Proceedings of the 1987 Winter Simulation Conference
A. Thesen, H. Grant, W. David Kelton (eds.)

MODELING OF A MANUFACTURING CELL USING A GRAPHICAL SIMULATION SYSTEM BASED ON SMALLTALK-80

Timothy Thomasma
Onur M. Ulgen

Department of Industrial and Systems Engineering
University of Michigan - Dearborn
4901 Evergreen Rd., Dearborn Mi 48128
(313) 593-5361

and

Production Modeling Corporation
214 Devonshire Road
Ann Arbor, Michigan 48104
(313) 761-2750

ABSTRACT

A simulation model of a manufacturing cell is constructed in order to test the accuracy of an analysis of the cell done without simulation. The analysis does a good job of ranking alternative designs according to throughput, but overestimates the throughput by about fifteen percent. The model is built using a simulation system written in Smalltalk-80 that features icon-based programming and animation. The experience of using the simulation system to build this particular model is described.

1. ANALYSIS AND SIMULATION IN DESIGN OF ROBOT CELLS

Simulation is widely considered to be the most useful tool in analysis and design of complex manufacturing systems. There are difficulties with its use, however, which have led manufacturing engineers to take a second look at analytic techniques (Suri and Hildebrant 1984). Simulation models are usually difficult to construct and modify, which inhibits their use in evaluating a large number of design alternatives. Recently, several software packages have been developed which make it easy to build simulation models of a variety of manufacturing systems. Once the models are built, however, using them still requires a great deal of time for gathering output and doing statistical analysis. A tool like MANUPLAN (Suri and Diehl 1985) that is based on queueing theory enables users to construct and run models quickly and requires no statistical analysis of the output. Some companies have used it instead of simulation in the early stages of design efforts to identify promising alternatives to be studied using detailed simulations (Haider, Noller and Robey 1986).

Flexible manufacturing systems usually have features that cannot be modeled using MANUPLAN or the newer icon-based factory simulation systems. Their performance is highly dependent on their material handling systems, which cannot be easily characterized by a set of parameters. Therefore, in order to study these systems, detailed simulation models are usually built in traditional simulation languages, or using physical models (Godziela 1986, Diesch and Malstrom 1985), primarily to study the material handling system and its interaction with other system components.

This paper concerns the study of a relatively simple manufacturing cell consisting of three grinding machines and a gauge served by a dual-bridge overhead gantry robot. Only one part type is served in the system and the only sources of variability are the random breakdowns of the grinding machine, a slight error in the gauge's measurements, and, for each part, a 0.2% probability that it will be out of specification. All processing times, travel times, loading time, etc. are deterministic. An attempt is made to balance the system and predict its throughput using an analytic model. Then the system is simulated in detail. If the simulation corresponds to the results of the analytic model, then there is reason to hope that analytic models may be as useful in initial design of flexible manufacturing systems as they are for other systems. The simulation is built using Smalltalk-80 to illustrate the advantages that object-oriented simulation has for this sort of modeling.

2. A ROBOT CELL

The manufacturing cell under study is sketched in Figure 1. The diagrams in Figure 2 indicate routes along which the robots move and the exact locations of the points they visit. There is a diagram for each of the two scenarios that are to be studied.

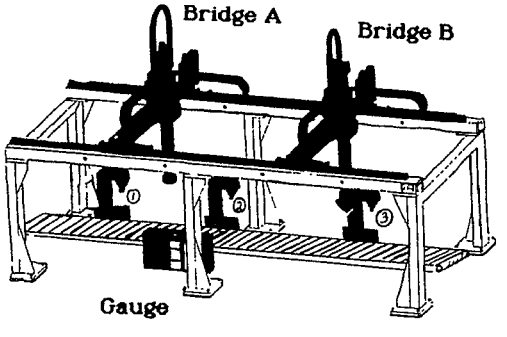
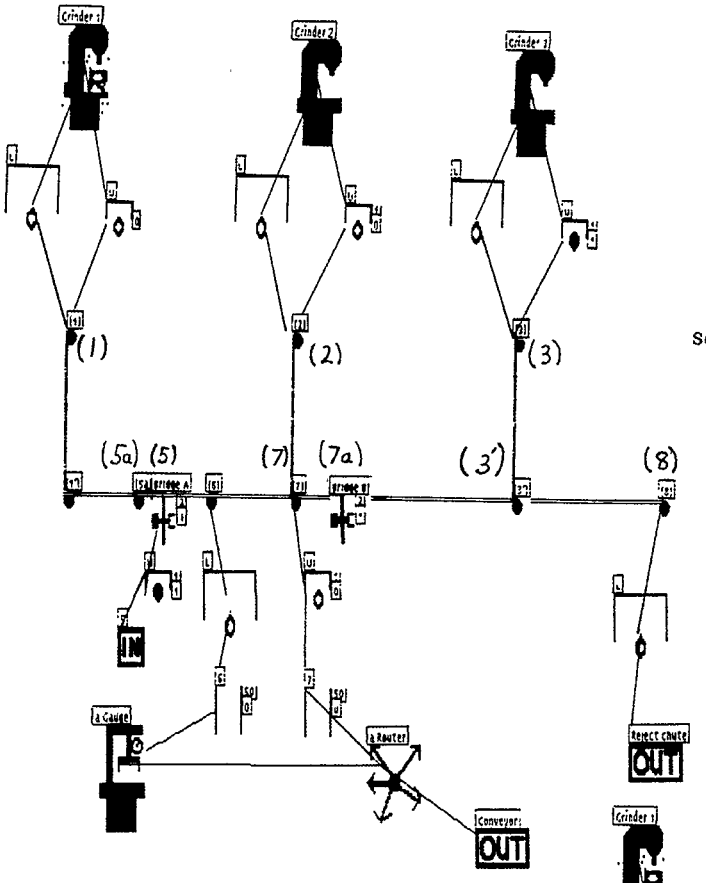
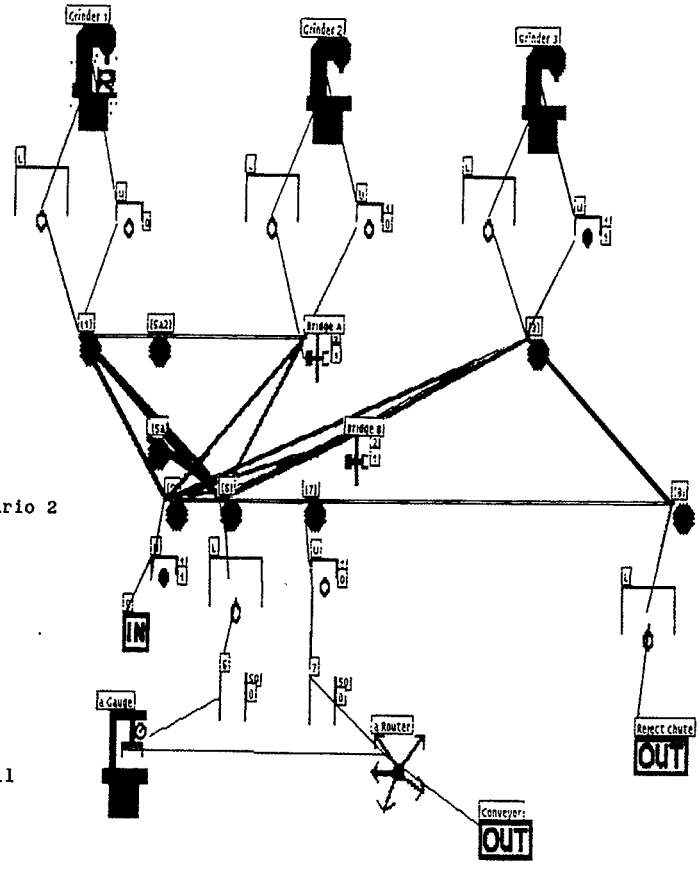


Figure 1: Sketch of Robot Cell



Scenario 1



Scenario 2

Figure 2: Diagrams of Routes of Robots in Cell

The rough parts arrive in the cell at location (5). It is assumed that there is always a rough part available at location (5). Any of the grinding machines can serve the rough parts. Bridge A takes rough parts to machines (1) and (2), alternately. If a machine is down, Bridge A takes the part to the available machine. If both machines (1) and (2) are down, Bridge A waits at location (5a) for the first available machine.

Bridge B takes rough parts to machine (3). Bridge B also picks rejects from location (7) and delivers them to location (8). If machine (3) is down and there are no rejects at location (7), then Bridge B waits at location (3').

The "interference area" is the area between locations (5a) and (7a). Bridge A cannot move into it unless Bridge B is outside it or moving out of it and Bridge B cannot move into it unless Bridge A is outside it or moving out of it. Location (6) in the interference area is the location where finished parts are dropped by the bridges. Whenever a bridge takes a rough part to a machine, it picks up the finished part and delivers it to location (6) to be gauged.

The gauging time is 12 seconds. If a part does not appear to meet specification on the first measurement, it is measured again. About 5% of the parts that are measured twice (which requires a total of 24 seconds) are found to be good on the second measurement. The other 95% are rejected as defective. The reject rate for the manufacturing cell is 0.2%. Defective parts are produced completely at random, except that 75% of them occur individually, 20% occur in doubles, and 5% occur in triples.

Each machine can process one part at a time and has a constant cycle time. Each machine gets cleaned for 111 seconds after it processes ten parts. Cleaning takes place as soon as the tenth cycle is completed, whether or not the finished part has been unloaded. Unloading and loading of the machine can be done while it is being cleaned. The machines break down randomly. They are down about 10% of the time. At the end of each eight-hour shift, any machines that are down are repaired in time for the start of the next shift.

3. ANALYSIS

From the results reported by Diesch and Malstrom (1985) it appears that the effect of the efficiency of the material handling system in a flexible manufacturing system is nearly independent of the effects of efficiencies of the machines in the system. If that is true of our robot cell, then the throughput of the cell can be found as the minimum throughput allowed by either the material handling system, the grinding operation or the gauging operation.

Normally, the measurement operation requires 12 seconds. Defective parts are measured twice for a total of 24 seconds. In addition, some good parts are measured twice, since 5% of those parts measured twice are found to be good on second measurement. Therefore, out of every 9500 parts, an average of 19 (0.2%) will be defective and, on an average, one good part will be measured twice. The average time required to measure these 9500 parts is

$$12*9500+12*20 = 114240$$

seconds. In this time 9481 good parts will be produced. The expected throughput of the gauging operation is therefore

$$(9481/114240)*3600 = 298.77$$

good parts per hour.

The average throughput for the grinding operation depends on the cycle time c. The time required to process ten parts on one machine, assuming no downtime, is

$$10c + (\text{loading-unloading time}) + (\text{cleaning time})$$

$$= 10c + 9*6 + 111$$
$$= 10c + 165$$

seconds. Loading-unloading time is counted only nine times because one unload-load is done during each cleaning cycle. If downtime d is considered, then the time t required to process 10 parts is

$$t = 10c + 165 + d.$$

On average,

$$d = (1/10)\ t.$$

Therefore the expected time to process 10 parts is

$$(100c + 1650)/9$$

seconds. Expected throughput is

$$6480/(2c + 33)$$

parts per hour, including 0.2% defective parts. Expected throughput of good parts for three machines, then, is

$$(.998)*3*6480/(2c + 33) = (19401.12)/(2c + 33)$$

parts per hour.

The throughput of the system cannot be more than the maximum of 298.77 and (19401.12)/(2c + 33). The maximum cycle time which should give the highest throughput allowed by the gauging operation is obtained by solving the equation

$$298.77 = (19401.12)/(2c + 33)$$

for c; that is c = 15.98 seconds

The time required by the robots for moving and loading-unloading parts is dependent on the control logic for the robot system. We investigate the two scenarios in Figures 3 and 4. Positions of the robots are plotted against time in these figures for the case when all three machines are operating and there are no defective parts to move. The control logic for the robots is the same in both scenarios. In Figure 3 the robots are constrained to move either parallel or perpendicular to the conveyor. In Figure 4 diagonal motion is allowed.

These two scenarios were chosen to test the predictive accuracy of the analysis done in this section. The only difference between the two is the difference in motion of the robot bridges. There is good reason to believe that the difference between direct diagonal motions and motion that is constrained parallel or perpendicular to the conveyor should make a noticable difference in material handling efficiency

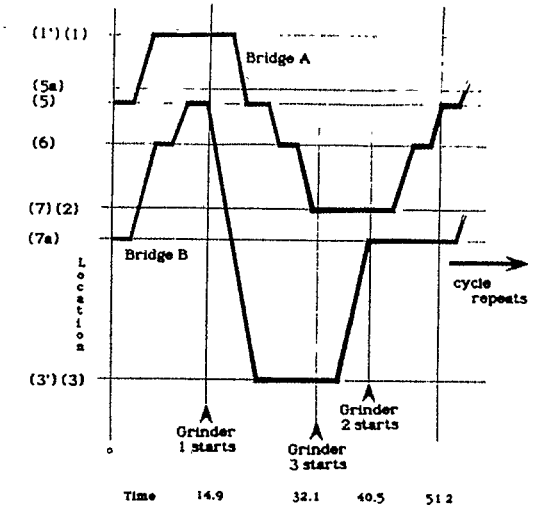


Figure 3: Robot Time - Position Plot for Scenario 1

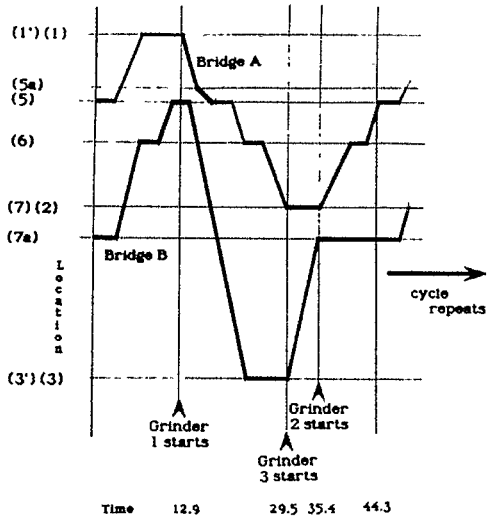


Figure 4: Robot Time - Position Plot for Scenario 2

between the two scenarios: the material handling of scenario 2 should be more efficient than that of scenario 1. One would hope that the analysis would predict the throughput well for each scenario. At least one hopes that the analysis will pick the scenario with the higher throuhput, since analysis that provides accurate ranking of design alternatives, even if it does not accurately predict throughput, would still be useful in identifying the best alternatives as candidates for detailed simulation studies.

In the first scenario, three parts can be moved every 51.1 seconds for a throughput of 211 parts per hour, assuming none of the three machines are down. Diagrams similar to Figure 3 can be drawn for each of the cases when one or more machines are down. When calculations are done for each case and weighted by its case's probability of occurring, throughput is found to be 196 parts per hour. A similar calculation for the second scenario gives 224 parts per hour.

On the basis of these calculations, we expect that at a cycle time of 16, the cell's throughput will be bound by the material handling, with throughputs of 196 and 224, respectively, for scenarios 1 and 2. The cycle time that will balance the system for either scenario depends on how many machines are working. Again, taking a weighted average, we find that the balancing cycle times are 37 for scenario 1 and 31 for scenario 2. If we run the machines at a cycle time of 41 in scenario 1, we expect throughput to be bound by the machine operation, and we expect it to be

$$(19041.12)/(2*41 + 33) = 169$$

parts per hour. Similarly, if we run the machines in scenario 2 at a cycle time greater than 31, say at 34, we expect throughput to be

$$(19401.12)/2*34 + 33) = 192.$$

In the sections that follow, we use a simulation model of the manufacturing cell to test the above calculations. We will make four sets of runs:

1) Scenario 1 with cycle time 16. Expected throughput: 196
2) Scenario 2 with cycle time 16. Expected throughput: 224
3) Scenario 1 with cycle time 41. Expected throughput: 169
4) Scenario 2 with cycle time 34. Expected throughput: 192.

4. CONSTRUCTION OF SIMULATION MODEL

The graphical simulation system used in modeling the manufacturing cell was written in the Smalltalk-80 object-oriented programming language (Goldberg and Robson 1983). The system enables the user to develop much of the model in a programming-free environment. The input to the simulation is based on the graphic layout of the model and the user's settings of a number of parameters. The icons used in the graphic layout represent the actual system components (robot, machine, gauge, storage space). Each icon has an interactive multi-level menu system to describe the characteristics of the actual system component being represented by that icon. The animation of the simulated system is also an integral part of this graphical package. More details concerning the system can be found in Ulgen and Thomasma (1987).

Since the system is modular, it is easy to quickly construct models with a variety of alternative configurations of icons. The library icons can be augmented as necessary to support additional operations or machines. These advantages of modular systems were noted by Medeiros and Sadowski (1983). Because of our system's graphical interface and animation capability, and because it was written in an object-oriented language, the simulation models of our robot cell could be developed and verified especially quickly and easily.

Table: 1: Class SourceAlwaysHasParts

```
File: SourceAlwaysHasParts.pp

From Smalltalk-80 version T2.1.3, of March 13, 1986 on 13 August 1987 at 8:49:18 am

Source subclass: #SourceAlwaysHasParts
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'Simulation Objects'

SourceAlwaysHasParts comment: ''


SourceAlwaysHasParts methodsFor: 'model definition

runBy: aSimulator
        "An arrival event must be scheduled at time 0 in order to get the simulation started.
        TT 8/3/87"

        super runBy: aSimulator.
        itsSimulator scheduleEvent: self at: 0.0 task: 1


SourceAlwaysHasParts methodsFor: 'tasks'

moveit
        "A SourceAlwaysHasParts always has a part to move to the next object.
        TT 8/3/87"

        self arrives
```

Several icons could be used without modification: the Sinks, Storage Facilities for queues of parts to be gauged and for rejects, and a Router to decide whether a part should go out of the system or to the reject queue. The Source provided in the system would not model the condition that rough parts are always available at location (5). A new class of objects, called "SourceAlwaysHasParts", was created as a subclass of Source to model this feature. All the code for class SourceAlwaysHasParts is listed in Table 1. The methods "runBy:" and "moveit" are present in Source, but behave differently than desired. The methods in Table 1 override Source's "runby:" and "moveit" methods. All the other methods in Source for graphics, animation, modeling of transfer of parts to other objects, etc. are inherited by SourceAlwaysHasParts and did not need to be rewritten.

In the same way, a subclass of Workstation had to be made to model the gauge's behavior that the time required to do an operation is conditional on whether the part is good or defective, and to model the gauge's error rate on first measurement. Again, only two methods from Workstation needed rewriting. The "accept:" method only required the modification indicated in the box in Table 2. Thus the "accept:" method could be copied from Workstation to Gauge and Gauge's copy modified. Once class Gauge was created, it was very easy to test it using the simulation laid out in Figure 5. A general impression of the correctness of the Gauge could be obtained by watching the animation of this small simulation. Its trace and output statistics were also checked for correctness and consistency.

In order to model the grinding machines, another subclass of Workstation, called "Grinder", was written. Grinder is more complicated than Gauge. It was also written as a subclass of Workstation, but instead of requiring rewriting of two methods, thirteen methods had to be rewritten and eight had to be added. This was much simpler than rewriting an

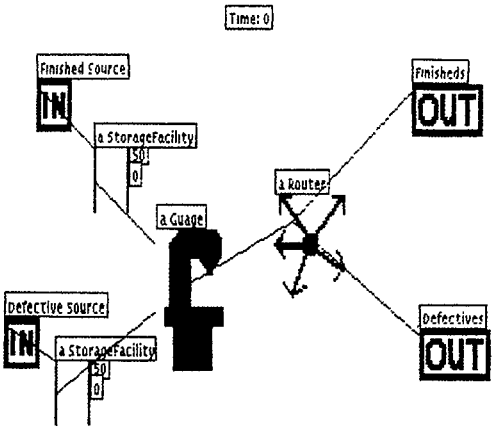


Figure 5: Simulation to Test Gauge

entire new version of Workstation, which has over 40 methods. The method classification structure in Smalltalk-80 made even this larger job fairly simple to do. Pieces of code to be modified could be found very quickly and easily. Existing code could often be easily found to use as templates for the new methods. Two experimental simulations were run to verify and validate the Grinder: one (Figure 6) to test the downtime distribution, including the behavior that Grinders are completely repaired at the end of each shift, and the second (Figure 7) to test the distribution of defective part production. Writing, testing and debugging the Grinder took about ten hours, including the time taken to run the test simulations.

Table: 2: Class Gauge

```
File: Gauge.pp

From Smalltalk-80 version T2.1.3, of March 13, 1986 on 20 August 1987 at 7:55:21 am

Workstation subclass: #Gauge
        instanceVariableNames: 'rejectProbability '
        classVariableNames: ''
        poolDictionaries: ''
        category: Simulation Objects'

Gauge comment: ''


Gauge methodsFor: 'initialization'

initialize
        "TT 6/30/87"

        super initialize.
        rejectProbability ← Bernoulli parameter: 0.0001054741


Gauge methodsFor: 'tasks'

accept: aPart
        "TT 8/7/87"

        | time n wname |
        (cantHandles includes: ((aPart partType) name)) ifTrue: [ffalse].
        (state = 'idle') ifTrue: [ "Part can be accepted."
                time ← itsSimulator simTime.
                workpiece ← aPart.

                "Update statistics."
                (traceOn) ifTrue: [ itsSimulator writeEvent: label, ' receives ', (workpiece fullLabel) ].
                waitingToShowAccept ← true.
                numberReceived ← numberReceived + 1.
                idleTime ← idleTime + (time - timeLastStateChange).
                timeLastStateChange ← time.

                "Schedule the time when the processing will be done."
                (processingProbability = nil) ifTrue: [ whenDone ← nil ] ifFalse: [
                        n ← processingProbability next.
                        wname ← (workpiece partType) name.
                        (wname = 'Defective') ifTrue: [n ← n*2 ].
                        (wname = 'Finished') ifTrue: [
                                ((rejectProbability next) = 1) ifTrue: [ n ← n*2 ]
                                ].
                        ].
                (n < 0) ifTrue: [ whenDone ← time ] ifFalse: [ whenDone ← (time + n) ].
                itsSimulator scheduleEvent: self at: whenDone task: 2.
                state ← 'working'.
                (whenBreaks = nil) ifFalse: [ (time > whenBreaks) ifTrue: [ self breakdown ]].
                ftrue
                ]

        ifFalse: [ ffalse ]
```

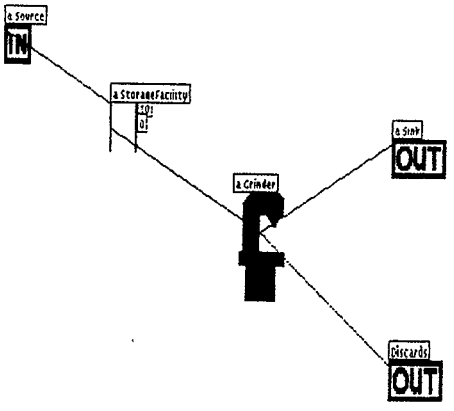


Figure 6: Simulation to Test Downtime in Grinder

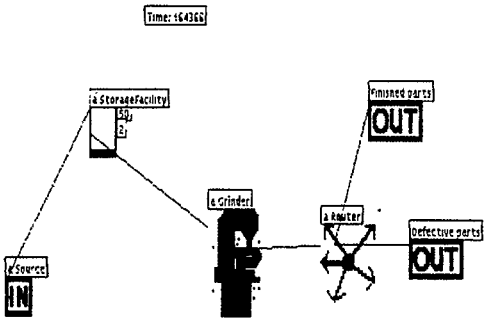


Figure 7: Simulation to Test Defective Part Production

Table 3: Class RealSimpleRobot

```
File: RealSimpleRobot.pp

From Smalltalk-80 version T2.1.3, of March 13, 1986 on 20 August 1987 at 7:14:12 am

MaterialHandler subclass: #RealSimpleRobot
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'Simulation Objects'

RealSimpleRobot comment: ''


RealSimpleRobot methodsFor: 'initialization'

initialize: aSimulator
        "The order in which these things are done is important!
        TT 6/22/87"

        | sp1 sp2 car1 |
        super initialize: aSimulator.

"Step 1:  Define the StopPoints."
        sp1 ← StopPoint new: aSimulator.
        sp1 label: '(1)'.
        sp2 ← StopPoint new: aSimulator.
        sp2 label: '(2)'.

"Step 2:  Define the routes out of each StopPoint."
        sp1 addRoute: sp2 acceleration: 1.45 deceleration: 1.45 velocity: 1.1.
        sp2 addRoute: sp1 acceleration: 1.45 deceleration: 1.45 velocity: 1.1.

"Step 3:  Put the StopPoints into the dictionary 'stopPoints'."
        stopPoints at: '(1)' put: sp1.
        stopPoints at: '(2)' put: sp2.

        actionsAtStopPoints at: '(1)' put: [ self actionsAt1 ].
        actionsAtStopPoints at: '(2)' put: [ self actionsAt2 ].

"Step 4:  Define the Carriers and put them into the dictionary 'carriers'."
        car1 ← (Carrier new: aSimulator).
        car1 label: 'Real Simple Robot'; isAt: (stopPoints at: '(1)'); acceleration: 1.45; deceleration: 1.45;
          velocity: 1.1; handledBy: self.
        carriers at: 'Real Simple Robot' put: car1.

"Step 5:  have each of the carriers arrive at the StopPoints that they are to be initially located at."
        self arrivalOf: car1


RealSimpleRobot methodsFor: 'actions at StopPoints'

actionsAt1
        "TT 6/23/87"

        | car sp |
        sp ← (stopPoints at: '(1)').
        car ← carriers at: 'Real Simple Robot'.
        (((car whereIsIt) = (sp label)) & ((car status) = 'idle')) ifTrue: [
                ((car numberIn) = 0) ifTrue: [
                        sp itsHere: car
                        ]
                ifFalse: [
                        sp release.
                        car startMoving: (stopPoints at: '(2)')
                        ]
                ]

actionsAt2
        "TT 8/4/87"

        | car sp |
        sp ← (stopPoints at: '(2)').
        car ← carriers at: 'Real Simple Robot'.
        (((car whereIsIt) = (sp label)) & ((car status) = 'idle')) ifTrue: [
                (car numberIn = 0) ifTrue: [
                        sp release.
                        car startMoving: (stopPoints at: '(1)').
                        ]
                ifFalse: [
                        sp itsHere: car
                        ]
                ]
```

The most difficult part of the model to build was the Gantry Robot. The system provides a class called "MaterialHandler", which is not an executable icon in itself. Subclasses of MaterialHandler must be made which define the material handling machines (carriers), set up the nodes and routes along which the carriers move and specify the behavior of each carrier at each node (stopPoint). As an example of a subclass of MaterialHandler, Table 3 lists the source code for RealSimpleRobot, which picks parts from

location (1), moves them to location (2), where it places them, and returns empty to location (1).

The source code listing for GantryRobot required fifteen pages. Of that, twelve pages defined the robot control logic, which required four pages to specify in English. Again, using Smalltalk-80 helped in that interactive debugging of the model could be done and time could be spent concentrating on the important parts of the model building effort, namely, the control logic for the robots. The animation provided by the simulation system was especially valuable in validating and verifying the control logic of the Gantry Robot system. Once Scenario 1 was complete, four hours were required to write, test, and debug the simulation model for Scenario 2.

5. OUTPUT ANALYSIS

The steady-state condition of this manufacturing cell is reached very quickly, almost at the beginning of its operation, just after the machines are first loaded by the robots. Since this is true, there is no advantage in using the batch method for output analysis. Each run was made using a different random number stream. The first ten minutes of simulated time were discarded for each run. For each alternative model four replications, each of 10 hours duration, were used to obtain a confidence interval for the system mean with an alpha level of 0.05. Table 4 gives the estimate of means and the corresponding confidence intervals for the replication method.

Besides the throughput of the system, which is the main concern, other statistics were also gathered to observe the behavior of the system. These included the percentages of times spent by the bridges in moving, waiting, or loading; times spent by each grinder in processing, being repaired, being cleaned, or waiting for delivery of parts; percent times idle and in use for the gauge; queue sizes at locations (6) and (7), number of defective parts produced and number of parts lost due to machine breakdown.

The observed throughtputs for each scenario and cycle time fell significantly below predicted times, although the designs giving highest and lowest throughputs were correctly chosen by the initial analysis. In every simulation run, well over 99% of the rough parts were successfully processed by the system; therefore we were justified in ignoring handling of defective parts in our analytic model. Our model correctly predicted all other statistics except the idle times for the grinders and the amount of time bridge A spends waiting either for Bridge B to leave the interference area or for one of the grinding machines to become ready to accept parts. In Scenario 1, c = 41 and in Scenario 2, c = 34, the analysis predicted that the system would be bound by the grinder's processing times, and therefore idle time for grinders was expected to be zero. In fact the percentages of time idle were those presented in Table 5. In the c = 16 cases, the analytic model predicted that Bridge A would spend less than 10% of its time waiting. In fact Bridge A waited 17% of the

Table 4: Throughput

Model / Experimental Condition	Throughput				
	Expected	Observed			
		Mean	Std. Dev.	Confidence interval L.L.	U.L.
Scenario 1					
Cycle Time:16	196	173	3.2	168	178
Cycle Time:41	169	137	2.1	134	140
Scenario 2					
Cycle Time:16	224	182	4.2	175	189
Cycle Time:34	192	166	7.2	155	177

Table 5: Percent Time Idle for Grinder

Model / Experimental Condition	Grinder 1	Grinder 2	Grinder 3	Average
Scenario 1				
Cycle Time:41	21	21	19	20
Scenario 2				
Cycle Time:34	12	12	11	12

time in Scenario 1 and 25% of the time in Scenario 2. The analytic model did not accurately model the cleaning cycles of the grinding machines. We consider this the most likely explanation for the discrepancy between the analysis and the simulation model.

The runs were done on a Tektronix 4405 workstation with 1 MB of memory running the standard Xerox Smalltalk-80 system. Each run took about one hour. Tektronix makes available an enhanced Smalltalk system for machines with more memory, which provides faster execution, among other advantages. Smalltalk is an interpreted language. Execution would be faster if the simulation program could be fully compiled once it were written, verified and validated using the interpretive environment.

6. CONCLUSIONS

It is possible, at least in simple cases, to evaluate alternative designs of flexible manufacturing cells and choose the most promising candidates without simulation. The candidates must then be studied using simulation in order to predict system throughput accurately. Computer programs to assist in preparing diagrams like those in Figures 3 and 4 would be very helpful in doing these initial analyses.

It is very easy to construct many simulation models or parts of models in an icon-based simulation system with a graphical user interface. Having an icon-based simulation system written in Smalltalk is particularly advantageous, since, if there aren't enough icons in the simulation system's icon library, they often can be easily written, added to the library, and placed in the simulation. The modularization encouraged by the icon library and the availability of interactive animation makes verification and debugging relatively easy. How difficult it will be to build a simulation model for any particular problem using a graphical simulation system based on Smalltalk depends on whether the system already has in it objects that are similar to the objects in the system that is to be modeled. For our particular study there was no existing object similar to the gantry robot system, which was a major component of our manufacturing cell. Therefore, the effort required to build the simulation in Smalltalk was similar to the effort required to write code for the simulation in a traditional simulation language. Now that the gantry robot system has been done in Smalltalk, however, it will be much easier to build simulation models of other robot cells.

REFERENCES

Diesch, K. H. and Malstrom, E. M. (1985). Physical simulator analyzes performance of flexible manufacturing systems. Industrial Engineering 17 (6), 66-75.

Godziela, R. (1986) Simulation of a flexible manufacturing cell. In: Proceedings of the 1986 Winter Simulation Conference (J.R. Wilson, J. O. Henriksen, S. D. Roberts, eds.). Institute of Electrical and Electronics Engineers, Washington D.C., 621-627.

Goldberg, A. and Robson, D. (1983). Smalltalk-80: The langauge and its Implementation, Addison Wesley, Reading, Mass.

Haider, S. W. , Noller, D. G., Robey, T. B. (1986) Experiences with analytic and simulation modeling for a factory of the future project at IBM. In: Proceedings of the 1986 Winter Simulation Conference, (J.R. Wilson, J. O. Henriksen, S. D. Roberts, eds.). Institute of Electrical and Electronics Engineers, Washington, D.C., 641-648.

Medeiros, D. J. and Sadowski, R. P. (1983). Simulation of robotic manufacturing cells: a modular approach Simulation 40, (1), 3-12.

Suri, R. and Diehl, B. W. (1985). MANUPLAN - a recursor to simulation for complex manufacturing systems. In: Proceedings of the 1985 Winter Simulation Conference, (D. T. Gantz, G. C. Blais, S. L. Solomon, eds.). Institute of Electrical and Electronics Engineers, San Francisco, California, 411-420.

Suri, R. and Hildebrant, R. R. (1984). Modeling flexible manufacturing systems using mean value analysis. Journal of Manufacturing Systems 3 (1), 27-38.

Ulgen, O. and Thomasma, T. (1987). Graphical Simulation using Smalltalk-80. In: Proceedings of the SAE/ESD International Computer Graphics Conference, (N. Spewock, E. D. Goodman, K. A. Kline, Eds.). Society of Automotive Engineers, Detroit, Michigan, 317-326.