Introduction

For several years, graduate students and faculty of the Engineering Systems Research Center at U.C., Berkeley have been studying new methods of planning and scheduling in a computer integrated manufacturing environment, with particular emphasis on large scale integrated circuit fabrication. One part of this work, focusing on short interval scheduling, uses simulation models as a primary research tool. We have built two versions of the same basic model (programmed in C) to study two different problems (one deals with machine down time and the other with setup times). These have proven to be efficient for studying particular problems, but are difficult and time consuming to modify. We are convinced that our research will be more effective: (1) if it were easier to build special purpose models tailored to the research question at hand; and (2) if we had better interfaces to graphics output.

Commercially available factory simulators are inadequate for this research for a variety of reasons. Existing packages such as SIMKIT, SLAM, SIMAN and EXCELL have their own weaknesses. Typically, they are hard to develop and to modify. They do not allow for adding new dispatching decisions or release decision. Also, it is hard to add more machines to existing environment or change the route the product flows. For these various reasons, we had developed a new simulation package having flexibility and modularity.

In this paper, based on experiences gained in the application of object oriented programming, we discuss unique features of the simulator developed in OOPS and ways to take advantage of features in developing and using manufacturing simulation software written in the OOPS

1. Development of the Simulator

Simulation is the conceptually simplest approach for modeling dynamic stochastic system such as the semiconductor manufacturing system. As customer demand of products

is diversifying and manufacturing technology requires more complex operations, manufacturing systems become complicated in terms of production scheduling and control. This is particularly true for the semiconductor manufacturing industries for which the simulator is developed. Because of this complexity, problems such as long production cycle time, excessive inventory of final and intermediate products, and low utilization of resource can result and production simulators are used as indispensable tools for solving these problems.

Since those simulators strongly depend upon the system being researched, they are not adaptable to other kinds of production systems. This problem causes a duplication of efforts for developing a new simulator for each individual production system. Therefore, it is hoped to develop a general simulator with adaptability to a great variety of production systems and reusability of its modules. The adaptability eliminates the necessity of developing a new simulator and the reusability reduces the duplication of efforts. The object oriented programming (OOP) is adopted for our research to achieve an adaptability and a reusability.

Selection of suitable software medium for our research is evaluated from two different perspectives, i.e., computer languages and software paradigms. A programming paradigm is a set of related concepts useful in conceptualizing, analyzing, and solving a problem.[11] A programming language is a medium of expression.

1.1. Computer Languages

Special Purpose Simulation Packages

Special purpose simulation packages have been developed for modeling manufacturing systems. MAP/1[16], Simfactory[8] and Excell[7] are a few of those packages. These packages are designed to be used by non-programmers and contain good user interfaces. But generally they are weak in flexibility and difficult to modify to encompass new control strategies or systems.

General Purpose Simulation Languages/Packages

High-level languages such as GPSS[13], SLAM[10], SIMSCRIPT[3], SIMAN[9] etc. aim to make modeling a little easier. The simulation languages mentioned above were implemented in FORTRAN as a collection of subroutines with a set of control statements. This results in a reduction in the amount of code one has to write to model a desired situation. Large applications involve complex development since these kinds of simulation languages are adequate in modeling typical simple systems. Young et al.[17] observe that the current simulation languages are designed to support only the modeling and analysis of process and material handling flow. Many requirements of computer integrated manufacturing are not easily performed. More drawbacks in designing simulations based on these languages are discussed by Shannon et al.[14]

General Purpose Languages

Fortran was, by far, the most popular language for implementing simulation in the 1970's because of the ability to handle numeric computations conveniently and efficiently. In recent years, PASCAL and C have emerged as competitors to Fortran. These newer general purpose languages provide data structures that make the abstraction process a little easier. An advantage in using a general purpose language is the ability to analyze virtually limitless number of problems. While the programmer has the flexibility to write the programs in an idiosyncratic style, this is the enemy of software reusability. As a part of our research, a special purpose simulation of wafer fabrication operations called FABSIM was written in C language.[12] This program contained about 6000 lines of C source code, but represented only the small set of real world phenomena that were important for that particular research. A second version of the program was written to expand its scope slightly and this required an additional 2000 lines of source code and considerable effort by a second student in understanding the original program. Our experience raises serious productivity concerns regarding extendibility and maintenance of such programs.

1.2. Programming Paradigms

Procedural Programming Languages

As mentioned earlier, the early simulation were carried out in general purpose languages which follow the procedural programming paradigm. While programming in a procedural language, one is required to specify a sequence of steps to be carried out to achieve a desired effect. The discussion related to the general purpose languages such as Fortran, C etc. in the previous section of this paper is quite applicable in this context too.

Declarative Programming Paradigm

This paradigm was made popular by the language Prolog.[4] In a declarative programming paradigm, one is expected to declare the facts and rules of the problem, and leave the work of searching for the solution to the built-in inference system of the implementation language. Popular implementation of declarative style of programming are achieved through the use of mathematical logic in programming. Application of Prolog in prototype simulation is discussed by Adelsberger[1], and Ben-Arieh[2].

Functional Program Paradigm

The most popular representative of this paradigm is the language LISP (List Processing language)[15]. In a functional language, one defines expressions which, upon evaluation, define values. A program may be understood as a definition of the desired value.[5] The current trend in simulation work in LISP is towards the development of high-level implementations. An example is Simulation Craft, a simulation system developed by Cranegie Group. An important advantage to be gained by programming in a language such as LISP is the symbolic manipulation capabilities offered by such language. The disadvantages include requirement of large amount of computing resources and problems in portability and communication with existing incompatible shop-floor information sources.

Object Oriented Programming (OOP) Paradigm

Object-Oriented computing as understood and practiced today owes its popularity to the ideas made popular among the software engineering community by the software Smalltalk[6]. In general, two key elements of an object-oriented programming paradigm are object and message. Object is a collection of private data and behavior that it is capable of. Data is stored in instance variables which cannot directly accessed by other objects. Each behavior is defined by a procedure known as a method. The concept of an object combines the attributes of data and procedures. This feature of confining all relevant data in a module and restricting its access to a set of predefined procedures is commonly referred to an encapsulation. Object communicate with each other by sending messages. When an object receives a message, it activates a corresponding method. Exchanging a message is similar to calling a function in languages such as Fortran or C with one important difference. Function invokes a predetermined procedure whereas in OOPS, the decision as to which procedure (or method) to be invoked is left to the object.

Some of the major benefits to be gained from object-based computing are as follows:

(*1*)*Representation of real world events.* By bundling the data and procedures directly affecting an object in one place, OOPS allows us to use a natural one-to-one correspondence with physical objects. For example, it is possible to have software objects which directly correspond to the physical object in a manufacturing system such as machines, operators, routes etc. The use of objects and messages allows one to come up with an easily understandable system design.

(*2*)*Reusability.* Since modules that are developed once are independent of other objects, they can be used in any application as a part of it. In addition, using the inheritance, a new object is created based on another object which has similar methods. Moreover, the message requesting a service from an object is independent of the method that it invokes. This allows reusability without the constraints associated with the reuse of subroutine libraries.

(*3*)*Simplicity.* Since relationship between two objects depends only on the encapsulated

data of both objects and is independent of other relations, the application system has less complexity. The complexity of the system increases efforts of developing the system exponentially. Hence, less complexity makes all phase of development including design, programming, test and maintenance much easier.

(*4*)*Adaptability.* This is strongly related to other advantages. The collection of objects and relations through the message emulates a behavior of the actual physical system. For a certain system, it is easy to figure out what modules we need and what kinds of properties they should have. Upon figure out this requirement, each module can be independently modified and assembled to a complete simulator.

(*5*)*Programmer productivity.* While programming in OOPS, we create a software object by describing the most common features/behavior required of a class of objects of the system to be modeled. This is called a 'class' object. Any other object which is declared as a subclass of this class object (now a superclass) inherits all the data and methods from its superclass. For example, we may say Electric Motor as a class of object which represents objects that convert electrical energy to mechanical. Next, we may declare 'D.C. Motors' as a subclass of 'Electric Motors'. When you declare D.C. Motors, we need not describe the properties exhibited by the electric motors ingeneral. We only need to describe the specialization or any additional functionality that is unique to that specific motor. This concept of inheritance eliminates the need for rewriting whole code and maintaining code in different places.

After reviewing characteristics of the various software medium introduced so far, it is emerging that a general purpose language should be selected as our research tool. In regard to software paradigm, the object oriented programming has strong edges over the other candidates considering flexibility and reusability of the simulator to be developed.

2. Model Description

First of all, to develop the simulator, careful analysis of the manufacturing system is

carried out. This includes identification of objects, internal flow of information, jobs to be performed by each objects as well as overall control mechanism of the simulator. As a result, the current simulator has been configured for the semiconductor system with the following characteristics.

.Multiple products.

.Arrival of jobs are dynamic and stochastic with a known distribution.

.Alternative routes for a product.

.Alternative work stations are identical in their processing capabilities.

.Machines break down with a known distribution.

.A work station has M identical machines.

.Machine and work force are constraint resources.

.Transportation time between work station has known distribution.

.Release control, dispatching and route selection are decisions.

As the characteristics of the system to be simulated are determined, specification of behavior and objects of the system is followed by identification of all the events inside the simulated model. At this phase, programming language for our research has been selected among the existing OOP languages. The appropriateness and expectation of the use of the OOP facilities are the prime guidelines for the language selection. Smalltalk, C++ and Objective-C have been emerged as possible tools for our research. Considering support in terms of documentations, library of software components and portability of our research, the software Objective-C has been chosen as our implementation language. Finally, programming of objects are done on a Micro-Vax in the Vax Station configured to Unix (BSD 4.3).

3. Execution of the Simulator

After functionality of the developed object is checked, we conceptually configure real factory system for which we want to simulate. At this stage, a careful consideration is

required to make sure that the two systems are matching exactly same. A main program to bind necessary objects for execution is prepared. Our experience reveals that the main program does not exceed 100 lines in its size. During compilation, cross checking of message table is performed by the Objective-C compiler to make sure that all the necessary objects have been included in the program. After successful compilation, the simulator can be executed by triggering a message flow to an object which, in turn, starts another message to the object specified in a method. This message flows continue until a predetermined condition are satisfied and the simulation ends. Therefore, message flow among the objects and methods inside object are two key element of the simulator and, also, distinctive feature of the system. As input to the simulation system, distribution of parameters, horizon of simulation run as well as release/dispatching rules are required. We obtain valuable system information including average waiting time, throughput time, percent utilization of work station, time scaled cumulative amount of work at each work station as output. At user's request, a report can be generated. During simulation run, animation of work station can be obtained at any time. Graphics requires use of graphics terminals such as TEK4207 or Micro-VAX work station terminal. We find, during test run of the simulator, the system developed in the OOPS has numerous attractions. For example, a system with two machines can be modified to the one with twenty machines by changing parameters inside main program. A system developed for a flow shop can be easily transformed to a pure job shop system by changing recipe data which identifies sequence of jobs to be performed by each product. New product or control methodology can be incorporated with minor change of objects concerned. One of drawbacks of OOPS found during experiments is running time efficiency. As the complexity of the system simulated increased, time required to run the simulator grows exponentially compared to linear growth pattern of the equivalent simulator written in C

4. Conclusions

We found that the simulator developed in the object oriented language has numerous advantages over the existing simulators in terms of simplicity, flexibility and adaptability. This unique programming system can be applied to variety of research fields including CIM (computer cntegrated manufacturing) system and AI (artificial Intelligence) area. In this regard, two basic fronts for further R&D (research and development) are discussed below.

Further testing of the reusability. Objects in our library has been tailored to be used for various simulation with a different set of configuration. This means that the simulator can be reused for simulating most of the I.C. manufacturing environment. On the other hand, there may be a certain system for which our simulator cannot be used because of considerable modification required. Therefore, development of library of generic object which covers wider areas of I.C. manufacturing system remains open.

Computational cost of implementing the Objective–C code. Our experience showed the simulator performs 3 to 6 times slower than an equivalent simulator written in C for simulation of simple system. Further research is necessary to find performance of the simulator for real world problems.

References

1) Adelsberger, H. N., "Prolog as a Simulation Language," *Proceedings of the 1984 Winter Simulation Conference*, pp 501–504, 1984.

2) Ben–Arieh, D., "Manufacturing System Application of a Knowledge Based Simulation," *Proceedings of the 8th Annual Conference on Computers and Industrial Engineering*, pp 459–463, 1986.

3) CACI, *SIMSCRIPT II.5 Programming Language*, CACI Inc., Los Angeles, 1983.

4) Clocksin, W. F. and C. S. Mellish, *Programming in Prolog*, Springer–Verlag, Berlin Heidelberg, 1981.

5) Davis, R. E., "Logic Programming is not Circuit Design," *Proceedings of IEEE COMPCON '84*, pp 53–62, 1984.

6) Goldberg, A. and D. Robson, *Smalltalk–80: The Language and its Implementation*, Addison–Wesley, Reading MA, 1983.

7) Jones, C. V. and W. L. Maxwell, "A System for Manufacturing Scheduling with Interactive Computer Graphics," *IIE Transactions*, pp 298–303, 1986.

8) Klein, B., "SIMFACTORY Tutorial," *Proceedings of the 1986 Winter Simulation Conference*, pp 530–542, 1986.

9) Pegden, C., *Introduction to SIMAN*, Systems Modeling Corporation, PA, 1985.

10) Pritsker, A. A. P. and C. Pegden, *Introduction to Simulation and SIMAN*, Halstead Press, John Wiley & Sons, 1987.

11) Ramamoorthy, C. V., S. Shekhar and V. Garg, "Software Development Support for AI Programs," *IEEE Computer*, pp 30–40, January 1987.

12) Resende, M. G. C., "Shop Floor Scheduling of Semiconductor Wafer Manufacturing," ESRC 87–1, Engineering Systems Research Center, University of California at Berkeley. September 1987.

13) Schriber, T. J., *Simulation Using GPSS*, Wiley, New York, 1974.

14) Shannon, R. E., R. Mayer and H. H. Adelsberger, "Expert Systems and Simulation," *Simulation*, pp 306–310, June 1985.

15) Winston, P. H. and B. K. P. Horn, *LISP*, Addison–Wesley, 1981.

16) Wortman, D. B. and R. J. Miner, "Designing Flexible Manufacturing Systems using Simulation," *Proceedings of AUTOFACT 6*, pp 24–29, October 1986.

17) Young, R. E., J. Vesterager, K. E. Wichman and J. Heide, "Simulation Uses in CIM Development," *International Journal of Computer Integrated Manufacturing*, Vol. 1, No. 1, pp 50–54, 1988.