Proceedings of the 1995 Winter Simulation Conference
C. Alexopoulos, K. Kang, W.R. Lilegdon, and D. Goldsman, eds.

ADVANTAGES OF THE ACTIVITY SCANNING APPROACH IN THE MODELING OF COMPLEX CONSTRUCTION PROCESSES

Julio C. Martinez
Photios G. Ioannou

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

ABSTRACT

Construction processes are very complex and include highly interdependent components subject to complex activity startup conditions. A simulation language based on activity cycle diagrams can express the logic of complex simulation models very effectively. This paper uses Stroboscope, a three-phase activity scanning simulation language based on extended activity cycle diagrams, to model the operations of a quarry. Although simplified to allow a complete discussion within the space limitations of this paper, this example is representative of the complex nature of construction operations.

1 STROBOSCOPE

Stroboscope (Martinez, Ioannou and Carr 1994; Martinez and Ioannou 1994) is an acronym for STate and ResOurce Based Simulation of COnstruction ProcEsses. It is a programming language specifically designed to model construction operations. Stroboscope models are based on a network of interconnected modeling elements and on a series of programming statements that give the elements unique behavior and control the simulation.

The character of Stroboscope arises from its ability to dynamically access the state of the simulation and the properties of the resources involved in an operation. The state of the simulation refers to such things as the number of trucks waiting to be loaded; the current simulation time; the number of times an activity has occurred; and the last time a particular activity started. Access to the properties of resources means that operations can be sensitive to properties (such as size, weight, and cost) on an individual (the size of the specific loader used in an operation) or an aggregate basis (the sum of the weights of a set of steel shapes waiting to be erected).

Stroboscope modeling elements have attributes, defined through programming statements, that specify how they behave throughout a simulation. Attributes represent such things as the duration or priority of an activity, the discipline of a queue, and the amount of resource that flows from one element to another. Most attributes can be specified with expressions and have default values that provide the expected behavior. Expressions are composed of constants; system maintained variables that access the state of the simulation and the properties of resources; user-defined variables; logical, arithmetic, and conditional operators; and scientific, statistical, and mathematical functions.

The attributes of Stroboscope modeling elements allow simulation models to consider uncertainty in any aspect (not just time), such as the quantities of resources produced or consumed (e.g., the volume of rock resulting from a dynamite blast). Attributes also allow models to dynamically select the routing of resources and the sequence of operations; to allocate resources to activities based on complex selection schemes; to combine resources and dynamically assign properties to the resulting compound resource; and to activate operations subject to complex startup conditions not directly related to resource availability (e.g., do not blast rock until all crews of all trades have left the vicinity, the wiring has been inspected, and there are less than 10 minutes left in the current shift).

2 EXAMPLE: QUARRY OPERATIONS

The effectiveness of activity scanning to model complex processes will be illustrated using a Stroboscope example that optimizes the performance of a rock quarry. The purpose of the quarry is to produce aggregate for use in construction. Quarry operations include drilling holes into the rock face, loading the holes with explosives, wiring the explosives, blasting the rock, loading rock into trucks, hauling the rock to a crusher,

and crushing the rock to produce aggregate. The performance of the system is measured in terms of the cost per CM (Cubic Meter) of aggregate, and the time required to produce 500,000 CM.

The quarry operates two 4-hour periods daily with a 1-hour lunch break. The indirect costs of running the quarry are $150/hr of work time.

2.1 Drill and Blast

The drilling crews drill holes into the rock face. The number of drilling crews, *NDC*, is to be determined. A shooting crew subsequently loads the holes with explosives, wires them, and shoots them. There is only one shooting crew. Each blast requires 25 holes drilled and loaded. Each drilling crew can drill one hole at a time. When 25 holes are drilled, the shooting crew loads the holes with explosives one at a time. In order to shoot the rock, 25 holes must be loaded. The shooting crew announces that the site must be cleared (i.e., the drilling crews, loader, and haulers must go away to safety). While the site is being cleared, the shooting crew wires the explosives. After all people and equipment clear the site, the rock is shot. The shooting crew inspects the shot (to make sure that all explosives have been either detonated or neutralized) and announces that "all is clear to return to work."

For safety reasons, the shooting crew cannot load the next batch of 25 holes with explosives before the previous batch of 25 loaded holes is shot. The number of holes drilled is limited to one set of 25 holes ahead of those already loaded. Thus, the maximum number of drilled holes is 50 and the maximum number of loaded holes is 25. The cost of a drilling crew is $60/hr and the cost of the shooting crew is $40/hr.

Each blast produces 1200 CM of rock. Due to site limitations, the amount of rock already shot (waiting to be loaded and hauled) must not exceed 2400 CM.

2.2 Load and Haul

There is only one loader: a single 3 CM shovel. It loads 12 CM haulers. The haulers haul downhill and return uphill. The number of haulers, *NHL*, is to be determined. The loaded haulers dump, one at a time, into a hopper at the crusher. The loader costs $50/hr. Haulers cost $40/hr when operating and $35/hr when idle.

2.3 Crushing

The crusher has a 96 CM hopper. The crusher does not operate with less than 12 CM of uncrushed material loaded in its hopper. The crusher costs $65/hr when operating and $55/hr when idle and waiting for rock.

2.4 Objectives and Strategy

The purpose of the study is to determine the number of drilling crews and haulers that give the lowest cost per CM of aggregate. The time required to produce the 500,000 CM is also of interest.

Since the system output is the production of aggregate, the load and haul operations that supply rock to the crusher must never stop during work hours (if possible). Thus, the optimal strategy for the operation of the quarry is to shoot only during the lunch break and/or right after the afternoon shift (i.e., "shoot during breaks") when the load & haul operations stop anyway.

Individual activities that start close to a break are not preempted by the end of a shift. To compensate for this "working into the breaks", activities do not start when there are less than 6 minutes left in a shift.

In order for shooting to take place during a break, all the necessary conditions must be satisfied at least 6 minutes before the end of the shift. The decision and announcement to shoot can take place as much as 12 minutes before the end of the shift (but not earlier).

Shooting during work hours can occur only when there is no rock to load and haul (i.e., the loader and haulers are idle and waiting for more rock to be blasted), and there are 25 holes already loaded with explosives. In this case, it does not make sense to wait until the next work break in order to shoot because all operations would then stop during work hours anyway (with the possible exception of drilling extra holes).

3 SOLUTION

Stroboscope models consist of a graphical network and a series of programming statements (the network is also defined via programming statements). This discussion will go through the network and explain all the details required to solve this problem completely.

3.1 The Network

Figure 1 shows the network for this model. At an abstract level, the Combi activities (rectangles with cutoffs in the top-left corner), Normal activities (rectangles), and Queues (large circles with a slash in the bottom right corner) shown in this figure are similar in appearance and function to Cyclone modeling elements (Halpin and Riggs 1992); Consolidators are shown as rectangles with a semicircle in the left end. The links connecting nodes are named by convention with 2 letters that abbreviate the type of resource that flows through them followed by a number. The abbreviations in this model are summarized in the drawing. *DC*, for example, stands for drilling crews and *HL* for haulers.

Resources move from node to node in the direction of the link arrows. Queues are storage locations for inactive resources. Combis, Normals, and Consolidators are classes of activities that require and take hold of resources for a certain amount of time. Combis activate themselves when enough resources are available in the Queues that precede them. Upon activation they remove from the preceding Queues the resources needed to support the activity they represent. Normal activities start when activated by any of the preceding nodes. They receive the resources released by the ending activity. A Consolidator accumulates resources until a condition is satisfied. When the condition is satisfied, the Consolidator releases all the accumulated resources to the successors. The Consolidator is then ready to accumulate more resources and release them when the condition is met again.

Figure 1 shows the drilling crew going through 2 cycles. A number of *NDC* drilling crews are initially in the *DrlCrwReady* Queue. The *DrillHole* Combi removes one of them through link *DC1* in order to drill a hole. After the hole is drilled, the drilling crew returns to *DrlCrwReady* through link *DC2*. When an upcoming blast is announced, the *ClearDrlCrew* Combi removes drilling crews from *DrlCrwReady* (through *DC3*). The crews are later released to the *DrlCrwClrd* Queue (through *DC4*). After the inspection of a shot, the *SetUpDrlCrw* Combi removes drilling crews from *DrlCrwClrd* (through *DC5*) and later releases them to the *DrlCrwReady* Queue (through *DC6*).

The *DrillHole* Combi produces a hole and releases it to the *AcmDrilledHoles* Consolidator (through *HO1*). *AcmDrilledHoles* accumulates holes until it has a group of 25 (the consolidating condition). All 25 holes are then released to the *HlsRdyToLoad* Queue (through *HO2)*. The *LoadHoles* Combi removes holes from the *HlsRdyToLoad* Queue one by one (through *HO3*). This happens only when *LoadHoles* can simultaneously

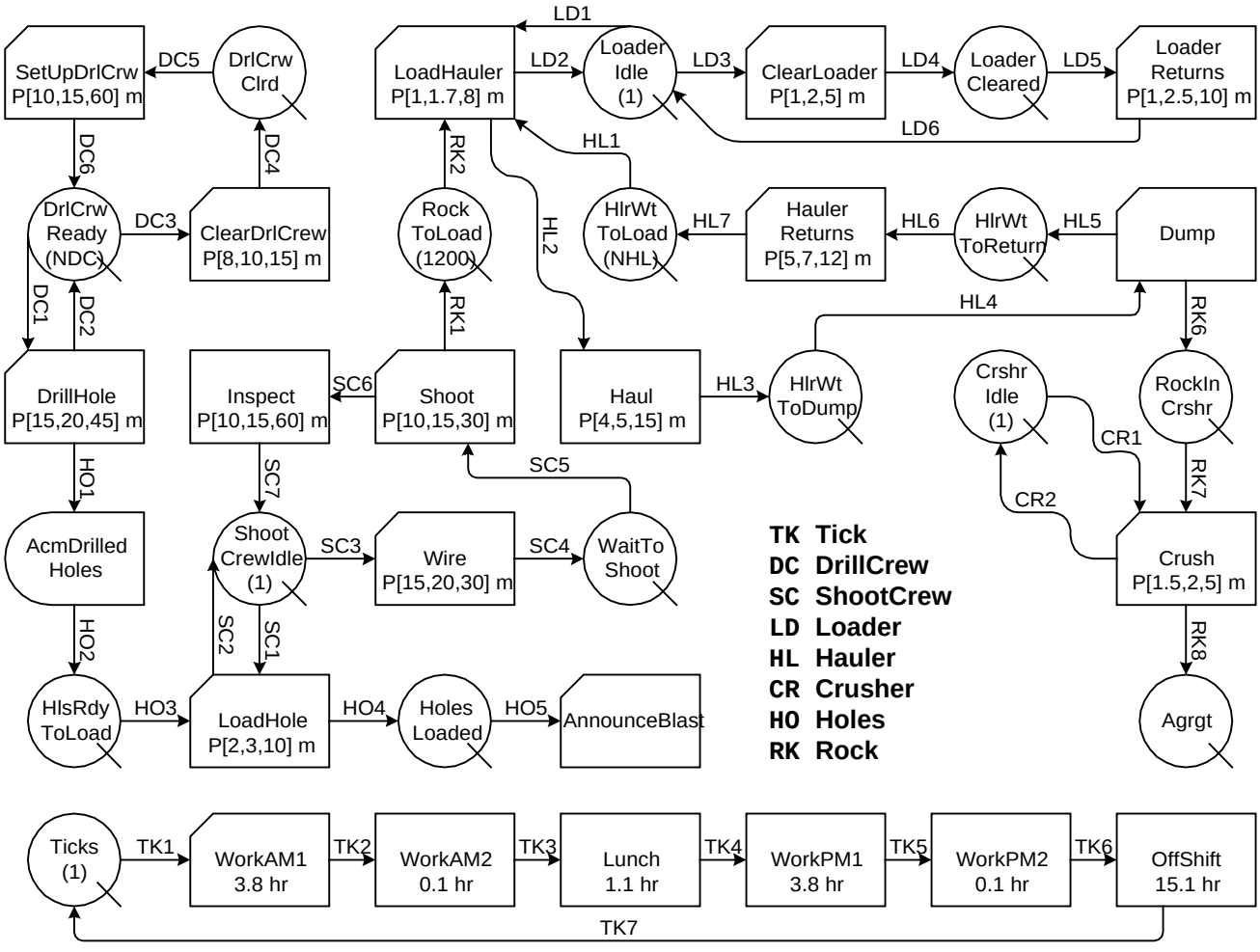

Figure 1: Stroboscope Network for the Quarry Operations

remove the shooting crew from the *ShootCrewIdle* Queue (through *SC1*). After a hole is loaded, the hole is released through link *HO4* to the *HolesLoaded* Queue and the shooting crew is released through link *SC2* to the *ShootCrewIdle* Queue. The *AnnounceBlast* Combi removes holes from the *HolesLoaded* Queue through link *HO5*. Twenty-five holes are needed to announce the shooting. Thus, *AnnounceBlast* removes 25 holes from *HolesLoaded* (in order to do so *HolesLoaded* must contain at least 25 holes). *AnnounceBlast* does not take time. Holes do not cycle through the network. They are created by *DrillHole* and destroyed when unreleased by *AnnounceBlast*.

The shooting crew is involved in another cycle in addition to the one between *ShootCrewIdle* and *LoadHole*. When a blast is announced, the *Wire* Combi removes the shooting crew from *ShootCrewIdle* through link *SC3*. After the wiring ends, the shooting crew is released through link *SC4* to the *WaitToShoot* Queue. The *Shoot* Combi then removes the shooting crew from *WaitToShoot* for the actual blasting (though *SC5*). After the shooting, the shooting crew is released to the *Inspect* Normal through link *SC6*. When the inspection is complete, the shooting crew returns to the *ShootCrewIdle* Queue through link *SC7*.

The *Shoot* Combi produces 1200 CM of rock. This rock is released to the *RockToLoad* Queue through link *RK1*. The *LoadHauler* Combi removes rock in smaller amounts (4 buckets of 3 CM per hauler loaded) through *RK2*. *LoadHauler* also removes the loader from the *LoaderIdle* Queue (through *LD1*) and a hauler from the *HltWtToLoad* Queue (through *HL1*). When *LoadHauler* finishes it returns the loader to *LoaderIdle* through link *LD1*. *LoadHauler* does not release rock. The rock is implicitly carried by the hauler, which leaves through link *HL2* towards the *Haul* Normal.

After hauling, the hauler enters the *HlrWtToDump* Queue through link *HL3*. The *Dump* Combi then removes the hauler through link *HL4*. When *Dump* finishes, the rock that was implicitly attached to the hauler is released to the *RockInCrshr* Queue (through *RK6*). The hauler itself is released through link *HL5* to the *HlrWtToReturn* Queue. The *HaulerReturns* Combi removes the hauler through link *HL6*. It later releases it to the *HlrWtToLoad* Queue through link *HL7* to complete the hauler cycle.

The loader is involved in another cycle in addition to the one between *LoaderIdle* and *LoadHauler*. When a shot is announced, *ClearLoader* removes the loader through link *LD3* and moves it to the *LoaderCleared* Queue through link *LD4* (off site). When it is time for the loader to resume work, *LoaderReturns* removes the loader from *LoaderCleared* through link *LD5* and later releases it to the *LoaderIdle* Queue through *LD6*.

The *Crush* Combi removes rock from the *RockInCrshr* Queue through link *RK7*. *Crush* also removes the crusher from Queue *CrshrIdle*. When *Crush* finishes, it releases the rock (which is now aggregate) to the *Agrgt* Queue (through *RK8*). *Crush* returns the crusher to *CrshrIdle* (through *CR2*).

At the bottom of Figure 1 there is an isolated network fragment that represents a 24-hour clock and through which a "tick" cycles. The tick is initially in the *Ticks* Queue. The *WorkAM1* Combi removes it through *TK1*. The tick is then sequentially released to the *WorkAM2*, *Lunch*, *WorkPM1*, *WorkPM2* and *OffShift* Normals (spending some time in each). *OffShift* then returns the tick to the *Ticks* Queue, where the tick cycle starts again.

3.2 Detailed Analysis of Model

Figure 1 shows resource types, nodes, and links. For this discussion it also shows activity durations and initial Queue contents. For "real" problems the latter are typically given by complex formulas that do not ordinarily "fit" in a drawing.

Queues indicate their initial content with a number shown in parenthesis below the Queue's name. Queues that are initially empty show only the name of the Queue. For example, at the beginning of the simulation, the *HlrWtToLoad* Queue contains *NHL* haulers, and the *HolesLoaded* Queue is empty. The Combis and Normals indicate their duration below their name. Thus, the "P[2,3,10] m" in the *LoadHole* Combi indicates that the duration for loading a hole follows a Beta pdf with 5th percentile of 2m (minutes), mode of 3m, and 95th percentile of 10m. The activities in the "clock cycle" have deterministic durations in hours as indicated.

By default, Stroboscope networks behave as follows: Only Queues can precede Combi activities. Combi activities are scanned for startup; their instances can start when none of the Queues that precede the Combi are empty. When a Combi instance starts, the Combi removes one unit of resource from each of the preceding Queues. The instance holds resources for a period of time determined by the Duration attribute of the activity, which if not specified is zero. After the time has elapsed, the instance terminates and releases its resources to the successor nodes. By default, the instance releases the resources it had acquired.

Only other activities can precede Normal activities. Normal activities are not scanned; their instances start when an instance of a predecessor finishes. It receives any resources released through the links that connect it to the terminating predecessor. The instance starts even if the terminating predecessor does not release resources. Once an instance of a Normal activity starts, it determines its duration and terminates in the same way as Combi instances (described above).

Consolidator activities, like Normals, can only follow other activities. A Consolidator activity reacts to the reception of resources and not to the termination of a predecessor (although resources are only released to it when an instance of a predecessor terminates). When a Consolidator receives a resource, it checks its ConsolidateWhen attribute. If the attribute returns 0 (FALSE), the Consolidator simply holds the resource. The Consolidator keeps accumulating resources until the ConsolidateWhen returns non-zero (TRUE). The Consolidator instance then terminates and releases the accumulated resources through the outgoing links.

Queues take a passive role. They receive resources released by terminating activity instances. The resources remain in Queues until removed by a Combi.

The cycle taken by the "tick" is a "clock". The *WorkAM1* Combi removes the tick from the *Ticks* Queue and holds it for 3.8 hours. The instance then terminates and an instance of *WorkAM2* starts. Every time an instance terminates, an instance of the successor starts. The "tick" never spends time in the *Ticks* Queue; when the tick enters the Queue, *WorkAM1* removes it and starts immediately.

The purpose of the clock is twofold. First, it forces the occurrence of discrete events at the instantiation and termination of each activity instance. These are important times in the simulation that correspond to the beginning of each shift, and to 12 and 6 minutes before the end of each shift. Second, the clock allows the definition of useful variables. These variables (in Stroboscope syntax) are defined in terms of the state of the simulation:

```
VARIABLE WorkTime
  '!Lunch.CurInst & !OffShift.CurInst;

VARIABLE BreakComingUp
  'WorkAM2.CurInst | WorkPM2.CurInst';
```

The name of an activity followed by a dot and the word "CurInst" returns the number of instances of the activity currently taking place. The ampersand "&" is the logical AND operator, the pipe "|" is the logical OR operator, and the exclamation "!" is the logical NOT operator. *WorkTime* returns TRUE whenever there are no instances of the *Lunch* or *OffShift* activities. *BreakComingUp* returns TRUE when instances of the *WorkAM2* or *WorkPM2* activities are taking place. Whenever *WorkTime* or *BreakComingUp* are evaluated in expressions, they return an up to date value that depends on the state of the simulation.

Two other variables, whose values we will change explicitly at certain points in the processing of the simulation are defined below:

```
SAVEVALUE ShootingAlert 0;
SAVEVALUE HolesUninspected 0;
```

ShootingAlert will have a value of one (TRUE) between the time a shooting is announced and the time it is inspected. *HolesUninspected* will keep track of the number of holes in the system. This includes holes drilled and not loaded, holes loaded not yet shot, and holes shot not yet inspected. The actual code that maintains the accuracy of these variables appears later. The initial value of both variables is 0 (specified with the definition).

The test-head (i.e., the procedure to determine if a Combi instance can start) of Combi activities is divided into two parts. The first part is not related to resource availability. The second part depends on the resources available in the Queues that precede the Combi. Both parts of the test-head must return TRUE for a Combi activity to start.

The first part is given by the Combi's Semaphore attribute. This attribute must return TRUE before the second part of the test-head is checked. The default Semaphore always returns TRUE.

The second part of the test-head is given by the Enough attribute of each of the incoming links. The Enough for all incoming links must return TRUE for the second part of the test head to succeed. The default Enough returns TRUE when the Queue connected by the link to the Combi is not empty.

If Combi *DrillHole* and link *DC1* are left with their default attributes (Semaphore, and Enough), *DrillHole* will start whenever *DrlCrwReady* contains at least one crew. Thus, a Semaphore must be used to recognize the fact that holes cannot be drilled during non-work hours, (or) during a shooting alert, (or) when the maximum number of holes allowed have already been drilled. This Semaphore is set with the following statement:

```
SEMAPHORE DrillHole 'WorkTime &
  !ShootingAlert & HolesUninspected<50';
```

This Semaphore returns TRUE when *WorkTime* is TRUE, *ShootingAlert* is FALSE, and *HolesUninspected* is less than 50.

As soon as *DrillHole* starts (and every time it does), the value of *HolesUninspected* must be incremented:

```
ONSTART DrillHole ASSIGN
  HolesUninspected HolesUninspected+1;
```

DrillHole instances obtain only one resource, a drilling crew from *DrlCrwReady*. When an instance of *DrillHole* terminates, the drilling crew returns to *DrlCrwReady*. There is, however, no hole to release through link *HO1*. Since the default is to release whatever was acquired, zero holes will be released

through *HO1*. The ReleaseAmount attribute of link *HO1* modifies this behavior and releases one hole:

```
RELEASEAMT HO1 1;
```

The *CearDrlCrew* Combi, if left with the default Semaphore, will start whenever there is a drilling crew in *DrlCrwReady*. Drilling crews are only cleared from site when an upcoming blast has been announced. Thus the following statement is necessary:

```
SEMAPHORE ClearDrlCrew ShootingAlert;
```

The same Semaphore should be applied to the *ClearLoader* Combi. There is no need to model the clearing of the haulers explicitly. Clearing the loader, which is slow, gives the haulers enough time to retreat to a safe place.

The *SetupDrlCrew*, *LoadHauler*, *LoaderReturns* and *HaulerReturns* activities cannot take place during a shooting alert or when not in working hours. The Semaphore for these activities should be set to `'WorkTime & !ShootingAlert'` (i.e., start only during working hours and if there is no shooting alert).

The default *ConsolidateWhen* attribute of a Consolidator always returns TRUE. Hence, the default is very rarely used. *AcmDrilledHoles* needs to accumulate 25 holes before releasing all of them through *HO2*. This is done by setting the Consolidator's ConsolidateWhen attribute:

```
CONSOLIDATEWHEN AcmDrilledHoles
  'AcmDrilledHoles.Holes.Count>=25';
```

The name of an activity followed by a dot, the name of a resource type, another dot, and the word "Count", returns the amount of resource of the given type in the instance of the activity that is "in context". Thus, *AcmDrilledHoles*.*Holes*.Count returns the number of holes in *AcmDrilledHoles*. The Consolidator will "consolidate" when the number of holes accumulated reaches 25. The ConsolidateWhen attribute is evaluated only when (and every time) a resource enters the Consolidator. When the Consolidator instance "consolidates" (finishes), it terminates and releases all the resources to *HlsRdyToLoad*.

The *LoadHole* Combi requires a Semaphore to stop it from starting when too many holes have already been loaded. The following Semaphore considers this fact as well as other issues mentioned earlier:

```
SEMAPHORE LoadHole '!ShootingAlert &
  WorkTime & HolesLoaded.CurCount<25';
```

The name of a Queue followed by a dot and the word "CurCount" returns the current content of the Queue. Thus, *LoadHole* will start only if a blasting has not been announced, (and) during working hours, and if the number of holes already loaded is less than 25.

The test-head for the *AnnounceBlast* Combi is a little bit more complex (both the Combi's Semaphore and the incoming link's Enough attribute must be specified). The Semaphore looks as follows:

```
SEMAPHORE AnnounceBlast
 'RockToLoad.CurCount<=1200 &

 (RockToLoad.CurCount<12|BreakComingUp)';
```

The first part of the test-head is successful if the amount of rock in *RockToLoad* is at most 1200 CM (so that the result of a new blasting does not make the total amount of rock exceed the maximum of 2400 CM), and either there is not enough rock to fill a hauler or the shift will end soon.

If the Semaphore for *AnnounceBlast* returns TRUE, the second part of the test-head must be checked. It is not sufficient for *HolesLoaded* not to be empty (the default). *AnnounceBlast* needs 25 loaded holes and the Enough attribute of link *HO5* must reflect this fact:

```
ENOUGH HO5 'HolesLoaded.CurCount>=25';
```

A link's Enough attribute is a logical expression. Its value is interpreted as TRUE (non-zero) or FALSE (zero). The fact that *HO5*'s Enough returns TRUE when the number of loaded holes is at least 25, does not imply that 25 holes will be removed from *HolesLoaded* when *AnnounceBlast* starts. This information must be specified explicitly by setting *HO5*'s DrawAmount attribute:

```
DRAWAMT HO5 25;
```

AnnounceBlast is a zero duration activity that marks a very important event in this model. Its main purpose is to activate the shooting alert:

```
ONSTART AnnounceBlast ASSIGN
                  ShootingAlert 1;
```

As soon as blasting is announced, and while the loader and drilling crews clear the site, the shooting crews begin to wire the loaded holes. Since wiring is done only when a blasting has been announced, it needs a Semaphore to prevent it from starting otherwise:

```
SEMAPHORE Wire ShootingAlert;
```

After the holes are wired, the shooting crew goes to the *WaitToShoot* Queue. Shooting, however, cannot start until the loader and drilling crews clear the site. Thus, *Shoot* needs the following Semaphore:

```
SEMAPHORE Shoot 'LoaderCleared.CurCount &
   DrlCrwClrd.CurCount==NDC';
```

When the loader is cleared it is in *LoaderCleared*. When a drilling crew is cleared it is in *DrlCrwClrd*. *NDC* is a decision variable for this model that indicates the number of drilling Queues. The Semaphore returns TRUE when the loader and all the drilling crews are cleared.

Instances of *Shoot* do not hold rock. Thus, the rock that results from the shooting should be specified explicitly so that it can be released through link *RK1*:

```
RELEASEAMT RK1 1200;
```

The inspection of a shot starts immediately after the shooting. When the inspection is complete, the following variables need to be updated:

```
ONEND Inspect ASSIGN HolesUninspected
   HolesUninspected-25;
ONEND Inspect ASSIGN ShootingAlert 0;
```

The statements reduce the number of uninspected holes in the system by 25 and "turn off" the shooting alert each and every time *Inspect* ends.

In addition to the conditions required by its Semaphore, startup of the *LoadHauler* Combi requires that sufficient resources be available. The loader, a hauler, and 12 CM of rock are required. The attributes of link *RK2* must be modified so that *LoadHauler* can start only if 12 CM of rock are available, and so that when it does start, 12 CM are removed from *RockToLoad*:

```
ENOUGH RK2 RockToLoad.CurCount>=12;
DRAWAMT RK2 12;
```

The only resource requirement for starting an instance of the *Dump* Combi is that a hauler be available in *HlrWtToDump*. Additional conditions must be met: A hauler cannot dump while another hauler is dumping or when not in working hours, and there must be enough space in the crusher's hopper to hold the dumped rock. Thus, the following Semaphore:

```
SEMAPHORE Dump 'WorkTime & !Dump.CurInst
   & RockInCrshr.CurCount<=84';
```

"!Dump.CurInst" returns TRUE only when the number of current instances of *Dump* is zero (FALSE). This limits the number of simultaneous dumps to 1.

The rock is hauled implicitly by the hauler. Instances of *Dump* do not hold rock (i.e., no rock links enter *Dump*). Thus the amount of rock released by *Dump* must be specified explicitly:

```
RELEASEAMT RK6 12;
```

The crusher operates during working hours and must leave at least 12 CM of rock in the hopper (in addition to the 12 CM it crushes). Thus:

```
SEMAPHORE Crush WorkTime;
ENOUGH RK7 'RockInCrshr.CurCount >= 24';
DRAWAMT RK7 12;
```

The crusher is a good example of the flexibility that comes about by separating the concept of sufficiency (Enough) and actual use (DrawAmout). The crusher requires that the hopper contain at least 24 CM of rock, but it only removes 12 CM.

3.3 Source Code

The entire model for the quarry operation was presented in the previous subsections. There are no elements of the model that are not either shown directly in Figure 1, or explicitly and individually discussed above.

There is a one to one mapping of the information in Figure 1 to Stroboscope statements. The following statements are examples. They define a resource type, a Queue, a Combi, a link, the initial content of a Queue, and the duration of an activity:

```
GENTYPE Hauler;
QUEUE HlrWtToLoad Hauler;
COMBI LoadHauler;
LINK HL1 HlrWtToLoad LoadHauler;
INIT HlrWtToLoad NHL;
DURATION LoadHauler Pertpg[1,1.7,8]/60;
```

3.4 Experiments

Once the model has been defined, it is possible to perform automated experiments that perform multiple replications for various alternatives and that incorporate variance reduction techniques such as common random numbers or antithetic sampling. The facilities include the ability to manipulate numerous statistics about the performance of a run and to establish confidence intervals on the derived quantities.

At the heart of the experimentation section is the control statement that makes the simulation run, which for this model is the following:

```
SIMULATEUNTIL Agrgt.CurCount>=500000;
```

This statement is sufficient to perform a single run with fixed values of *NDC* (number of drilling crews) and *NHL* (number of haulers).

More elaborate programming can produce Figure 2 below (copied directly from Stroboscope's output). For a pre-defined set of *NDC* and *NHL* combinations, replications were performed until the 90% confidence interval on unit cost (for each pair) was at most $0.02/CM. Figure 2 shows *NDC*, *NHL*, *Rpl* (the number of replications required), and the 90% confidence intervals on total time and unit cost.

NDC	NHL	Rpl	Total Time 90% CI Days	Unit Cost 90% CI $/CM
1	6	3	[381.8,383.9]	[3.64,3.66]
1	7	2	[351.7,352.4]	[3.57,3.57]
1	8	4	[333.2,334.5]	[3.58,3.59]
1	9	4	[324.2,326.1]	[3.67,3.69]
2	6	3	[381.7,383.3]	[4.01,4.02]
2	7	4	[351.3,352.9]	[3.90,3.92]
2	8	5	[333.2,334.7]	[3.90,3.91]
2	9	6	[324.5,326.1]	[3.98,4.00]
3	7	6	[350.8,352.3]	[4.23,4.25]
3	8	2	[333.1,333.5]	[4.21,4.22]
3	9	8	[324.5,326.0]	[4.29,4.31]
3	10	2	[321.0,321.4]	[4.43,4.43]

Figure 2: Stroboscope Output for Quarry Operations

4 IMPLEMENTATION

A 32-bit MS Windows dynamic link library provides compilation and simulation services to a command-line compiler or to an integrated development environment (editor–debugger–compiler driver). All are WIN32 programs developed in C++. They run native on Windows NT and Windows 95, or under MS Windows 3.1 if extended with the Win32s subsystem.

A graphical user interface is also available. It is used to draw a model network interactively, and then to generate a Stroboscope language model file or to run the model directly through OLE automation.

The entire system and documentation are available via anonymous FTP from grader.engin.umich.edu.

5 CONCLUSION

The quarry example uses only a small subset of Stroboscope's modeling capabilities. Nevertheless, it illustrates how a system based on three-phase activity scanning can model complex systems of the kind frequently found in construction. It also shows the one-to-one correspondence between an activity cycle diagram (that can be understood by most managers) and the actual model. There is no need to define artificial entities, to differentiate between resources that serve and those being served, to wonder what "entities" might be for this system, etc. This directness in representation makes it easy to construct and debug Stroboscope models involving complex activity and resource interactions with a minimum of training. Students with no prior experience in simulation are able to model problems of this complexity after about six lectures.

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

Halpin, D.W., and L.S. Riggs. 1992. *Planning and analysis of construction operations*, John Wiley & Sons, New York, NY.

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

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 of the 1994 Winter Simulation Conference*, Lake Buena Vista, FL.

AUTHOR BIOGRAPHIES

JULIO C. MARTINEZ is Horace H. Rackham Pre-Doctoral Fellow and a Ph.D. candidate 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 in Construction Engineering and Management from the University of Michigan in 1993. His research interests are in computer applications to Construction Engineering and Management.

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 is currently the Chairman of the Computing in Construction 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 is currently serving 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.