Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

EZSTROBE - GENERAL-PURPOSE SIMULATION SYSTEM BASED ON ACTIVITY CYCLE DIAGRAMS

Julio C. Martinez

Civil & Environmental Engineering Department
Virginia Tech
Blacksburg, Virginia 24061-0105, U.S.A.

ABSTRACT

This paper introduces EZStrobe, a very simple but powerful general-purpose simulation system. Although designed for modeling construction operations, EZStrobe is domain independent and useful for modeling a wide variety of systems in any discipline. EZStrobe is based on Activity Cycle Diagrams and employs the Three-Phase Activity Scanning paradigm. It is therefore naturally adept for complex systems where many resources collaborate to carry out tasks as is typical in construction. The paper describes the basic system concepts. The paper also presents and explains several examples of increasing complexity to illustrate the range of modeling capabilities.

1 INTRODUCTION

Several simulation systems have been designed specifically for construction (e.g., Halpin 1992, Martinez 1996). These systems all use some form of network based on Activity Cycle Diagrams to represent the essentials of a model, and employ clock advance and event generation mechanisms based on Activity Scanning or Three-Phase Activity Scanning. These systems are designed for both simple (e.g., CYCLONE) and very advanced (e.g., STROBOSCOPE) modeling tasks but do not satisfy the need for a very easy to learn and simple tool capable of modeling moderately complex problems with little effort. EZStrobe is designed to fill this void in currently existing simulation tools and to facilitate the transition to more advanced tools (e.g. STROBOSCOPE) as the system is outgrown.

2 ACTIVITY CYCLE DIAGRAMS AND ACTIVITY SCANNING

Activity Scanning models are prepared based on the various activities that can take place in an operation. The modeler focuses on identifying activities, the conditions under which the activities can happen, and the outcomes of the activities when they end. For an earth-moving operation where wheel loaders load trucks from a stockpile, for example, the modeler may identify activities as shown in Table 1.

These models are typically represented using Activity Cycle Diagrams (ACDs), which are networks of circles and squares that represent idle resources, activities, and their precedence. The ACD of Figure 1 for example, is a graphical representation of the information in Table 1. The rectangles represent activities (resources collaborating to achieve a task), the circles represent queues (idle resources), and the links between them represent the flow of resources. ACDs of this type are typically used to express the main concepts of a simulation model -- other details of the model such as startup conditions not related to resource availability, are not shown. The ACD is used as a guide for coding the model using a general-purpose or simulation programming language.

Table 1 - Activities, conditions and outcomes for earthmoving operation

Conditions Needed to Start	Activity	Outcome of Activity
Wheel loader idle at source. Empty truck waiting to load. Enough soil in stockpile.	Load	Wheel loader idle at source. Loaded truck ready to haul.
Loaded truck ready to haul.	Haul	Loaded truck ready to dump.
Loaded truck ready to dump.	Dump	Dumped soil. Empty truck ready to return.
Empty truck ready to return.	Return	Empty truck waiting to load.

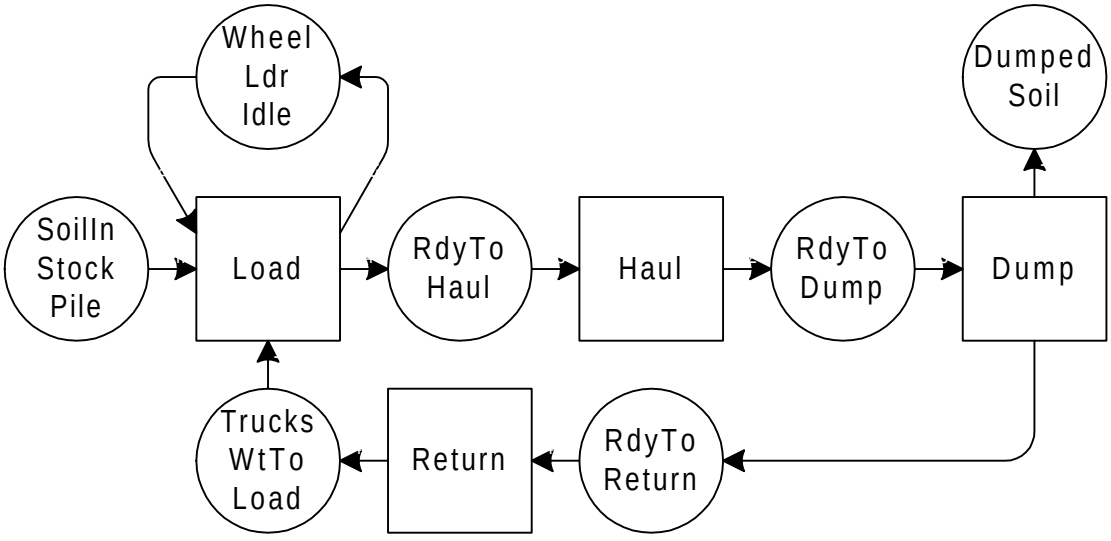

Figure 1 - Conventional ACD for earthmoving operation

3 EZSTROBE ACDS

EZStrobe ACDs are annotated extensions of the standard ACD's described above. The EZStrobe ACD for the same earthmoving operation described in Table 1 and Figure 1 is shown in Figure 2.

The network of Figure 2 is more compact than the one in Figure 1. Superfluous queues have been removed to indicate that some activities immediately follow their predecessors because the conditions needed for them to start are completely satisfied by the predecessor's outcome. Hauling, for example, immediately follows loading, making it unnecessary to show trucks in a 'ready to haul' state.

Unlike the ACD of Figure 1, the annotations of the EZStrobe ACD of Figure 2 make it a complete and unambiguous representation of the operation. The "1000" written in the bottom of *SoilInStkPl* indicates that at the beginning of the operation the Queue will contain 1000 units of resource (cubic meters). The first part of the annotation shown on the link that connects *TrkWtLd* to *Load* (">0") indicates that one of the conditions needed for

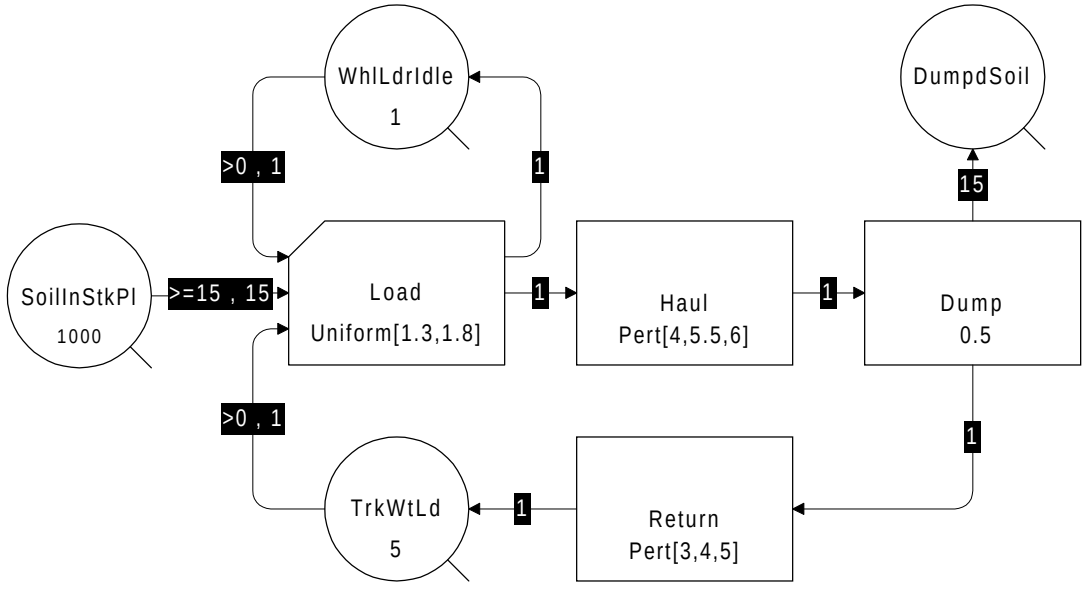

Figure 2 - EZStrobe ACD for earthmoving operation

Load to start is that more than zero units of resource exist in *TrkWtLd*. The other two conditions needed for *Load* to start are that at least 15 units of resource exist in *SoilInStkPl* and that more than zero exist in *WhlLdrIdle*. The second part of the annotations on those links (",1", ",15", and ",1") indicate that 1, 15, and 1 units will be removed (if possible) from *TrkWtLd*, *SoilInStkPl*, and *WhlLdrIdle* every time *Load* starts. The "Uniform[1.3,1.8]" shown inside *Load* indicates that its duration is sampled from a uniform distribution with minimum 1.3 and maximum 1.8 (minutes). The "15" shown on the link that connects *Dump* to *DumpdSoil* indicates that one of the outcomes of *Dump* is the insertion of 15 units of resource into *DumpdSoil*.

In EZStrobe models, all activity startup conditions and outcomes are in terms of resource amounts. Resources that reside in the same location are assumed to be indistinguishable, interchangeable, and exist in bulk quantities (i.e., their amounts can be expressed with real numbers and are not limited to integers). EZStrobe does not enforce the type of resources and the units with which they are measured -- the modeler is responsible for maintaining consistency.

3.1 EZStrobe Modeling Elements

The modeling elements that can be used in EZStrobe, the precedence rules that govern them, and their explanation follow.

A Queue is a named element that holds idle resources. The name of the Queue is shown at the center. At the beginning of a simulation Queues hold a certain number of resources. This number is shown below the Queue name. Resources are placed in Queues when they are released by terminating instances of preceding Activities. They are removed from Queues by starting instances of succeeding Combi Activities. A Queue can follow any other node except another Queue. A Queue can only precede a Combi Activity.

A Combi Activity is a named element that represents tasks that can start whenever the resources that are available in the Queues that precede it are sufficient to support the task. The name of the Combi is shown at the center. The number at the top is the priority that the Combi has over other Combis when competing for resources in preceding Queues. A Combi with a high priority has a chance to start before a Combi with a lower priority. Priorities can be negative and the default value is zero (e.g., when the priority is not specified it is assumed to be zero). The formula at the bottom of the Combi is used to determine the duration of its instances. The duration formula typically samples from a probability distribution. Therefore, different instances of the same Combi can have different durations. A Combi can only follow a Queue, but can precede any other node except another Combi.

A Normal Activity is a named element that represents tasks that start whenever an instance of any preceding Activity ends. The name of the Normal is shown at the center. The formula at the bottom of the Normal is used to determine the duration of its instances. The duration formula typically samples from a probability distribution. Consequently, different instances of the same Normal can have different durations. A Normal can follow any node except a Queue, and can precede any node except a Combi.

A Fork is a probabilistic routing element. It typically follows an Activity but can also follow another Fork. When a preceding activity instance finishes, the Fork chooses one of its successors. If the chosen successor is a Normal Activity then the Normal Activity starts. If the chosen successor is a Queue then the Queue receives any resources routed through the Fork. If the chosen successor is another Fork, then the second Fork will choose one of its successors. The relative likelihood that a particular successor will be chosen depends on the "P" property of the Branch Link that emanates from the Fork towards the successor (see Brach Link below).

A Draw Link connects a Queue to a Combi. A Draw Link shows two pieces of information separated by a comma. The first part is the condition necessary for the successor Combi to start as a function of the content of the predecessor Queue. The text ">0", for example, indicates that the content of the Queue must be greater than zero in order for the Combi to start. EZStrobe supports six relational operators to express this condition: less than (<), less than or equal (<=), greater than (>), greater than or equal (>=), equal (==), and not equal (!=). The second part is the amount of resource that the Combi will **attempt** to remove from the predecessor Queue in the event that the Combi does start. The Combi may not be able to remove the amount attempted if that amount is greater than the content of the Queue, in which case the entire content is removed.

A Release Link connects an Activity to any other node except a Combi. The text shown on a Release Link indicates the amount of resource that will be released through the Link each time an instance of the predecessor activity ends.

A Branch Link is used to connect a Fork to any other node except a Combi. The text shown on a Branch Link indicates the value of the "P" property for that Link. The "P" property establishes the relative likelihood that the successor connected by the Branch Link will be selected every time the Fork needs to choose a successor.

3.2 Supplementary Input and Simulation Output

Because an annotated EZStrobe ACD is a complete representation of an operation, in most cases no further basic input is required to run simulations. For simulations that do not naturally stop (i.e., that can potentially run

forever), it is necessary to specify a simulation termination condition. In EZStrobe this condition can be set by specifying a limit on simulation time or on the number of times a particular activity starts.

The purpose of simulating an operation is to obtain statistical measures of performance. By default, EZStrobe will produce a report containing the simulation time of the report and information on the activities and queues of the model. For each queue, the report shows the content at the time of the report (Cur), the total amount of resource to ever enter (Tot), the average waiting time (AvWait), the time-weighted average content (AvCont), the time-weighted standard deviation of the content, the minimum content (MinCont), and the maximum content (MaxCont). For each activity, the report shows the current number of times that the activity is being performed at the time of the report (Cur), the total number of times it has started (Tot), the time at which the first instance started (1stSt), the time at which the last instance started (LstSt), the average duration (AvDur), the standard deviation of the duration (SDDur), the minimum duration (MinD), the maximum duration (MaxD), the average time between successive starts (AvInt), the standard deviation of the time between successive starts (SDInt), the minimum time between successive starts (MinI), and the maximum time between successive starts (MaxI). An output for the model shown in Figure 2, for example, is shown in Table 2.

Note from the output that *SoilInStkPl* contains 10 units of resource (cubic meters) at the time of the report. This is because 15 units are required to enable *Load* start.

More detailed statistics regarding the historical content of queues are available in the form of cumulative histograms. To obtain a histogram for a queue it is necessary to specify the range and number of collection bins. EZStrobe will additionally create an underflow and an overflow bin. Specifying 3 bins between 1 and 4 for *TrkWtLd*, for example, produces the additional output shown below:

```
Detailed statistics on content of queue
                TrkWtLd

     Content      TotTime  %Time
  ============================
      < 1.00       132.94  82.47
      < 2.00       147.77  91.67
      < 3.00       151.94  94.26
      < 4.00       155.31  96.35
     >= 4.00         5.89   3.65
```

The output indicates that *TrkWtLd* was empty (its content was less than 1, i.e., zero) 82.47% of the time, and contained exactly 4 or more trucks 3.65% of the time.

Table 2 - Simulation Output for the Model Shown in Figure 2

```
Statistics report at simulation time 161.195

Queue       Res    Cur     Tot AvWait AvCont SDCont MinCont MaxCont
============================================================
DumpdSoil   ezs 990.00  990.00  80.32 493.29 301.72    0.00  990.00
SoilInStkPl ezs  10.00 1000.00  74.38 461.45 298.67   10.00 1000.00
TrkWtLd     ezs   5.00   71.00   0.80   0.35   0.92    0.00    5.00
WhlLdrIdle  ezs   1.00   67.00   0.88   0.36   0.48    0.00    1.00

Activity Cur Tot 1stSt  LstSt AvDur SDDur MinD MaxD AvInt SDInt MinI MaxI
=================================================================
Dump       0  66  6.94 156.70  0.50  0.00 0.50 0.50  2.30  1.15 0.90 5.13
Haul       0  66  1.58 150.85  5.32  0.31 4.55 5.88  2.30  1.11 1.30 5.30
Load       0  66  0.00 149.30  1.55  0.15 1.30 1.80  2.30  1.10 1.30 5.42
Return     0  66  7.44 157.20  3.98  0.32 3.30 4.76  2.30  1.15 0.90 5.13
```

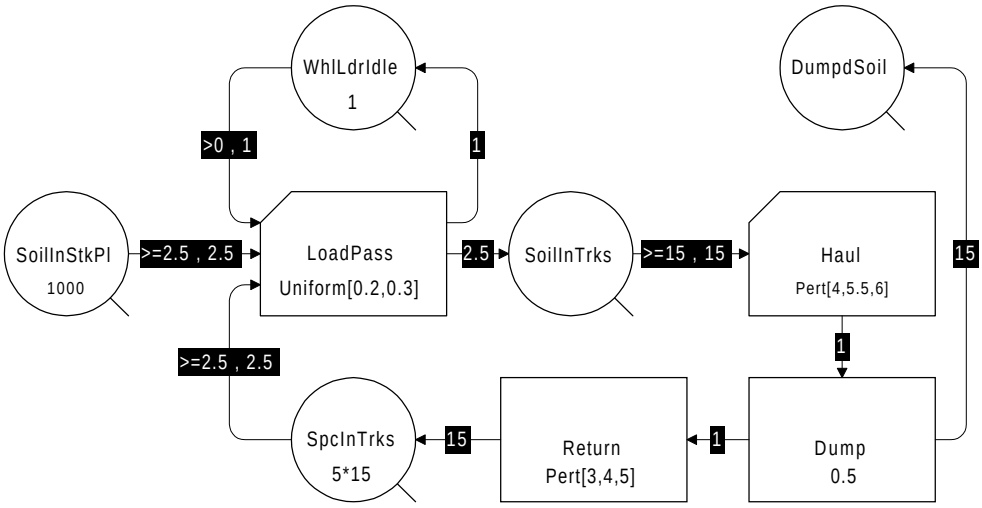

Figure 3 - ACD for Earthwork Operation With Multiple Loader Passes

3.3 Matching Cycles by Modeling Actual Resource Flows

EZStrobe has the ability to model resource amounts in bulk quantities and flexibility for specifying the startup conditions of activities. This allows it to model actual resource flows, conversion, and cycle matching. The classical example of modeling the loading of a truck on a per-pass basis, shown in Figure 3, illustrates this better.

In the model of Figure 3 the original truck cycle is broken up into three parts. Trucks flow only on the path *Haul-Dump-Return*, a resource that represents 'unoccupied space in trucks' flows on the path *Return-SpcInTrks-LoadPass*, and a resource that represents 'soil contained in trucks' flows on the path *LoadPass-SoilInTrks-Haul*. The *LoadPass* activity converts 2.5 cubic meters of unoccupied space into 2.5 cubic meters of soil in trucks, the *Haul* activity converts 15 cubic meters of soil into 1 truck, and the *Return* activity converts 1 truck into 15 cubic meters of unoccupied space. Thus, in this model the *LoadPass* activity represents loading one bucket-full of soil onto a truck. The *LoadPass* activity will take place 6 times for each truck that returns from the dumpsite (i.e., it takes six 2.5 cubic meter passes to consume 15 cubic meters of unoccupied space in trucks). Similarly, the *Haul* activity takes place once after every 6 terminations of *LoadPass* (i.e., six 2.5 cubic meter passes are needed to fill a truck -- Haul requires that the content of *SoilInTrks* be at least 15 before it will start). Note that in order to create the 5 trucks of this model it is necessary to initialize *SpcInTrks* with 75 cubic meters.

Running the model of Figure 3 produces the output shown in Table 3. The statistics reported for this model differ from those obtained for the model of Figure 2. The activity *LoadPass* is performed 6 times more than the others making the total count and average waiting times at the *WhlLdrIdle* Queue different from those in the previous run.

Table 3: Results From Running the Model in Figure 3

```
Statistics report at simulation time 160.338

Queue       Res    Cur     Tot AvWait AvCont SDCont MinCont MaxCont
================================================================
DumpdSoil   ezs 990.00  990.00  79.15 488.74 299.55    0.00  990.00
SoilInStkPl ezs   0.00 1000.00  75.27 469.47 298.25    0.00 1000.00
SoilInTrks  ezs  10.00 1000.00   0.71   4.44   4.62    0.00   15.00
SpcInTrks   ezs  65.00 1065.00   1.23   8.20  11.74    0.00   75.00
WhlLdrIdle  ezs   1.00  401.00   0.15   0.37   0.48    0.00    1.00

Activity Cur Tot 1stSt  LstSt AvDur SDDur MinD MaxD AvInt SDInt MinI MaxI
=============================================================================
Dump       0  66  6.91 155.92  0.50  0.00 0.50 0.50  2.29  1.21 0.35 6.69
Haul       0  66  1.51 151.13  5.30  0.34 4.19 5.90  2.30  1.07 1.40 5.71
LoadPass   0 400  0.00 151.89  0.25  0.03 0.20 0.30  0.38  0.52 0.20 4.41
Return     0  66  7.41 156.42  4.05  0.35 3.40 4.84  2.29  1.21 0.35 6.69
```

This is because the wheel loader visits the Queue 6 times for each truckload rather than just once.

The statistics for the *SpcInTrks* Queue are now in terms of cubic meters of unoccupied space rather than in terms trucks waiting. EZStrobe can collect and report further additional statistics about the "integral content" of a queue. The integral content of a queue is the content of a queue divided by some value and **truncated**. In this model, the integral content of *SpcInTrks* when the divisor is 15 represents the number of trucks waiting (e.g., 75 cubic meters of unoccupied space in trucks implies 5 completely empty trucks, and 67.5 cubic meters of unoccupied space in trucks implies 4 completely empty trucks). If, for example, integral statistics for *SpcInTrks* with divisor 15 were requested and named *TrucksWait* (integral statistics must be uniquely named), and if 3 histogram bins between 1 and 4 were requested for it; EZStrobe would produce the following additional output:

```
IntStat    CurVal Mean   SD Min   Max
==================================
TrucksWait 4.00   0.27 0.71 0.00 5.00

 TrucksWait integral statistics histogram

       Range     TotTime  %Time
   ============================
       < 1.00      134.00  83.57
       < 2.00      150.80  94.05
       < 3.00      154.82  96.56
       < 4.00      158.83  99.06
      >= 4.00        1.51   0.94
```

The first part of the above output contains one row for each queue for which integral statistics were requested. The values are analogous to the content of a fictitious queue that holds the truncated integral equivalent of the original queue. Additionally, there is one histogram for each integral equivalent for which histograms were requested.

3.4 Probabilistic Branching

EZStrobe can probabilistically select one among several successors to an activity for resource routing and activation. This is achieved with a Fork and the Branch Links that emanate from it. The EZStrobe ACD of Figure 4 illustrates this by expanding the model of Figure 2 to include the possibility of a truck breakdown.

In the model of Figure 4 there is a 5% chance that a truck will break down after dumping and that repairs will take between 10 and 60 minutes. The probability of a particular branch being selected is calculated by dividing its P value by the sum of the P values of all the branches that leave the link. Thus, the probability of the activity *Repair* starting when *Dump* ends is 5/(95+5)=0.05. Regardless of whether a truck breaks down or not, the *DumpdSoil* Queue will receive 15 units of resource (cubic meters of soil) because it is connected directly to *Dump*.

4 MODELING COMPLEX LOGIC

EZStrobe's essential modeling concepts have already been presented in the previous sections of this paper. EZStrobe's capability to model systems of moderate complexity, however, may not be obvious without an illustrative example. Consider a more complex version of an earthmoving operation where the haul road has a narrow portion that allows travel in only one direction (i.e., either loaded traffic or empty traffic, but never both simultaneously). The direction of travel is established by the first truck to arrive at the segment when it is empty. That direction is maintained until all the trucks that have arrived at the segment in that same direction have passed. If at that time trucks are waiting at the other end, then the direction of travel is reversed. The EZStrobe ACD for this operation is shown in Figure 5.

In order to understand the model it is necessary to have a clear picture of how this operation is implemented in practice. In this model, the haul road is divided into three segments, with the narrow segment in the middle. The green light is given to a loaded truck if it arrives to an empty segment, or if it arrives to the segment while the current direction is for loaded vehicles and the last vehicle in the segment has traveled enough to allow the trailing one to enter. The converse is true for empty trucks.

In the model of Figure 5 it is assumed that it takes 0.3 minutes for a truck to travel enough to allow a trailing truck to enter. The remainder of the narrow segment requires 1.45 minutes of travel time. The logic that implements access to the narrow segment in the model is in the center of the ACD. It consists of the following nodes: *SegSpots*, *LdGreen*, *SetEtGreen*, *EtGreen*, and *SetLdGreen*.

SegSpots represents the number of spots available to accommodate trucks in the segment. It is initialized to an arbitrarily large number that is guaranteed to exceed the segment capacity (100 in this case). A spot is consumed before each time a truck enters the segment loaded (*EnterLd*) or empty (*EnterEt*). A spot is returned after each time a truck exits the segment loaded (*FinishLd*) or empty (*FinishEt*). Thus, whenever the content of *SegSpots* is less than 100, it is because there are some trucks traversing the segment. When the content is greater than or equal to 100 (the "greater than" portion is not really necessary) the segment is empty.

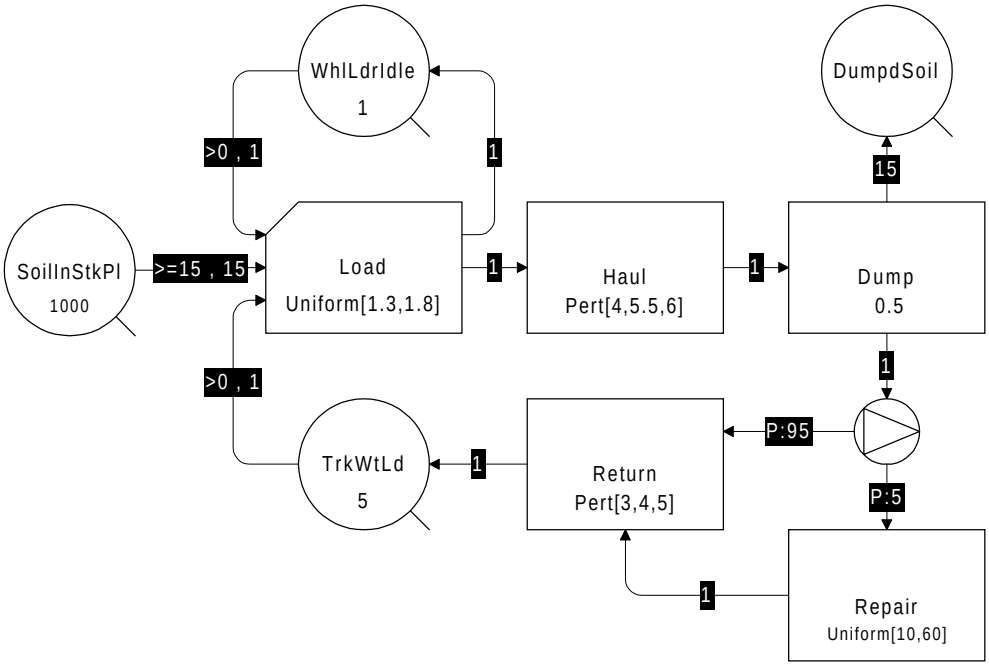

Figure 4 - ACD for Earthmoving Operation With Truck Breakdown and Repair

In order to change the side of the segment in which the green light is currently turning on and off (*SetEtGreen* and *SetLdGreen*), the narrow segment must be empty (otherwise entering trucks will face oncoming trucks head-on). This condition is expressed by the ">=100" that appears in the first part of the annotations for the links that connect *SegSpots* to *SetEtGreen* or to *SetLdGreen*. Note that the second part of the annotations for these two links is "0" to indicate that *SetEtGreen* and *SetLdGreen* do not actually remove spots from *SegSpots* when they start.

In order to not continuously and infinitely change the side of the segment holding the green, *SetEtGreen* and

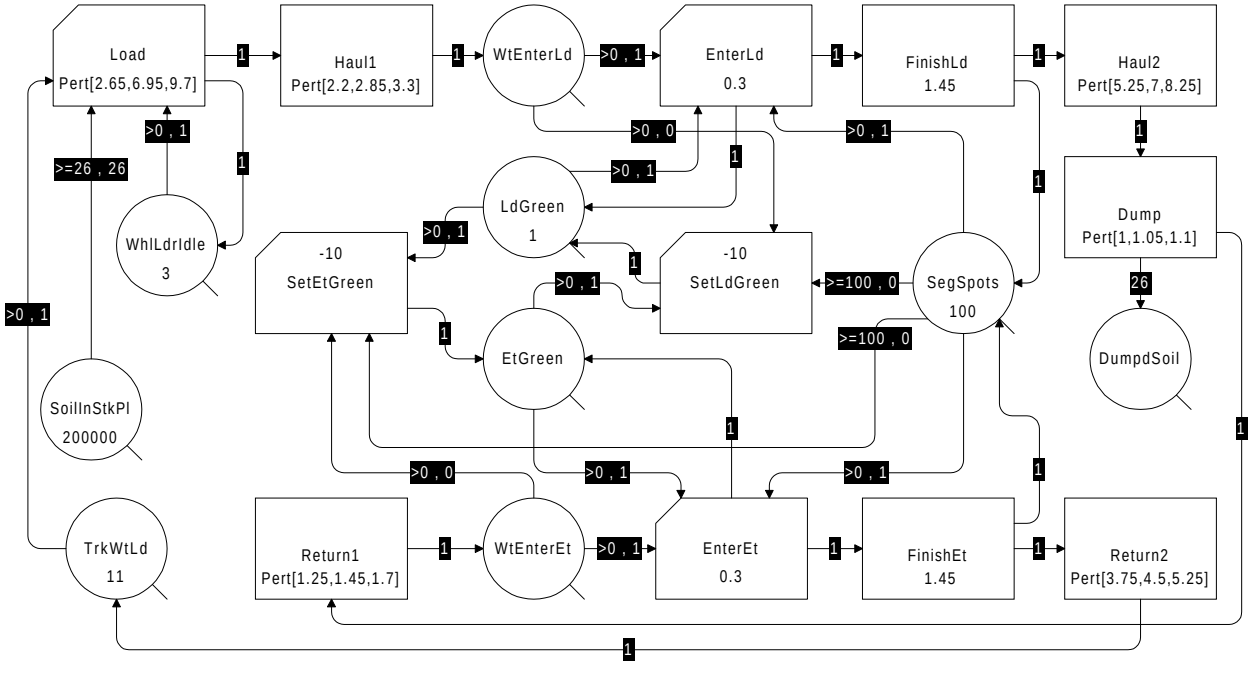

Figure 5 - ACD for Earthmoving Operation With Unidirectional Narrow Segment

SetLdGreen take place only if 1) trucks are waiting or arrive at the end of the segment to which they will set the green and if 2) the green is currently at the other side. The first condition for setting the green to the loaded side, for example, is indicated by the annotation (">0,0") on the link that connects *WtEnterLd* to *SetLdGreen* -- at least one truck (more than zero) is required in *WtEnterLd* to allow *SetLdGreen* to start, but no truck is removed. The second condition for setting the green to the loaded side, for example, is indicated by the link that connects *EtGreen* to *SetLdGreen* -- the green must be in *EtGreen* and will be moved from there to *LdGreen* instantaneously (*SetLdGreen* has no duration).

The green, however, is not always in *LdGreen* or *EtGreen*. Before a truck enters the segment loaded (*EnterLd*), the green must be turned on in the loaded side (*LdGreen* must contain the green). While that truck is entering the segment (during the first 0.3 minutes of their trajectory into the segment), the green is turned off but still in the loaded side (the green is removed from *LdGreen* when *EnterLd* starts and returned when it ends). This prevents other loaded trucks from entering the segment when the former is entering and ensures a minimum separation between trucks of 0.3 minutes. The same logic applies to the empty side.

The priorities of *SetEtGreen* and *SetLdGreen* are set to below normal for the extremely unlikely (and probabilistically impossible) situation where a truck arrives at the segment at the exact same moment at which the last truck in that same direction is just exiting the segment and there are trucks waiting in the other direction. In this case the negative priority prevents a switch in direction by allowing *EnterLd* or *EnterEt* to take hold of the green before *SetEtGreen* or *SetLdGreen*. Positive priorities, on the other hand, could be used to enable a switch in direction.

The conditions and resource removals that can be expressed in the link that connects a queue to a Combi are quite powerful. This example illustrates how it is possible to model moderately complex logic by using conditions and resource removal options of only a few forms.

5 CONCLUSION

This paper presented the basic elements of the EZStrobe modeling system and illustrated them with examples that progressed in complexity. EZStrobe is a simple system that is ideal as a first simulation tool and that can prove useful for modeling many operations that do not incorporate extremely complex logic or require uniquely identifiable resources with distinct characteristics. In addition, the EZStrobe concepts prove very useful in transitioning to STROBOSCOPE, the advanced and programmable simulation system in which EZStrobe is implemented and which can be used to model any operation regardless of its complexity. EZStrobe can be obtained from the web at: http://strobos.ce.vt.edu.

ACKNOWLEDGMENTS

The support of the National Science Foundation (Grant CMS-9733267) for portions of the work presented here is gratefully acknowledged. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the author and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

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

Martinez, J.C. 1996. STROBOSCOPE - State and Resource Based Simulation of Construction Processes. Doctoral Dissertation. Department of Civil and Environmental Engineering, University of Michigan, Ann Arbor, MI.

AUTHOR BIOGRAPHY

Julio C. Martinez is an Assistant Professor in the Department of Civil and Environmental Engineering at Virginia Tech. He received his Ph.D. in Civil Engineering at the University of Michigan in 1996; an MSE in Construction Engineering and Management from the University of Michigan in 1993; an MS in Civil Engineering from the University of Nebraska in 1987; and a Civil Engineer's degree from Universidad Catolica Madre y Maestra (Santiago, Dominican Republic) in 1986. He designed and implemented the STROBOSCOPE simulation language as part of his Ph.D. dissertation research. In addition to discrete-event simulation, his research interests include construction process modeling, decision support systems for construction, scheduling of complex and risky projects, and construction management information systems.