Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

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

Julio C. Martinez

200 Patton Hall
Virginia Tech
Blacksburg, VA 24061-0105 USA

ABSTRACT

EZStrobe is a very simple but powerful general-purpose simulation system designed for modeling construction operations, but domain independent and thus 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 develops an earthmoving example in increasing levels of complexity and detail to illustrate the range of modeling capabilities. This is a revised version of paper with the same title that appeared in a previous Winter Simulation Conference (Martinez 1998).

1 INTRODUCTION

Several simulation systems have been designed specifically for construction (e.g., Halpin 1992, Martinez 1996). These systems 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. Martinez and Ioannou (1999) describe in detail the differences between Activity Scanning and other paradigms. 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.

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.

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 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.

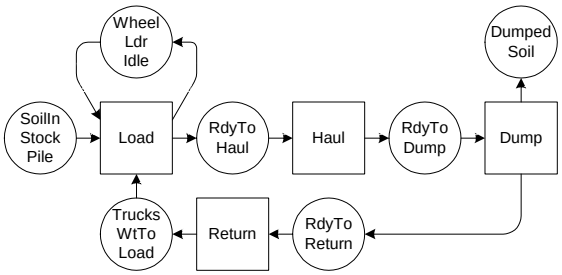


Figure 1: Conventional ACD for earthmoving operation

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.

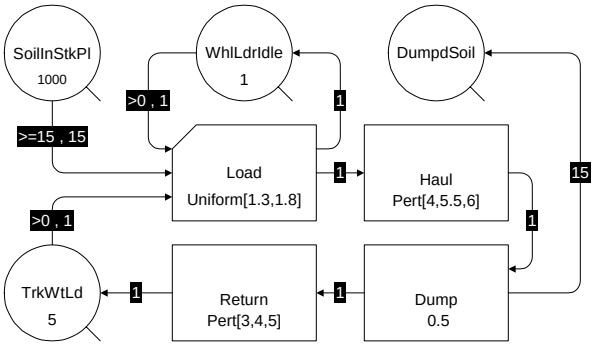


Figure 2: EZStrobe ACD for earthmoving operation

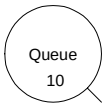
The network of Figure 2 is more compact than the one in Figure 1. Some queues such as *RdyToHaul* in Figure 1 are superfluous because they link activities that immediately and unconditionally follow each other. Such queues have been removed to indicate that some activities immediately follow their predecessors 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 *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.

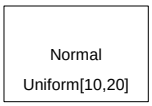
3.1 Basic EZStrobe Modeling Elements

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



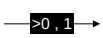
A **Queue** is a named element that holds idle resources. The name of the Queue is shown at the center. At the beginning of a simulation Queues hold a certain number of resources. This number is shown below the Queue name. Resources are placed in Queues when they are released by terminating instances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

A **Conditional Activity** (**Combi**) 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 Conditional Activity is shown at the center. The number at the top is the priority that the Conditional Activity has over other Conditional Activities when competing for resources in preceding Queues. A Conditional Activity with a high priority has a chance to start before a Conditional Activity 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 Conditional Activity is used to determine the duration of its instances. The duration formula typically samples from a probability distribution. Therefore, different instances of the same Conditional Activity can have different durations. Conditional Activities can only follow Queues, but can precede any other node other than a Conditional Activity.

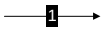


A **Bound Activity** (**Normal**) is a named element that represents tasks that start whenever an instance of any preceding Activity ends. The name of the Bound Activity is shown at the center. The formula at the bottom of the Bound Activity is used to determine the duration of its instances. The duration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

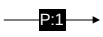
A **Fork** is a probabilistic routing element. It typically follows an activity but can also follow another Fork. When a preceding activity instance finishes, the Fork chooses one of its successors. If the chosen successor is a Bound Activity then the Bound Activity starts. If the chosen successor is 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 Conditional Activity. A Draw Link shows two pieces of information separated by a comma. The first part is the condition necessary for the successor Conditional Activity 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 Conditional Activity 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 Conditional Activity will attempt to remove from the predecessor Queue in the event that the Conditional Activity does start. The Conditional Activity 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 Conditional Activity. 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** connects a Fork to any other node except a Conditional Activity. 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. A report for the model shown in Figure 2, for example, is shown below.

```
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
```

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). Note from the output that *SoilInStkPl* contains 10 units of resource (cubic meters) at the time of the report. Those resources remained in *SoilInStkPl* because they were not enough to enable *Load* to start (which requires 15) one more time.

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 < 1, i.e., zero) 82.47% of the time, and contained exactly 4 or more trucks 3.65% of the time.

3.3 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 3 illustrates this by expanding the model of Figure 2 to include the possibility of a truck breakdown.

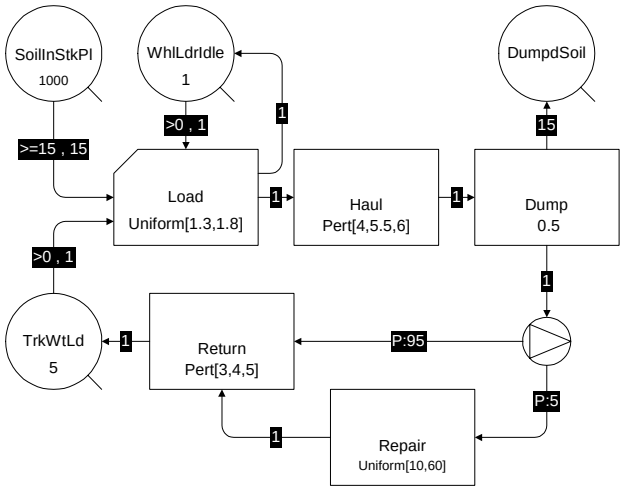


Figure 3: ACD for earthmoving operation with truck breakdown and repair

In the model of Figure 3 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 detailed and complex version of an earthmoving operation where 1) an excavator is used instead of a wheel loader and its cycle is modeled explicitly and 2) 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). An EZStrobe ACD that incorporates these details and complexities is shown in Figure 4 and explained in the following two subsections.

4.1 Modeling The Excavator Cycle

Wheel loaders load trucks with material that has already been excavated and stockpiled. Excavators, on the other hand, dig material from their undisturbed natural state. This is done in a cycle where the excavator swings empty from the truck loading position to the digging position, excavates, swings loaded from the digging position to the truck loading position, waits for a truck if one is not already there, and dumps the excavated material unto the truck.

The components of the excavator cycle are represented by the *SwingEmpty*, *Excavate*, *SwingLoaded*, *ExcWtDmp*, and *DumpBucket* nodes located in the top left part of the ACD. In this cycle, *DumpBucket* is the only Conditional Activity. According to the ACD, the conditions needed for *DumpBucket* to start are that a truck be under the excavator (*TrkUndrExc* contains a truck) and that the excavator be waiting to unload its bucket unto a truck (*ExcWtDmp* contains the excavator). The link that connects *TrkUndrExc* to *DumpBucket* indicates, however, that zero trucks are removed from *TrkUndrExc* when *DumpBucket* starts. This is consistent with reality because the truck needs to be under the excavator to receive a bucket load, but remains under the excavator after receiving the load. In this model, the truck that is under the excavator and the soil that it contains are represented by two separate queues, *TrkUndrExc* and *SoilInTrk*. Every time the *DumpBucket* activity concludes, 2.5 m3 of soil are placed in *SoilInTrk*.

The *Haul1* conditional activity takes place whenever there is a full truck under the excavator (*TrkUndrExc* contains a truck and *SoilInTrk* contains 15 m3 of soil). It is when this activity starts that the truck leaves the spot under the excavator and that the *TrkUndrExc* and *SoilInTrk* queues are cleared. *Haul1* represents the truck traveling loaded from the load area towards the entrance to the narrow portion of the road.

In this detailed modeling of the truck loading, the next truck to be loaded needs to enter the loading area and position itself under the excavator. This is represented by the *EnterArea* conditional activity. According to the ACD, *EnterArea* takes place when there is at least one truck waiting to be loaded (the content of *TrkWtLd* is >0), maneuvering space is available (the content of *ManeuvSpc* is >0), and there is no truck under the excavator (*TrkUndrExc* is empty). The link that connects *TrkUndrExc* to *EnterArea*

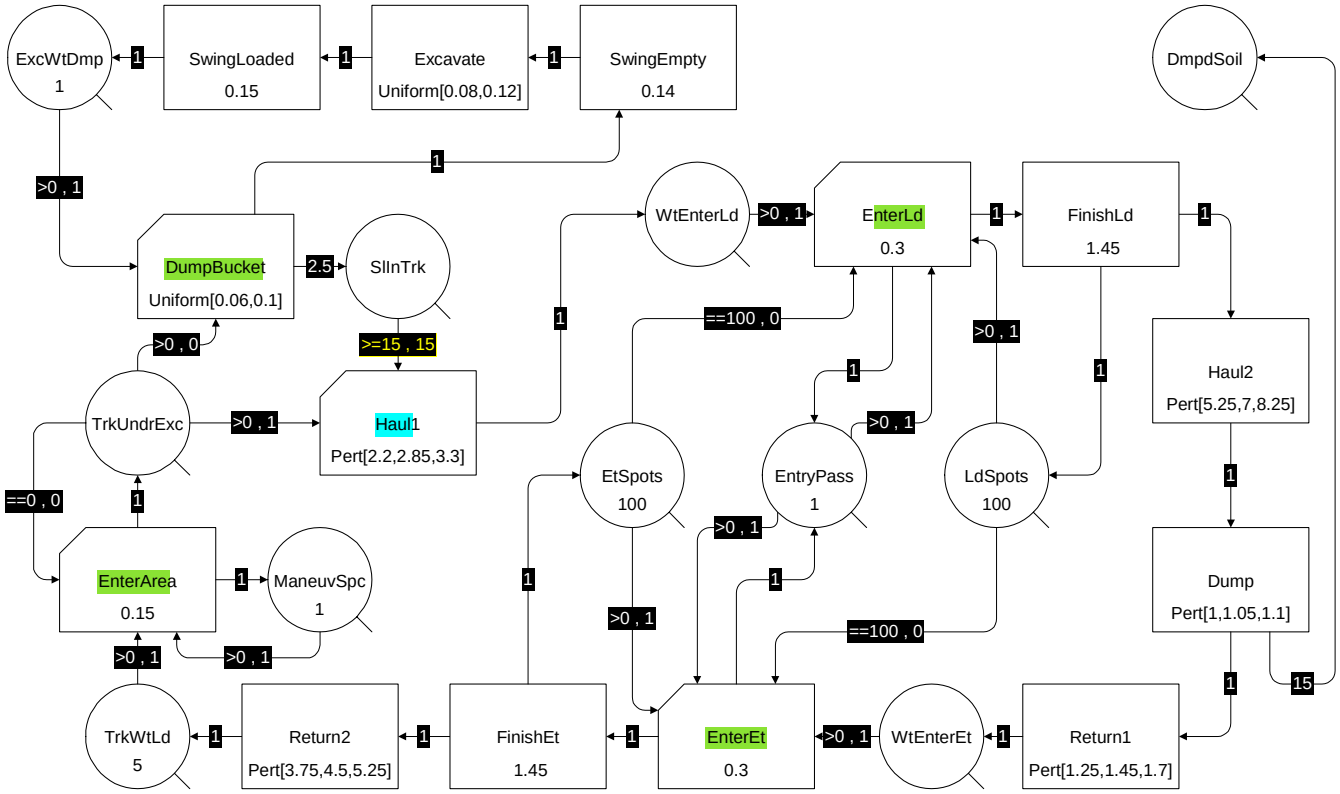


Figure 4: ACD for earthmoving operation with unidirectional narrow segment

indicates that no truck is removed from *TrkUnderExc* when *EnterArea* starts. This is specified for clarity. Regardless of the number specified, no truck will be removed because the activity can only start when none are there. The maneuvering space needs to be modeled explicitly because otherwise several trucks could simultaneously enter the loading area. When *EnterArea* concludes, a truck is placed under the excavator and the maneuvering space is made available.

In cases where the earthmoving operation is clearly undertrucked, trucks will receive their first bucket load immediately upon entering the area because the excavator will be waiting with its load ready to be dumped unto a truck. In cases where the operation is clearly overtrucked, the truck will be positioned to receive a load before the excavator has completed its cycle. Thus, the more detailed modeling of the loading captures aspects of the operation that cannot be represented by simply repeating the excavator cycle 6 times for each truck, as illustrated in the previous version of this paper (Martinez 1998) using EZStrobe, and as traditionally done when modeling with CYCLONE.

4.2 Modeling The Narrow Segment

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 as long as trucks keep arriving at the segment in that same direction or until the segment is again empty. If trucks are waiting at the other end when the segment becomes empty, then the direction of travel is reversed to allow those trucks to travel. The portion of the EZStrobe ACD in Figure 4 that models this narrow segment consists of the following nodes: *WtEnterLd*, *EnterLd*, *FinishLd*, *EtSpots*, *EntryPass*, *LdSpots*, *FinishEt*, *EnterEt* and *WtEnterEt*.

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. A truck arriving to the narrow segment is allowed to enter if no trucks are traversing the segment in the opposite direction. In addition, for reasons of physical space, a vehicle must wait until the vehicle ahead of it has traveled enough into the segment to allow it to enter (i.e., the entrance to the segment must be cleared). In the model discussed here it is assumed that this takes 0.3 minutes, and is represented by the *EnterLd* (entering loaded headed towards the dump site) and *EnterEt* (entering empty returning to the load area) Conditional Activities. The resource initially placed in the *EntryPass* queue ensures that only one truck enters the narrow segment at a time from either end. This single resource is required for both *EnterLd* and *EnterEt* to start and is removed from *EntryPass* at the startup of either ac-

tivity. While *EntryPass* is empty neither *EnterLd* nor *EnterEt* can take place.

The remainder of the narrow segment requires 1.45 minutes of travel time. This is represented by the *FinishLd* and *FinishEt* activies which are bound to *EnterLd* and *EnterEt*, respectively (i.e., an instance of *FinishLd* or *FinishEt* starts every time an instance of *EnterLd* or *EnterEt* finishes). Because the time to traverse the remainder of the narrow segment is larger (1.45 min) than the time to enter it (0.30 min), it is possible for several instances of *FinishLd* or *FinishEt* to take place simultaneously (i.e., several trucks with 0.3 min separation may be traversing the narrow segment).

Every time *EnterLd* starts it removes a resource from *LdSpots*. Every time *FinishLd* ends it deposits a resource in *LdSpots*. Because *LdSpots* is initialized with a large number (100), its content will never drop to zero. In effect, the number of resources below 100 in *LdSpots* is a count of the number of loaded trucks currently traversing the segment. When the content of *LdSpots* is 100, it is because no loaded trucks are currently traversing the narrow segment. This information is very valuable, and is used as one of the conditions needed to allow empty trucks to enter the narrow segment (the ==100 in the link that connects *LdSpots* to *EnterEt*).

Likewise, *EtSpots* and the links that connect it to *EnterEt* and from *FinishEt* maintain and provide information about the number of empty trucks traversing the narrow segment. The condition that no empty trucks be traversing the segment (i.e., that the content of *EtSpots* is 100) is similarly used as one of the conditions necessary for a loaded truck to enter the segment.

Thus, according to the ACD of Figure 4, for a loaded truck to enter the narrow segment, the following 4 conditions are required: 1) The content of *WtEnterLd* must be greater than 0 (i.e., a loaded truck must be waiting to enter), 2) The content of *EtSpots* must be 100 (i.e., no empty trucks can be traversing the narrow segment), 3) The content of *EntryPass* must be greater than zero (i.e., the entrance to the segment must be cleared), and 4) The content of *LdSpots* must be greater than zero (which will always happen).

When *EnterLd* does start (e.g., a loaded truck enters the narrow segment), it 1) Acquires one resource from *WtEnterLd*, 2) Leaves the content of *EtSpots* intact, 3) Acquires the resource in *EntryPass* and 4) Acquires a resource from *LdSpots* (this resource will not be returned to *LdSpots* until the instance of *FinishLd* that is bound to the starting instance of *EnterLd* finishes).

By modeling the empty direction of the narrow segment the same way, the desired operation is achieved.

There are many ways in which a given operation can be modeled. Too see a slightly different approach to modeling the narrow segment with EZStrobe, see the previous version of this paper (Martinez 1998).

The conditions and resource removals that can be expressed in the link that connects a queue to a Conditional Activity 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 MODELING AND PARAMETERIZING LARGE OPERATIONS

EZStrobe has some advanced features that allow parameterizing input, customizing output, defining model behavior dependent on the dynamic model state, building multi-page models, publishing models to be run over the web, and animation of running models for model verification (debugging). The diagram of Figure 5 is a multi-page model of the same operation represented in Figure 4, and briefly illustrates some of these advanced features. Each of the boxed portions of Figure 5 is independent and disconnected from other boxed portions. They can each be placed in separate pages, they can all be disconnected parts of a very large page, or some of the smaller boxed items can be grouped in one page. For the purposes of this discussion, each boxed portion of Figure 5 is assumed to be placed in a separate page named as indicated in the top of the box. Thus, the model now contains 6 separate pages.

5.1 Fusion Queues and Multi-page Models

Fusion Queues are nodes that look like ordinary Queues but are drawn with dashed line type (e.g., *WtEnterLd* in the top left part of the "Narrow Segment" page in Figure 5). Fusion Queues must bear the name of an ordinary Queue that exists elsewhere in the model (e.g., the *WtEnterLd* ordinary Queue is on the right edge of the "Excavator Cycle and Truck Loading" page in Figure 5). A model can contain several Fusion Queues with the same name of an ordinary Queue. All such Fusion Queues are assumed to be one and the same as the ordinary Queue they are named after. *WtEnterLd*, for example, receives resources from *Haul1* in page "Excavator Cycle and Truck Loading" and provides resources to *EnterLd* in page "Narrow Segment". This capability can be used to break up a model into separate pages or to reduce the number of links that cross each other in complex pages. In order to do this, it may be necessary to replace a Bound Activity (Normal) with a Queue followed by a Conditional Activity (Combi). The *Haul2* Bound Activity in Figure 4, for example, was replaced by the *WtHl2* Queue and the *Haul2* Conditional Activity in Figure 5. Fusion Queues were inspired by the Fusion Places used in some extended Petri Nets and described by Sawhney (1999) in the modeling of steel erection operations.

5.2 Parameterizing Models

The performance of a given system (e.g., expected cost per cubic meter for moving earth) depends on the values of the key decision variables (e.g., the number of trucks to use) and other variable data (e.g., the hourly cost of the equipment used). These values are often used in different parts of the model and their definition should be located in a single place to facilitate experimentation and avoid the mistakes that result from inconsistent changes. EZStrobe "Parameters" allow the model designer to assign a symbolic name and description to these values. The name of the parameter can then be used throughout the model. The "Model Parameters" page in the model of Figure 5 shows how the amount of soil to be moved; the number of trucks to be used; and the hourly cost of trucks, excavator and indirects; have been defined as parameters. The initial number of trucks in the *TrkWtLd* Queue, for example, has been specified by using the name of the parameter (*nTrucks*) rather than its value.

By using parameters it is possible to create generic models that adapt to a wide range of similar operations. Such models can be reused by specifying appropriate parameter values.

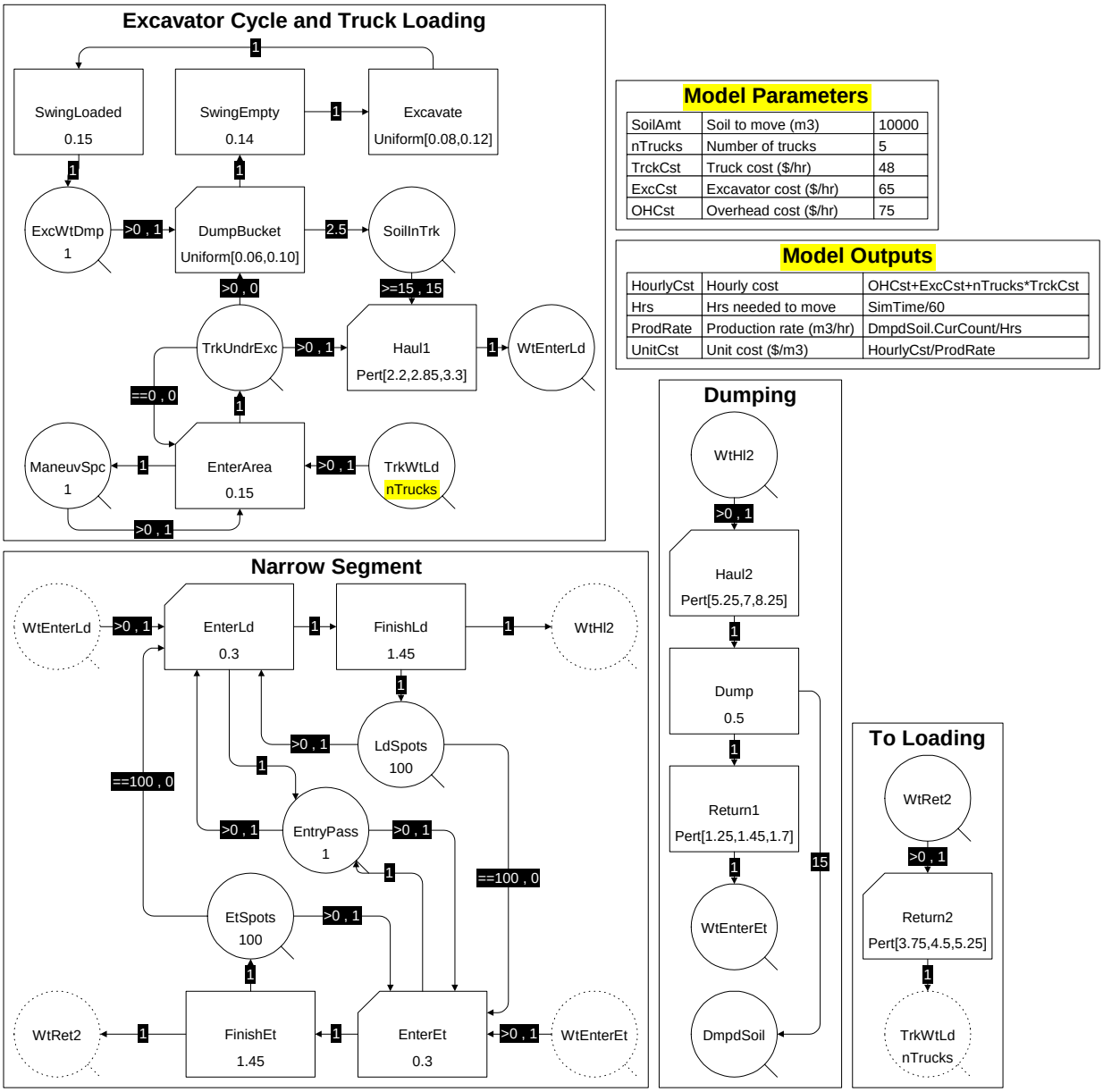


Figure 5: Multipage, parameterized model with custom outputs

5.3 Customizing Output

Typical decisions about a system often depend on measures of performance that must be calculated from statistical output. In the case of the earthmoving operation described in this paper, the most important measure of performance is probably the cost per cubic meter. Other important measures of performance may include the time required to move the material and the production rate. EZStrobe "Results" allow the definition of performance measures with formulas that depend on parameters, statistics from model execution, and other Results. The "Model Outputs" page in Figure 5 shows how some Results have been defined.

When models contain "Parameters" and/or "Results", the output includes both as shown below:

```
** Model input parameters **

Amount of soil in m3 : 10000
Number of trucks     : 5
Truck cost ($/hr)    : 48
Excavator cost ($/hr): 65
Overhead cost ($/hr) : 75

** Calculated results after simulation **

Hrs needed to move material: 51.7635
Production rate (m3/hr)    : 193.283
Unit cost ($/m3)           : 1.96603
```

When running multiple replications, EZStrobe collects statistics about each "Result" automatically and presents them in a table such as the one below:

```
** Calculated results after simulation **

    Run               Hrs          ProdRate          UnitCst
=======     ==============   ==============   ==============
      1          51.7634796        193.282988        1.96602921
      2          52.0920414        192.063888        1.97850832
      3          51.7088068         193.48735        1.96395268
      4           51.841363        192.992611         1.9689873
      5          52.1091735        192.000742        1.97915902
=======     ==============   ==============   ==============
Average          51.9029729        192.765516        1.97132731
Std Dev          0.18656115       0.692378978      0.0070857808
Minimum          51.7088068        192.000742        1.96395268
Maximum          52.1091735         193.48735        1.97915902
```

In addition, a model with Parameters and Results can be converted to an equivalent web-ready STROBOSCOPE model. When such a model is invoked over the web and the server runs the STROBOSCOPE Server Extensions, a form prompting for the values of the parameters is presented and the results are sent back after the simulation has run on the server.

5.4 Dynamic Model Behavior

Each of the numerical data that appears on an EZStrobe ACD, such as those used to define activity durations or the amount of resource to be released by a link, can be specified with a dynamic formula. These formulas can contain function calls (e.g., Sin, Log), arithmetic operators, variables that represent the dynamic model state (e.g., the current content of a Queue or the number of times an activity has taken place), model parameters, and model results.

This capability enables modeling quite complex situations such as dynamic logic and non-stationary activity duration distributions. The duration of the *Excavate* activity in Figure 5, for example, could be set to: *Uniform[0.08,0.12]*Excavate.TotInst^-0.12* to represent a stochastic learning effect in which the excavation times tend to decrease as experience is gained. "Experience" in this example is represented by *Excavate.TotInst*, which dynamically returns the number of times that the *Excavate* activity has started. Refer to (Martinez and Ioannou 1999) for more examples of this. Although those examples are modeled with STROBOSCOPE, they are equally applicable to EZStrobe.

5.5 Model Animation for Verification

The first model of a complex system is rarely a correct representation of the modelers' understanding of the real system. By running the model and analyzing its results it is often possible to detect some errors, other errors may not be readily apparent and may go undetected. Trace files of the simulation run can help, but it is an extremely cumbersome process that becomes unmanageable for most non-trivial models.

EZStrobe offers graphical and interactive model verification (debugging) by means of model animation. Similar ideas have been used by (Huang and Halpin 1994) and (Shi 2000) for other objectives (e.g., study of transients and communication to people without modeling knowledge). EZStrobe's animation capabilities are designed specifically for the model developer to understand and gain confidence in the model's correctness. The animator graphically illustrates the dynamic state of the simulation (e.g., current content of queues and number of ongoing activity instances) and the events that take place during simulation (e.g., when an instance of an activity starts or ends, when a queue receives resources, or when resources flow through links).

Figure 6 shows a snapshot of the EZStrobe animation controller and a portion of a model being animated. The animator is set to stop after every event, and is currently at simulation time 9.7. The thick red border on *DumpBucket* indicates that it is terminating an instance. Specifically, the "20" on the top right indicates the specific instance that is finishing (zero based count). The "1/21" in the top middle indicates that one instance is currently taking place (the one terminating) and that 21 have started since model execution began. The blue thick border on *SwingEmpty* indicates that it is starting an instance (this happens while *DumpBucket* is finishing). The "20" on the top left of the activity indicates the specific instance that will start. The "0/20" in the top middle indicates that no instances are taking place (the one currently starting does not count until after its duration has been sampled), and that a total of 20 have begun. The thick line used for the link that connects *DumpBucket* to *SlInTrk* indicates that 2.5 resource units are being sent to the Queue. The "5" in the top of *SlInTrk* indicates its current content. If the "Continue" button is

pressed on the controller, the link's line will return to normal thickness, *SlInTrk's* border will turn thick, and the number on top will be updated to "7.5".

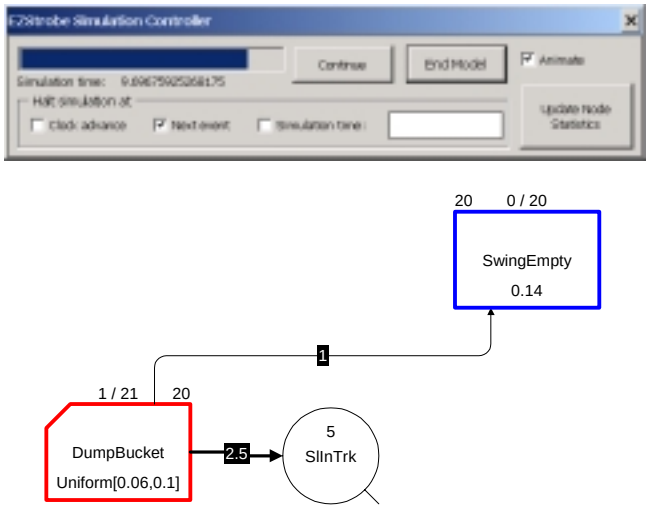


Figure 6: EZStrobe Animation Snapshot

While model animation is halted, it is possible to update and subsequently inspect the entire state of the simulation (by pressing the "Update Node Statistics" button).

The model animation capabilities prove to be very useful for individuals who are learning the system to grasp precisely how the EZStrobe modeling methodology works, and to learn by "experimenting and seeing".

6 CONCLUSION

This paper presented the basic elements of the EZStrobe modeling system and illustrated them with an increasingly complex earthmoving example. 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.

Huang, R.Y. and Halpin, D.W. 1994. "Visual Construction Operations Simulation - The DISCO Approach", *Journal of Microcomputers in Civil Engineering*, (9) 175-184.

Martinez, J.C. 1998. "EZStrobe -- general-purpose simulation system based on activity cycle diagrams". *Proceedings of 1998 Winter Simulation Conference*, pp. 341 – 348.

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.

Martinez, J.C. and Ioannou, P.G. 1999. "General Purpose Systems for Effective Construction Simulation", Journal of Construction Engineering and Management. ASCE. 125 (4), pp. 265-276.

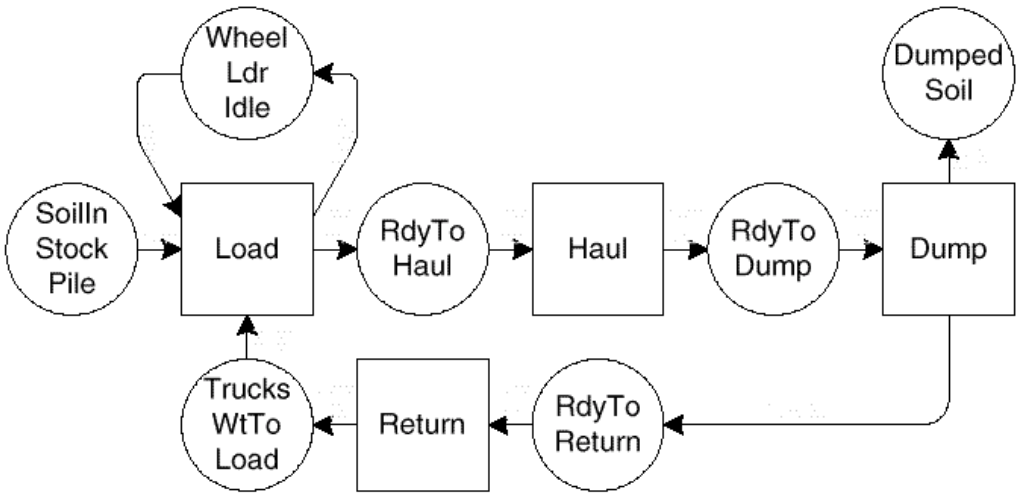
Sawhney, A.; Mund, A. and Marble, J. 1999. "Simulation of the structural steel erection process". *Proceedings of the 1999 Winter Simulation Conference*. pp. 942-947.

Shi, J.J. 2000. "Object-Oriented Technology for Enhancing Activity-Based Modeling Functionality". *Proceedings of the 2000 Winter Simulation Conference*.

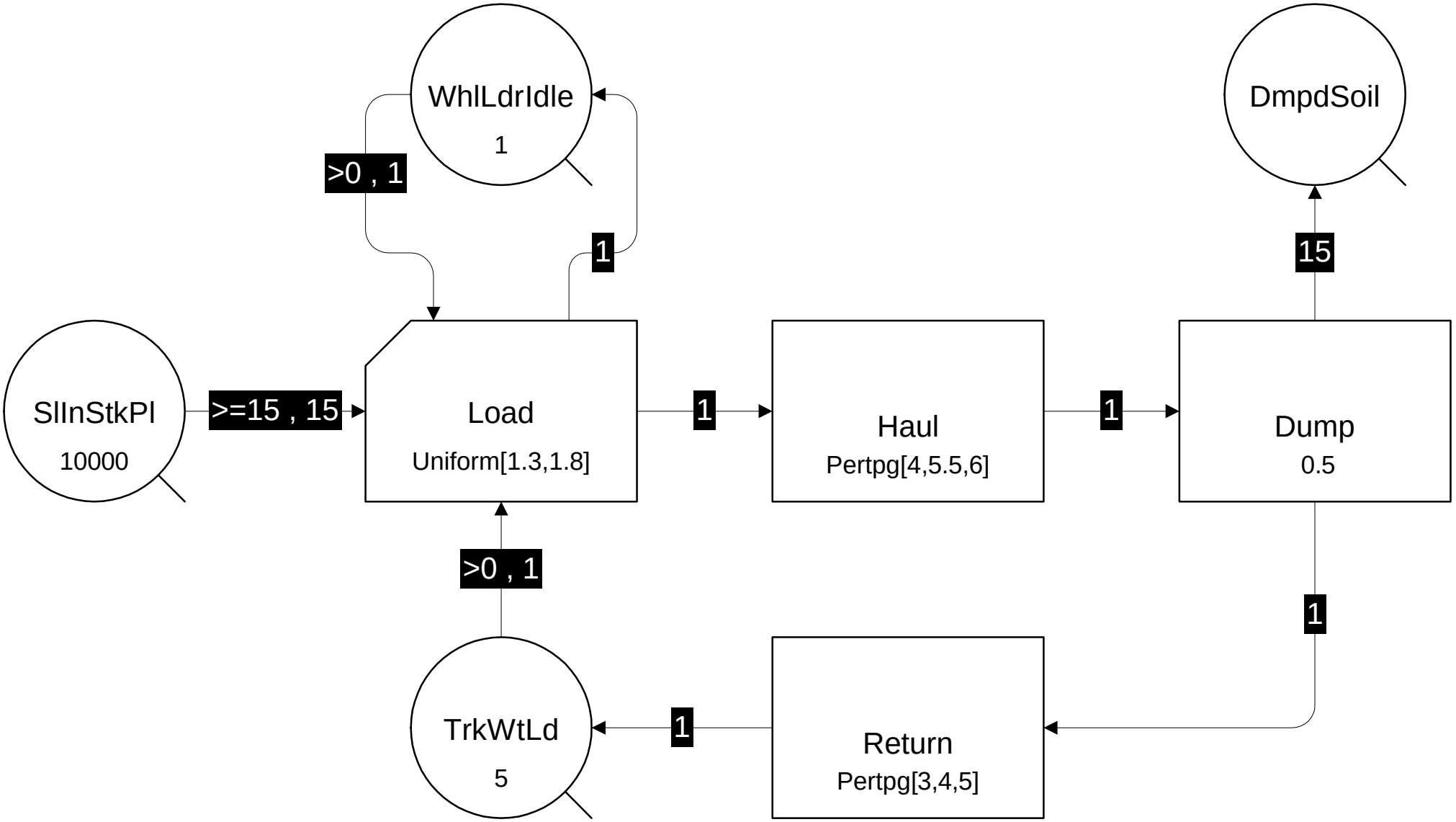
AUTHOR BIOGRAPHY

JULIO C. MARTINEZ is Assist. Prof. in the Via Dept. of Civil and Env. Eng. at Virginia Tech. He received his Ph.D. in Civil Eng. at the Univ. of Michigan in 1996; an MSE in Construction Eng. and Mngmt. from the Univ. of Michigan in 1993; an MS in Civil Eng. from the Univ. of Nebraska in 1987; and a Civil Eng. degree from Universidad Catolica Madre y Maestra (Santiago, Dominican Republic) in 1986. He designed and implemented the STROBOSCOPE simulation language and a host of other tools based on it. In addition to discrete event simulation, his research interests include advanced visualization and animation, construction process modeling, decision support systems for construction, scheduling of complex and risky projects, and construction management information systems.

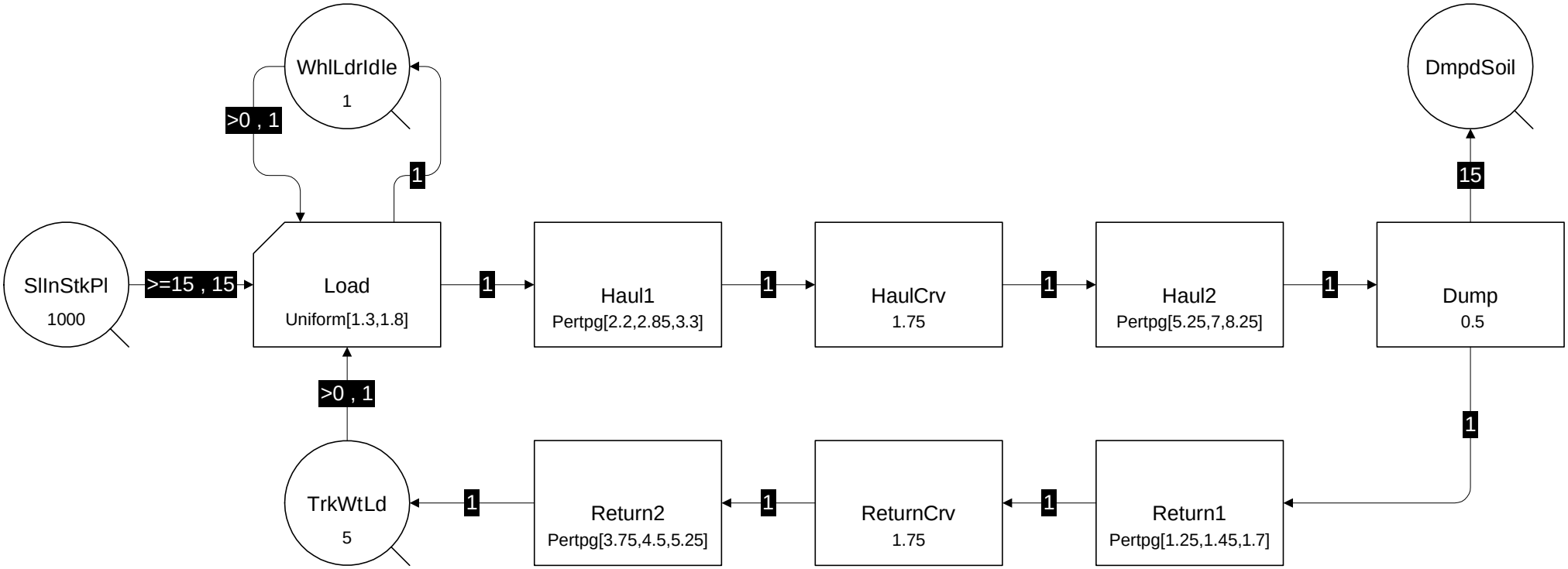
Conditions Needed to Start	Activity	Outcome of Activity
Wheel loader idle at source. Empty truck waiting to load. Enough soil in stockpile.	Load	Wheel loader idle at source. Loaded truck ready to haul.
Loaded truck ready to haul.	Haul	Loaded truck ready to dump.
Loaded truck ready to dump.	Dump	Dumped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



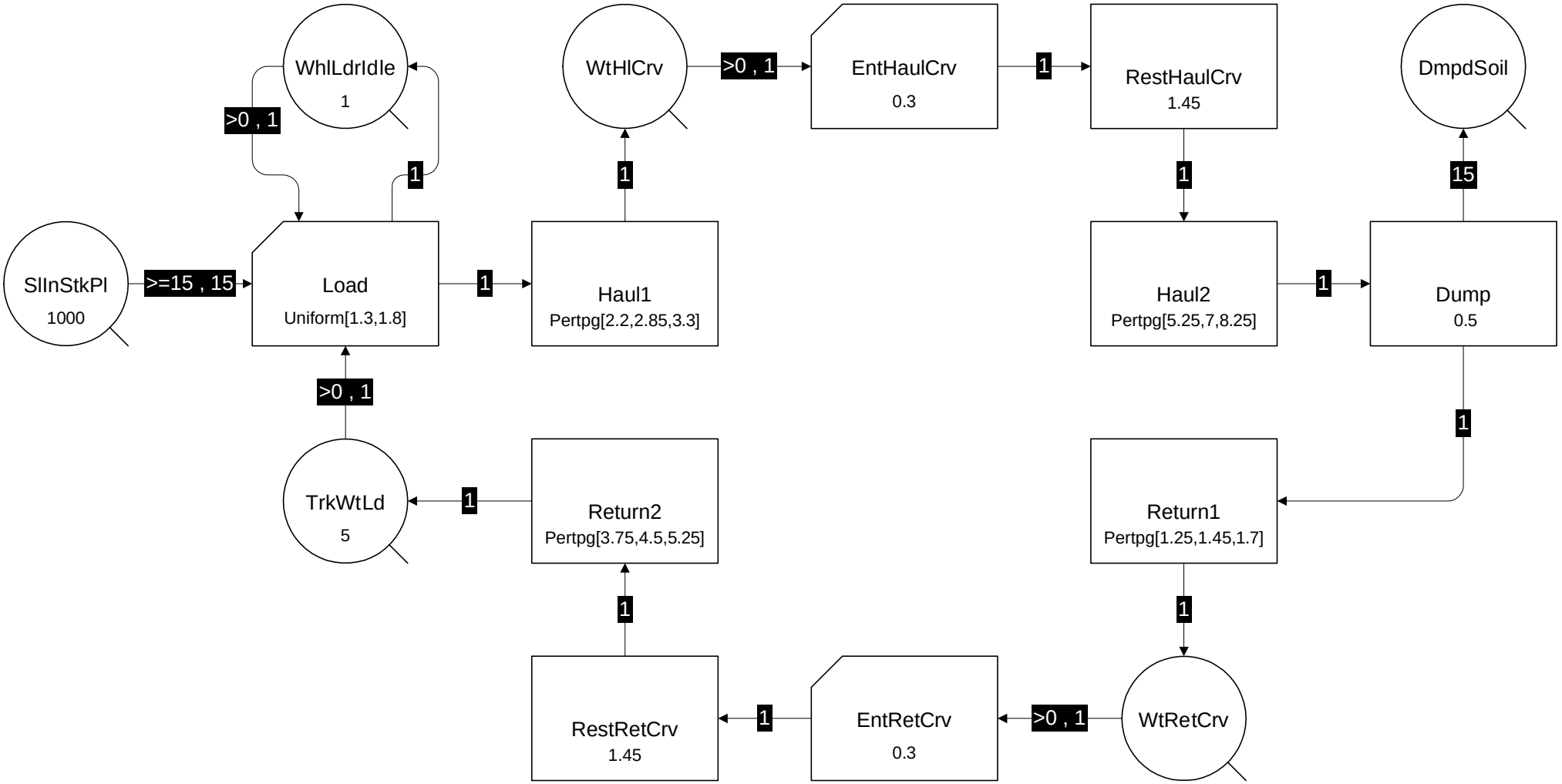
This is a classic model for the movement of 10000 m3 of soil using 15 m3 trucks and a wheel loader. The loading time is that required for the wheel loader to completely fill a truck. There is no restriction for dumping, which can take place immediately after hauling.



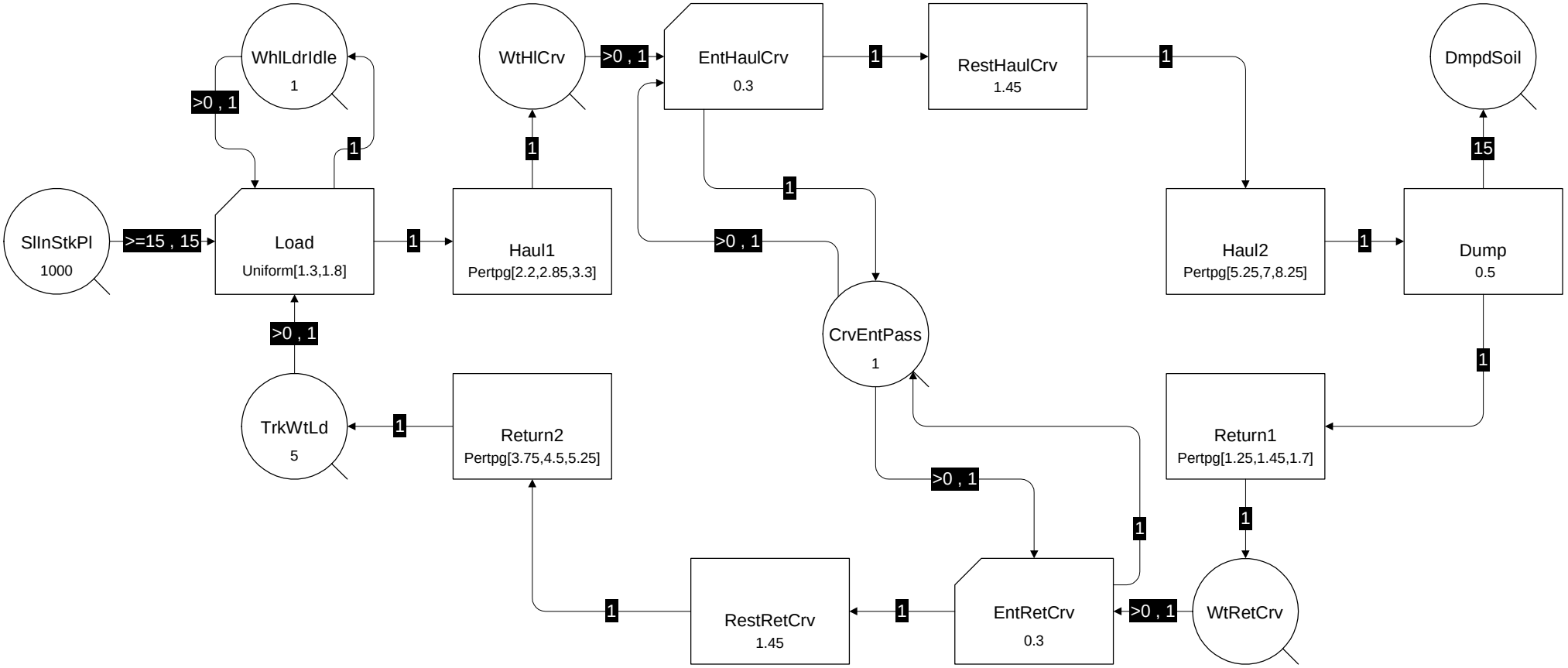
In this model the haul road has been broken into three parts. Haul now consists of a Haul1, HaulCrv, and Haul2. Similarly, Return now consists of Return1, Return Crv, and Return2. HaulCrv and ReturnCrv are for crossing a narrow, curved segment. This segment will eventually allow traffic in only one direction with no passing (single file loaded or empty traffic but not both at the same time). This is the first modification in a series of steps required to implemen t this.



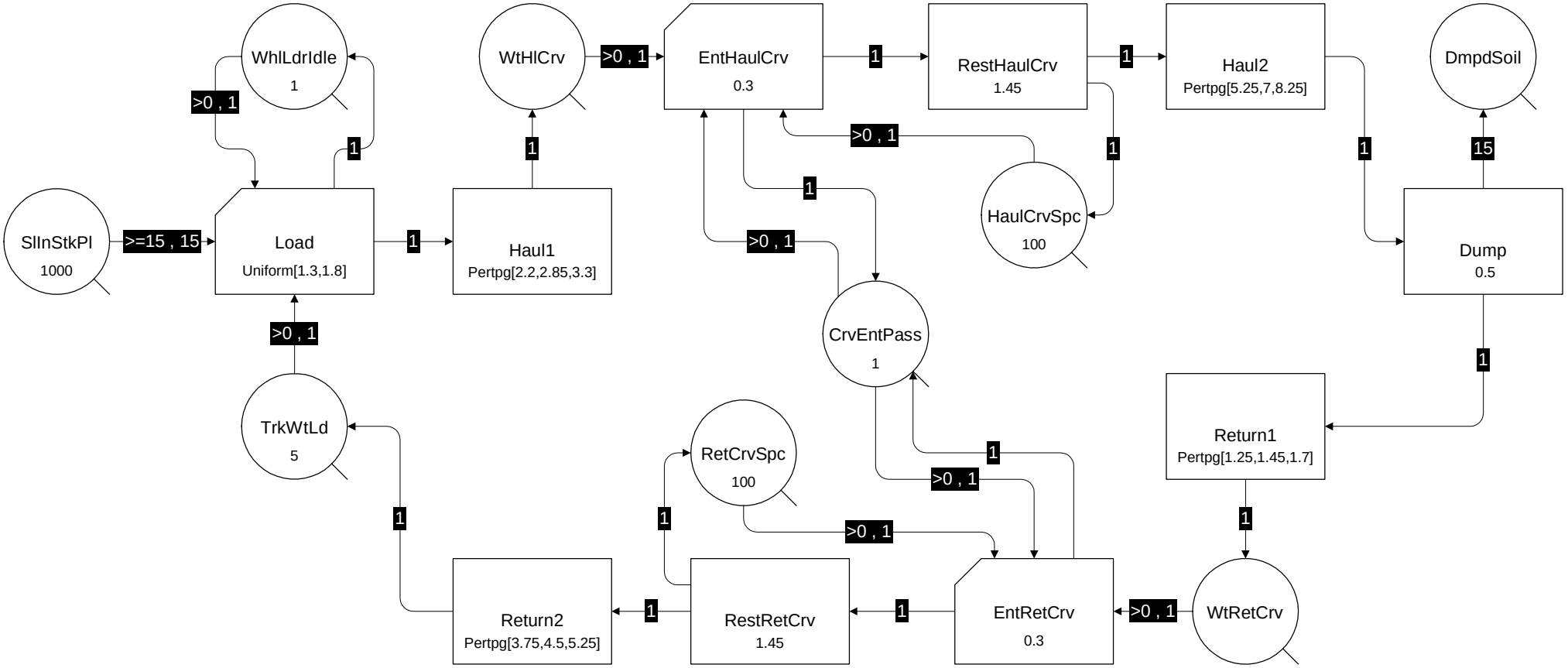
This model is functionally equivalent to the previous model. The haul and return over the narrow curve has been broken up into two parts, and preceded by re spective queues. WtHlCrv and WtRetCrv are conceptually necessary since crossing over the narrow segment is not an immediate consequence of arriving at the segment. A truck may need to wait for traffic to switch to its direction or for the truck ahead to make space. The crossing of the curve itself has been broken up into two par ts, one to represent the entry into the curve and the other to represent the traversal of the remainder. This is done in preparation to recognize that although more than one truck can traverse the curver at the same time, only one can be entering it.



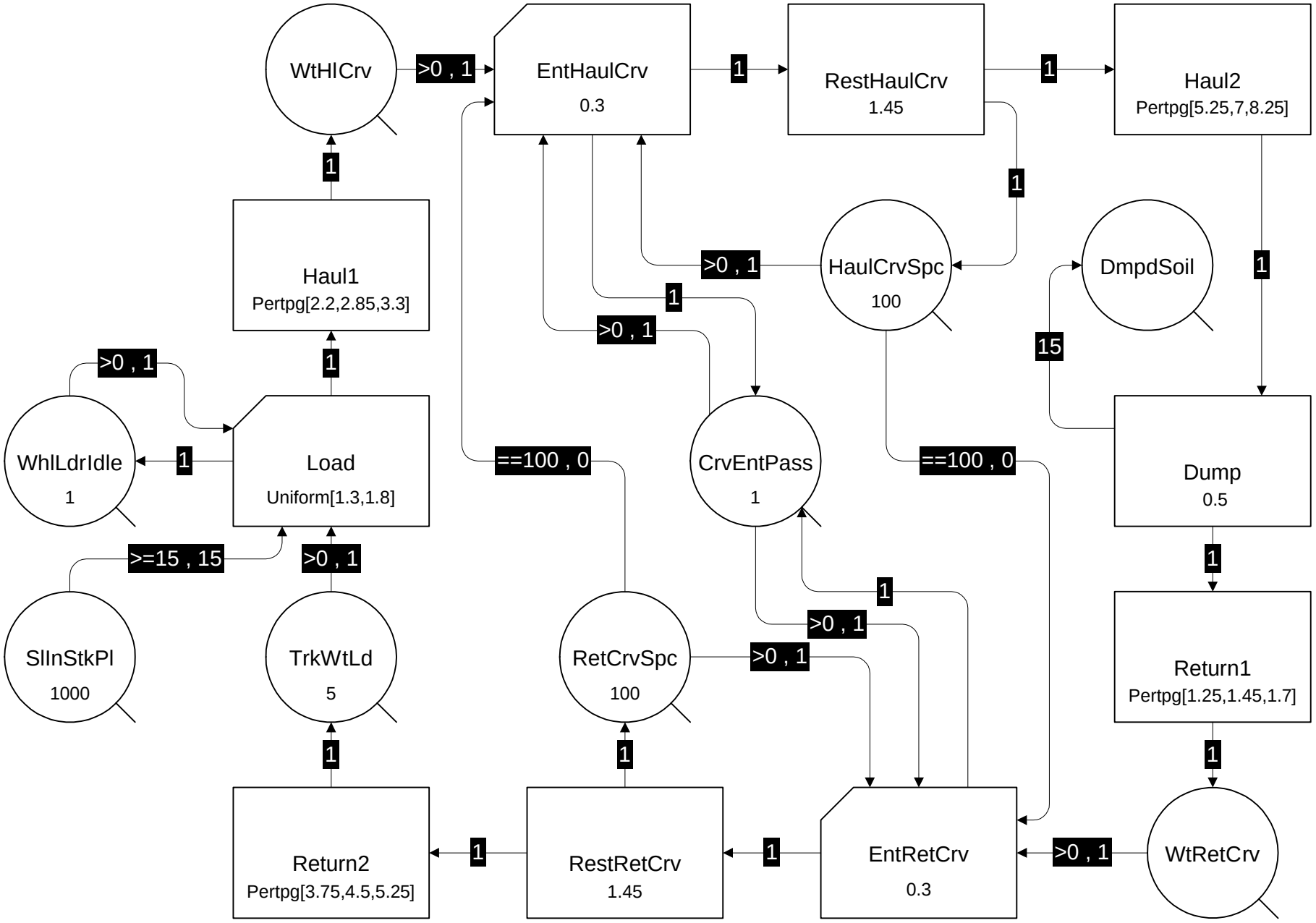
In this model, the entry of trucks into the curve has been forced to be sequential by the introduction of CrvEntPass, which is initialized with only one resource. Now only one truck can enter the curve at a time (from either side). This model still does not prevent head-on collisions.



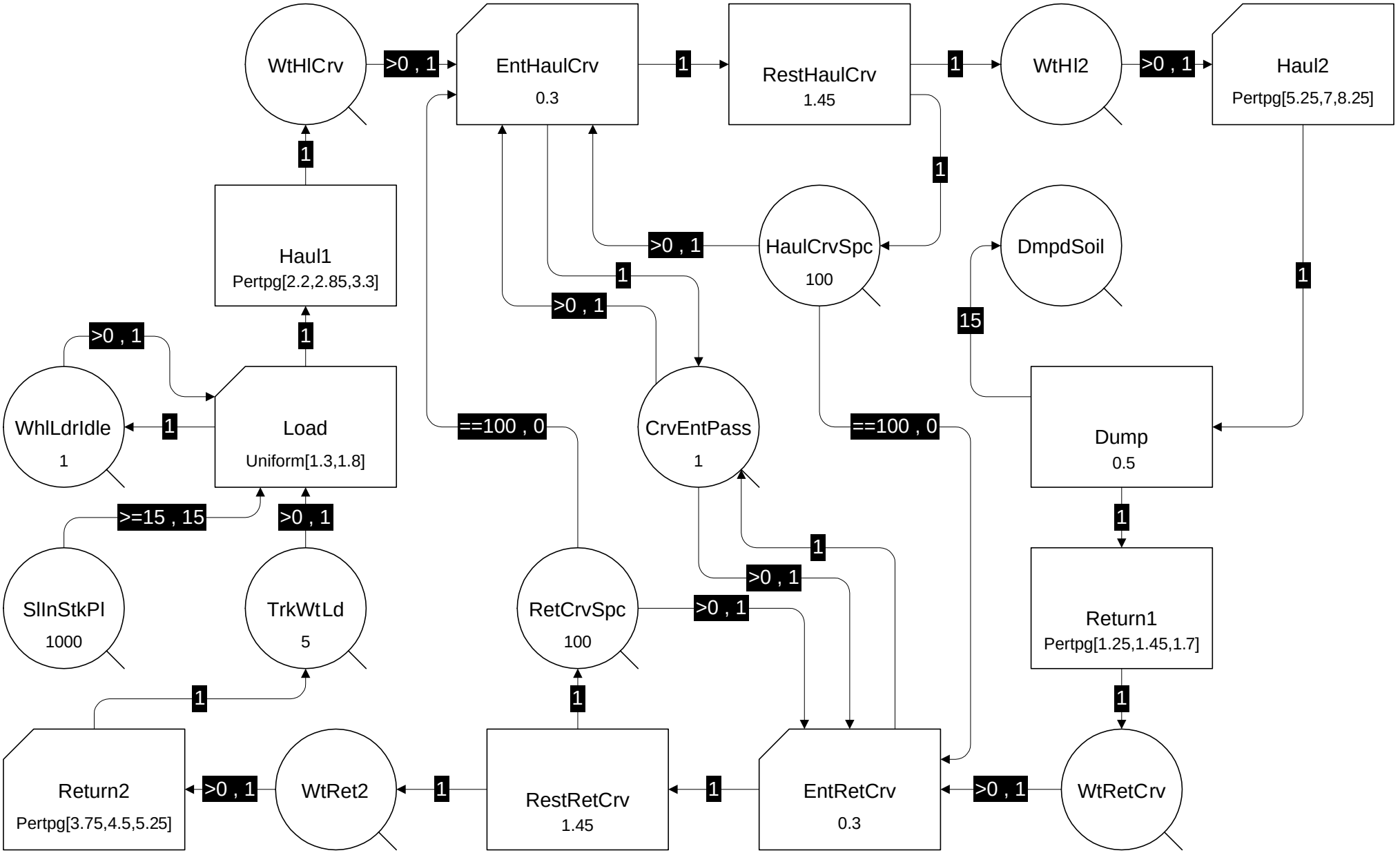
In this model, the HaulCrvSpc and RetCrvSpc queues have been added to provide dynamic information about the number of trucks that are traversing the curve while hauling or returning. Each of the queues is initialized with an arbitrarily large number (100) of resources. Whenever a truck enters the curve, a resource in the corresponding queue is consumed (making its content less than 100). Whenever a resource finishes traversing the curve, a resource is added to the corresponding queue (incrementing its content, perhaps to 100 if it is the last truck traversing the curve). In this model this information is not used, but it is necessary for the subsequent step.



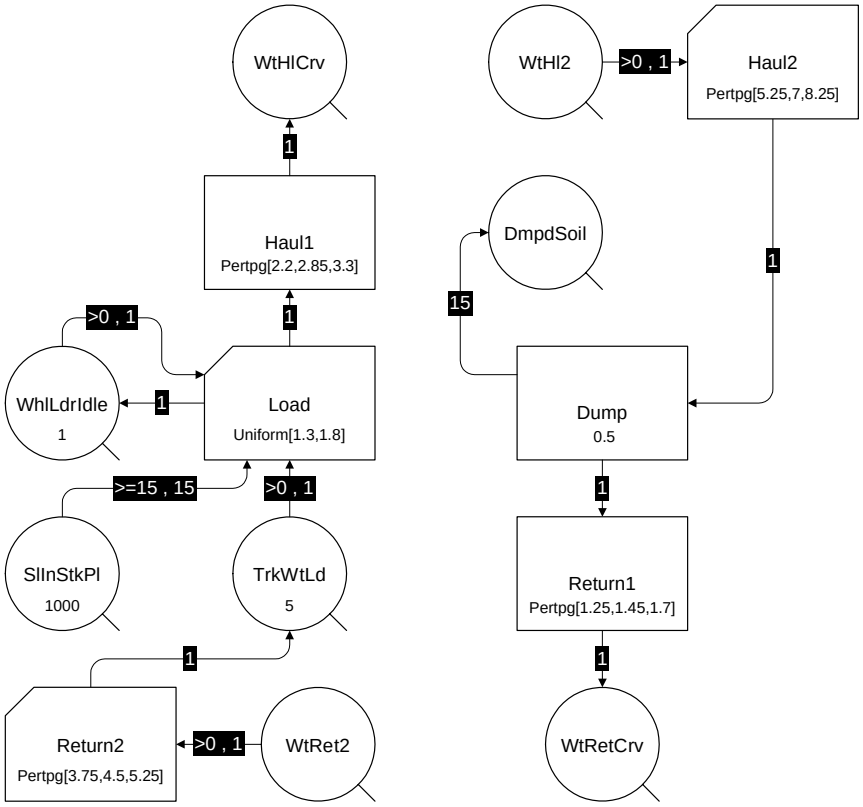
In this model, the nodes have been rearranged and the information provided by the contents of HaulCrvSpc and RetCrvSpc has been used to implement the single-lane unidirectional logic of the curve. This is done by the links with inscription "==100,0" that connect HaulCrvSpc to EntRetCrv and RetCrvSpc to EntHaulCrv. Essentially, a hauling truck is allowed to enter the curve only when there are no returning trucks in it (i.e., the RetCrvSpc queue contains 100 resources). The same applies to returning trucks.

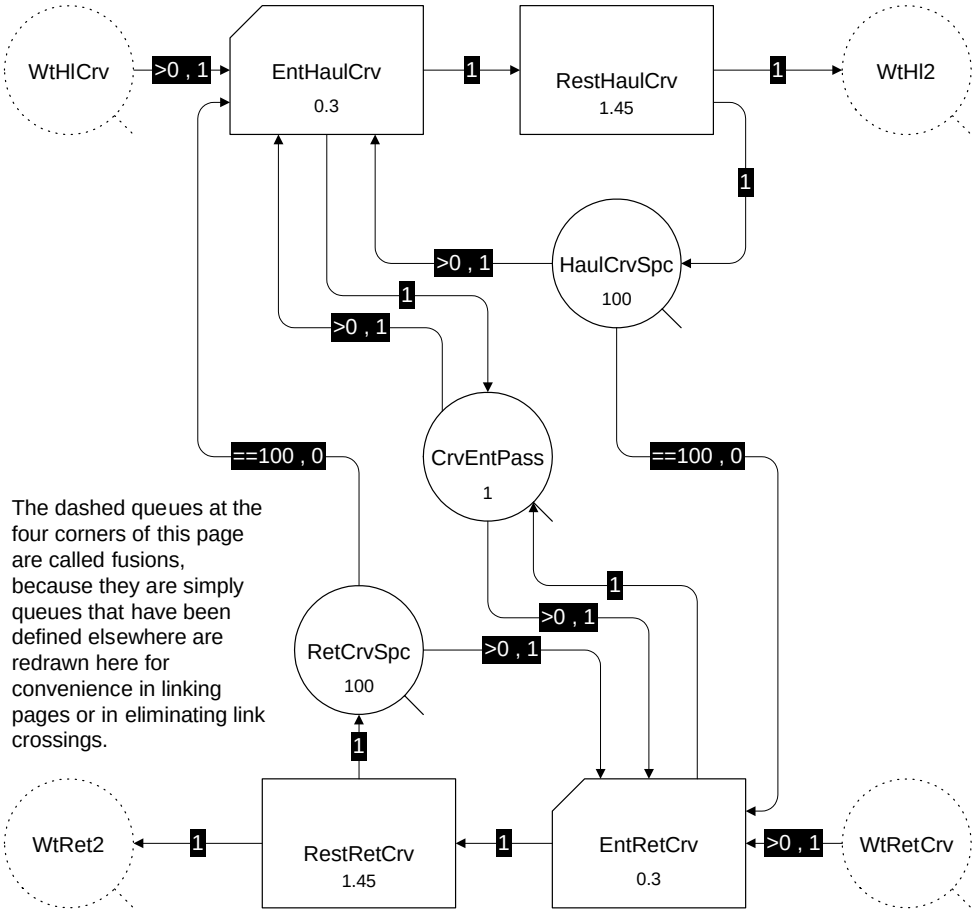


This model has turned the Haul2 and Return2 normal activities into combi activities by the introduction of WtHl2 and WtRet2. Conceptually, the model is the same as before. This has been done, however, to enable the partitioning of this model into two separate pages.



Here, the model has been broken up into two pages. One page (the 2nd page) with the logic of the curve and another page with the rest of the model (this page). This page is linked to the next one by the WtHlCrv, WtHl2,WtRetCrv and WtRet2 queues (fusions of these queues appear in the next page).

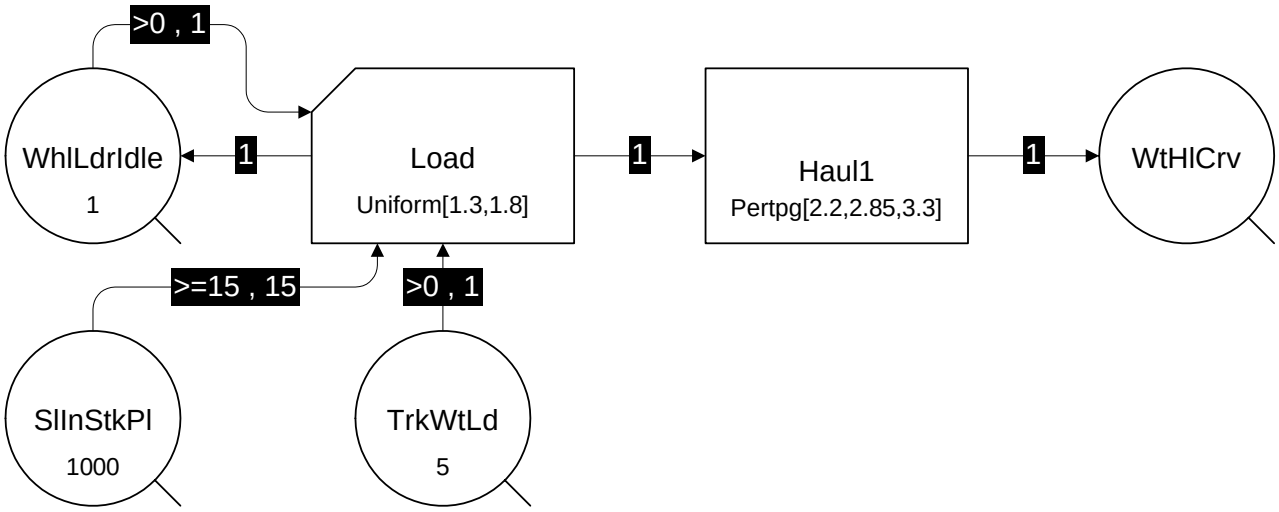




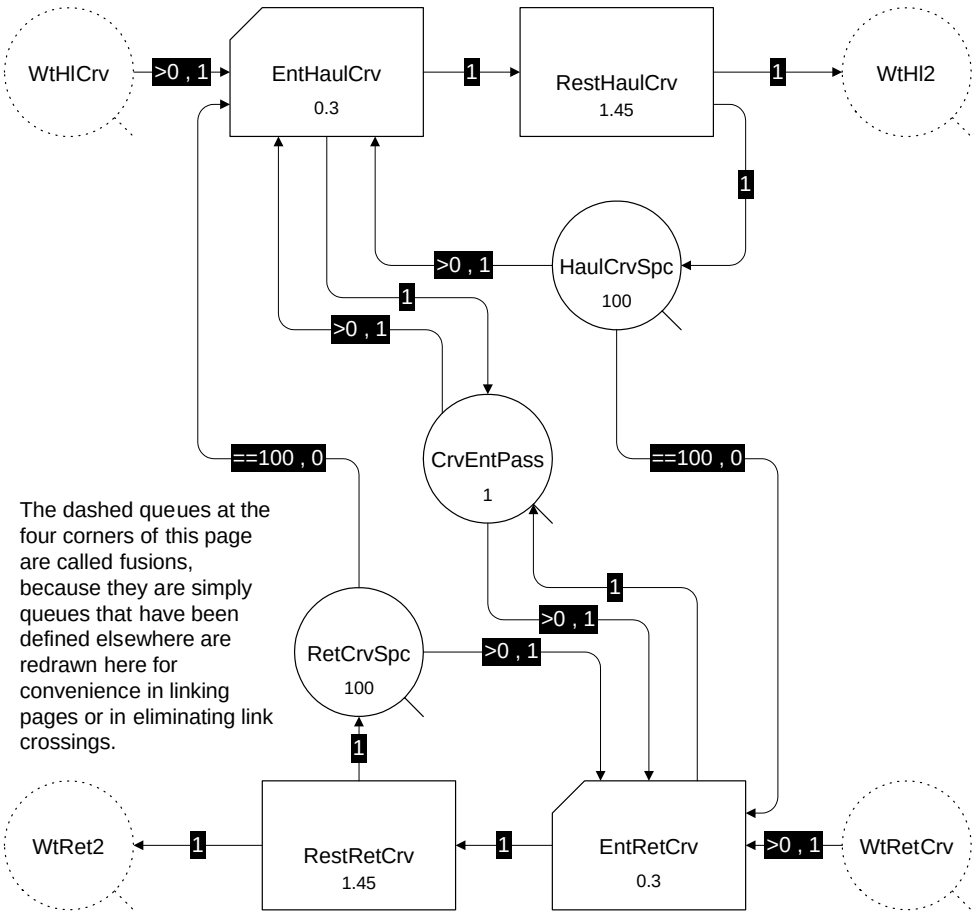
The dashed queues at the four corners of this page are called fusions, because they are simply queues that have been defined elsewhere are redrawn here for convenience in linking pages or in eliminating link crossings.

EZStrobe Tutorial Model 9 Page 1/3

This portion of the model, the first of three pages, contains the loading and hauling to the curve. It links to other parts of the model by fusions of the WtHlCrv and TrkWtLd queues.

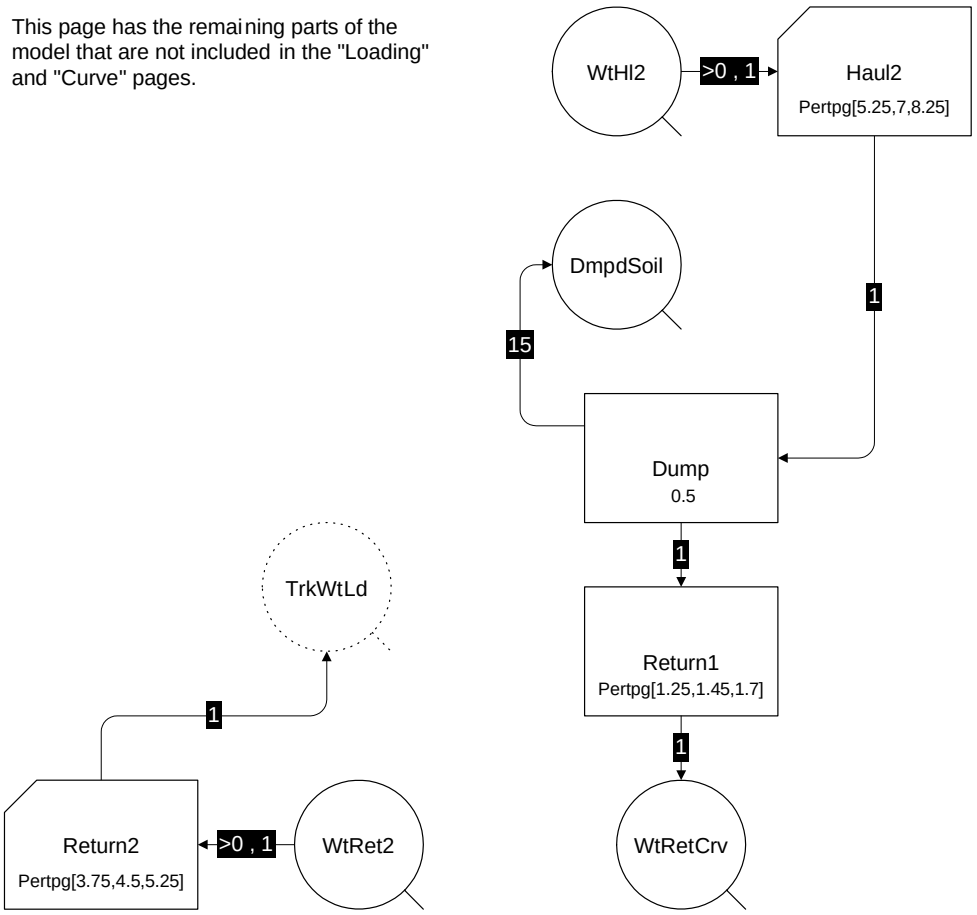


EZStrobe Tutorial Model 9 Page 2/3



The dashed queues at the four corners of this page are called fusions, because they are simply queues that have been defined elsewhere are redrawn here for convenience in linking pages or in eliminating link crossings.

This page has the remaining parts of the model that are not included in the "Loading" and "Curve" pages.

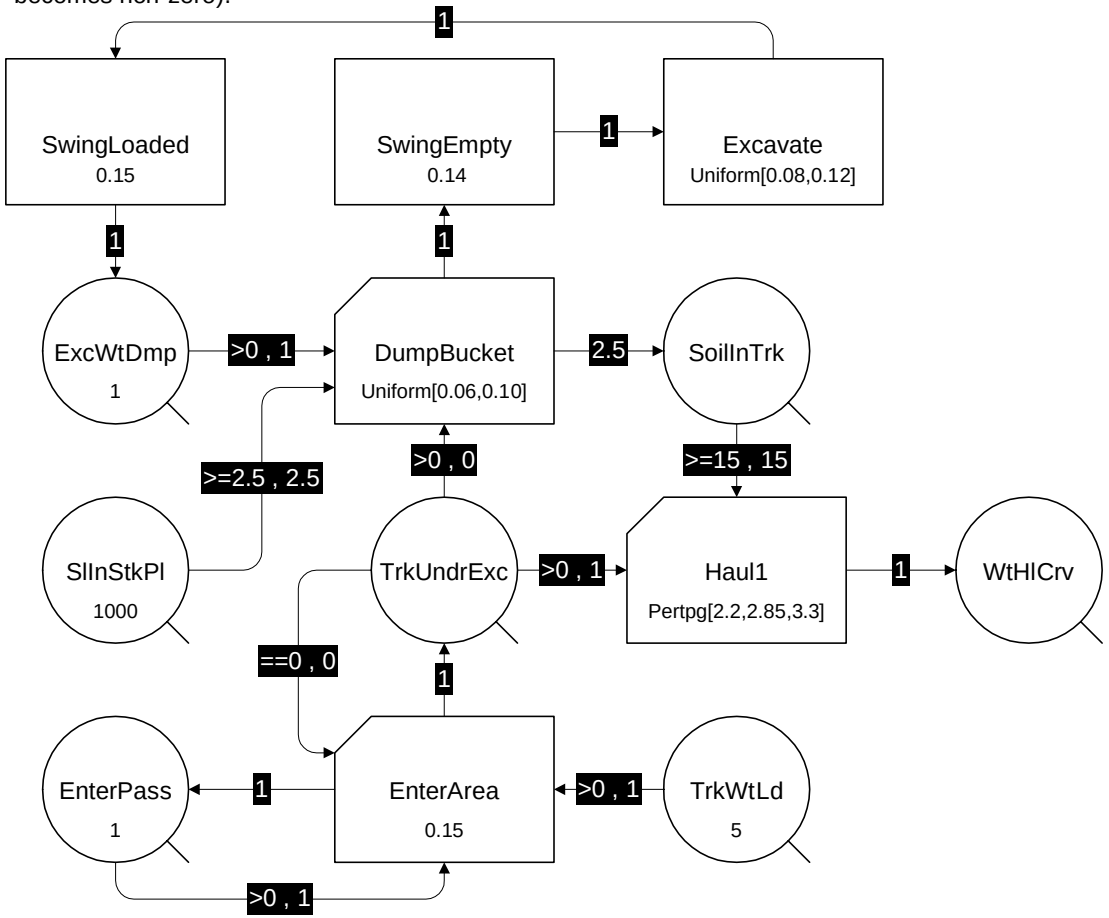


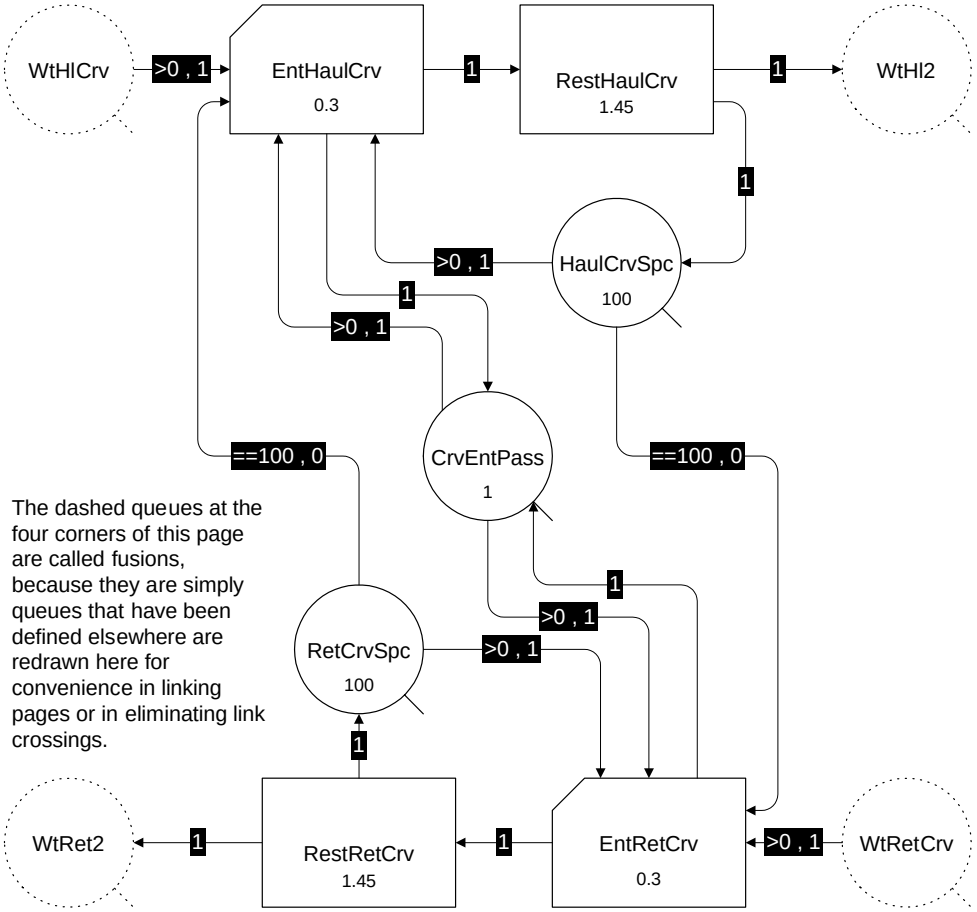
Here, the loading has been modeled in much more detail, representing the complete cycle of the excavator (switch from a wheel loader). The excavator's dump to the truck, empty swing, actual digging, and loaded swing are modeled explicitly. In this model the excavator will wait for a truck with its bucket ready to dump if it has time to do so after a previous truck has left. TrkUnderExc contains a truck while it is being loaded, hence the average waiting time there is the time required to load 15 m3 of soil into a truck with a 2.5 m3 excavator.

Notice the link from TrkUnderExc to EnterArea. Its function is to allow EnterArea to start only if there is no truck under the excavator. Under these circumstances it can obviously not remove a truck from TrkUnderExc regardless of the number placed after the comma in the link (because none are available). A zero is used to make this explicit.

Also notice the link from TrkUnderExc to DmpBucket. It indicates that there must be a truck in TrkUnderExc in order for the excavator to dump its bucket, but it also indicates that when DmpBucket starts it should not remove a truck from TrkUnderExc.

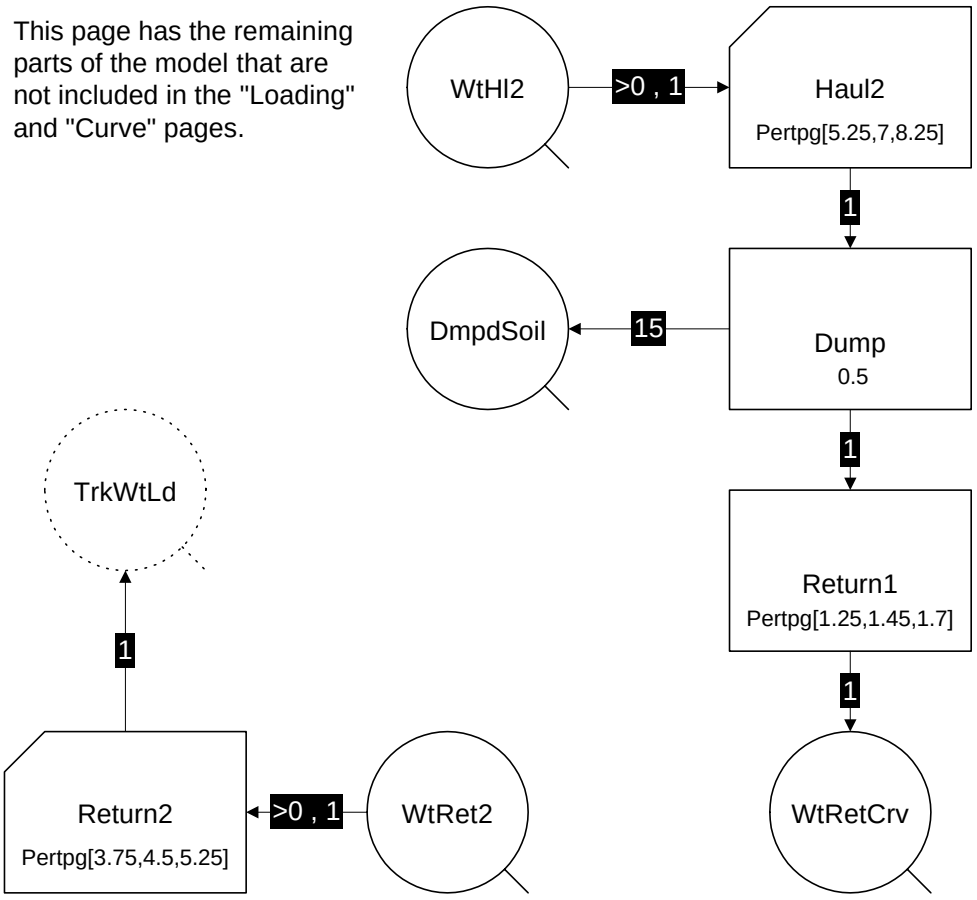
The EnterPass queue is necesary to prevent more a truck from entering the load area while another truck is entering (it is only after a truck completes its entry that the content of TrkUnderExc becomes non-zero).





The dashed queues at the four corners of this page are called fusions, because they are simply queues that have been defined elsewhere are redrawn here for convenience in linking pages or in eliminating link crossings.

This page has the remaining parts of the model that are not included in the "Loading" and "Curve" pages.



SoilAmt	Amount of soil in m3	10000
ExcCap	Excavator capacity in m3	2.5
TruckCap	Truck capacity in m3	15
nTrucks	Number of trucks	5
TrckCst	Truck cost ($/hr)	48
ExcCst	Excavator cost ($/hr)	65
OHCst	Overhead cost ($/hr)	75

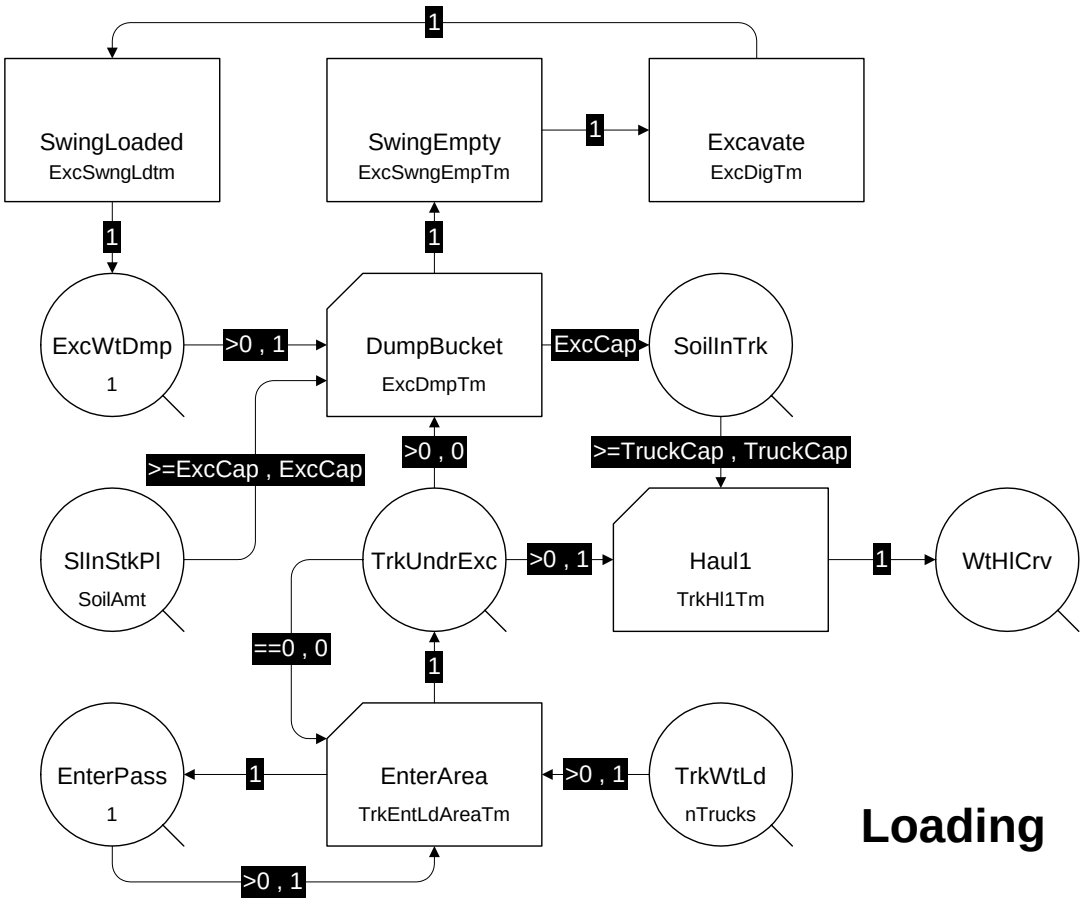
This model has been parameterized so that inputs are defined in one place and accessed symbolically throughout the model.

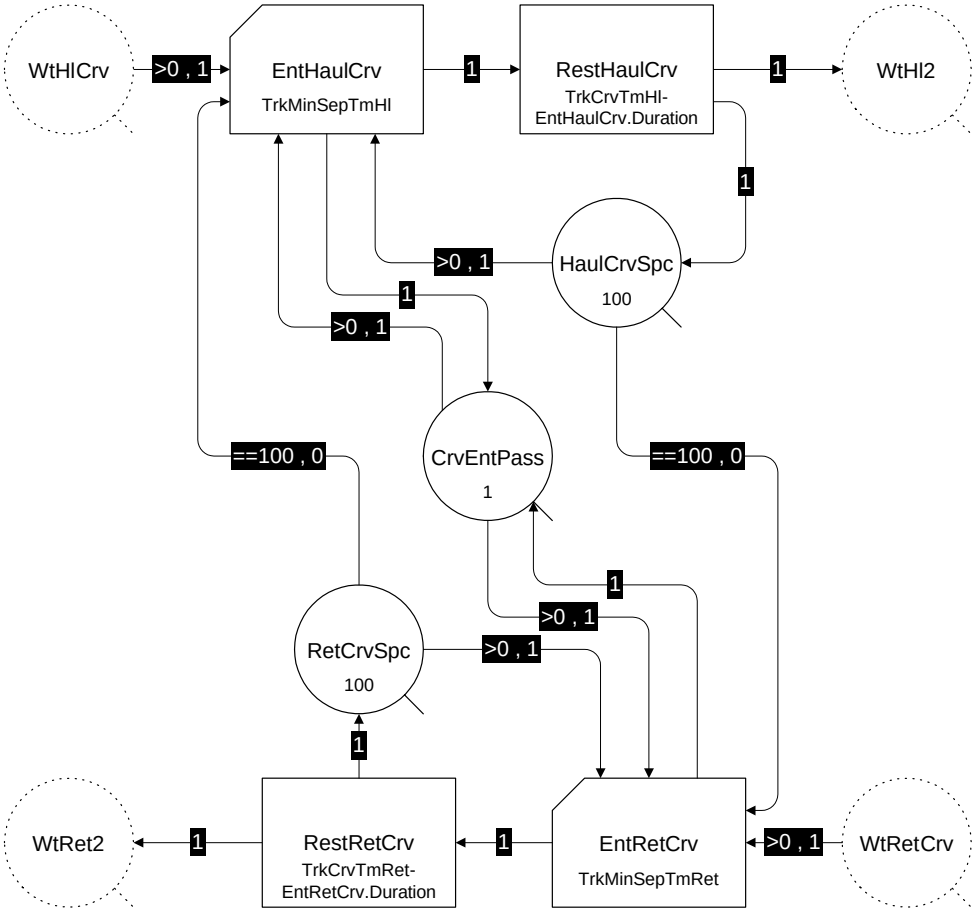
ExcDmpTm	Excavator dumping time	Uniform[0.06,0.10]
ExcSwngEmpTm	Excavator swing empty time	0.14
ExcDigTm	Excavator digging time	Uniform[0.08,0.12]
ExcSwngLdtm	Excavator swing loaded time	0.15

TrkEntLdAreaTm	Truck entry to load area time	0.15
TrkHl1Tm	Truck time for first haul	Pertpg[2.2,2.85,3.3]
TrkMinSepTmHl	Truck minimum separation time for hauling	0.3
TrkCrvTmHl	Truck time to traverse curve loaded	1.75
TrkHl2Tm	Truck time for second haul	Pertpg[5.25,7,8.25]
TrkDmpTm	Truck time to dump	0.5
TrkRet1Tm	Truck time for first return	Pertpg[1.25,1.45,1.7]
TrkMinSepTmRet	Truck minimum separation time returning	0.3
TrkCrvTmRet	Truck time to traverse curve returning	1.75
TrkRet2Tm	Truck time for second return	Pertpg[3.75,4.5,5.25]

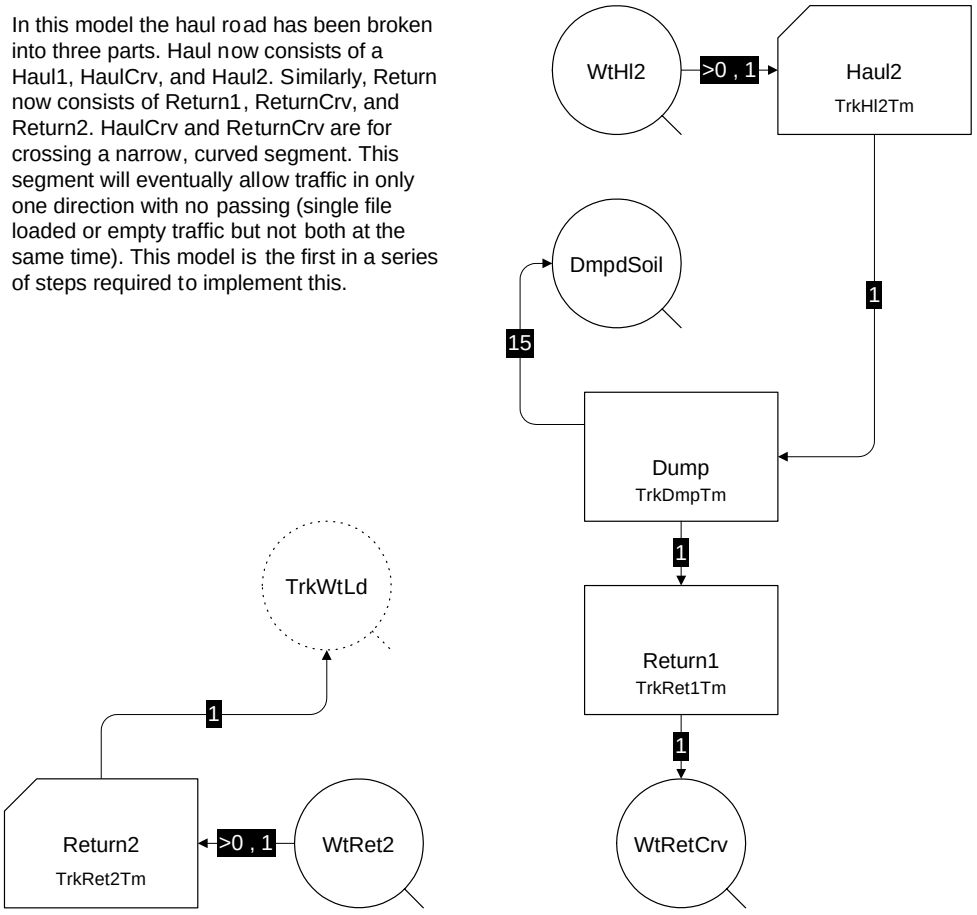
HourlyCst	Hourly cost	OHCst+ExcCst+nTrucks*TrckCst
Hrs	Hrs needed to move material	SimTime/60
ProdRate	Production rate (m3/hr)	DmpdSoil.CurCount/Hrs
UnitCst	Unit cost ($/m3)	HourlyCst/ProdRate

Important results have been extracted from the model and displayed explicitly.
This model is ready to be published in the web.





In this model the haul road has been broken into three parts. Haul now consists of a Haul1, HaulCrv, and Haul2. Similarly, Return now consists of Return1, ReturnCrv, and Return2. HaulCrv and ReturnCrv are for crossing a narrow, curved segment. This segment will eventually allow traffic in only one direction with no passing (single file loaded or empty traffic but not both at the same time). This model is the first in a series of steps required to implement this.



STROBOSCOPE Web Server: Earth-moving over curve interactive model

Amount of soil in m3	:	10000
Excavator capacity in m3	:	2.500
Truck capacity in m3	:	15
Number of trucks	:	5
Truck cost ($/hr)	:	48
Excavator cost ($/hr)	:	65
Overhead cost ($/hr)	:	75

Proceed to DirtOverCurve.str step 2

STROBOSCOPE Web Server: Earth-moving over curve interactive model results

```
Hrs needed to move material: 51.9423
Production rate (m3/hr)    : 192.329
Unit cost ($/m3)           : 1.97578
```

Proceed to DirtOverCurve.str step 3

EZStrobe User's Guide

This guide explains how to install and use the EZStrobe simulation software. For information on the modeling method itself please refer to the document contained in EZStrobe.pdf, which can be extracted from EZSTrobe.exe (see below).

Installation Requirements

EZStrobe requires Windows 95 or later, or Windows NT 4.0 or later. In addition, it requires STROBOSCOPE version 1.3.4 or later and Visio version 4.5 (with support for VBA), both of which must be installed prior to installing EZStrobe.

Installation Procedure

To install EZStrobe, run the provided EZStrobe.exe file to extract the files necessary for installation. To start the installation process, double-click Setup.vsd (one of the files extracted from EZStrobe.exe) or open it from within Visio and follow the instructions. 'Setup.vsd' ensures that EZStrobe is installed correctly. It will inform you if it detects any errors during installation. If you are running Windows NT make sure you are using an account with administrative privileges.

Visio will not recognize double-clicks if you are running Windows NT version 4.0 with Service Pack 3. To correct this you should install a hot-fix supplied by Microsoft. The hot-fix can be obtained from the following URL:

ftp://ftp.microsoft.com/bussys/winnt/winnt-public/fixes/usa/nt40/hotfixes-postSP3/getadmin-fix/admnfixi.exe

Running EZStrobe

To run EZStrobe you can either select EZStrobe when you launch Visio and it asks you to select a drawing template, or you can use the menu sequence File -> New -> EZStrobe from within Visio. Create your model by dragging and dropping modeling elements from the stencil. The Stencil, EZStrobe.vss, is a window with green background that lies along the left edge of the Visio workspace. Never open the EZStrobe Stencil directly (only the template, EZStrobe.vst).

When you drag Combis, Normals, and Queues from the stencil unto the drawing, a dialog box appears where you can supply the properties of the node. You can later edit these properties by double-clicking over the node. If dialog boxes do not appear upon drag or double-click, it is due to a faulty installation -- please try re-installing or contact the author if problems persist.

After you drag a Link, its ends must be connected to nodes. Not all parts of nodes can be targets of connections, only certain points along their perimeter. You can tell that a Link end is over a connectable portion of a node if while moving the Link's end, it shows a bold square around the grabbed end. A proper connection will be red when the link is selected.

Edit model options and run the model by selecting from the menu that pops up when you right-click over an empty area of the page.

Meaningful dynamic information exposed by EZStrobe via system-defined and system-maintained variables. In the following tables, the word *Activity* (in italics) should be replaced by the name of an actual Activity in the model; the word *Queue* (in italics) should be replaced by the name of an actual Queue in the model.

Global Variables (accessible all the time):

Variable Form	Description
SimTime	The current value of the simulation clock
Activity.AveDur	The average value of the duration of the instances of *Activity*
Activity.AveInter	The average inter-instantiation time between instances of *Activity*
Activity.CurInst	The current number of instances of *Activity*
Activity. InContext	Returns 1 if an instance of *Activity* is being created or terminated, and 0 otherwise
Activity.FirstStart	The time at which the first instance of *Activity* started
Activity.LastStart	The time at which the last instance of *Activity* started
Activity.MaxDur	The maximum value of the durations of the instances of *Activity*
Activity.MaxInter	The maximum of the inter-instantiation times between instances of *Activity*
Activity.MinDur	The minimum value of the durations of the instances of *Activity*
Activity.MinInter	The minimum of the inter-instantiation times between instances of *Activity*
Activity.SDDur	The standard deviation of the durations of the instances of *Activity*
Activity.SDInter	The standard deviation of the inter-instantiation times between instances of *Activity*
Activity.TotInst	The total number of instances of *Activity* that have been created
Queue.LastAmtReceived	The amount of resource that last entered *Queue*
Queue.AveCount	The time-weighted average of the content of *Queue*
Queue.AveWait	The average waiting time for resources that have entered *Queue*
Queue.CurCount	The current content of *Queue*
Queue.MaxCount	The maximum content experienced by *Queue*
Queue.MinCount	The minimum content experienced by *Queue*
Queue.SDCount	The time-weighted standard deviation of content experienced by *Queue*
Queue.TotCount	The total amount of resource that has entered *Queue*

Instance Variables (accessible only when an instance of Activity is starting or terminating):

Variable Form	Description
Activity.Duration	The duration of the instance of *Activity* that is starting or ending
Activity.Instance	The instance number of the instance of *Activity* that is starting or ending

Function Prototype	Function Description
Abs[val]	Absolute value of val
Acos[val]	ArcCosine of val
Asin[val]	ArcSine of val
Atan[val]	ArcTangent of val
AtanXdivY[x,y]	ArcTangent of x/y
AveCount[queue]	Average content of queue
AveDur[activity]	Average duration of activity
AveInter[activity]	Average time between succesive starts of activity
AveWait[queue]	Average wait at queue
Beta[a,b]	Sample from a unit Beta distribution
Confidence[SD,level,nSamples]	Half width of a confidence interval
Cos[val]	Cosine of val
Cosh[val]	Hyperbolic cosine of val
CurCount[queue]	Current content of queue
CurInst[activity]	Current number of instances of activity
Duration[activity]	Duration of the instance in context of activity
Erlang[order,mean]	Sample from an Erlang distribution
Exp[val]	Exponential function of val
Exponential[mean]	Sample from an Exponential distribution
FirstStart[activity]	Time at which the first instance of activity started
Gamma[a,b]	Sample from a Gamma distribution
InContext[Activity]	Indicates whether there is an instance in context of the activity
Instance[activity]	Instance number of the instance in context of activity
Int[val]	Integer part of val
LastAmtReceived[Queue]	Returns the last amount received by GenQueue
LastRnd[]	Retrieve the last number returned by Rnd[]
LastStart[activity]	Time at which the last instance of activity started
Ln[val]	Natural logarithm of val
Log[val]	Base 10 logarithm of val
Max[val1,val2]	Maximum of val1 and val2

Function Prototype	Function Description
MaxCount[queue]	Maximum content of queue
MaxDur[activity]	Maximum duration of activity
MaxInter[activity]	Maximum time between succesive starts of activity
Min[val1,val2]	Minimum of val1 and val2
MinCount[queue]	Minimum content of queue
MinDur[activity]	Minimum duration of activity
MinInter[activity]	Minimum time between succesive starts of activity
Mod[val,div]	Remainder of val / div
Normal[mean,sd]	Sample from a Normal distribution
NormalInv[mean,stdev,cumulative]	Inverse of the Normal distribution
Pert[p0,Mode,p100]	Sample from a Beta distribution
Pertpg[p5,Mode,p95]	Sample from a Beta distribution
Rnd[]	Sample a number uniformly distributed between 0 and 1
Round[expression,decimals]	Round expression to the specified number of decimal places (which can be negative)
ScaledBeta[low,high,a,b]	Sample from a scaled Beta distribution
SDCount[queue]	Standard deviation of content of queue
SDDur[activity]	Standard deviation of duration of activity
SDInter[activity]	Standard deviation of time between succesive starts of activity
Sin[val]	Sine of val
Sinh[val]	Hyperbolic sine of val
Sqrt[val]	Square root of val
StdNormalInv[Cumulative]	Inverse of the Standard Normal distribution
Tan[val]	Tangent of val
Tanh[val]	Hyperbolic tangent of val
tInv[alpha,degFreedom]	Inverse of the t Distribution
TotCount[queue]	Total content of queue
TotInst[activity]	Total number of instances of activity
Triangular[low,mode,high]	Sample from a triangular distribution
Uniform[low,high]	Sample from a Uniform distribution