Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

SCALABLE SIMULATION MODELS FOR CONSTRUCTION OPERATIONS

Photios G. Ioannou
Julio C. Martinez

Civil & Environmental Engineering Department
University of Michigan
Ann Arbor, Michigan 48109-2125, U.S.A.

ABSTRACT

Construction operations are often repetitive not only in terms of time (the same tasks are performed over and over) but also in terms of space (the same tasks are repeated at several places, such as different floors in a high-rise building). Thus, construction simulation models in general must be cyclic to represent temporal repetitions, but also scaleable to represent spatial repetitions. This paper presents the mechanisms for preprocessor replacement and automatic code generation that have been designed and implemented to facilitate the development of scaleable simulation models in STROBOSCOPE, a general-purpose discrete-event simulation system developed by the authors. A relatively complex simulation model for the vertical transportation of people serves as an example to illustrate how to develop a completely scaleable model for the operation of an elevator in a building with any number of floors.

1 INTRODUCTION

Construction operations are typically repetitive in nature and have traditionally been modeled using simulation systems based on cyclic activity networks. Thus, repetitive construction processes are most often represented as cycles through the same set of nodes of a simulation network.

There is an entire class of problems, however, for which this approach does not work. A typical example is the vertical transportation of people (e.g., construction labor) in a high-rise building (e.g., with more than 50 floors) using multiple elevators that may or may not span the entire height of the building. In such cases, the queuing behavior at each floor, although similar to that of other floors, needs to be modeled separately. For example, to collect statistics about the number of people waiting for elevator service at each floor and for each direction of travel, the model must include two queues per floor, one for the people waiting to go up and another for those waiting to go down. Thus, a building with 50 floors would require the arduous task of defining 98 queues (the 1st and 50th floors only need one queue) and at least 196 links (one link for entering and another for leaving each queue). Lumping all these queues into two global queues (one for all the people waiting to go up and one for all the people waiting to go down) trades this problem for another. While it eliminates the need to define multiple queues and links, it also prevents the automatic collection of any per-floor statistics. Thus, the problem now requires the definition of two statistics collectors per floor, while ensuring that these collectors receive the correct data every time a person enters or leaves the global waiting queues at a particular floor.

The same type of problem occurs when modeling a horizontal transportation system, such as the inter-terminal bus service at an airport. As the number of repetitive units increase (such as the floors in a building or the number of terminals at an airport), simulation models must be augmented with new modeling elements to represent the new floors or terminals.

A convenient solution to this type of problems is to create scaleable simulation models at the "meta" level that can create detailed simulation models at the simulation language level. Thus, a scaleable simulation model must have the capability to generate simulation code that defines any number of simulation elements (e.g., activities, queues, links, etc.) as necessary to match their real counterparts. This paper describes the facilities for statement preprocessing and automatic code generation provided within STROBOSCOPE (STate and ResOurce Based Simulation of COnstruction ProcEsses), a simulation language developed by the authors based on activity-scanning. The elevator problem is used as an example to illustrate these capabilities.

2 MODELING ELEVATOR OPERATIONS

The facilities for creating scaleable simulation models will be described by using as an example a building

served by a single elevator. This fairly complex problem has been adapted from (Law and Kelton 1991) and is used to illustrate the effectiveness of preprocessing and automatic code generation to create a scaleable model that once verified (and even animated) can provide an accurate representation of the elevator's complicated control logic for any number of floors.

The problem is of interest in high-rise building construction (50+ stories) as subcontractors add money to their bid to account for lost time in hoisting personnel. If at the bidding stage a general contractor can show the subcontractors a video (or an animation) and supporting documentation that the proposed hoisting system has been optimized and that waiting times will be as short as possible, there may be cost benefits to all parties (including the owner). In order to do this, project managers have expressed the need for models depicting the movement of labor that can provide queue-related statistics for different hoisting policies. In general, such models should be able to analyze and optimize the operation of multiple elevators and hoists (some of which may or may not span the entire building height) in very tall buildings.

In addition to being interesting, this problem was also selected because the "classical" elevator problem (even with simplified control logic) is one of the most difficult problems for simulation languages to model and verify. It has been estimated, for example, that the elevator problem (with much simpler control logic than the example described below) requires an expected modeling effort of 20-30 hours of work (Chisman 1996).

3 EXAMPLE PROBLEM STATEMENT

A five-story building is served by a six-person elevator. People arrive to the ground floor (floor 1) with independent exponential interarrival times having a mean of 1 minute. A person will go to each of the four upper floors with probability 0.25. It takes the elevator 15 seconds to travel up or down one floor. For simplicity it will be assumed that the elevator loading and unloading time at any floor is zero. The length of stay of a person at a particular floor is distributed uniformly between 15 and 120 minutes. When a person leaves floor i = 2, 3, 4, 5, he or she will go to floor 1 with probability 0.7, and will go to each of the other three floors with probability 0.1. A person coming down to floor 1 departs from the building immediately.

When the elevator is going up, it will continue in that direction if a current passenger wants to go to a higher floor or if a person on a higher floor wants to get on the elevator. When the elevator is going down, it will continue in that direction if it has at least one passenger or if there is a waiting passenger at a lower floor. When the elevator stops at floor i = 2, 3, 4 while going up (down), it picks up only those people at that floor that want to go up (down). If the arriving elevator does not have enough room to get all the people waiting at a particular floor, the excess remain in queue.

The elevator decides at each floor what floor it will go to next. It does not change directions between floors. At the start of the simulation the elevator is at rest at its base floor. This is the same floor the elevator returns to whenever it is idle. The best choice for the base floor is made by minimizing the average of individual delays over all floors and all people. An integral part of the modeling requirements for this problem is the collection of statistics about the number of people waiting and the corresponding waiting times at each floor and for each direction of travel.

4 SIMULATION METHODOLOGY

A scaleable simulation model for this problem has been developed using STROBOSCOPE, a general-purpose simulation programming language based on activity-scanning.

A complete description of STROBOSCOPE appears in (Martinez 1996). Example applications can be found in (Martinez, Ioannou, and Carr 1994; Martinez and Ioannou 1994, 1995; Ioannou and Martinez 1995, 1996a, 1996b). The STROBOSCOPE program, its documentation, and several examples are available via anonymous ftp from "grader.engin.umich.edu."

5 SIMULATION MODEL NETWORK

The STROBOSCOPE cyclic activity network for the elevator problem is shown in Figure 1. From a tactical point of view, this network can be divided into two parts.

The eight queues enclosed by the dashed lines (*Q1U*, *Q2U*, *Q3U*, *Q4U*, *Q2D*, *Q3D*, *Q4D*, and *Q5D*) as well as the 16 links that enter and depart these queues represent the scaleable portion of the network. The four queues ending in *U* represent the people waiting at the corresponding floor for the elevator to go up. Similarly, the four queues ending in *D* represent the people waiting for the elevator to go down. Thus, each of these queues is specific to a particular floor and a direction of movement (up or down). To increase the number of floors in the building it is necessary to define additional network elements: two queues and four links per floor.

The remainder of the simulation network represents the cycle of loading and moving the elevator from one floor to the next. The main network elements in this cycle are the activities *EnterElev* and *MoveElev*. Notice that this cycle is not specific to any particular floor, nor does it depend on the number of floors.

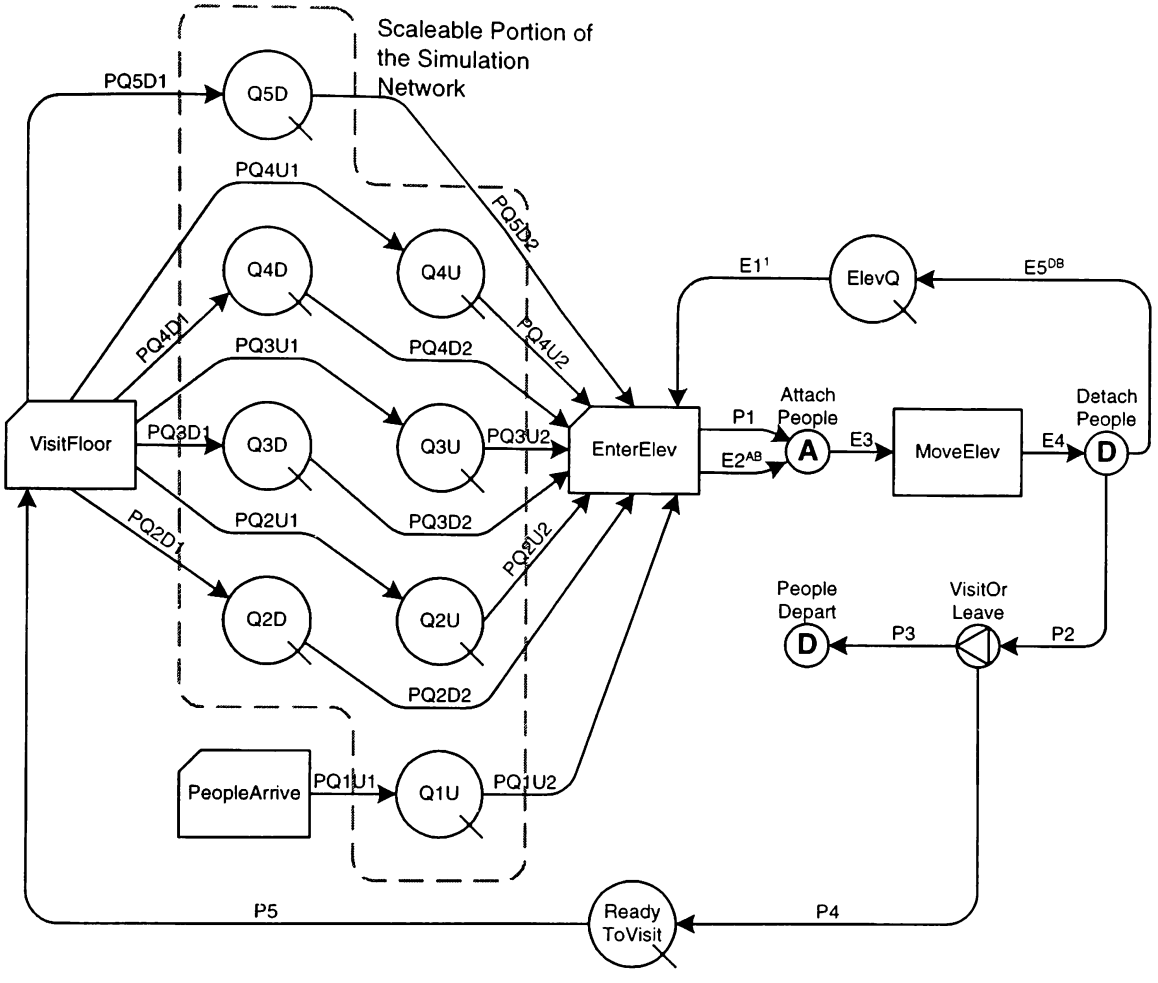

Figure 1: STROBOSCOPE Network for the Elevator Model

There are two types of resources in this model, the *elevator* and *people*. For convenience both are defined as compound characterized resources. Although people could have been modeled just as well as simple characterized resources with no static properties, compound characterized resources are easier to define in this case because there is no need to define subtypes. Furthermore, compound resources provide a base to which other resources (generic, simple characterized, or compound characterized) can be attached and detached. Thus, defining the elevator as a compound resource makes it extremely simple to model a loaded elevator as an elevator with people attached.

The attachment of people (a compound resource) to the elevator (another compound resource that serves as the assembly base) is performed by the assembly node *AttachPeople*. At the end of activity *MoveElev*, the people that have reached their destination and need to unload are detached from the elevator by the disassembly node *DetachPeople*. The rest of the people inside the elevator are in transit and remain attached to the elevator which is then routed to the queue *ElevQ* ready to start another cycle.

The people detached from the elevator are routed to either the disassembly node *PeopleDepart* (i.e., to floor 1 and out of the building), or to the queue *ReadyToVisit*. From there each person that exited the elevator starts a separate instance of the combi (conditional) activity *VisitFloor* where it spends time visiting the associated floor.

The most crucial element of this model is to specify the precise conditions under which the combi activity *EnterElev* can start. This combi is preceded by nine queues (eight *people* queues and one *elevator* queue). Obviously, the only hard requirement for *EnterElev* to be able to start is the availability of the elevator (i.e., that the *ElevQ* queue be non-empty). This requirement, however, does not hold true for the eight people queues. Because the elevator should be able to move to another floor even when all these queues are empty (this occurs

when the elevator is idle and returns to its base floor). To allow this, the *Enough* attributes of the links joining the eight people queues to the *EnterElev* combi must be set so that they are always *true*. Thus, from a resource availability point of view, only the *Enough* attribute of link E1 matters (by default this is determined by the contents of queue *ElevQ*).

However, the availability of the elevator should not be sufficient for *EnterElev* to start. In addition, at least one of three conditions must also be satisfied: either the elevator contains people in transit, or the total contents of the eight people queues are nonzero (there are somewhere people waiting for the elevator), or the elevator is not at its base floor (and should return there). These three conditions constitute the *semaphore* for *EnterElev*. The *semaphore* for a combi activity is the first logical condition that is evaluated every time the combi activity attempts to start. If the *semaphore* is *false* then the attempt fails. If the *semaphore* is true, then STROBOSCOPE continues checking and evaluates the *Enough* attributes for the links joining the combi to its preceding queues. A combi is able to start only if its *semaphore* and the *Enough* expressions for all these links evaluate to *true*.

Once the decision to start *EnterElev* is made (because its *semaphore* is *true* and the elevator is waiting in *ElevQ*), the elevator is certain to move to another floor. The issue now is to determine the direction of travel: *up* or *down*? This decision is made on the *beforedraws* event of the *EnterElev* combi (i.e., before the combi *EnterElev* draws the elevator and possibly any people as new passengers). On this event, the model sets the *EDirection* property of the elevator to one of three values: "1" to indicate "going up", "-1" to indicate "going down" and "0" to indicate "idle at base floor".

The expression that gets evaluated to determine *EDirection* at this event is a complex conditional statement that implements the following logic: The elevator keeps its current direction (up or down) if it contains people in transit, or if there are people waiting at higher/lower floors and the current direction is up/down. If none of these conditions are true, then if there are any people waiting to board, the elevator switches direction; otherwise, it moves in the direction of its base floor.

Once the decision has been made to move the elevator in a specific direction (i.e., *EDirection* is set), the next issue is to decide which one (if any) of the eight people queues will be allowed to have the people waiting there enter the elevator. This is accomplished by evaluating the *drawwhere* attribute of the links from each of the eight people queues to the *EnterElev* activity. The *drawwhere* attribute of a link is a filter that is evaluated for each of the characterized resources residing in the preceding queue to determine those that can be drawn through the link. Only those characterized resources that pass the filter can pass though the link. Clearly, the statements specifying the *drawwhere* attributes for these links are repetitive and should be generated automatically as described in the next section.

Two *SaveProps* (i.e., assignable properties) for the *people* compound resource are *CurFloor* and *Direction* (i.e., the floor on which the person is now and the direction he/she wants to move to). The *drawwhere* attribute of the links simply states that only those people that are on the same floor as the elevator and want to move in the elevator's current direction should be allowed to board. The number of people allowed to board is regulated by the link's *drawuntil* attribute. The expression for this attribute is the same for all links and allows loading people until the number of passengers equals the capacity of the elevator. Again, the *drawuntil* statements are repetitive and their specification in a generic manner is presented in the next section.

After the elevator loads the people currently waiting at the same floor and wishing to go in its own direction (up to its capacity), the combi activity *EnterElev* can start. Upon start-up, the elevator property *EPeopleCount* is updated to reflect the current number of passengers inside the elevator.

The *EnterElev* combi has zero duration and terminates immediately after it starts. Its termination leads to the assembly node *AttachPeople* where the newly loaded passengers are attached to the elevator compound resource (and join the rest of the passengers already attached). The loaded elevator is then routed as one compound resource to the *MoveElev* activity where it spends an amount of time equal to the duration of a one-floor move.

Just before the *MoveElev* activity ends and releases the elevator, the elevator property *ECurrentFloor* is updated to reflect its new current floor position. The elevator is then routed to the disassembly node *DetachPeople*. Here all passengers are for a moment detached from the elevator compound resource to allow those people that have reached their destination to depart and flow through link P2. The *releasewhere* attribute of link P2 compares each person's *NextFloor* property to the elevator's *ECurrentFloor* property. Those that match flow through P2 and leave the elevator. The remaining passengers are in transit. They are reattached to the elevator compound object and are released to the *ElevQ* queue via link E5 to start another cycle.

If the elevator is on floor 1 then the departing passengers are routed by the fork *VisitOrLeave* to the disassembly node *PeopleDepart* where they are destroyed (there are no exit links). Otherwise, they are routed to the *ReadyToVisit* queue where each person

starts a separate instance of the *VisitFloor* combi. On release through link P2, each person's *CurFloor* property is updated to reflect the floor he/she is currently on. On flow through link P4, each person's next floor destination (at the conclusion of the visit to the current floor) is determined through simple Monte Carlo sampling and is assigned to its *NextFloor* property. At the same time, a person's *Direction* property is determined based on the values of *NextFloor* and *CurFloor*.

When an instance of the *VisitFloor* combi activity finishes, it releases the person it holds to one of the seven people queues that follow. The recipient queue is determined by the *releasewhere* attributes of the seven links out of *VisitFloor*. The *releasewhere* attribute for each of these eight links allows a person to flow through only if its current floor (*CurFloor*) and direction of movement (*Direction*) match those of the succeeding queue. Thus, exactly one of these links will allow the person to flow through. It should be obvious that the specification of the *releasewhere* attributes of the links out of *VisitFloor* are repetitive and can be generated automatically as described below.

6 STATEMENT PREPROCESSING AND AUTOMATIC CODE GENERATION

For the elevator model to be scaleable, there is clearly a need to generate automatically the STROBOSCOPE statements that define all the *up* and *down* queues (two queues per floor), as well as all the links in out of these queues (two links per queue). The number of floors in the building should be the only variable that controls the size of the model. However, in addition to these statements (that have an obvious effect on the *size* of the simulation network), we must also be able to generate statements that define the necessary queue or link *attributes* (*drawwhere*, *drawuntil*, etc.) as well as those statements necessary for the collection of statistics and the communication of the results.

In STROBOSCOPE this is accomplished by using the preprocessor operator $<*Arg*>$, where *Arg* stands for the preprocessor replacement expression. A simulation model file is a text file that is read, pre-processed, and executed one statement at a time. As each statement is read, STROBOSCOPE searches for the two-character sequence "$<" (i.e., a *dollar sign* immediately followed by a *less than sign*). If the sequence "$<" is found, then what follows is treated as the preprocessor replacement expression. This expression extends to but does not include the next occurrence of ">$" (i.e., a *greater than sign* followed by a *dollar sign*). The entire preprocessor replacement expression is treated as a mathematical formula, which is then evaluated, truncated to an integer number, and used to replace the entire sequence $<*Arg*>$ as a string. After this substitution, the resulting statement is parsed and executed like any other ordinary STROBOSCOPE statement.

For example, the following statements:

```
QUEUE Q$<Log[10]>$U People;
LINK PQ$<Sin[3.14/2]>$U2
                Q$<Cos[0]>$U EnterElev;
```

are equivalent to:

```
QUEUE Q1U People;
LINK PQ1U2 Q1U EnterElev;
```

Statement preprocessing when coupled with control statements that implement *While-Wend* loops provide STROBOSCOPE with a powerful mechanism for automatic code generation. This mechanism is best illustrated with an example. The following short snippet of STROBOSCOPE code uses a *While-Wend* loop (controlled only by the number of floors in a building) to define the scaleable elements of the simulation network. These are the 4 "going up" queues (Q1U to Q4U), the 4 "going down" queues (Q2D to Q5D), the 8 incoming links (PQ1U1 to PQ4U1, and PQ2D1 to PQ5D1), and the 8 outgoing links (PQ1U2 to PQ4U2, and PQ2D2 to PQ5D2). Thus, for a building with *N+1* floors this short set of statements defines *2N* queues and *4N* links:

```
VARIABLE nFloors 5;
SAVEVALUE I 1;
WHILE I<=nFloors-1;
    QUEUE Q$<I>$U People;/ Ith Up Queue
    IF I==1; /1st floor links (exception)
        LINK PQ1U1 PeopleArrive Q1U;
        LINK PQ1U2 Q1U EnterElev;
    ELSE; /Intermediate floor links (rule)
        LINK PQ$<I>$U1 VisitFloor Q$<I>$U;
        LINK PQ$<I>$U2 Q$<I>$U EnterElev;
    ENDIF;
    QUEUE Q$<I+1>$D People;/ (I+1) down Q
    LINK PQ$<I+1>$D1 VisitFloor Q$<I+1>$D;
    LINK PQ$<I+1>$D2 Q$<I+1>$D EnterElev;
ASSIGN I I+1;
WEND;
```

Similarly, the following *While-Wend* loop defines several STROBOSCOPE statements that specify the behavior of the above nodes and links. In particular, these statements set the *Enough* attributes of all the links entering activity *EnterElev* to *true*; define *drawwhere* filters that allow only the correct people to board the elevator; define *drawuntil* attributes to allow people to board for as long as there is room in the elevator; define statistics collectors for each queue and supply data for the number of people that could not board the elevator

because it was full; collect statistics about each person's overall waiting time; and define filters that route each person that wants to leave a floor to the correct waiting queue at the elevator. The following statements include detailed comments and should be self-explanatory:

```
ASSIGN I 1;
WHILE I<=nFloors-1;
 /set the ENOUGH for links PQxU2 to TRUE
 ENOUGH PQ$<I>$U2 !FALSE;
 /Define statistics collectors NoFitxU
 COLLECTOR NoFit$<I>$U;

 /Before EnterElev draws any resources
 / collect in NoFitxU the number of people
 /  at QxU that do not fit to board.
 /   (provided people do wait at QxU
 /     and the elevator is going up
 /      and the elevator is at floor x)
 BEFOREDRAWS EnterElev COLLECT NoFit$<I>$U
  PRECOND 'Q$<I>$U.CurCount &
         EDirection==1 & ECurFloor==$<I>$'
  Max[Q$<I>$U.CurCount-SpaceLeftInElev,0];

 /Draw via link PQxU2 only those people
 / whose floor and direction match those
 /  of the elevator:
 DRAWWHERE PQ$<I>$U2
             'EDirection==Direction &
              ECurFloor==CurFloor';

 /When drawing a person through link PQxU2
 / collect in OverallWait the waiting time
 /  for that person in queue QxU
 ONDRAW PQ$<I>$U2 COLLECT OverallWait
              SimTime-TimeIn;

 /Keep drawing people through link PQxU2
 / until there is no space in the elevator
 DRAWUNTIL PQ$<I>$U2 !SpaceLeftInElev;

 / Release through link PQxU1 (x<>1)
 /  (to queue QxU) only those
 /  people that are currently
 /  at floor x and want to go up
 IF I!=1;
  RELEASEWHERE PQ$<I>$U1  'CurFloor==$<I>$
              & Direction== 1';
 ENDIF;

 /****From here down "x" means "I+1"****

 /set the ENOUGH for links PQxD2 to TRUE
 ENOUGH PQ$<I+1>$D2 !FALSE;

 /Define statistics collectors NoFitxD
 COLLECTOR NoFit$<I+1>$D;

 /Before EnterElev draws any resources
 / collect in NoFitxD the number of people
 /  at QxD that do not fit to board.
 /   (provided people do wait at QxD
 /     and the elevator is going down
 /      and the elevator is at floor x)
 BEFOREDRAWS EnterElev
   COLLECT NoFit$<I+1>$D
      PRECOND 'Q$<I+1>$D.CurCount &
      EDirection==-1 & ECurFloor==$<I+1>$'
   'Max[Q$<I+1>$D.CurCount-
        SpaceLeftInElev,0];

 /Draw via link PQxD2 only those people
 / whose floor and direction match those
 /  of the elevator:
 DRAWWHERE PQ$<I+1>$D2
              EDirection==Direction &
              ECurFloor==CurFloor;

 /When drawing a person through link PQxD2
 / collect in OverallWait the waiting time
 /  for that person in queue QxD
 ONDRAW PQ$<I+1>$D2 COLLECT OverallWait
              SimTime-TimeIn;

 /Keep drawing people through link PQxD2
 / until there is no space in the elevator
 DRAWUNTIL PQ$<I+1>$D2 !SpaceLeftInElev;

 / Release through link PQxD1
 /  (to queue QxD) only those
 /  people that are currently
 /  at floor x and want to go up
 RELEASEWHERE PQ$<I+1>$D1
      'CurFloor==$<I+1>$ & Direction==-1';

 / Increment the counter I and loop
 ASSIGN I I+1;
WEND;
```

The above examples are not the only places where the elevator model requires the use of preprocessing and automatic code generation. This capability is also needed to produce the required output at appropriate points during simulation run-time and after the simulation is completed. For example, automatic code generation is used to generate STROBOSCOPE statements that produce animation instructions for PROOF Animation (by writing to a text file at appropriate points during simulation run-time). Similarly, at the end of simulation,

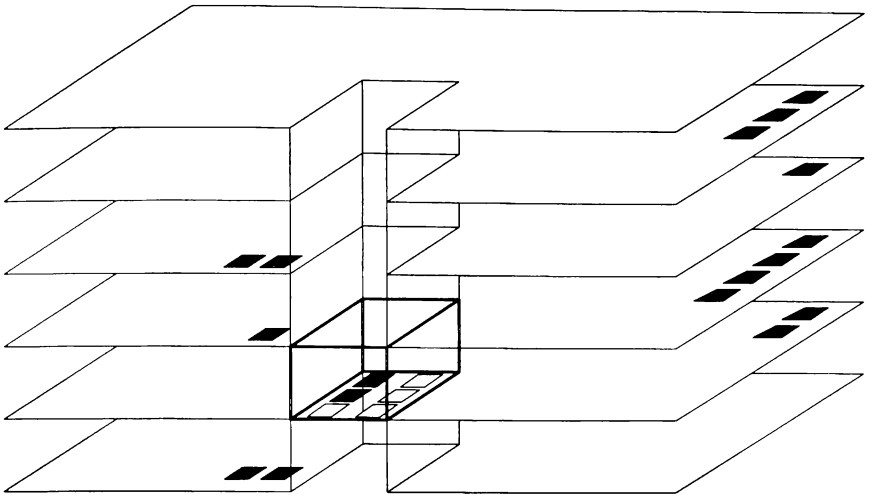

Figure 2 - PROOF Animation of Elevator Model

automatic code generation provides a convenient way to create statements that print the desired statistical results on a *per-floor*, or *per-queue* basis. In both cases, the necessary STROBOSCOPE statements are repetitive in nature (i.e., they repeat over the queues, their links, or both). Thus, generating these statements with a *While-Wend* loop is a natural solution to the problem of making the model scaleable.

7 ANIMATION

The STROBOSCOPE simulation model for the elevator problem described above was designed to be scaleable with respect to animation as well. By simply changing a single parameter that represents the number of floors in a building, it is possible to produce the correct animation trace file (ATF) that could then be processed in playback mode by Proof Animation to show the operation of the elevator for the selected number of floors.

Figure 2 shows a snapshot of the animation for a building with 5 floors. The building in this figure is part of the layout background, whereas the elevator (shown as a box with six slots) and the people (shown as colored squares) are instances of classes defined in the animation layout file. Every time a new person enters the building a new square is created and placed on a "line path" (a predefined trajectory) where it queues waiting for the elevator. When the elevator arrives, a person going in the same direction is moved from its "path queue" to one of the empty six slots inside elevator. A person exiting the elevator is placed on another "path queue" on the right-hand side of the corresponding floor where it stays for the duration of the person's visit to that floor. At the end of this visit, a person is removed from the "path queue" on the right-hand side of the floor and is placed on another "path queue" on the left-hand side of the floor where it waits for the elevator to pick it up. Figure 2 shows two people inside the elevator, two people waiting for the elevator at floors 1 and 4 and one person waiting at floor 3. In addition, there are two, four, one, and three people still visiting floors 2, 3, 4, and 5. Even a short playback of the animation makes it obvious that the elevator policy has been implemented correctly.

The only and rather obvious difficulty in scaling the entire animation is that the background drawing representing the building itself must also be scaled to have the correct number of floors. Fortunately, PROOF Animation stores the layout file as a CAD drawing in text format and this makes it possible to produce a layout file with the correct number of floors in the building as part of the output of a completely scaleable simulation run.

8 CONCLUSION

After a parametric model's accuracy is verified through statistics collection and animation, scaling it to represent, for example, a 50+ story building becomes just a matter of changing a single number to define the required number of floors.

The model presented above is not the only way to make the elevator problem scaleable. As mentioned earlier, it is possible to lump the separate "up" and "down" queues for each floor into two *global* "up" and "down" queues for the entire building (or even just one queue with unspecified direction). In this case, automated code generation would be necessary to define many statistics collectors that provide the same statistical information on a per floor basis as provided automatically by the deleted queues.

Alternatively, it is also possible to eliminate entirely the use of *collectors* (i.e., STROBOSCOPE objects) for gathering per floor statistics. For the elevator problem, for example, it is possible to store statistical information on a per floor basis in one- or two-dimensional arrays. In essence, this approach uses arrays to perform manually the functions that collectors do automatically. No matter what approach is used, however, it is very difficult to create completely scaleable models that do not have any need for statement preprocessing and automated code generation.

ACKNOWLEDGMENTS

The development of STROBOSCOPE has been supported by the University of Michigan Horace H. Rackham School of Graduate Studies and the National Science Foundation (Grant No CMS-9415105).

REFERENCES

Chisman J.A. 1996. *Industrial cases in simulation modeling*, Duxbury Press, ITP, Inc.

Ioannou, P. G., and J.C. Martinez. 1995. Evaluation of alternative construction processes using simulation. In *Proceedings of the 1995 Construction Congress*, San Diego, CA, October 22-26, ASCE.

Ioannou, P. G., and J.C. Martinez. 1996a. Animation of complex construction simulation models. In *Proceedings, Third Congress on Computing in Civil Engineering*, Anaheim, CA, June 17-19, ASCE.

Ioannou, P. G., and J.C. Martinez. 1996b. Comparison of Construction Alternatives Using Matched Simulation Experiments. *Journal of Construction Engineering and Management*, ASCE, (122) 3.

Law A.M., and W.D. Kelton. 1991. *Simulation Modeling and Analysis*, 2nd ed. McGraw Hill, New York, NY.

Martinez, J. C., Ioannou, P. G., and R. I. Carr. 1994. State and resource based construction process simulation. In *Proceedings of the First Congress on Computing in Civil Engin.*, ASCE, Washington, DC.

Martinez, J.C., and P.G. Ioannou. 1994. General purpose simulation with STROBOSCOPE, In *Proceedings, 1994 Winter Simulation Conference*, Orlando FL, December 11-14, IEEE, Piscataway, New Jersey.

Martinez, J.C., and P.G. Ioannou. 1995. Advantages of the activity scanning approach in modeling complex construction processes. In *Proceedings, 1995 Winter Simulation Conference*, Washington, DC, December 3-6, IEEE, Piscataway, New Jersey.

Martinez J.C. 1996. STROBOSCOPE—State- and resource-based simulation of construction processes, Ph.D. thesis, Civil and Environmental Engineering Dept., Univ. of Michigan, Ann Arbor, Michigan.

AUTHOR BIOGRAPHIES

PHOTIOS G. IOANNOU is Associate Professor in the Dept. of Civil and Environ. Engin. at the Univ. of Michigan. He received a Diploma in Civil Engin. from the National Technical Univ., Athens, Greece, in 1979; and a SMCE and Ph.D. from MIT in 1981 and 1984. He has just completed a six-year term as Chairman of the Computing in Construction Technical Committee of the ASCE. He co-developed the UM-CYCLONE construction process simulation system with R.I. Carr, supervised the design and development of COOPS by L.Y. Liu, and served as chairman of J.C. Martinez's Ph.D. dissertation committee. His research interests are primarily focused on the areas of construction decision support systems and construction process modeling.

JULIO C. MARTINEZ is Post Doctoral Research Fellow in Civil Engineering at the University of Michigan. He designed and implemented the STROBOSCOPE simulation language as part of his Ph.D. dissertation research. He received a Civil Engineer's degree from Universidad Catolica Madre y Maestra (Dominican Republic) in 1986, an MS in Civil Engin. from the University of Nebraska in 1987, and an MSE and a Ph.D. in Construction Engineering and Management from the University of Michigan in 1993 and 1996. His research interests are in computer applications to construction engineering and management.