Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

OBJECT ORIENTED PROGRAMMING LANGUAGES FOR DEVELOPING SIMULATION-RELATED SOFTWARE

Timothy Thomasma

University of Michigan - Dearborn
Industrial and Systems Engineering
4901 Evergreen Road
Dearborn, Michigan 48128

James Madsen

Production Modeling Corporation
One Parklane Blvd. Suite 1604 West
Dearborn, Michigan 48126

ABSTRACT

A comparative study has been done in which a portion of an icon-based simulation program generator is implemented in each of four object oriented programming languages that are available for MS-DOS and PC-DOS based personal computers. The languages studied are MODSIM II, Objective-C 4.0, Smalltalk/V 286, and Zortech C++. These new languages and new versions of older languages produce code that runs much more quickly than code produced by earlier object oriented programming languages, thereby eliminating much of the execution speed penalty commonly associated with object oriented programming. The choice of which object oriented language to use is dependent on the syntax one feels most comfortable with, the appropriateness to the job at hand of the classes in the libraries provided with the language, and the quality of support given by the programming environment supplied with the language.

1. RECENT DEVELOPMENTS IN OBJECT ORIENTED SIMULATION

In the course of doing a computer simulation study there are a number of pieces of software that are useful, in addition to the model itself. Simulation program generators [Ulgen and Thomasma 1989] can often be used to quickly produce models from descriptive data fed to them describing a real system. There are programs that assist in analysis of data and management of simulation models, experimental frames, and results of runs [Grant and Starks 1988]. Some programs fit probability distributions to observed data [Law and Vincent 1988]. Others assist the user in preparing animations of simulation runs [Poorte and Davis 1989]. All of this is in addition to the compiler used to generate executable code from the simulation language statements.

These pieces of software are being packaged together in the form of simulation support environments. It is likely that future simulation environments will include other kinds of software, e.g., artificial intelligence software for assistance in system design and design and management of experiments. The people who are developing simulation support environments are and should be people with expertise in simulation.

There has long been interest in applying object oriented programming to the construction of simulation models. In fact, object oriented programming was first supported in the Simula simulation language [Birtwistle et. al. 1979]. Object oriented programming is particularly appropriate for implementing hierarchical, modular simulation modeling and distributed simulation. Hierarchical, modular simulation modeling and distributed simulation are rarely done using anything other than an object-oriented programming language or approach.

Object oriented programming is also seen as an important technique for producing all kinds of software. Recently there has been a proliferation of object oriented programming languages, some of which are supported and promoted aggressively by leading software companies. For PC-DOS and MS-DOS machines, one can choose an object-oriented programming language from this (not necessarily all-inclusive) list: Smalltalk/V 286, ObjectWorks for Smalltalk-80, C++ (from at least five different companies), Turbo Pascal 5.5, Objective-C, MODSIM II, Eiffel, GoldWorks II, Actor, C_Talk. Similar lists exist for the Macintosh, for OS/2 machines, and for Unix workstations. Some of these languages and new versions of older languages have been very recently released.

Previous experience with earlier versions of many of these languages has led to the conclusion that object oriented programming languages support faster program development, but the programs that are developed run more slowly than programs developed using a standard programming language. Therefore, it has been considered that the best applications for object oriented programming are in the software elements that support the model-building process, especially those that involve a high degree of user interface or artificial intelligence [Alasuvanto, et.al. 1988, Thomasma and Ulgen 1988]. One should use these tools to assist in building simulation models that are actually compiled by standard compilers, such as C, FORTRAN, GPSS, SLAM, SIMAN, etc.

Improvements in object oriented programming languages are changing this picture. In a recent comparative review [Doyle 1990] it was found that Zortech C++ produces simulation programs that run very nearly as fast as programs written in C and that Smalltalk/V 286 produces simulation programs that run faster than those written in Simscript II.5.

These new developments led us to undertake a comparative review of our own of several object oriented programming languages. We are primarily interested in how suitable they are for writing simulation program generators. We also provide new empirical results on run-time efficiency of simulation programs written in some of these languages.

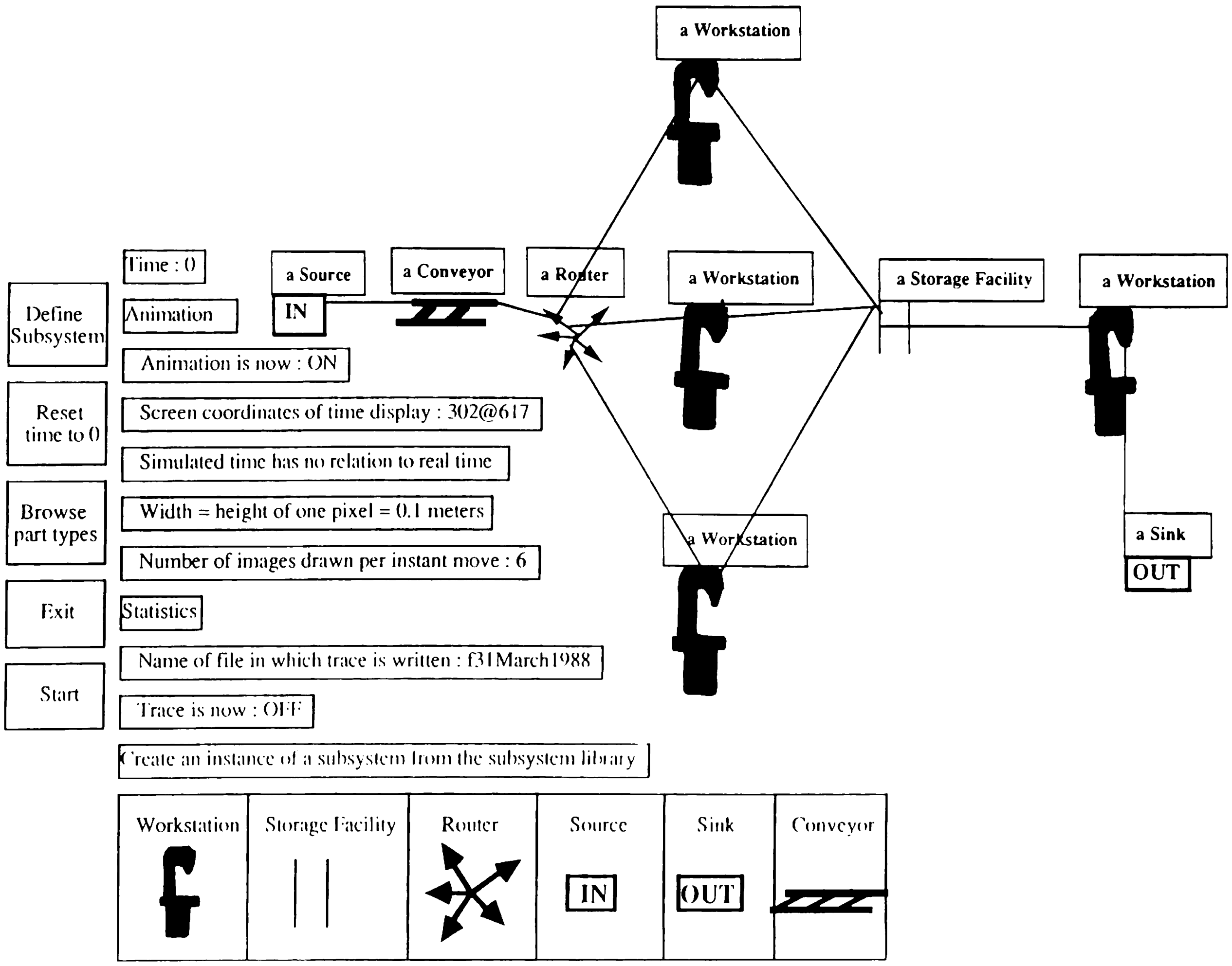
2. TEST PROGRAMS

The program that we are using in this experiment is a portion of SmartSim [Thomasma and Ulgen 1988]. It uses icons and popup menus to capture information from the user that describes the buffers and workstations in a manufacturing system. Part routings are indicated by arrows that the user draws on the layout showing the paths of the various parts through the system. Figure 1 shows how the user interface looks. Once the information is obtained, the program writes it to a file, which is intended to be read by a data-driven simulation program written in SIMAN.

In addition to designs for simple versions of the classes in SmartSim's class hierarchy (Figure 2), the design of this program includes specifications for user interface objects, including Button, TextButton, EditField, Menu, ParameterMenu, and Image. This is done in order to implement the program in object oriented programming languages that don't have class libraries for interactive graphics.

In order to compare execution efficiency of the various languages, simplified versions of SmartSim's Simulator, StationarySimulationObject, Part, PartType, Event, Source, Sink, Conveyor, StorageFacility, Router, and Workstation are implemented. Then the model pictured in Figure 1 is built and run (without animation).

In the system shown in Figure 1 parts enter the Source with an exponential interarrival time at an averages of 14 seconds between arrivals. These parts are placed on the Conveyor and travel down it for 20 seconds before accumulating at the end. The Conveyor has a maximum capacity of 10 parts. If a part cannot be placed on the Conveyor, it leaves the system. Parts are taken from the Conveyor and placed on an idle Workstation. The decision as to which Workstation to send the part to is made by the Router. Processing times for these three Workstations are each normal with mean 40 seconds and standard deviation 5 seconds. Each of these Workstations is subject to random breakdown. Breakdowns occur between 200 and 1000 seconds apart, distributed uniformly. Repair



Time : 0

Animation is now : ON

Screen coordinates of time display : 302@617

Simulated time has no relation to real time

Width = height of one pixel = 0.1 meters

Number of images drawn per instant move : 6

Name of file in which trace is written : f31March1988

Trace is now : OFF

Create an instance of a subsystem from the subsystem library

Figure 1. The Test Program's User Interface

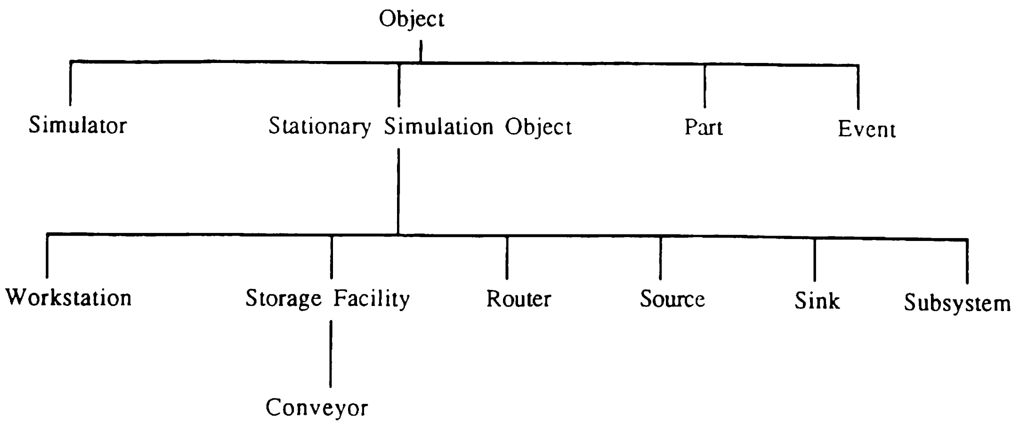


Figure 2. SmartSim's Class Hierarchy

times are exponential with mean 15 seconds. When processing is finished, the parts are placed in the StorageFacility, which has a capacity of 10 parts, before being processed on the last Workstation, after which time they leave the system at the Sink. The last Workstation is not subject to random breakdowns. Its processing time is normal with mean 12 seconds and standard deviation 2 seconds

All of this code was prototyped in Smalltalk-80 on a Tektronix 4405 and thoroughly tested and debugged prior to the start of the study. The random number generator in Smalltalk-80 [Goldberg and Robson 1989] is used if no other random number generator is provided by the language.

3. LANGUAGES USED IN THE STUDY

The languages we are studying are MODSIM II, version 1.1, Objective-C 4.0, Smalltalk/V 286, and Zortech C++. Zortech C++ is a complete compiler that comes with an extensive library and program development environment.

MODSIM II and Objective-C operate as C preprocessors. MODSIM II translates its source code into Turbo C 2.0. Objective-C translates its source code into Microsoft C 5.0. Once translated by MODSIM II or Objective-C, the code must then be compiled, using Turbo C or Microsoft C compilers, respectively.

The libraries and programming environment for Objective-C are limited at present in the MS-DOS version, although there are third-party libraries available. Other versions (for Unix workstations) provide much larger libraries and more programming support tools.

MODSIM II is the only language in the group that supports simulation directly, and this support is built into the language, as well as added in a library. The libraries presently provided with MODSIM II are otherwise rather limited. However, the next version (becoming available in Summer 1990) offers Simgraphics as a graphics library.

Smalltalk/V 286 is an interpreted language. Its programming environment is the most elaborate and its class libraries are the most extensive of any of the languages tested. Smalltalk/V 286 provides no direct support for simulation modeling.

Objective-C and Smalltalk/V 286 have been available the for the longest time for MS-DOS. Zortech C++ has received positive reviews in the trade magazines. MODSIM II is the first commercial implementation of the Army's ModSim language [Herring 1990].

Newer object oriented languages tend to be extensions of other programming languages. Smalltalk is quite different from the usual programming languages and it is partly for that reason that people find it hard to learn. In order to learn Objective-C, it is very helpful to already know both C and Smalltalk.

The newer languages, C++ and MODSIM II, are extensions of C and Modula-2, respectively. The latest version of Turbo Pascal (5.5) also is extended to support object oriented programming, generally along the lines of the way in which C++ extends C. Therefore, all other things being equal, a C programmer is most likely to feel comfortable with C++ and an Ada, Pascal, or Modula-2 programmer is most likely to feel comfortable with Turbo Pascal 5.5 or MODSIM II.

4. EXPERIMENTS

We wanted to estimate the degree of difficulty we can expect if we use each of these languages to develop simulation and simulation-support programs. We wanted to find out how well a detailed object oriented design can guide efforts to implement programs in several object oriented languages. Prior to performing the experiments, we each were experienced Smalltalk-80 and Objective-C programmers. MODSIM II and C++ were new languages to us. We read all accompanying documentation on these two new languages before starting on these experiments.

All the experiments were done on identical 386-based IBM-compatible computers, running under MS-DOS at 25MHz. In order to do the port from Smalltalk-80, the same bottom-up implementation pattern would be used in each case, as follows.

1. Implement the user interface classes, if needed
2. Add Simulator, if needed
3. Add Stationary Simulation Object and submenus.
4. Code and run the example executable simulation model.

In order to save time, we did parts of this in each language and used our experiences to extrapolate figures on effort to implement the entire program. We implemented the necessary user interface objects in Objective-C and C++ and coded and ran the example simulation model in Smalltalk/V 286 and MODSIM II.

5. OUTCOMES

5.1 Ease of Program Construction

We expected Smalltalk/V 286 and Zortech C++ do the best job, overall, of supporting program construction, since they have the best programming support environments and most extensive libraries. We did not expect MODSIM II and Objective-C to be as good. Their libraries are limited and the need for three steps (translate, compile, link) in order to get executable programs from source code slows down the pace of work.

In fact, we found that MODSIM II, in its present version, was as good as C++ in speeding software development, and that the new release is likely to be better. We found that manually porting the portion of SmartSim from Smalltalk-80 to our four target languages takes 26 hours for Smalltalk/V 286, 105 hours for MODSIM II, 171 hours for C++ and 257 hours for Objective-C. We estimate that when Simgraphics becomes available for use with MODSIM II, this time will be cut from 105 hours to 72 hours.

The numbers presented above are only estimates, based on extrapolations from two programmers' experiences in porting portions of one program. We feel that at most the numbers indicate that this sort of programming requires similar effort in MODSIM II and C++, greater effort in Objective-C, and somewhat less effort in Smalltalk/V 286. One reason why the port to Smalltalk/V 286 took so little time is that Smalltalk/V and Smalltalk-80 are very similar languages.

The surprising degree of suitability of MODSIM II could be attributed to at least three factors. One is that MODSIM II comes with a very powerful intelligent compiler that manages the entire process of translating, compiling, and linking. It also does configuration management, much as a Unix or C "make" utility does, but it creates and maintains its own "make file" automatically. Compilation does in fact take a long time, but the compiler automates a lot of tasks that usually fall to the programmer. This makes it particularly useful to support programming teams.

Secondly, the MODSIM II language is well designed and well documented. The language elements are familiar to anyone who knows a language like Pascal, C, or Ada and who is familiar with the concepts of object oriented programming. The language elements usually work in just the way a person with that background would expect them to. About 85% of what we needed to know in order to write our program in MODSIM II we learned just by reading the tutorial. Finally, both the compiler and runtime error messages pinpointed precisely which statements in which modules were causing errors.

We had expected to be able to port our work to Objective-C much more easily than we did. Objective-C is designed to incorporate the best features of both Smalltalk and C and some important simulation work is being done in that language [Najmi and Lozinski 1989]. However, Objective-C for MS-DOS has very little programming support. In particular, it does not help the programmer very much in dealing with dynamic memory allocation problems. Also, the libraries that come with the system contain only collection types of data structures. One must even build routines to mix text and graphics, since these are not provided in the Microsoft C subroutine library.

Since MODSIM II is the only one of the languages that directly supports simulation modeling, we expected step 4 of the experiments to be easiest to do using that language. However, the Smalltalk-80 simulation was event oriented and proved to be rather difficult to recast that in the process orientation that MODSIM II supports. For this reason, even though the Smalltalk/V 286 port required writing of classes for event handling and random number generation which are already supported in MODSIM II, step 4 for

Smalltalk/V 286 was done in half as much time as for MODSIM II.

5.2 Execution Efficiency of Simulation Programs

We ran the simulation for 600,000 seconds of simulated time in Smalltalk/V 286 and in MODSIM II. One would expect that, because Smalltalk/V 286 is interpreted and MODSIM II produces a .EXE file, the program run in Smalltalk/V 286 would run more slowly. In fact, the Smalltalk/V 286 version finished in 20 minutes, while the MODSIM II version required 38 minutes. Similarly, Doyle [1990] found that a Simscript II.5 version of a simulation program required 56% more time to run than a Smalltalk/V 286 version on an AT-286 machine.

In our earlier study [Thomasma and Ulgen 1988] we found that a Smalltalk/V version of a simulation program required three times a long to run as a SIMAN version. Smalltalk/V is an early version of Smalltalk/V 286. In order to see how much faster Smalltalk/V 286 is than Smalltalk/V, we ran our Smalltalk/V 286 example in Smalltalk/V on the same AT-386 machine that we used for the MODSIM II and Smalltalk/V 286 timings. The Smalltalk/V run required 46 minutes. If we give SIMAN a rank of 1.0, then, based on these results, Smalltalk/V 286 has rank 1.33, MODSIM II has rank 2.5, and Smalltalk/V has rank 3.0. One should note that this is the first release of MODSIM II and later releases are likely to be faster. Also, the SIMAN that was involved in the 1988 study is not the most recent version of SIMAN.

It is becoming clear that one need not pay a significant run time penalty in order to use object oriented programming languages in order to build simulation programs. The C++ language has many features that allow the programmer to develop programs that run very fast. If one uses that data reported by Doyle [1990], Zortech C++ would be given a rank of 0.15.

The performance of Smalltalk systems has improved markedly in the last two years. Programs written in the Smalltalk/V 286 interpreter have been observed to run faster than equivalent programs produced by compilers. The latest Smalltalk systems (ObjectWorks for Smalltalk-80 by ParcPlace and Smalltalk/V PM by Digitalk) produce compiled code.

All four of these languages (Objective-C, MODSIM II, Smalltalk/V 286, and Zortech C++) are quite reliable. Once a detailed design is done and a prototype is built, it is fairly straightforward to reimplement it in any one of these programming languages. The choice of language to use is dependent on which syntax one feels most comfortable with, the appropriateness to the job at hand of the classes in the libraries that one has available, and the quality of the programming support environment that is available. Run-time efficiency is a secondary concern. All the object oriented programming languages produce programs that run as efficiently as programs written in corresponding non-object-oriented languages, and new releases of the object oriented programming languages are continually improving in this respect.

REFERENCES

Alasuvanto, J., E. Eloranta, M. Fuyuki, T. Kida, and I. Inoue (1988), "Object Oriented Programming in Production Management: Two Pilot Systems," *International Journal of Production Research 26*, 5, 765-776.

Birtwistle, G.M., O.J. Dahl, B. Myhrhaug, and K. Nygaard (1979), *Simula BEGIN*, Second Edition, Studentlitteratur, Lund.

Doyle, R.J. (1990), "Object-Oriented Simulation Programming," In *Object Oriented Simulation*, A. Guasch, Ed. Society for Computer Simulation, San Diego, CA, 1-6.

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

Grant, M.E. and D.W. Starks (1988), "A Tutorial on TESS: The Extended Simulation Support System," In *Proceedings of 1988 Winter Simulation Conference*, M.A. Abrams, P.L. Haigh, and J.C. Comfort, Eds. IEEE, Piscataway, NJ, 136-140.

Herring, C. (1990), "ModSim: A New Object-Oriented Simulation Language," In *Object Oriented Simulation*, A. Guasch, Ed. Society for Computer Simulation, San Diego, CA, 55-60.

Law, A.M. and S.G. Vincent (1988), "A Tutorial on UNIFIT: An Interactive Computer Package for Fitting Probability Distributions to Observed Data," In *Proceedings of 1988 Winter Simulation Conference*, M.A. Abrams, P.L. Haigh, and J.C. Comfort, Eds. IEEE, Piscataway, NJ, 188-193.

Najmi, A. and C. Lozinski (1989), "Managing Factory Productivity Using Object-Oriented Simulation for Setting Shiftly Production Targets in VLSI Manufacturing," In *Proceedings of AUTOFACT*, SME, Dearborn, MI, 3-1 through 3-14.

Poorte, J.P. and D.A. (1989), "Computer Animation with CINEMA," In *Proceedings of 1989 Winter Simulation Conference*, E.A. MacNair, K.J. Musselman, and P. Heidelberger, Eds. IEEE, Piscataway, NJ, 147-154.

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

Ulgen, O.M. and T. Thomasma (1989), "Computer Simulation Modeling in the Hands of Decision-Makers," *Simulation and AI, 1989*, W. Webster, Ed. Society for Computer Simulation, San Diego, CA, 89-95.