Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

DESIGN AND IMPLEMENTATION OF A HIGHLY REUSABLE MODELING AND SIMULATION FRAMEWORK FOR DISCRETE PART MANUFACTURING SYSTEMS

Hemant C. Bhuskute
Manoj N. Duse
Jagannath T. Gharpure
David B. Pratt
Manjunath Kamath
Joe H. Mize

Center for Computer Integrated Manufacturing
Oklahoma State University
Stillwater, Oklahoma 74078, U.S.A.

ABSTRACT

This paper describes ongoing research in the area of "Advanced Modeling Methodologies for Simulation of Complex Discrete Part Manufacturing Systems" in the Center for Computer Integrated Manufacturing at Oklahoma State University. We focus on presenting the design and implementation details of a modeling and simulation environment under development. First, we briefly describe the concept of model reusability and then show how the concept of separation of physical, information, and control elements of a model can lead to model reusability. Next, we outline the structure of our modeling and simulation environment and discuss the implementation of this structure in Smalltalk-80.

1 INTRODUCTION

Most simulation languages available today are well suited for developing single-use models of small to moderate sized systems. These languages support the analyst in model building and validation tasks such as identifying system components, abstraction of these components in the standard constructs, building the system model, carrying out simulation experiments, collecting results, and performing statistical analysis. The analyst generally concentrates on the model building activities of a specific scenario. Generally, the assessment of different alternatives is done in terms of changes in system parameters such as service patterns, arrival patterns, and failure rates, and their impact on system performance. These kinds of alternatives are normally incorporated into the basic model through small embellishments. Major changes in the system model such as changes in the structure of the system and their impact on the system performance cannot be easily handled. Thus, the traditional languages facilitate model reusability only to a limited degree. In this paper, we present an implementation of a highly reusable modeling and simulation framework for discrete part manufacturing systems.

In Section 2, we present some background information. In Section 3, we discuss some fundamental issues for incorporating reusability in a simulation framework. Section 4 extends the reusability concepts as applied to our modeling and simulation framework for discrete part manufacturing systems. The next section deals with the design and implementation issues of the modeling and simulation environment. Section 6 presents an illustrative example of the interactions between the simulation objects during a simulation execution. Section 7 presents concluding remarks.

2 BACKGROUND

Advancements in object-oriented programming (OOP) and related technologies such as knowledge engineering and modeling formalisms were the fundamental catalysts which caused a major paradigm shift in the modeling, analysis, and design of complex manufacturing systems. A research team in the Center for Computer Integrated Manufacturing (CIM) at Oklahoma State University has been exploring alternative approaches to the modeling and simulation of complex systems since 1985. The research efforts were directed mainly at conceptualizing new approaches to model construction, utilization, and

maintenance within an operating environment. The fundamental guiding factors in the quest for an alternative approach were the concepts of "plug compatibility of building blocks" and "modularity in the model building process." Over time these factors led to the concepts of a reusable simulation framework and separation of physical, information, and control objects during the abstraction of real world entities. There was a need for a framework that readily supports implementation of these concepts. Due to the well known advantages of OOP languages, as documented by Adiga [1989], Cox [1986], and Meyer [1988], the research team chose to implement these concepts in the object-oriented paradigm. This paradigm allows incremental revisions of the modeling and simulation environment. After an initial implementation in Smalltalk-V [Beaumariage 1990, Digitalk 1986], we shifted to a more powerful environment, Smalltalk-80 [Goldberg and Robson 1989].

In previous technical publications we have shared our experiences in modeling and simulation using OOP with the research community [Beaumariage and Mize 1990, Basnet et al. 1990, Pratt et al. 1991]. Here we present, in greater detail, the design and implementation of our framework for modeling and simulation of discrete part manufacturing systems.

3 REUSABLE SIMULATION FRAMEWORK REQUIREMENTS

Reusability is a relativistic concept, as it cannot be defined without referring to the experimental time frame. The following paragraphs describe reusability for short and long time frames.

Short-time-frame reusability: To achieve reusability in this context, the simulation/modeling approach should allow the modeler to repetitively use the basic model building blocks that correspond to the real world system components, to either reconfigure the model, or allow minor modifications to the building blocks.

Long-time-frame reusability: The real world system structure may change drastically. Some of the subsystems which were considered part of the external environment may have to be explicitly considered as part of the system under study, i.e., the system boundary itself may change. From these considerations, over the long term, the reusability features of the simulation framework should allow the modeler to:

1. reuse a subset of existing building blocks along with newly defined building blocks to experiment with a reconfigured system, and

2. expand the boundary of the system description to easily incorporate new subsystems without changing the existing building blocks of the current system. This means that the building blocks defined in the earlier model remain valid even if the system boundary has been expanded. To realize this, the simulation framework should provide for an easy coupling of the existing building blocks with the new blocks.

The reusability features in a simulation framework go a long way in supporting the model maintenance requirements. Consider for example a simulation model of a manufacturing system. As the manufacturing system evolves, its structure changes, it interacts differently with its environment, and the environment itself may change. Traditional modeling efforts do not lend themselves easily to updating the manufacturing system model in order to reflect the current manufacturing system structure.

However, if the model is formulated in a framework with reusability features, the model itself can evolve to keep in step with the manufacturing system evolution. In the present day philosophy of CIM, a manufacturing system is constantly evolving, therefore requiring constant changes in the system model. The simulation framework with reusability features easily lends itself to such model maintenance tasks.

The choice of Smalltalk-80 for implementing the code for our simulation environment was due to the fact that it is one of the purest OOP languages. Many of the object-oriented characteristics can be traced to the SIMULA 1 language [Meyer 1988]. Simula, though popular in academia in Europe and throughout the world, has never gained widespread use in the commercial environment [Kreutzer 1986]. Smalltalk includes an explicit message passing paradigm creating a programming style which has since become known as OOP [Kreutzer 1986, Meyer 1988]. The concepts underlying OOP can be exploited for simulation modeling [King and Fisher 1986; Mize et al. 1989; Thomasama and Ulgen 1988; Ulgen et al. 1989]. For details on the language Smalltalk-80 and OOP, readers may refer to Goldberg and Robson [1989], Cox [1987], or Pascoe [1986].

4 SIMULATION FRAMEWORK FOR MANUFACTURING SYSTEMS

The OOP paradigm has been employed for modeling manufacturing systems by several researchers [Adiga and Gadre 1990; King and Fisher 1986; Sanderson et al. 1991]. Adiga and Gadre [1990] describe modeling of a flexible manufacturing system. Their emphasis is on the modeling methodology and its translation into software using OOP. Glassey and Adiga [1989] present a conceptual design of a software library for simulation of semiconductor manufacturing systems. They have

identified three goals in their research as (1) ease of assembling special purpose simulation models, (2) ease of modification of object library and (3) the run time efficiency of the model assembled from the library of objects. The first two goals directly lead to reusability. Sanderson et al. [1991] describe design and implementation of a Hierarchical Simulation Language, which is interpreter based and hence offers certain advantages of portability and modifiability (during program execution).

In this paper, we present a framework with reusability features for modeling and simulation of discrete part manufacturing systems. While many aspects of the framework may apply equally well to the modeling of other systems, our focus to date has been limited to discrete part manufacturing. Expansion to other areas is an element of our future research agenda.

The concept of reusability was the main goal in designing the simulation framework for discrete part manufacturing systems. We have used the concept of separation for abstraction of a real world entity. According to this concept, any entity in the real world system has three dimensions to it, viz. physical, information, and control. The emphasis on these dimensions differ with each entity.

While representing the real world entities as modeling objects, the modeler may separately focus on the three dimensions. Figure 1 illustrates this concept. Every real world entity can be abstracted in its three dimensions. The model object space is correspondingly three dimensional. Thus, the model object is in fact a true representation of the real world object, as it has all three corresponding dimensions.

However, the modeler can visualize each model object in terms of its projections on the three planes, viz., the *physical plane*, the *information plane* and the *control plane*. This results in a structured approach to abstraction of real world objects.

The primary purpose for the abstraction of a real world object influences the main focus on the object. Thus, abstraction automatically results in the separation of the above three facets of an object. Despite the three dimensions of an object, depending on our primary focus, we may still classify the object as if it is one dimensional. Thus, in a model we can represent elements of the real world system with physical, information and control modeling objects. These three types of objects are defined as [Pratt et al. 1991]:

Physical Object: A physical object is an object with a tangible correspondent in the real world system. An object can be classified as a physical object if the primary focus of the modeler's interest in the object is its physical extent or characteristics, e.g., parts, work stations, and material handlers.

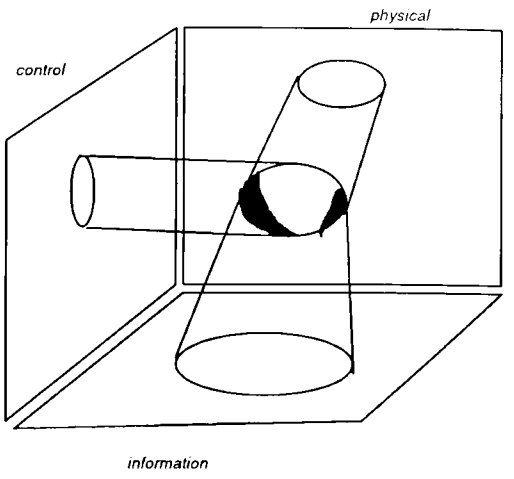

Figure 1: Modeling Object Projection

Information Object: An information object may or may not have a real tangible correspondent in the real world system. An object can be classified as an information object if the primary focus of the modeler's interest in the object is its information content, e.g., bill of materials, item master, and routing.

Control Object: A control object is a logical object which typically has no tangible correspondent in the real world system. An object can be considered a control object if its primary function is to (1) evaluate the state of a given system, (2) exercise a logic algorithm, and (3) signal an appropriate action be taken, e.g., a work station queue controller.

In the framework for simulation of discrete part manufacturing systems, we have exploited the above concepts of separation during the process of abstraction. Thus, an object can be abstracted in one, two, or all three planes. When some aspect of the real world object changes, e.g., the physical aspect, the primitive object representing that physical aspect (defined above as the physical object) can be changed independently of the other aspects. Thus the model object can be easily updated to keep in step with the most recent changes in the real world object.

In the process of model building, the modeler can couple several primitive objects to form a coupled object which corresponds to a real world object. The coupled object can be stored as a single entity in the object library [Pratt et al. 1991]. One of the primary advantages of a coupled object is that it can exhibit multiple behaviors. The behaviors are those inherited from its several parent classes.

In our framework we have implemented the separation of physical, information and control objects through several classes as described in the next section.

5 STRUCTURE OF THE MODELING AND SIMULATION ENVIRONMENT

The structural elements that define the modeling environment are illustrated in Figure 2. The dotted box indicates the implementation counterpart which is a set of classes responsible for implementing the corresponding structural element. The mapping between a structural element and its implementation counterpart is represented by a heavy line. Thus, the class library consists of the following four categories: model representation classes, class *SimModel*, simulation classes, and user interface classes. These categories are described below.

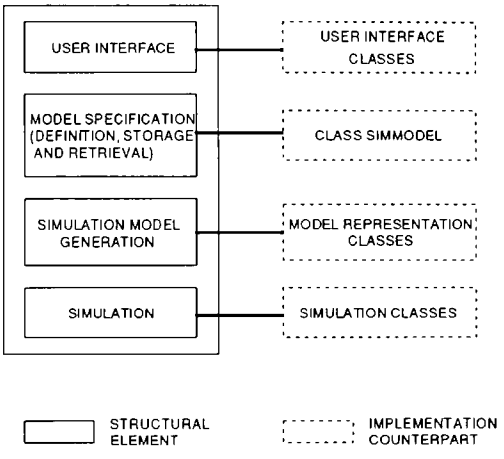

Figure 2: Modeling Environment Structure

5.1 Model Representation Classes

Model representation classes can be grouped into three categories: classes representing physical elements, classes representing information elements, and classes representing control/decision elements.

5.1.1 Classes Representing Physical Elements

These classes collectively model the behavior of the physical aspects of a manufacturing system. The manufacturing system (Figure 3) consists of a plant having a pair of buffers (input and output), several material handlers, and a number of work centers. Each work center in turn consists of a pair of buffers (input and output), several material handlers, and a number of work stations. The work stations perform the machining or assembly operations on the work flow items (parts). The movement of work flow items between the work stations is accomplished by the material handlers. The work flow items enter the plant, move through the work stations they need to visit (as defined in the part routings), and finally exit the plant through the plant output buffer as finished goods.

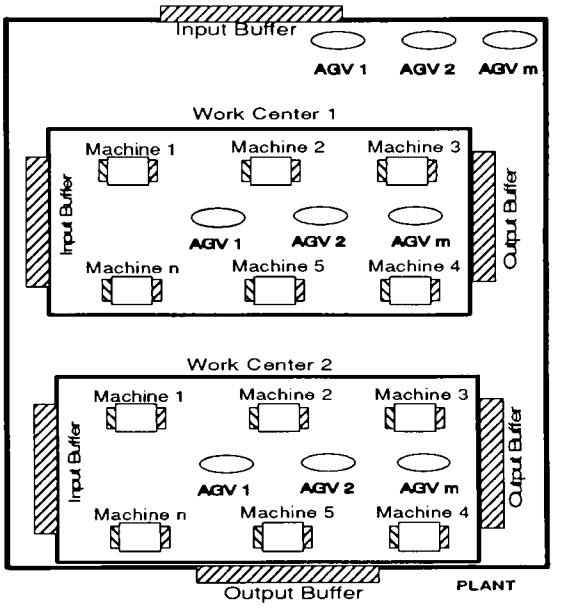

Figure 3: OOP View of a Manufacturing System

In addition to classes *Plant*, *WorkCenter*, *WorkStation*, *AssemblyWorkStation*, and *Material-Handler*, which function as described above, the following additional classes are used to model the physical elements of a manufacturing system.

Queue: This class embodies the behavior of a queue that holds work flow items waiting for some type of service. A number of queue disciplines such as, FIFO, LIFO, Earliest Due Date, etc., are defined for this class. A user can specify any one of these queue disciplines.

CapacitatedQueue: This class embodies the behavior of a queue that can hold at most the number of entities defined by the queue capacity.

Buffer: This class represents a storage location. All the work flow items entering a buffer are stored in a queue. The buffers are defined for a plant and work centers to accommodate work-in-process inventory.

5.1.2 Classes Representing Information Elements

These classes collectively model the behavior of the informational aspects of a manufacturing system.

CustomerOrder: This class represents a customer order describing the part type ordered and the quantity required. Customer order generators create the

customer orders to model a steady stream of incoming orders from the customers in the real system. On receipt, a customer order is exploded using the bill of materials structure for the given part to determine requirements for lower level components. Based on the quantities on hand, shop orders and/or purchase orders are generated for the required components.

ShopOrder: This class represents an order released to the production shop. Shop orders can specify a lot size of more than one. This produces excess inventory that can be used to satisfy subsequent customer orders.

Operation: This class represents a machining operation description. An operation consists of set-up time and processing time distributions for a given part on a specified machine.

Routing: This class defines the flow of parts through the work stations in the manufacturing system. A routing for a part is defined as a sequence of operations.

BOM: This class defines the parent/child structural relationships between parts in the manufacturing system.

ItemMaster: This class contains special information about each part defined in the bill of materials. ItemMaster contains information such as lot size, lead time, on hand quantity, and pending orders.

5.1.3 Classes Representing Control Elements

These classes collectively model the behavior of the control aspects of a manufacturing system.

QueueController: This class models a controller that monitors and manages the input and output queues of a work station or material handler. Based on the prescribed queue discipline, the queue controller provides the work station with the next part from the input queue.

AssemblyQueueController: This class models a controller that monitors and manages the input and output queues of an assembly station. AssemblyQueueController determines if there are enough components available to initiate an assembly operation.

WorkCenterController: This class is responsible for controlling the work center activities. WorkCenterController releases incoming work flow items to its work stations and provides decision support for routing work flow items in process.

5.2 Class SimModel

This class is responsible for providing a template to hold information about the entire simulation model of the manufacturing system. SimModel allows the user to create, modify, add or delete portions of the model. Based on the user information SimModel instantiates all the objects required for the execution of the simulation. This arrangement separates model definition (i.e., specification) from the simulation entities (i.e., object instantiations) and enables the user to initiate a new simulation run by using the information in the template.

5.3 Simulation Classes

These classes provide a basic framework for discrete system simulation and statistics collection. The classes Simulation, DelayedEvent, and SimulationObject are taken from Goldberg and Robson [1989].

Simulation: The purpose of this class is to schedule actions to occur according to the simulated time.

DelayedEvent: This is an active process, which when delayed for a specific amount of time, is placed on the queue sorted with respect to the resumption time.

SimulationObject: This class models the simulation entity and is an abstract class. It has to be further refined to faithfully represent the system being modeled.

Tracked Number: This class and its subclasses, *ObsTrackedNumber, TimeTrackedNumber, TrackedNumberWithCollection,* and *TrackedNumberWithHistogram,* define the repositories for various types of simulation statistics. The message protocols consist of methods for clearing statistics, collecting observations, calculating summary statistics, and printing results.

RandomNumberGenerator and probability distribution classes : Class RandomNumberGenerator provides a stream of random numbers required for simulation. The probability distribution classes implement a variety of random variate generators.

OrderGenerator: An order generator creates customer orders with a prescribed inter-arrival distribution. These orders trigger the pull logic that explodes each customer order and generates appropriate shop and purchase orders.

WorkFlowGenerator: A work flow generator creates work flow items according to a given inter-arrival distribution and pushes them into the manufacturing system.

5.4 User Interface Classes

These classes collectively provide a variety of user interfaces for model definition, modification, and simulation experimentation with a manufacturing system.

SimView: This class offers the main menu for the modeling and simulation environment. As depicted in Figure 4(a) on the next page, the Simulation Launcher View provides a user with a list of options such as plant

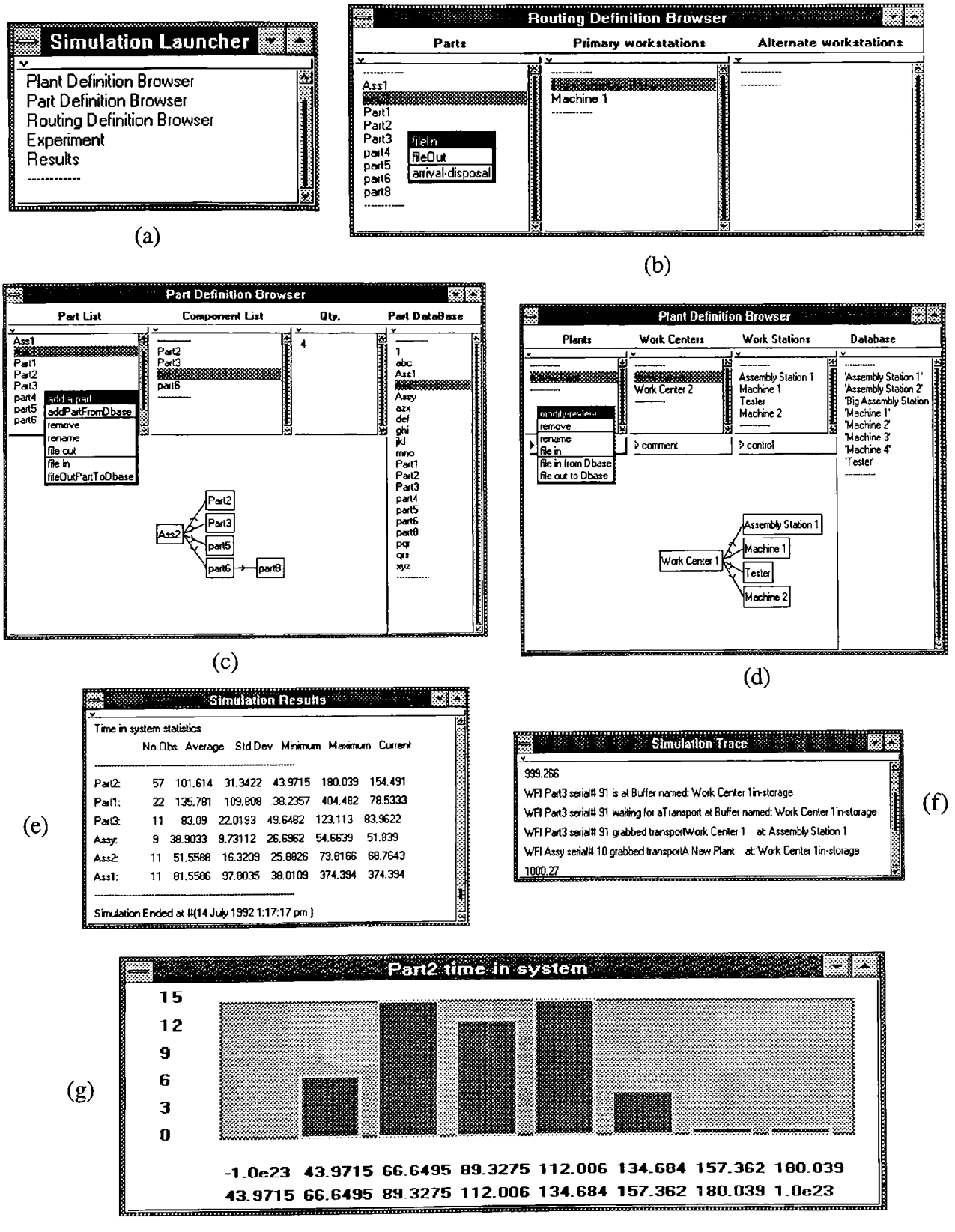

Figure 4: User Interface Windows

definition, part definition, routing definition, experimentation, and simulation results.

RoutingDefinitionView: This class offers a user interface for the routing definition. As depicted in Figure 4(b), the routing definition view provides the user with a list of work flow items and a list of predefined work stations, so that an appropriate routing can be defined for each part.

PartDefinitionView: This class offers a user interface for part definition or creation of the BOM structure. A Part definition view is depicted in Figure 4(c). Using the BOM database, the user can access previously available BOM structures in order to define a new BOM for the current manufacturing system.

PlantDefinitionView: This class offers a user interface for plant definition. As depicted in Figure 4(d), the plant definition view provides the user with plant, work center, and work station databases for quick plant definition or reconfiguration. The pop-up menus available in the plant definition view provide the user a mechanism to enter the required input parameters for the plant, work center, or work station.

ExperimentView: This class offers a user interface for defining the experimental simulation parameters such as, simulation termination time, initial seed for the random number stream, "Trace on" time, "Statistics clear" time, and histograms.

ResultView: This class offers a user interface for depicting the simulation results. As depicted in Figures 4(e), 4(f), and 4(g), the result view provides a summary report of all the statistics collected, trace output for a simulation run, and prints histograms.

6 ILLUSTRATIVE EXAMPLE

In this section, we use the movement of a WorkFlowItem (WFI) through the plant as a vehicle for explaining the interaction between various classes. This interaction takes place in the form of message passing. The description is not meant to be comprehensive in terms of describing all the message passing, but is intended to portray the overall scheme of simulation. Though the environment is capable of modeling both pull and push systems, we do not attempt to go into the details of their implementation. Rather, we use a simple case for the sake of clarity.

Consider an arrival of a customer order for a plant operating under the pull philosophy. This results in the following message: **Plant *arrivedOrder:anOrder.*** This message results in a chain of messages to several classes which first check for available inventory of the required part. If the number available is less than the order size then the appropriate BOM is accessed and part requirements are exploded in a recursive fashion. This leads to the release of shop orders for all required parts (placing of purchase orders in case of parts at the lowest level). Typical classes involved in these interactions are ItemMaster and ShopOrder. At this time the order specific information such as parent customer order and queue selection rules is also preserved.

As a response to a purchase order, a WFI corresponding to the ordered part type enters the simulation after a delay of time equivalent to the order lead time. When a WorkFlowItem is generated, a process corresponding to its life cycle is created which encapsulates tasks to be executed during the simulation (Figure 5).

```
[done] whileFalse:
[currentOperation := workCenterController
      nextWorkStation: self.
  machine := Plant active resourceNamed:
      (currentOperation machine).
  self moveMajorAndAcquireResource:
      machine.
  self completeOperationsAtLocation].
```

Figure 5: Life Cycle of a Typical Work Flow Item

The work center controller accesses the routing for the part, and then dynamically chooses the appropriate work station based on feasible alternate work stations and the specified queue selection rule.

Now the work flow item, in response to the message ***moveMajorAndAcquireResource: machine*** invokes a series of messages by which it tries to acquire the appropriate material handler for moving from the plant input buffer to the work center input buffer and then to the input queue of the desired resource *machine*. It then tries to acquire that resource and if the resource is not available, then the WFI is ***paused.*** (This suspends the process corresponding to the tasks of this WFI). When the resource is acquired, the process corresponding to tasks for the WFI is ***resumed,*** which restarts the process corresponding to the life cycle of the WFI. The message ***completeOperationsAtLocation*** schedules the process to be restarted after the processing time. This sequence of actions is repeated until the WFI finishes its operations. In case the resource is of type assembly station, assemblyQueueController also checks for the availability of components required for an assembly in different input queues, before trying to allocate the resource. If the resource (assembly station) is allocated, the WFI task-processes corresponding to each WFI which goes into assembly are terminated by the message ***finishUp*** sent to each of them and then a new

WFI (hence a task-process) representing the assembly is generated after the time required for completing the assembly.

When all the stages in the routing of the WFI are completed (causing [done] shown on line 1 of Figure 5 to become "True"), the WFI sends the message ***Plant iAmDone:aWFI*** which causes the pending shop order, against which the WFI was produced, to be filled. When the desired end item is produced in the required quantities, the pending customer order is satisfied.

7 SUMMARY AND CONCLUSIONS

The work on this project is still largely conceptual and exploratory. The proposed approach to modeling based on the separation of physical, information and control elements within an object-oriented environment is more revolutionary than evolutionary. The prototype environment has demonstrated encouraging potential for enhanced modeling power, but a formidable research agenda remains. The research will have major impacts, both in the conceptual area of reusable models of complex systems and in the applied area of engineering practice.

ACKNOWLEDGMENT

The authors acknowledge the support provided by the AT&T Foundation and the Oklahoma Center for the Advancement of Science and Technology. We also would like to acknowledge the valuable contributions of Prof. Lawrence Leemis, Steve Tretheway, and Mike Oltman (The University of Oklahoma) to this project.

REFERENCES

Adiga, S., (1989), "Software Modeling of Manufacturing Systems: A Case for an Object-Oriented Programming Approach," *Annals of Operations Research, 17,* 363-378.

Adiga, S. and M. Gadre, (1990), "Object-Oriented Software Modeling of a Flexible Manufacturing System," *Journal of Intelligent and Robotics Systems, 3,* 147-165.

Basnet, C., P. Farrington, D. Pratt, M. Kamath, C. Karacal, and T. Beaumariage, (1990), "Experiences in Developing an Object-Oriented Modeling Environment for Manufacturing Systems," *Proceedings of the 1990 Winter Simulation Conference,* O. Balci, R. Sadowski, and R. Nance, Eds., IEEE, Piscataway, NJ, 477-481.

Beaumariage, T.G., and J.H. Mize, (1990), "Object-oriented Modeling: Concepts and Ongoing Research," *Proceedings of the SCS Multiconference on Object-oriented Simulation,* A. Guasch Ed., SCS, 7-12,

Beaumariage, T.G., (1990), "Investigation of an Object-oriented Modeling Environment for the Generation of Simulation Models," Ph.D. Dissertation, School of Industrial Engineering and Management, Oklahoma State University, Stillwater, OK.

Cox, B., (1986), *Object-oriented Programming: An Evolutionary Approach*, Addison-Wesley, Reading, MA.

Digitalk, Inc., (1986), *Smalltalk/V: Tutorial and Programming Handbook*, Digitalk, Inc., Los Angeles, CA.

Goldberg, A. and D. Robson, (1989), *Smalltalk-80 The Language*, Addison-Wesley, Reading, MA.

Glassey, C.R. and S. Adiga, (1989), "Conceptual Design of a Software Object Library for Simulation of Semiconductor Manufacturing Systems," *Journal of Object-Oriented Programming*, 2,4, 39-43.

King, C.U. and E.L. Fisher, (1986), "Object-Oriented Shop-Floor Design, Simulation, and Evaluation," *Proceedings of the 1986 Fall Industrial Engineering Conference*, Institute of Industrial Engineers, Norcross, GA, 131-137.

Kreutzer, W., (1986), *System Simulation Programming Styles and Languages*, Addison-Wesley, Reading, MA.

Law, A.M. and S.W. Haider, (1989), "Selecting Simulation Software for Manufacturing Applications: Practical Guidelines and Software Survey," *Industrial Engineering*, 31, 5, 33-46.

Meyer, B., (1987), "Reusability: The Case for Object-Oriented Design," *IEEE Software*, 4, 2, 50-64.

Meyer, B., (1988), *Object-Oriented Software Construction*, Prentice Hall International (UK) Ltd., Hertfordshire, Great Britain.

Mitrani, I., (1982), *Simulation Techniques for Discrete Event Systems*, Cambridge University Press, Cambridge, Great Britain.

Mize, J.H., T.G. Beaumariage, and S.C. Karacal, (1989), "Systems Modeling Using Object-Oriented Programming," *Proceedings of the 1989 Spring Conference*, Institute of Industrial Engineers, Norcross, GA, 13-18.

Nance, R.E., (1984), "Model Development Revisited," *Proceedings of the 1984 Winter Simulation Conference*, S. Sheppard, U.W. Pooch, and C.D. Pegden, Eds. IEEE , Piscataway, NJ, 75-80.

Pratt, D.B., P.A. Farrington, C.B. Basnet, H.C. Bhuskute, M. Kamath, J.H. Mize, (1991), "A Framework for Highly Reusable Simulation Modeling: Separating Physical, Information, and Control Elements," *Proceedings of the 24th Annual*

Simulation Symposium, A.H. Rutan (Ed.), IEEE Computer Society Press, 254-261.

Pritsker, A.A.B., (1986), *Introduction to Simulation and SLAM II*, Third Edition, Halsted Press, New York, NY.

Sanderson D.P., R. Sharma, R. Rozin, and S. Treu, (1991), "The Hierarchical Simulation Language HSL: A Versatile Tool for Process-Oriented Simulation", *ACM Transactions on Modeling and Computer Simulation, 113-153.*

Thomasma, T. and O.M. Ulgen, (1988), "Hierarchical, Modular Simulation Modeling in Icon-based Simulation Program Generators for Manufacturing," *Proceedings of the 1988 Winter Simulation Conference*, M. Abrams, P. Haigh, and J. Comfort, Eds. IEEE , Piscataway, NJ, 254-262.

Ulgen, O.M., T. Thomasma, and Y. Mao, (1989), "Object-oriented Toolkits for Simulation Program Generators," *Proceedings of the 1989 Winter Simulation Conference*, E.A. MacNair, K.J. Musselman, P. Heidelberger, Eds. IEEE, Piscataway, NJ, 593-600.

Zeigler, B.P., (1976), *Theory of Modeling and Simulation*, Robert E. Krieger Pub. Co., Malabar, FL.

AUTHOR BIOGRAPHIES

HEMANT C. BHUSKUTE is Research Associate in the Center for CIM within the School of Industrial Engineering and Management at Oklahoma State University. He holds a B.E. in Electrical Engineering from the University of Bombay, an M.M.S.E. in Manufacturing Systems Engineering from Oklahoma State University and is currently completing his Ph.D. in IE at Oklahoma State. His research interests include discrete event simulation, object-oriented modeling and parallel processing. He is a member of IIE, ASQC, ORSA, Tau Beta Pi, and Alpha Pi Mu.

MANOJ N. DUSE is Research Associate in the Center for CIM within the School of Industrial Engineering and Management at Oklahoma State University. He holds a B.E. in Mechanical Engineering from the University of Poona and an M.S. in Industrial Engineering from NITIE, Bombay, India. He is currently completing his Ph.D. in IE at Oklahoma State University. His research interests include object-oriented modeling, knowledge based simulation and performance evaluation of queueing networks. He is a member of IIE, Tau Beta Pi, and Alpha Pi Mu.

JAGANNATH T. GHARPURE is Research Associate in the Center for CIM within the School of Industrial Engineering and Management at Oklahoma State University. He holds a B.E. in Electrical Engineering from The M.S. University of Baroda and a M.S. in Industrial Engineering from NITIE, Bombay, India. He is currently completing his Ph.D. in IE at Oklahoma State University. His research interests include discrete event simulation, object-oriented modeling, AI and neural networks. He is a member of Tau Beta Pi, and Alpha Pi Mu.

DAVID B. PRATT is Assistant Professor of Industrial Engineering and Management at Oklahoma State University. He holds B.S., M.E, and Ph.D. degrees in IE from Oklahoma State University. His research and teaching interests include enterprise-wide systems integration, manufacturing systems modeling, total quality management, and applied operations research. He is a registered Professional Engineer, a certified Fellow in Production and Inventory Management, and an ASQC Certified Quality Engineer. He is a member of IIE, NSPE, APICS, TIMS, and ASQC.

MANJUNATH KAMATH is Assistant Professor of Industrial Engineering and Management at Oklahoma State University. He received the B.Tech. degree in Mechanical Engineering from the Indian Institute of Technology, Madras, India, in 1982, the M.E. degree in Automation from the Indian Institute of Science, Bangalore, India, in 1984, and the Ph.D. degree in Industrial Engineering from the University of Wisconsin-Madison, in 1989. His primary areas of interest are stochastic modeling and queueing theory, analytical performance modeling of manufacturing systems, object-oriented modeling, simulation of discrete-event systems, and Petri nets. He is a member of IIE, IEEE, and ORSA.

JOE H. MIZE is Regents Professor of Industrial Engineering and Director of the Center for Computer Integrated Manufacturing at Oklahoma State University. His research interests are in advanced modeling/simulation environments and manufacturing systems integration. He has authored six engineering texts and has edited 90 others. He is a Fellow and Past President of the Institute of Industrial Engineers and is a Member of the National Academy of Engineering.