Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

Bridging the gap: Transferring logic from a simulation into an actual system controller

Roger McHaney
Control Engineering Company
Harbor Springs, Michigan 49740

ABSTRACT

The use of simulation in the design stages of complex control systems has warranted the development of new methodology for transferring logic from model to actual system. This paper emphasizes the importance of maintaining simulation integrity during actual system implementation and presents four methods to aid in accomplishing this. These four methods of logic transfer are discussed and compared using the example of an AGV system simulation.

1.0 INTRODUCTION

Discrete event simulation in manufacturing has traditionally been used as a tool for studying the behavior of dynamic real-world systems (Law, 1986; Lajeunesse, 1984; Christy and Watson, 1983; Schriber, 1984). Some common applications use simulation to prove that proposed systems or system changes are feasible. Following development, validation, and verification of a simulation, some questions such as these can be answered :

How Many ?

a) How many machines, conveyors, pallets, carts, AGVs, and stands are needed for the system to operate.
b) How many men are required for desired production.
c) How many shifts should be operated.

Which ?

a) Which station is a bottleneck.
b) Which process flow is best.
c) Which schedule is best.
d) Which product mix is best.

What ?

a) What is the maximum through-put possible in the system.
b) What is the expected average through-put.

Another growing area of application for simulations is the modeling of control algorithms for complex materials handling systems. In these cases, it is essential that the actual system logic closely duplicates the logic used in the simulation. Otherwise, the simulation results will not be accurate and the actual system will not perform as expected. Often, after a model is developed and a proposed system is proven to work, the simulation phase is over. Control system development will then be turned over to a group of software design engineers who reformulate the control algorithms. The result can be changes in the operating characteristics of the system. Steps can be taken to minimize this potential pitfall and its effects. Examples of measures that can be taken to preserve the integrity of the algorithms developed with simulation are explored in this paper, as well as several different methods of transferring simulation logic.

2.0 PERFORMING A LOGIC TRANSFER

The simulations being dealt with in this paper are used as design tools. They are typically precursors to the actual system being built. It is the goal of the simulation team to identify all logic necessary to implement a controller for the system. In addition, the simulation team must effectively communicate its developed and debugged algorithms to the system design team. This insures that the required logic is incorporated into the actual controller. Four methods of facilitating logic transfer have been devised. These are *Philosophic Transfer*, *Pseudocode Transfer*, *Data Base Transfer*, and *Actual Code Transfer*. These four transfer

methods are not completely independent, rather they can be thought to exist on a continuum such as this :

It is possible to utilize a simulation logic transfer methodology that falls somewhere in between two categories on the continuum. To better understand what these categories entail, a detailed description of each is presented.

2.1 Philosophic Transfer

The most common and perhaps least efficient method of transporting simulation logic into actual system control logic is termed a philosophic transfer. Under this scenario, logic for the real-world system is based on the key ideas and assumptions used in the simulation. This information is presented to the system design team either verbally or in the form of a written report. The design team then evaluates the simulation findings and uses them as a guide or criterion in implementing the actual system software.

In many simple cases, the philosophic transfer works quite well. However, there are some inherent shortcomings that can cause inefficiencies. One of these inefficiencies is duplication of effort. The system software is designed, coded, debugged, and tested. This same process has already been used in the development of the simulation. Another potential shortcoming is the possibility that a piece of information was not communicated properly resulting in either a misinterpreted or omitted portion of logic. A subtle difference in logic can produce a discrepancy that renders the simulation results invalid. The philosophic transfer method tends to make simulation validation (Carson, 1986) a more complex task. Since the two softwares were developed in a somewhat independent fashion, the differences become harder to identify. Whether the simulation accurately depicts the actual system becomes a difficult question with an answer that is hard to prove.

The best way to insure that these problems do not occur when employing the philosophic transfer method is to form a system design team consisting of both simulators and software engineers. A common design can be arrived at and ideas can be tested with a simulation. When the design is finalized and the simulation is complete, the system software can be written. If the project has emphasized communication and structure, the transfer of the philosophy contained in the simulation should be successful. Duplication of effort has been minimized by doing the initial design work collectively. The ideal philosophic transfer approach to implementing a simulation in the real world is characterized by team work and communication.

2.2 Pseudocode Transfer

The second method of transferring data from a simulated controller to an actual controller is the pseudocode transfer method. This approach is very similar to the philosophic transfer but it takes the level of detail a step further. It evolved from the need to insure that all assumptions incorporated into a simulation were addressed and made apparent to the actual system design team. In this scenario, pseudocode (Page-Jones, 94-96) is generated by simulation personnel as a method of documenting what has been implemented. This pseudocode is analyzed and rewritten for the actual controller by system software engineers.

The pseudocode method of transfer offers the advantage of being more detailed. Therefore, the number of omissions and oversights occuring are expected to be fewer than would be seen when using the philosophic transfer method. Simulation validation is also made easier. The logic rules used in the control system can be examined and verified as being identical to those in the simulation. Differences are identified readily and the pseudocode provides a common record for both the simulation and system software package.

A disadvantage associated with using this method of transfer is duplication of programming effort. The pseudocode has to be rewritten in the language of the controller, debugged, and tested. Some of the simulation team's time is also required to rewrite their logic in the form of pseudocode.

Communication between the simulation team and the system design team is important under this transfer scenario; but not nearly as crucial as when using the philosophic transfer method. By placing a structured and detailed pseudocode document in the hands of the design team, little additional interaction will be required of the simulators. It is not until the verification and validation phase of the project, when the completed system software is compared to the simulation, that further communication will be required.

2.3 Data Base Transfer

The third method of transferring simulation logic to an actual system is known as a data base transfer. The essence of this transfer method is to transport a data base, developed in the simulation, and use it to drive the actual system controller. It is important to note that this method of data transfer is contingent upon software existing in both the simulation and controller. This requires that an initial development project must be undertaken to define the data base and delineate how it will be used. After this has been established and the simulation and controller software have been developed, the data base can be transferred and utilized. This type of transfer will most commonly be used in cases where many similar systems are being designed using a generic simulation and a generic controller.

The advantages to using this transfer method are quite apparent. The debugging of the data base and its testing are all done in the simulation phase. Nearly all duplication of programming effort is eliminated (after the initial system has been created). Changes to the simulation and actual system can be done easily and consistantly. Simulation validation and verification become much easier. In addition, communication and the passing of abstract concepts become less of an issue.

A potential drawback encountered when using the data base transfer method is the loss of flexibility. Although using a data base simplifies logic and provides the system engineers with a standard, unusual cases and exceptions become more difficult to incorporate. For this reason it is very important that the data bases be designed in a comprehensive manner.

2.4 Actual Code Transfer

The final method of data transfer to be examined is transporting actual code from simulation to system controller. This can be done either when the simulation has been written in a conventional language such as FORTRAN, or when using a specialized simulation language such as GPSS/H (Henriksen and Crane, 1983) or GPSS/PC (Cox, 1984) which has the capability of calling FORTRAN subroutines. These subroutines contain code that will be used in the system controller. The FORTRAN subroutines can be written either by the simulators or by the system design team prior to implementation in the simulation. Duplicated effort is reduced because the control logic is written only once in the simulation phase and then reused in the actual system.

Using this actual code transfer method requires that the simulators and system designers work very closely. It may be advantageous to form one team consisting of both simulation and design personnel. A disadvantage inherent when using this method is the requirement of a more concerted initial effort and additional time to debug the model. The simulator may not be familiar with the control algorithm and the system designer may not be familiar with the rest of the simulation. When the model is run and

debugged, some learning time might be required. However, testing and debugging the simulation is also testing and debugging the actual system. Little duplication of effort will occur when the finished software is transported to the actual system software.

3.0 TRANSFER METHOD COMPARISONS

The following chart compares two aspects of the four transfer methods previously described. These aspects are communication and duplication of effort. The first column in the chart refers to the level of communication that will be required to successfully implement each transfer method. The second column indicates the level of duplication of effort occurring between the simulation group and the system design team. These levels are rated on a scale from one to ten, with ten being the greatest amount of time required and one being the least. This scale is to be used only to compare the four methods of transfer and does not consider project complexity or any other issue.

Table 1 : Comparison of Transfer Methods

Transfer Method	Communication	Duplication of Effort
Philosophy	10	10
Pseudocode	6	5
Data Base	3	3
Actual Code	7	1

Communication is an important aspect in the success of any simulation. The author wishes to demonstrate that the time spent communicating information related to the system can be reduced using different methods of simulation transfer.

4.0 AN APPLICATION INVOLVING THE TRANSFER OF SIMULATION LOGIC

Most complex systems will use combinations of different logic transfer types within the same application. This is due to the nature of the logic being transferred, the importance of maintaining simulation integrity, and time constraints. One particular transfer method is not always superior to the others; they all perform specific functions and should be used in particular instances. In order to better illustrate logic transfers from simulation to actual controllers, a typical Automatic Guided Vehicle (AGV) system application will be referenced. The methodology used in this AGV simulation is very similar to what has been described in past literature (Harmonosky and Sadowski, 1984; Davis, 1986; Newton, 1985). It should be noted that in this AGV system example, several types of logic transfers will be used for different parts of the system logic. The optimal method of transferring simulation data is not being demonstrated in this paper. Instead a broad overview illustrating different transfer methods is presented.

4.1 What is an AGV System ?

AGVs are generally battery-powered, driverless vehicles that travel along paths which consist of wires buried in the floor. They are used for materials handling tasks. A central computer dispatches empty vehicles, optimizes routing, inhibits collisions, prevents system gridlocks, and provides empty vehicle management. Part of the mechanism for performing these tasks is magnet codes (or telsor cards) located in the floor at regular intervals and prior to intersections. These codes (also known as control points) maintain vehicle separation and identify vehicle location to the central computer. Vehicles will often have sonic or optic sensors to prevent collisions with path obstructions and other AGVs. Communications with the central computer will usually be implemented with radios or transmitted through the guidepath wires buried in the floor (Sadowski, 1987; Quinn, 1987).

Simulations of AGV systems consist of several major parts. These are:

1) Floor Logic
2) Vehicle Programming Algorithms
3) Through-put Modeling

The floor logic and vehicle programming algorithms are both areas of logic that need to be transferred from the simulation to the actual system controller. The third category, through-put modeling, is a constraint used to help derive a vehicle count. It is also instrumental in determining the floor logic and vehicle programming algorithms.

4.1.1 Floor Logic

Floor logic can be broken into three destinct categories: vehicle routing, control point layout, and gridlock avoidance. Each area represents some data which needs to be incorporated into the system controller.

4.1.2 Control Points

Control points are placed to facilitate vehicle movement throughout the system. The idea is to allow vehicles to travel as freely as possible and at the same time avoid collisions with other AGVs. The guidepath is reproduced as a system of control points connected by line segments. Vehicles move along one segment at a time. At each control point they receive instructions to either stop and wait for the segment (or intersection) ahead to clear, pick up or drop off a load, turn a corner, advance to a new destination, or simply continue straight ahead.

4.1.3 Routing

Vehicle routing typically will be logic associated with each guidepath intersection. This logic will enable the AGV to arrive at its destination in the most efficient manner. Most often this means taking the shortest path, however at times it may be advantageous to take a longer route through a less congested area.

4.1.4 Gridlock Avoidance

A gridlock occurs when vehicles occupy consecutive sections of guidepath which form a connecting loop structure. The result is that none of the vehicles can advance to the next location because another vehicle is occupying it. Figure #1 illustrates how a gridlock might look.

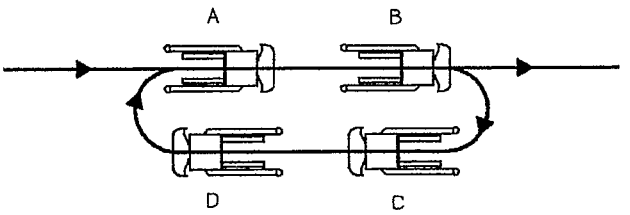

FIGURE #1 DEPICTING GRIDLOCK

AGV A MOVING TO POSITION OCCUPIED BY AGV B
AGV B MOVING TO POSITION OCCUPIED BY AGV C
AGV C MOVING TO POSITION OCCUPIED BY AGV D
AGV D MOVING TO POSITION OCCUPIED BY AGV A

Logic is added to the simulation to inhibit gridlock conditions. This logic may be of a preventative nature or may resolve the gridlock after it occurs. In either case, the logic must be documented and incorporated into the actual system.

4.1.5 Transferring the Floor Logic

In this example, the vehicle routing will be transferred using the philosophic method. The simulation team identifies any areas of guidepath that are congested, therefore warranting a longer route of travel as a bypass. If this condition exists, its presence will be relayed to the actual design team. In the absence of this condition, the simulation team will only communicate that the philosophy of choosing the shortest path between two points is the norm to which the system should be designed. The system engineers implement the vehicle routing logic based on this criterion.

Since control point placements are based on a documented convention, only the exceptions found to be necessary in the simulation are passed on to the system design team. Otherwise, a common philosophy that has been previously established will be independently employed by both teams.

The gridlock avoidance logic can be incorporated in the system controller using the pseudocode transfer method. The simulation team prepares pseudocode detailing the exact steps to be followed when the gridlock occurs. An example of pseudocode which represents resolution of a gridlock is as follows :

```
IF
   A VEHICLE IS PRESENT AT POINT CP100 AND IS
   TRAVELING TO POINT CP101
  AND
   A VEHICLE IS PRESENT AT POINT CP101 AND IS
   TRAVELING TO POINT CP102
  AND
   A VEHICLE IS PRESENT AT POINT CP102 AND IS
   TRAVELING TO POINT CP100
THEN
   CAUSE THE VEHICLE AT POINT CP102 TO TAKE
   AN ALTERNATE ROUTE AND GOTO POINT CP103
```

This pseudocode is then submitted to the system software designers for recoding and inclusion in their controller. As can be seen, the pseudocode is a detailed method of representing logic so it may be readily translated into virtually any application.

4.2 Vehicle Programming Algorithms

In order for an AGV system to provide peak service, it is crucial to develop an optimizing algorithm for moving the product. This algorithm usually involves two major areas of concern: dispatching vehicles to pick up loads and managing empty vehicles in a manner that minimizes deadhead (empty vehicle) travel. Often, these areas of concern will be represented in the form of certain logical guidelines. These guidelines are as follows :

4.2.1 Parking and Polling

Following completion of a load move, a vehicle at a drop-off point will check for any loads to be picked up at other qualifying stations. This process is called polling. If there are no loads ready to be picked up, the vehicle is assigned an intermediate parking location. Both the qualifying stations and the intermediate destination are represented in terms of a data base.

When a vehicle arrives at an intermediate parking location, polling of pick-up stations continues. If no loads are ready to be picked up, the vehicle either advances to the next intermediate parking location or remains at its current location until a load does qualify (Gray, 1988).

Chart #1 shows a typical vehicle programming data base. It is keyed by load drop-off locations and shows which pick-up stations qualify for service when a vehicle completes its drop. If no loads are ready to be picked up, the vehicle is dispatched to an intermediate parking location. This intermediate parking location is included as a field in each record.

CHART #1

Drop-off Location	Pick-up Locations Polled	Intermediate Parking Location
1 (battery)	**	50
4 (Slave)	4,3,2	51
5,6,7,8 (Ship)	4,2,3	51
15,16 (Pal Col)	14	60
17 (ASRS)	18,20,22	52
19 (ASRS)	20,22	52
21 (ASRS)	22	52
23 (Lift)	24,11,10	61
25 (Dye Pen)	25	58
26 (Prod)	26,27,28,29, 30,31,32	59
27 (Prod)	27,28,29,30, 31,32	59
28 (Prod)	28,29,30,31	59
29 (Prod)	29,30,31,32	59
30 (Prod)	30,31,32	57
31 (Prod)	31,32	57
32 (Prod)	32	57
33 (Dye Pen)	33	58
34 (Dye Pen)	34	55
35 (Prod)	35,34	55
36 (Prod & Dye Pen)	36,35,34	55
37 (Raw Matl)	37,36,35,34	55
38 (Raw Matl)	38,37,36,35, 34	55
39 (WIP)	39,40,41	54
40 (Dye Pen)	40,41,39	54
41 (Fin Prod)	41,40,39	54

** There are no pick-up stations between the drop-off location and the intermediate point. Therefore,the vehicle travels directly to the intermediate point before polling begins.

Chart #2 is a continuation of Chart #1 . It is keyed by each intermediate parking location. The pick-up stations which are polled during a vehicle's residence at its current location and the next intermediate parking location are included as records in this data base.

CHART #2

Parking Location	Pick-up Locations Polled	Next Intermediate Parking Location
51	18,20,22,25	52
52	24,10,11,37, 38,39,40,41, 35,36,34,33	58
53	39,40,41,38, 37,36,35,34	55
54	33	58
55	33	58
56	33	58
57	33	58
58	26,27,28,29, 30,31,32,14	60
59	30,31,32	57
60	12,13	61
61	18,20,22	52
62	**	63
63	**	64
64	4,3,2,20,18, 22,24,25,26, 27,29,33,30	*

** There are no pick-up stations between the parking location and the next intermediate parking location, therefore the vehicles travel to the next parking location before polling begins.

* Vehicles residing at location 64 will remain there until given a pick-up station destination.

4.2.2 Transfer of Vehicle Programming Algorithms

In an AGV system, vehicle programming can be represented in the form of data bases. Quite obviously, table driven logic developed in a simulation would be most efficiently transferred to the actual system in data base form. The actual control system would operate the AGV system using the rules developed in the simulation and represented by the data base. The advantages inherent in this data base transfer include the incorporation of completely debugged logic, ease of transfer, and reduction of duplication of effort.

4.3 Through-put Modeling

Although the through-put logic is never transferred from the simulation to the actual system controller, for the sake of completeness it will be briefly mentioned. Through-put is defined as the number of loads per unit time which must be picked up and delivered. In the simulation, through-put is used as a measuring stick to arrive at the number of vehicles required to achieve the desired results. It is possible to exercise the actual system controller by modifying this portion of simulation logic and using it to create a system emulator. This gives the system designers some insight concerning the amount of work being done on the factory floor.

5.0 SUMMARY

The widespread use of simulation in the design stages of complex control systems has warranted the development of new standards of communication and requires new techniques for the transfer of simulation logic. This has become an issue for several reasons. First, it is essential that simulation integrity be maintained in the actual control system. Second, without a logic transfer methodology, subtle differences between the simulation and real world system are prone to emerge. Third, duplication of effort can be minimized when using a logic transfer. This paper has listed four methods that can be used for transporting simulation logic into actual system software. Several of these methods and their advantages were further illustrated through the example of an AGV system.

REFERENCES

Carson, John S. 1986. "Convincing Users of a Model's Validity is Challenging Aspect of Modeler's Job." *Industrial Engineering* (June): 74-85.

Christy, David P. and Hugh J. Watson. 1983. "The Application of Simulation : A Survey of Industry Practice." *Interfaces* (Oct).

Cox, Springer. 1984. *GPSS/PC*. MINUTEMAN Software, Stow, MA.

Davis, Deborah A. 1986. "Modeling AGV Systems." *In Proc. 1986 Winter Simulation Conf.* (Washington D.C., Dec. 8-10). SCS, San Diego, Calif., 568-574.

Gray, Cheryl. 1988. *Generic Simulation Document.* Control Engineering Company, Harbor Springs, MI.

Harmonosky, Catherine M. and Randall P. Sadowski. 1984. "A Simulated Model and Analysis Integrating AGV's with Non-Automated Material Handling." *In Proc. 1984 Winter Simulation Conf.* (Dallas, Tex. Nov. 28-30). SCS, San Diego, Calif., 178-183.

Henriksen, James O. and Robert C. Crane. 1983. *GPSS/H User's Manual.* Wolverine Software Corporation, Annadale, Va.

Lajeunesse, Jim. 1984. "Eleven Steps To Simulating An Automated System." *Modern Materials Handling* (Nov 4) : 45-49.

Law, Averill M. 1986. "Introduction to Simulation : A Powerful tool For Analyzing Complex Manufacturing Systems." *Industrial Engineering* (May) : 46-63.

Newton, Dave. 1985. "Simulation Model Calculates How Many Automated Guided Vehicles Are Needed." *Industrial Engineering* (Feb) : 68-78.

Page-Jones, Meilir. 1980. *The Practical Guide to Structured Systems Design.* Yourdon Press, New York, N.Y.

Quinn, Edward B. 1987. "Discrete Event Computer Simulation of AGV Networks." *In Proc. AGVS 87.* (Pittsburg, PA., Oct 27-29), MHI, Charlotte, NC. 43-51.

Sadowski, Randall. 1987. "Computer Simulation of Automatic Guided Vehicles." *In Proc. AGVS 87.* (Pittsburg, PA., Oct 27-29), MHI, Charlotte, NC. 21-41.

Schriber, Thomas J. 1984. "A GPSS/H Model For A Hypothetical Flexible Manufacturing System." *In Proc. First ORSA/TIMS Special Interest Conference on Flexible Manufacturing Systems.* 168-182.

AUTHOR BIOGRAPHY

ROGER MCHANEY graduated Summa Cum Laude from Lake Superior State University in 1984 with a B.S. in Industrial Engineering Technology and an Associates degree in Computer Technology. He is currently a candidate for his M.B.A. degree at the same institution. In addition to being a part-time student, he is employed by Control Engineering Company (an affiliate of Jervis B. Webb Company) as a simulation analyst. In his five years there, he has developed a GPSS based software package to aid in simulating Automatic Guided Vehicle Systems and has used it to simulate over 100 AGV systems. His current interests include microcomputer applications of simulations and computer graphics.

Roger McHaney
Control Engineering Company
8212 Harbor Springs Road
Harbor Springs, MI 49740