plant.

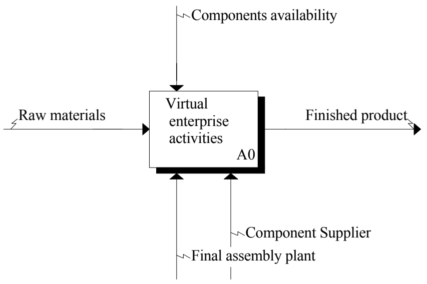

Figure 2: IDEF∅ diagram of a virtual enterprise

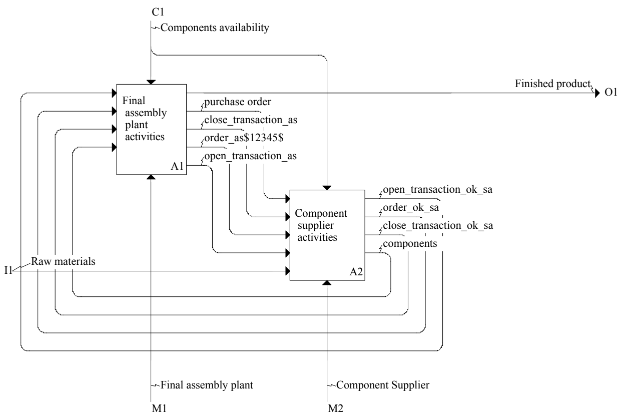

Figure 3: IDEF∅ Function Model of part of "Virtual Enterprise Activities"

The IDEF∅ models in this section illustrated the functional interactions among components. The sequence of these interactions will be explained in the next section using the finite state automata formalism.

4 MODELING OF BEHAVIORS AMONG MEMBERS USING FINITE STATE AUTOMATA

The coordination required between components suppliers and a final assembly plant has been modeled using the Deterministic Finite State Automata (DFSA) (Hopcroft and Ullman, 1979). The DFSA for the final assembly plant and the component suppliers are shown in Figures 4 and 5, respectively. The circles with numbers indicate the states or nodes. The arrows indicate actions that on completion will allow the system to proceed to the next state. The "I", "O", and "T" in Figures 4 and 5 denote incoming messages, outgoing messages, and tasks carried out, respectively. The "Ob" in Figure 4 denotes an observation to be performed. In addition, messages ending with "as" denote messages from the assembly plant to a component supplier. Similarly, messages ending with "sa" denote messages from a component supplier to the assembly plant.

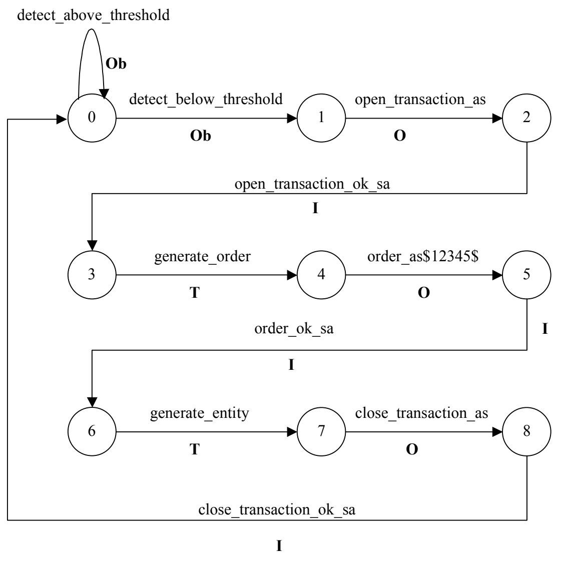

Figure 4: Finite state automata graph for assembly plant

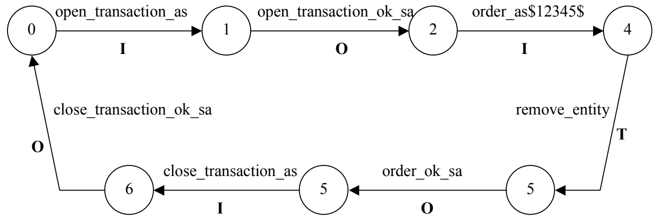

Figure 5: Finite state automata graph for supplier

Initially, both the component suppliers and the final assembly plant are at the zero state (node). In this state, the final assembly plant observes or checks the quantity of components available, every given time period. If the number is above the prescribed threshold (*detect_above_threshold*), the assembly plant remains at the same state. If the number of components falls below the threshold (*detect_below_threshold*), it moves to the next state. Once it has reached state 1, it will not check the quantity of components again until the entire transaction is completed and it returns to the zero state. The final assembly plant initiates the transaction between it and the supplier by sending the message *open_transaction_as*. It then waits for the response, *open_transaction_ok_sa*. Upon receiving this message it generates a purchase order specifying the component details and supplier details. The component details include the component ID, component name, quantity required, price, expected delivery date, etc. The supplier details include the supplier ID, supplier name, supplier address, etc. However, the current prototype developed in this paper uses only the following information:

- Component details: component ID, component name, and quantity required.
- Supplier details: supplier name.

This is represented by the task *generate_order*. After the successful generation of the purchase order, the final assembly plant sends the message *order_as$12345$*. The number enclosed by the $ signs is the purchase order ID. The supplier, on receiving the message, seizes the purchase order according to the purchase order ID given in the message. It then removes the requested quantity of components from its buffer, represented by the task, *remove_entity*. It then sends back the message *order_ok_sa* to the final assembly plant. On receipt of the message, the assembly plant generates the required quantity of components by the task *generate_entity*. The tasks *remove_entity* and *generate_entity* simulate the transportation of components from one place to another. Note that the automata graph needs to be accordingly changed after transportation systems are added to the virtual enterprise. After the generation of components, the assembly plant closes the transaction by sending the message *close_transaction_as* to the suppliers. The response *close_transaction_ok_sa* returns the final assembly and the suppliers to the initial state. Note that the supplier then remains in its initial state until it receives the initial message from the final assembly plant. Table 1 summarizes the messages and their meanings.

The final assembly plant maintains a different finite state automata graph for each supplier. The messages are differentiated by adding the suffix "#1" or "#2", the numbers corresponding to suppliers.

Table 1: Messages in the finite state automata graph and their meanings

Message	Meaning
detect_above_threshold	Message associated with the fact that the number of components at the final assembly plant is above the threshold.
detect_below_threshold	Message associated with the fact that the number of components at the final assembly plant is below the threshold.
open_transaction_as	Message sent from assembly to supplier to initiate a transaction between them.
open_transaction_ok_sa	Message sent from supplier to assembly to show that the supplier is ready for a transaction with the assembly plant.
generate_order	Message associated with a task done by assembly. A purchase order is created for the required quantity of components.
order_as$12345$	Message sent from assembly to supplier after the successful generation of the purchase order. The order number is enclosed within the $ signs.
remove_entity	Message associated with a task done by supplier. The quantity of components requested by the assembly plant is removed from storage.
order_ok_sa	Message sent from supplier to assembly after the successful completion of the *remove_entity* task.
generate_entity	Message associated with a task done by assembly. It generates the required number of components.
close_transaction_as	Message sent from assembly to supplier to close the transaction.
close_transaction_ok_sa	Message sent from supplier to assembly confirming the closure of the current transaction.

5 DISTRIBUTED SIMULATION

This section presents background for distributed simulation. The Department of Defense's High Level Architecture (HLA) (Kuhl et al. 1999) for modeling and simulation can certainly be regarded as the state of the art in distributed simulation. The HLA establishes a common high-level simulation architecture to facilitate the interoperability of all types of models and simulations. The Run-Time Infrastructure (RTI) software implements the specification and represents one of the most tangible products of the HLA. It provides services in a manner that is comparable to the way a distributed operating system provides services to applications.

An HLA-based simulation is called a *federation* (Kuhl et al. 1999). Each simulator that is integrated by the HLA RTI is called a *federate* (Kuhl et al. 1999). One common data definition is created for domain data that is shared across the entire federation. It is called the *federation object model* (FOM) (Kuhl et al. 1999). Note that the simulation models can be legacy simulation systems implemented in different languages. The direct interaction of the simulation federates with the Runtime Infrastructure is quite complex and cumbersome. An interface called Distributed Manufacturing Simulation (DMS) Adapter has been developed by the National Institute of Standards and Technology (NIST) to provide mechanisms for distributed simulation similar to those provided by the HLA RTI, but with a level of complexity that is manageable by the development resources available in the manufacturing community (McLean and Riddick 2000).

6 IMPLEMENTATION

Simulations for two component suppliers and a final assembly plant have been implemented using Arena™ 4.0, and they have been integrated into a distributed simulation using the HLA and the DMS adapter. The implementation is demonstrated in this section.

6.1 Assumptions

There are two component suppliers denoted by "suppliera" and "supplierb". The final assembly plant is denoted by "assembly". Assumptions and characteristics made for the demonstration are as follows:

- The assembly plant has some initial quantity of both component A and component B. It checks every given time interval to see if the component level goes below the prescribed threshold level. If it goes below, the transaction takes place as illustrated earlier using the finite state automata graph. Communications between the component suppliers and final assembly plant are performed through message exchange.
- The federation time is advanced or incremented every time interval denoted by the '*SimulationStepSize*'. The '*SimulationStepSize*' is a property available in the DMS Adapter. For details on

more methods and properties that are available, refer to the *DMS Adapter Reference Guide* (Riddick 2001).

- The component suppliers wait for the assembly to initiate the transaction. They maintain a minimum level of stock to ensure that the assembly plant gets its requirements immediately. In addition, enough raw materials are assumed to be available for the suppliers. In the final assembly plant, one unit each of component A and component B is used to produce a finished product denoted product AB. As soon as these products are produced, they are put into storage.

6.2 Modeling Using Arena™ 4.0

The above model has been implemented using Arena™ 4.0. The implemented modules are generic, and therefore the same modules have been used for component suppliers and the final assembly plant, with minor customizations. The simulation model can be broadly classified into two parts: 1) the time management part and 2) the actual model.

Arena modules needed for the time management part of the model are shown in Figure 6. Even though Arena modules are used in this presentation, exactly the same concepts can be used when implementing models with other discrete event simulation packages. As shown in Figure 6, one entity is created at zero time. It invokes the Visual Basic code contained in the VBA block, and delays for an amount of time ("a_time") determined from the Visual Basic code. This entity continues this procedure until the simulation is terminated. The pseudo code contained in the VBA block is shown in Figure 7. The first "if" condition checks whether the time of the local simulation is behind the current time of the global distributed simulation. If this gap is larger than the simulation step size (S_i), then it advances the local simulation by S_i. If the gap is smaller than S_i, then it advances the local simulation by the amount of the gap. In the latter case, the local simulation time becomes equal to the global distributed simulation time. Note that time advancement in the local simulation is performed by specifying "a_time" value and delaying the simulation for "a_time" amount of time. When necessary, the VBA block halts the local simulation until the simulation advance request has been completed. In other words, the local simulation needs to wait physically until all of the other legacy simulations within the same federation catch up to the current time of the global distributed simulation.

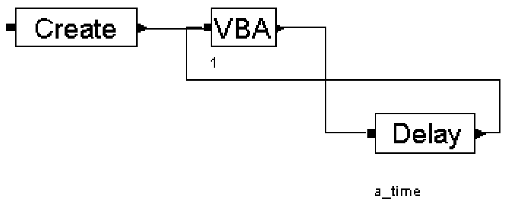

Figure 6: Time management blocks in Arena™ 4.0

```
C = current time in distributed simulation
Tnow = current time in local simulation

If Tnow <= C And (simulation advance has been completed) Then
   If (this is the first time after Tnow = C) Then
      Tell the RTI that I want to move forward
   End If

   If (C - Tnow) > S Then
      a_time = s
   Else
      a_time = ) = (C - Tnow)
   End If
Else
   While (simulation advance has not been completed)
      'do nothing -- physical halt
   Wend
End If
```

Figure 7: Pseudo code contained in VBA block in Figure 6

As discussed in Section 4, the coordination needed between the two component suppliers and the final assembly plant has been performed using the deterministic finite state automata (DFSA). The DFSA has been implemented in Arena™ 4.0, using global variable and arrays. The messages to be sent and the responses to be received are stored in a global (public) array structure. The pseudo code handling the message transactions is also contained in the VBA block on Figure 6. The pseudo code for the DFSA graph-based simulation is shown in Figure 8. This is the generic pseudo code required to handle any interaction between different companies, not only the interactions in the prototype discussed here.

```
Initialize to state 0 of DFSA graph

Loop
        Perform the next action to be done
        Proceed to the next state

End loop when simulation stops
```

Figure 8: Generic pseudo code for the DFSA graph-based simulation

The pseudo code for the DFSA graph-based simulation will vary, depending on the company's respective DFSA graph. The pseudo code for the supplier is shown in Figure 9.

There are two arrays and two counters declared globally. The array "theMessages" contains the messages to be received and the array "theResponses" contains the messages to be sent as responses. The counter "message_to_be_got" is associated with the array "theMessage". The counter "message_to_be_sent" is associated with the array "theResponses". The counters can take the values 0, 1 or 2 corresponding to each element of the array. The counters are used to indicate the current state of the simulation. For example, if the value of "message_to_be_got" is 1 and the value of "message_to_be_sent" is 0, then the model is in State 1 of the DFSA graph (see Figure 5). The "while" loop will be entered if there are any unread messages for the supplier model. When the next message is received, it will be checked in the first "if" loop to see whether it is the message that the simulation is waiting for. If it is the correct message, then it proceeds into the loop. It then sends the corresponding response and increments the counters. If the message *order_as$12345$*, is received, then the model gets the purchase order of the ID 12345. It then checks the quantity required by the assembly from the order. The requested quantity is then removed from its storage (task *remove_entity* of the DFSA in Figure 5). The "while" loop is executed until the last message is read.

```
theMessages=Array[open_transaction_as, order_as, close_transaction_as]
theResponses=Array[open_transaction_ok_sa, order_ok_sa, close_transaction_ok_sa]
counter message_to_be_got=0
counter message_to_be_sent=0

While (All message received is False)
        theNextMessage = Receive the next message

        If ( theNextMessage = theMessages(message_to_be_got) ) Then

                If (theNextMessage=order_as$12345$) Then
                        Get the Purchase order of the ID "12345"
                        Delete the specified quantities of the components
                End if

                Send the reply (theResponses(message_to_be_sent))
                Increment counter message_to_be_got by 1
                Increment counter message_to_be_sent by 1

                If (the counter value =3) Then
                        Reset counters =0
                End if
        End If
Wend
```

Figure 9: Pseudo code for the DFSA graph-based simulation for a supplier

The above pseudo code for the supplier showed in detail the exact implementation for the model. Since the pseudo code for the assembly plant is similar to the code for the supplier, it is not presented in this paper. Messages used The different types of actions that take place (Input, Output, Task and Observation) and the way they are handled are summarized in Table 2. If the desired action does not take place, it will only prevent the system from going on to the next state; it will not stop the simulation. Note that the 'system' in Table 2 refers to a single DFSA graph, either a supplier or an assembly plant. Note also that the messages that are sent across the simulations are exactly the same as those shown in the DFSA graph.

Table 2: Actions associated with the DFSA

Actions	Meaning	What happens	Mechanism
Input **(I)**	Get an input message to proceed to the next state	The system waits for the correct input message from the other system.	The system compares the messages received with the messages that it is waiting for. If they match, it proceeds to the next state.
Output **(O)**	Send an output message to proceed to the next state	The system sends the required output message	The output message that needs to be sent is sent and the system moves to the next state.
Task **(T)**	Perform a task to proceed to the next state	The system performs a task.	The task to be done is performed by the system. Example: the assembly performs tasks *generate_order* and *generate_entity*.
Observation **(Ob)**	Make a check to proceed to the next state	The system makes a check. The next state the system proceeds to depends upon the result of the check.	The assembly alone does this action. It checks the quantity of components left in order to find out whether a transaction is required.

6.3 Testing

The entire federation described so far has been successfully tested across platforms and versions of Arena™. Factors used in the experiment are as follows:

- Operating systems
  - Windows 98, Windows NT, and Windows 2000.
- Network environments
  - Internet and local arena network (LAN).
- Simulation packages
  - Arena™ 3.0 and Arena™ 4.0.

Even when the distributed simulation was conducted over the internet, no significant delay was noticed. Applying the proposed method in this paper to a more general virtual enterprise with more complicated interactions is left as future research.

A snapshot animation of the assembly plant is shown in Figure 10. The animation provide information that allows a user to examine the current status of the final assembly plant.

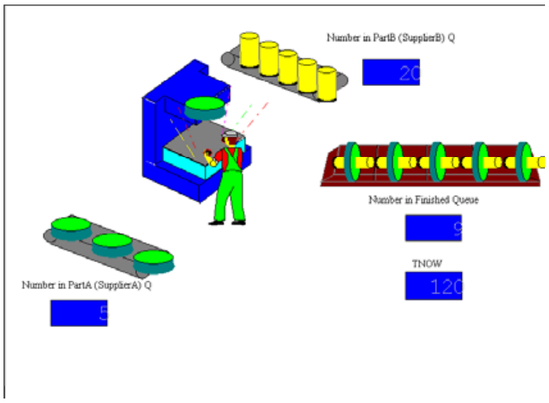

Figure 10: Animation of the final assembly plant

7 CONCLUSIONS

In this paper, a prototype for a distributed simulation that could be used to evaluate the viability of a virtual enterprise was presented. First, information flows as well as material flows among three members in a virtual enterprise were identified using the IDEF∅. Second, mechanisms have been described to govern time management and communications among member simulations in the distributed simulation. Finally, these mechanisms have been implemented for a commercial simulation package, and a sample virtual enterprise has been demonstrated. Based on the experience gained in the development of this paper, distributed simulation over the internet environment seems to be a promising technology for the evaluation of virtual enterprises.

ACKNOWLEDGMENT

This work was supported by the National Institute of Standards and Technology.

REFERENCES

Cheng, K., D. K. Harrison, and P. Y. Pan 1998. Implementation of agile manufacturing – an AI and Internet based approach. *Journal of Materials Processing Technology*, Vol. 76, pp. 96 - 101.

Hopcroft, J. E. and J. D. Ullman 1979. *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Publishing, Reading, MA.

Kuhl, F., R. Weatherly, and J. Dahmann 1999. *Creating Computer Simulations: An Introduction to the High Level Architecture*, Prentice-Hall, Upper Saddle River, NJ.

Mayer, R. J. 1992. *IDEF∅ Function Modeling Method Report*, Knowledge Based Systems Inc., College Station, TX.

McLean, C. and F. Riddick 2000. The IMS Mission architecture for distributed manufacturing simulation. In *Proceedings of the 2000 Winter Simulation Conference*, *Orlando,* FL.

Riddick, F. 2001. *The Distributed Manufacturing Simulation Adapter Reference Guide (Draft)*. National Institute of Standards and Technology.

AUTHOR BIOGRAPHIES

JAYENDRAN VENKATESWARAN is a graduate student and a teaching assistant in the Department of Systems and Industrial Engineering at The University of Arizona. His email and web addresses are `<jayendran_v@sie.arizona.edu>` and `<www.sie.arizona.edu/~jayendran_v/>`.

MOHAMMED YASEEN KALACHIKAN JAFFERALI is a graduate student and a research assistant in the Department of Systems and Industrial Engineering at The University of Arizona. His email address is `<kalachik@email.arizona.edu>`.

DR. YOUNG JUN SON is an assistant professor in the Department of Systems and Industrial Engineering at The University of Arizona. Dr. Son received his BS degree in Industrial Engineering with honors from POSTECH in Korea in 1996 and his MS and Ph.D. degrees in Industrial and Manufacturing Engineering from Penn State University in 1998 and 2000, respectively. His research interests include computer integrated manufacturing, simulation based shop floor control, distributed simulation, virtual manufacturing, and virtual enterprises. Dr. Son was the Rotary International Multi-Year Ambassadorial Scholar in 1996, the Council of Logistics Management Scholar in 1997, and the recipient of the Graham Endowed Fellowship for Engineering at Penn State University in 1999. He is an associate editor of the International Journal of Modeling and Simulation and a professional member of ASME, IEEE, IIE, INFORMS, and SME. His email and web addresses are `<son@sie.arizona.edu>` and `<www.sie.arizona.edu\faculty\son>`.