Proceedings of the Third Congress on Computing in Civil Engineering, ASCE
Anaheim, California, June 17-19, 1996
J. Vanegas and P. Chinowsky, eds.

Animation of Complex Construction Simulation Models

Photios G. Ioannou[1] A.M. ASCE and Julio Martinez[2]

Abstract

One of the major impediments in using computer simulation to model and optimize construction processes has been the fact that decision makers often do not have the training nor the time to check the validity of simulation models prepared by others and thus have little confidence in the results. Animation is an increasingly popular technique that can be used to verify, debug and validate simulation models. Perhaps most importantly, it can also show in an easy to understand manner the insights gleaned from building and running the underlying model, thus increasing its credibility significantly. This paper presents a simulation model for the movement of people inside a building served by a single elevator with fairly complex control logic that was verified using animation. The simulation model for this example was developed using Stroboscope and verified using Proof Animation.

Animation of Simulation Models

Recent advances in computer animation are a main contributor to the increased popularity of simulation modeling. Typically, the animation of a simulation model involves a graphical computer application that uses changes in the position, shape, or color of icons to illustrate changes in the state of the simulated system. These icons often represent interacting resources (labor, equipment, materials, space, etc.). The result is a precisely-described, smoothly-moving depiction of a complex system whose states are constantly changing. Animation of simulation models differs from cartoon-like animation in that it does not attempt to create photo-realistic sequences of frame-by-frame effects. It is the state of the system that the animation software has to depict faithfully. While showing changes in the system state, the animation produces a visual rendition (possibly to scale) that is meaningful to a human observer. Almost all major simulation languages on the market today support animation in one of two modes. In *concurrent* mode, animation is rendered while the system is being simulated (albeit slowly to allow for visual comprehension). In *playback* mode, animation is rendered

[1]Associate Professor, Civil & Environmental Engineering Department, University of Michigan, Ann Arbor, MI 48109-2125.
[2]Research Fellow, Civil & Environmental Engineering Department, University of Michigan, Ann Arbor, MI 48109-2125.

based on changes in the system that have been recorded earlier and saved to a disk file by a simulation model.

The animation of simulation models can serve a variety of purposes. During model development process animation is useful for *verifying* that the simulation model behaves correctly and represents what the model developer had in mind. It is not possible to animate what is not already simulated. Thus, the simulation model has to provide a level of detail than can support realistic animation. At the same time it can be used to *debug* the model and find out why it does not behave as intended. Debugging a complex model is a difficult and arduous task. Visualizing the changing states of an entire system and tracing its behavior through a segment of simulation language code or a simulation trace output file can be extremely time-consuming. In this respect, computer animation can serve as the ideal trace mechanism for revealing the complex interrelationships represented by the model.

Often the developer of a simulation model is not an expert in the process being represented. In this case, animation can be used for model *validation*. Animations of the model can be shown to a process expert that can point out flaws not just in the model itself but also in way the model developer understands the behavior of the system. Thus, animation can provide feedback and increase everyone's confidence that the model is indeed a realistic representation of the underlying system.

Perhaps the most important use of animation is to communicate the simulation model to others and boost the credibility of its results. This is particularly important when the audience includes managers and other key personnel that are directly responsible for making decisions that affect system performance but do not understand simulation modeling. In this case, animation is an indispensable tool for increasing everyone's confidence. Instead of taking the blind advice of others, managers can see for themselves the insights gleaned from building and running the underlying model and appreciate why some strategies make sense while others do not.

Modeling Elevator Operations

The remainder of this paper describes the simulation model for a complex system that was verified through animation. The example used is that of a building served by a single elevator. This fairly complex example has been adapted from (Law and Kelton 1991) and is used to illustrate the effectiveness of animation to verify that the model is an accurate representation of the elevator's complicated control logic.

The problem is of interest in high-rise building construction (50+ stories) because 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 and documentation that the proposed hoisting system has been optimized and that wait 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).

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.

Simulation Model

A simulation model for this problem has been developed using Stroboscope (an acronym for STate and ResOurce Based Simulation of COnstruction ProcEsses), a general-purpose simulation programming *language* based on the activity-scanning paradigm. A description of Stroboscope and example applications can be found in (Martinez 1996, Martinez Ioannou & Carr 1994, Martinez & Ioannou 1994, Ioannou & Martinez 1995). The Stroboscope program, its documentation, and the simulation and animation models for the elevator problem are available via *anonymous* ftp from “grader.engin.umich.edu.”

The Stroboscope cyclic activity network for this problem is shown in Figure 1 and can be logically divided into two parts. The first part contains the eight queues *Q1U*, *Q2U*, *Q3U*, *Q4U*, *Q2D*, *Q3D*, *Q4D*, and *Q5D*. The four queues ending in *U* represent the people (at the corresponding floor) that wait for the elevator to go up. Similarly, the four queues ending in *D* represent the people (at the corresponding floor) that wait for the elevator to go down. Thus, these queues are not only specific to a particular floor, they are also specific to a certain direction of movement (up or down). To increase the number of floors in the building it is necessary to define additional queues of this type (two queues per floor).

The remainder of the simulation model is generic and involves the cycle of loading and moving the elevator from one floor to the next. This part is primarily represented by the two

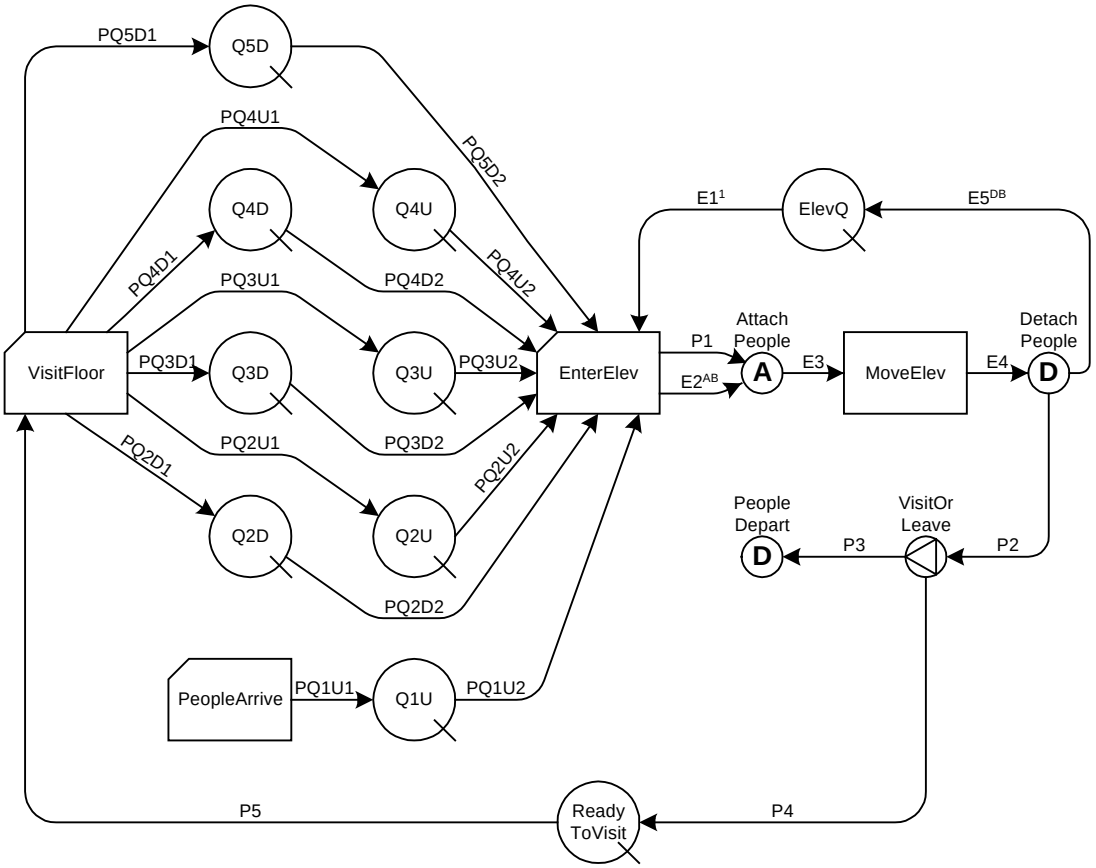

Figure 1 - Stroboscope Network for Elevator Model

activities *EnterElev* and *MoveElev.* It is not specific to any particular floor, nor does it depend on the number of floors.

There are two types of resources in this model, the *elevator* and *people*. For convenience both are defined as *compound-characterized* resources. A *compound-characterized* resource is not only easy to define but can also provide a base to which other resources (*generic*, *simple-characterized*, or even *compound-characterized* resources) 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* to the *elevator* (the *assembly base*) is performed by the *assembly* node *AttachPeople*. At the end of the *MoveElev* activity 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 *VisitFloor* where it spends time visiting the associated floor.

At the heart of this model is the answer to the question "when can the combi activity *EnterElev* start"? This combi is preceded by nine queues (eight *people* queues and one *elevator* queue). Obviously, the only "hard" requirement for the combi activity to start is that the *ElevQ* queue be non-empty. The elevator should be able to move to another floor even when all eight people 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 from the eight *people* queues must be set so that they are always "true." Thus, only the *enough* attribute of link E1 matters (by default this is determined by the contents of the *ElevQ* queue). However, the availability of the *elevator* resource is not 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* (a combi's *semaphore* is the first logical condition that is evaluated every time the combi activity attempts to start).

Once the decision is made to start *EnterElev* (because its *semaphore* is "true" and the elevator is waiting in *ElevQ*), the elevator is certain to move to another floor. The question now is in which direction: 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" indicates "up", "-1" indicates "down" and "0" indicates "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 (whether "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* combi. The *drawwhere* attribute of such a link acts like a filter. Only those characterized resources that pass the filter can pass though the link.

Two of the properties of 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.

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* elevator can start. Upon start-

up the elevator property *EPeopleCount* is updated to reflect the number of passengers inside the elevator. The duration of the *EnterElev* combi is zero.

The termination of *EnterElev* 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 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. 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. A person's *Direction* property is determined based on the values of *NextFloor* and *CurFloor*.

When the *VisitFloor* combi finishes, it releases the person it holds to one of the seven *people* queues that follow. The choice of 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.

Animation

The Stroboscope simulation model was designed to produce an animation trace file (ATF) that was then processed in playback mode by Proof Animation. Figure 2 shows a snapshot of the animation. In this figure the building 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

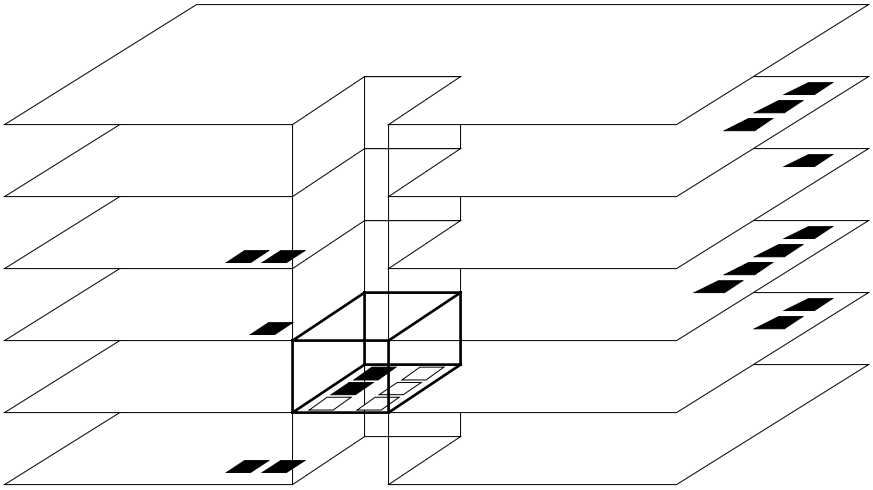

Figure 2 - PROOF Animation of Elevator Model

still visiting floors 2, 3, 4, and 5 respectively. Even a short playback of the animation makes it obvious that the elevator policy has been implemented correctly.

Conclusion

After the model's accuracy is verified through animation, scaling it to represent for example a 50+ story building is not difficult. For models such as this that require the repetitive definition of nodes and links over many floors, it is best to use Stroboscope's code-generating capabilities to automate the node and link definition process. Furthermore, there is really no need to define two people queues per floor (as done in the this example) unless one requires individual (and automatic) queue statistics for each direction and each floor. This means that it is in fact possible (and even easier to model) if all eight people queues in this example are merged into a single queue of people waiting for the elevator.

Appendix I - References

Chisman J.A., (1996), *Industrial cases in simulation modeling*, Duxbury Press, International Thomson Publishing, Inc.

Law A.M., and Kelton W.D. (1991), *Simulation Modeling and Analysis*, 2nd Ed., McGraw Hill.

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

Martinez J.C., Ioannou P.G., and Carr R.I. (1994), "State- and Resource-Based Construction Process Simulation", *Proceedings, ASCE First Congress on Computing in Civil Engineering*, Washington, DC, June 20-22, 1994.

Martinez J.C., Ioannou P.G. (1994), "General Purpose Simulation with Stroboscope", *Proceedings, 1994 Winter Simulation Conference*, Orlando, FL, Dec 11-14, 1994.

Martinez J.C., Ioannou P.G. (1995), "Advantages of the activity scanning approach in modeling complex construction processes", *Proceedings, 1995 Winter Simulation Conference*, Washington, DC, Dec 3-6, 1995.

Martinez J.C. (1996), *Stroboscope—State- and resource-based simulation of construction processes*, Ph.D. Thesis, Civil and Envir. Eng. Dept., Univ. of Michigan, Ann Arbor, MI