CENTER FOR CONSTRUCTION ENGINEERING AND MANAGEMENT

UM-CYCLONE
Discrete Event Simulation System
Reference Manual

Photios G. Ioannou
Assistant Professor

UMCEE Report No. 89-11
Civil and Environmental Engineering Department
UNIVERSITY OF MICHIGAN
Ann Arbor, Michigan
March 19, 1990

UM-CYCLONE
Discrete Event Simulation System
Reference Manual

(C) Photios G. Ioannou
Assistant Professor of Civil Engineering

March 19, 1990

UMCE 89-11
Department of Civil and Environmental Engineering
University of Michigan
Ann Arbor, MI 48109-2125
(313) 764-3369

Introduction

UM-CYCLONE is a discrete event simulation system that is both powerful and easy to learn and use. It can be used to model a variety of systems that can be represented as networks and which either exhibit queuing behavior due to limited resources, or are characterized by uncertainty either in activity durations or in the sequence of operations. Examples include construction projects, manufacturing systems, computer systems, service systems, etc.

UM-CYCLONE includes both the methodology for representing real systems as equivalent network models, and the computer programs that perform the simulation and produce the statistical results.

Before being input to the computer, UM-CYCLONE networks are typically developed by first drawing them on paper. This process is done by hand. Once a network is drawn, it can be submitted to the computer for performing the simulation. In most cases, the simulation results will indicate modeling errors that must be corrected in the network representation. The trial and error design process is repeated until the system behaves in the intended manner.

This manual describes the various modeling elements that are used to construct networks, explains how these elements interact during the simulation process, and describes the meaning of the various statistics produced by the computer program. Its objective is to provide all the information necessary for constructing accurate models of reality, and interpreting their results. These are the most difficult steps in learning how to use the system.

Additional information on how to install and use the UM-CYCLONE programs on IBM PC, PS/2, or compatible computers is provided in the UM-CYCLONE User's Guide. Example models are provided on the same disk as the UM-CYCLONE programs.

Elements of a UM-CYCLONE Simulation Model.

A UM-CYCLONE simulation model is a precedence network that consists primarily of two kinds of elements: Nodes and Links (Arrows). In this respect, a UM-CYCLONE model is similar to an *Activity on Node* (or *Precedence*) CPM or PERT network.

This similarity, however, does not extend very far. CPM or PERT scheduling networks have only one kind of node, the *Activity*, and are characterized by one simple precedence rule:

"An activity cannot start until after its direct predecessor activities have finished." As a result, CPM and PERT networks cannot contain loops (they are *acyclic*).

UM-CYCLONE networks, on the other hand, can have up to three different kinds of nodes: *Activities*, *Queues*, and *Consolidation Nodes*. They also have various rules of precedence, depending on the type of nodes being connected, which allow networks to contain loops (to be *cyclic*). The name CYCLONE stands for CYCLic NEtworks.

Typically, a UM-CYCLONE simulation model is constructed by drawing the network on paper using different node symbols to represent Queues, Activities and Consolidation Nodes. Each node is given a unique number that serves as its ID and differentiates it from other nodes in the network. Precedence is indicated by drawing arrows (*links*) from predecessor nodes to successor nodes. There can be any number of links entering or leaving each node in the network.

After the network is complete, it is input to the computer for performing the simulation. The simulation output and statistics are examined for possible modeling errors and the network is revised accordingly. Once it is ensured that the developed network is an accurate model of the real system, the simulation output is used to measure and optimize its performance. Thus, the development of a model follows the traditional trial-and-error design process.

The various kinds of UM-CYCLONE nodes and the corresponding precedence rules are explained below.

Queues

A Queue in a UM-CYCLONE model is shown graphically as a circle with a small diagonal line crossing its lower right side to make it look like the letter Q. Each Queue is identified by its own unique number (*QueNo*), an integer greater than zero, which is usually written inside and towards the top of the Queue node symbol. A Queue may also be given a name or description which is typically written in the center of the node symbol.

From a modeling standpoint, a Queue is simply a storage area for a particular resource that is needed for performing the directly following Activities. Examples of typical resources are labor, equipment, materials, permits, etc. In addition to *real* (or *physical*) resources, queues can

also be used to store any kind of *artificial* (or *virtual*) resource that may be needed for constructing an accurate model of the operation of a system. Examples are signals, flags, etc.

A Queue can be directly preceded by any combination of Activities or Consolidation Nodes. In other words, a Queue may be preceded by any type of node other than a Queue. This requirement stems from the fact that a Queue can only precede an Activity. An Activity preceded by one or more Queues is called a Combi. A Combi Activity can be preceded by any combination of Queues but not by any other type of node. Hence, Queues and Combi Activities are closely related in that Queues can only precede Combi Activities and Combi Activities can only be preceded by Queues.

UM-CYCLONE considers all the resource units stored in the same Queue, at the same point in time, as identical and interchangeable. They are treated as a uniform resource pool with no consideration to individual arrival times or other resource characteristics or attributes. Consequently, *different types of resources should be stored in different Queues* for the system to be able to tell them apart.

UM-CYCLONE represents each Queue as a separate integer variable. The value of this variable represents the number of resource units currently in the Queue. This simplicity is a direct consequence of the fact that resource units in the same Queue are considered identical.

During a UM-CYCLONE simulation, the contents of a Queue change every time a preceding Activity or Consolidation Node finishes, or a succeeding Combi Activity starts. The precedence rules for changing the number of units stored inside a Queue can be summarized as follows:

- When a directly preceding node (Activity or Consolidation Node) finishes, the contents of the Queue are increased by one unit: i.e., a unit enters the Queue.

- When a Combi Activity starts, the contents of each directly preceding Queue are decreased by one unit: i.e., a unit exits the Queue.

Queues cannot have negative contents. Hence, a direct corollary of the second rule is that *for a Combi Activity to start, its directly preceding Queues must not be empty*. This rule is restated and explained below.

From a modeling standpoint, the precedence rules stated above imply that the finish of a predecessor produces *an arrival of one resource unit* to a Queue, and the start of a successor produces *a departure of one resource unit* from the Queue. Thus, there is a one to one correspondence in the number of resource units that have entered a Queue and the number of resource units that can exit the Queue.

There are many modeling situations, however, that require that the number of resource units that can exit a Queue be different from the number of units that have entered the Queue. UM-CYCLONE accomplishes this by "splitting" each incoming resource unit into GEN number of units, where GEN is an integer greater than or equal to 1. This provides Queues with the capability of resource "generation" (GEN is an abbreviation for GENerate).

The first precedence rule can now be stated in a more general form:

- When a directly preceding node (Activity or Consolidation Node) finishes, the contents of the Queue are increased by GEN number of units.

When constructing a simulation network on paper, the Generation function of a Queue is indicated by writing GEN=X (*e.g.,* GEN=3) directly below the Queue node symbol. If GEN=1 this caption is implied and is usually omitted.

A simple example requiring resource generation is the case of a truck delivering precast beams to a construction site. The truck delivers three beams at a time, and arrives at the construction site every time the Activity "*Truck delivers beams*" finishes. This Activity is succeeded by the Queue "*Beams available*". Every time the Activity "*Truck delivers beams*" finishes, the contents of the "*Beams available*" Queue must be increased by 3 units. As a result, this Queue must have GEN=3, because the arrival of one truck increases the contents of the Queue by three beams.

A UM-CYCLONE network must include at least one Combi Activity. Since a Combi Activity can be preceded only by one or more Queues, it follows that a network should also include at least one Queue. Furthermore, at least one Combi Activity in the network must be able to start at the beginning of the simulation. Otherwise, none of the model's operations will be able to start.

In order for a Combi Activity to be able to start at the beginning of the simulation, it is necessary to initialize the directly preceding Queue(s) so that their contents are greater than zero. The initialization of Queues at the start of a simulation is an important step in defining the operation of a model. After the network is drawn, the initial contents of the Queues are specified by an integer (greater than or equal to zero) which is usually written inside and towards the bottom of the Queue node symbol. If the initial Queue contents are missing they are assumed to be zero.

An interesting situation occurs when specifying the initial number of resource units in a Queue associated with a GEN function. In this case, the initialization of the Queue is done in terms of *incoming* units, and not in terms of *departing* or *generated* units.

As an example, consider the case described above of a truck delivering three beams at a time to a construction site. If we wanted to start the simulation with some precast beams already available for work, then we must initialize the contents of the Queue "*Beams available*" to some nonzero value. Let's assume that we specify the initial value to be X. As soon as simulation starts, UM-CYCLONE applies the Queue's GEN function and multiplies X by 3. As a result, the initial number of resource units in the Queue equals 3X. Thus, the number X should not be interpreted as "*beams*", but rather as "*truckloads of beams*". Specifying X=1, for example, is equivalent to storing 3 beams. Notice that, because of this feature, it is impossible to initialize the Queue to a number of resource units that is not an integer multiple of 3 (the GEN value).

It must be emphasized that UM-CYCLONE describes the contents of Queues by simple integer numbers that are not associated with *units of measure*. UM-CYCLONE does not know how these numbers are interpreted by the user, whether they represent CY of soil or the number of available carpenters. It is the user's responsibility to select the appropriate units of measure and to make sure that the contents of Queues are increased or decreased by the correct amounts. Furthermore, the units of measure for one Queue should be compatible with those of other Queues and with those required by the following Combi Activities. Compatibility of units of measure is the primary reason why UM-CYCLONE defines a GEN function for Queues. It is also the main reason for the existence of Consolidation Nodes.

Activities

There are two kinds of Activities in UM-CYCLONE: *Combi Activities* and *Normal Activities*. Each activity, irrespective of its type, is represented graphically by a rectangle and is identified by its own unique number (*ActNo*), an integer greater than zero, written inside and towards the top of the node symbol. A small diagonal line forming a small triangle at the upper left corner of the rectangle indicates that the Activity is a *Combi*. The absence of this line indicates a *Normal* Activity. Activities may also be given a name or a description that is typically written inside the node symbol.

UM-CYCLONE Activities are similar to the activities for CPM or PERT networks. They represent work or tasks that need to be performed. The performance of Activities typically takes a certain amount of time. In addition, some Activities cannot be performed unless certain resources (such as material, labor, equipment, space, etc.) are available. These resource requirements are usually modeled by preceding the Activities with the appropriate Queue(s).

Activities that require resources are usually modeled as Combi Activities. Formally, a Combi Activity is an Activity that can only be directly preceded by any number of Queues, but not by any other kind of node. Similarly, a Normal Activity is an Activity that can be preceded by any combination of node types that does not include a Queue.

The distinction between Combi and Normal Activities is necessary because each Activity type follows a different precedence rule:

- A Combi Activity can start whenever all the directly preceding Queues are not empty. Equivalently, a Combi Activity cannot start if one (or more) of the directly preceding Queues is empty.

- A Normal Activity starts whenever any preceding node (Combi, Normal Activity, or Consolidation Node) finishes.

It is important to note that the first rule is a logical "*AND*" statement, whereas the second is a logical "*OR*". In order for a Combi Activity to start (to be True), the current contents of *all* preceding Queues must be nonzero (*all* Queues must be True). In contrast, a Normal Activity can start (can be True), whenever any one of its predecessor nodes is finished (is True). Thus,

Combi Activities follow a precedence rule that is logically similar to that of CPM networks, whereas Normal Activities follow a much more permissive rule.

Even though the two types of Activities may appear to be very different, it is easy to see that a Normal Activity is essentially a special case of a Combi. A Normal Activity is equivalent to a Combi Activity preceded only by one Queue, that is itself preceded by all the Normal Activity's predecessors. This Queue is empty at the start of simulation. Every time one of its predecessors finishes, the contents of the Queue are increased from zero to one. This allows the Combi Activity to start immediately, which in turn resets the contents of the Queue back to zero. Thus, the Combi Activity starts every time one of the Queue's predecessors finishes, just like a Normal Activity.

Both types of Activities can be succeeded by any combination of nodes that does not include a Combi. Obviously, if an Activity is followed by another Activity then the latter must be a Normal Activity because a Combi can only be preceded by Queues.

Every Activity, irrespective of its type, has a duration that represents the time span from its start to its completion. The duration of an activity can either be constant (deterministic) as in CPM, or probabilistic (random) as in PERT.

An Activity with a constant duration takes up the same amount of time every time it is performed. Its duration is specified as a fixed nonnegative real number. An Activity with a constant duration equal to zero is called a *dummy*.

In general, however, the duration of an Activity varies every time the Activity is performed. In this case, the duration of the Activity is considered a random variable described by a probability distribution. The type of probability distribution and its parameters must be specified as part of the Activity description. During simulation, the Activity duration is determined by random sampling from its associated distribution. Each Activity may have its own distribution with different parameters. An Activity's duration, or its distribution, is usually written at the bottom of its node symbol (*e.g.,* 5, N(5,3), b(10,12,15), etc.).

UM-CYCLONE does not associate Activity durations with any specific measure of time, such as minutes, hours, days, etc. It is up to the user to select an appropriate unit of time for specifying Activity durations and interpreting the simulation results. Furthermore, only one type

of time unit should be used in a given UM-CYCLONE model. The same time units must be used for specifying the durations of all Activities (as well as the *Simulation Time Limit*).

This is necessary because UM-CYCLONE keeps track of simulated time by increasing the value of a single real variable (TNOW), also known as *the simulation clock*. Successive values of the simulation clock are determined by adding a sampled Activity duration to the current value of the clock. In order for the resulting sum to have any meaning, both the clock and the Activity duration must be expressed in the same time units.

The selection of the *appropriate* time units for a given model is only a matter of convenience. By scaling all the time-related data by the same factor it is possible to express simulated time in minutes, hours, days, 8-hour shifts, or any other suitable time unit.

Consolidation Nodes

A Consolidation Node is shown graphically as a circle with the caption CON=X (*e.g.*, CON=5) written below it. Each Consolidation Node is identified by its own unique number (*ConNo*), an integer greater than zero, written inside and towards the top of the circle.

A Consolidation Node is a *cross* between a dummy Normal Activity and the logical opposite of a Queue with a GEN function. It resembles a dummy Normal Activity in that it has a *start*, a *finish*, no duration of its own, and the same precedence rules. It is the logical opposite of a Queue in that a Queue generates GEN number of resources for each incoming resource unit, whereas a Consolidation Node merges CON number of *starts* to produce one *finish*. CON is an integer greater than or equal to 1. As explained below, a Consolidation Node with CON=1 is superfluous and identical in function to a dummy Normal Activity.

A Consolidation Node can be preceded by any combination of node types other than a Queue, and can be succeeded by any combination of node types other than a Combi (just like a Normal Activity). Every time a direct predecessor to a Consolidation Node finishes, it sends one *start signal* to the Consolidation Node. The Consolidation Node has an internal counter that counts the number of start signals received. When this counter becomes equal to CON, the Consolidation Node *finishes* and sends a *start signal* to all its direct successors in the simulation network. At the same time, it resets its internal *start counter* to zero and gets ready to repeat the

cycle. Thus, a Consolidation Node finishes once, and outputs one *start signal*, for every CON *start signals* it receives. In other words, it *consolidates* CON *start signals* into one.

When a Consolidation Node finishes, the action taken by its direct successors depends on their type. If a successor node is a Normal Activity, then it starts immediately. If it is a Queue, then its contents are increased by GEN number of units. If it is another Consolidation Node, then it simply receives one incoming *start signal* and increments its own *start counter* by one.

The precedence rules for a Consolidation Node can be summarized as follows:

- A Consolidation Node receives a start signal whenever each of its direct predecessor nodes finishes. When a Consolidation Node receives a start signal it increases its internal counter of accumulated start signals by one.

- A Consolidation Node finishes whenever it accumulates CON number of start signals. When a Consolidation Node finishes, it resets its internal counter of accumulated start signals to zero.

A Consolidation Node does not have a predefined duration of its own. Instead, the amount of time between successive *finish* events is determined by how quickly the counter of *start signals* received reaches the value CON, which in turn depends on how often the *predecessor nodes finish*. Obviously, a Consolidation Node with CON=1 is identical in function to a dummy Normal Activity, because it finishes the same instance it starts.

The close relationship between the function of a Consolidation Node and the GEN function of a Queue can be illustrated by considering the example presented earlier about a truck delivering beams from a precasting shop to a construction site. A beam is loaded onto the truck every time the Activity "*Crane loads one beam onto truck*" finishes. This Activity is succeeded by the Consolidation Node "*Truck loaded with three beams*" (CON=3), which in turn is succeeded by the Activity "*Truck delivers beams*". This Activity is then succeeded by the Queue "*Beams available*" (GEN=3). The Consolidation Node "*Truck loaded with three beams*" counts how many beams have been loaded onto the truck and ensures that the Activity "*Truck delivers beams*" does not start until after the Activity "*Crane loads one beam onto truck*" has been performed three times.

Limits on the Number of Precedence Links

Even though, in general, there is no limit to the number of links in a network, the current version of UM-CYCLONE allows each Activity and Consolidation Node to have up to six direct successors and six direct predecessors. This constraint does not apply to Queues, which can be directly preceded or succeeded by any number of nodes.

The current limit of six incoming or outgoing links is rarely a problem in developing a simulation model. Furthermore, it can be easily overcome by introducing one or more dummy Activities immediately preceding or following the node that requires more than six links. The addition of a dummy can split each available link into six. This creates either an incoming or outgoing tree of links with dummy Activities as its nodes. The resulting tree can have as many terminal links as required.

Probabilistic Forks

The above precedence rules for the links in a UM-CYCLONE simulation network are *deterministic* in nature. When an Activity (or Consolidation Node) finishes, *all* its direct successors can start. If the following node is a Queue then its contents are increased by GEN units. If the following node is a Normal Activity then it starts immediately. If the following node is a Consolidation Node, then its *start counter* is increased by one. Thus, the finish of an Activity (or Consolidation Node) activates *all* outgoing links and thus *all* its direct successors.

There are cases, however, when it is desirable to activate *only one* of these links (and the associated successor node) based on a probabilistic selection process. This behavior is similar in function to a chance node in a decision tree, or a GERT network.

In order to accommodate the probabilistic selection of succeeding paths, UM-CYCLONE allows the links from a given Activity (or Consolidation Node) to its direct successors to act like a probabilistic fork with mutually exclusive branches. The probability of selecting a particular branch is written above the corresponding link and must be specified as part of the model description. Obviously, the sum of the probabilities for all the branches of a particular fork should equal 1. Furthermore, the set of links leaving a particular Activity (or Consolidation Node) must either be *completely deterministic* (no probabilities on the branches) or *completely probabilistic* (a probability must be specified for each branch).

When the Activity (or Consolidation Node) preceding a probabilistic fork finishes, UM-CYCLONE does not immediately activate all its successors. Instead, it performs Monte Carlo sampling, using the probabilities associated with the branches in the fork, to select and activate only one of the succeeding nodes. This selection procedure is performed every time the Activity (or Consolidation Node) preceding a probabilistic fork finishes and thus the activated successor is not always the same.

As an example requiring the use of a probabilistic fork, consider the situation where trucks arrive for inspection at a maintenance facility. Assume that the inspection of any given truck may lead to one of three outcomes: (a) *Routine maintenance* with probability 0.6; (b) *Minor repair* with probability 0.3; and (c) *Major repair* with probability 0.1. Each one of these alternatives may require different *down times*, different crews, and have different costs.

For simplicity, assume that this situation is modeled by five activities: "*Truck arrives*", "*Routine maintenance*", "*Minor repair*", "*Major repair*", and "*Truck departs*". The three Activities "*Routine maintenance*", "*Minor repair*", and "*Major repair*" are direct successors to "*Truck arrives*" and direct predecessors to "*Truck departs*". The arrival of a truck, however, should only activate one of these three activities. Furthermore, the selection of the successor should be in accordance with the probabilities given above.

To model this situation, the successor links from the Activity "*Truck arrives*", must be defined as a probabilistic fork with three branches. Each one of these branches leads to one of the Activities "*Routine maintenance*", "*Minor repair*", and "*Major repair*", and is assigned the associated probability. Obviously, each of these three Activities has a deterministic link to its successor "*Truck departs*". When the Activity "*Truck arrives*" finishes, UM-CYCLONE selects one of the branches based on Monte Carlo sampling, and activates the corresponding successor Activity. At the end of that Activity, irrespective of which one it is, UM-CYCLONE activates the "*Truck departs*" Activity.

Probabilistic forks provide UM-CYCLONE with very powerful and useful modeling capabilities. They can be used to model activity durations with discrete probability distributions, decision trees, Markov chains, GERT networks, queuing systems with random selection of servers, etc.

How a UM-CYCLONE Simulation Works

The above description of the various modeling elements provides some of the necessary background for understanding the structure of UM-CYCLONE networks. This section provides additional insight by describing how a UM-CYCLONE simulation works after the model is transferred from a paper drawing to the computer.

The easiest way to describe the operation of a UM-CYCLONE simulation is to examine its logical structure as represented by the following simplified pseudocode:

```
START; {Simulation}
TNOW=0;
ITER=0;
While ( ( TNOW <= SimTimeLimit ) and ( ITER <= ProdLimit ) )
  begin {While Loop}
    If a Combi Activity can start then
      begin {Generate Phase}
        Move Units to Combi Activity;
        Generate Combi Activity Duration, DUR;
        Calculate its End Event Time, EET=TNOW+DUR;
        Record EET in the Future Event List;
      end {Generate Phase}
    else
      begin {Advance Phase}
        Find the next EET in the Future Event List;
        Advance the simulation clock: TNOW=EET;
        Release resource units to the following nodes;
        If Flagged Activity finishes at TNOW then ITER=ITER+1;
      end; {Advance Phase}
  end; {While Loop}
Write Output File; {Simulation Statistics}
STOP. {Simulation}
```

The main processing for a UM-CYCLONE simulation occurs within a *While loop* controlled by two variables: TNOW and ITER. TNOW represents the current value of the simulation clock, and ITER is a counter that represents the number of times a particular Activity (the *Flagged Activity*) has finished. At the start of the simulation both of these variables are initialized to zero.

The statements within the *While loop* are repeated until either TNOW exceeds a specified time limit (*SimTimeLimit*), or ITER equals the specified production limit for the Flagged Activity (*ProdLimit*). By choosing appropriate values for these limits, it is possible to control the simulation duration either on the basis of simulated time, or system production.

Each UM-CYCLONE model must have exactly one Flagged Activity, even if the simulation is not intended to be controlled on the basis of production. The Flagged Activity can be any Activity in the network. It is shown graphically by drawing a triangular flag at the upper right corner of its rectangular node symbol, with the production limit written next to the flag.

While neither of the control limits is exceeded, the sequence of statements followed by the simulation is determined by the answer to the question "*Can a Combi Activity Start*?". If the answer is "*Yes*" then the simulation executes the statements in the "*Generate Phase*"; otherwise, it executes the statements in the "*Advance Phase*". After the statements in a particular phase are finished, the simulation returns to the question "*Can a Combi Activity Start*?" and the process is repeated again.

In order to answer this question, UM-CYCLONE must examine whether each one of the Combi Activities in the network can start. As soon as the system finds one Combi that can start, the answer is "*Yes*" and the system stops the search (at least temporarily). If all activities are examined, and no activity can start, then the answer is "*No*".

The order in which Activities are examined to determine which one can start is very important because it determines the *relative priority* of Combi Activities. Combi Activities can start whenever their directly preceding Queues are not empty. If two or more Combi Activities are preceded by the same Queue, then they are essentially competing for the same resource and their relative priority becomes an issue. As an example, consider a small network with two Combi Activities, 5 and 10, both preceded by Queue 30. If the number of resource units in Queue 30 is 1, then at most only one of the Combi Activities will be able to start: either 5 or 10, but not both.

UM-CYCLONE determines the relative priority of Combi Activities by sorting all Activities in ascending order of their ID numbers (ActNo). During simulation, the Activities are examined in the order in which they have been sorted. The Activity with the smallest ActNo is examined first, and the Activity with the largest ActNo is examined last. Thus, the relative priority of Combi Activities should be established by careful numbering of the nodes during the development of the UM-CYCLONE network. The underlying rule is very simple:

- Combi Activities with smaller ActNo numbers have priority over those with larger ActNo numbers.

For the small example network above, Activity 5 has priority over Activity 10 and will always be examined first (5 is smaller than 10). Since Activity 5 has no predecessor other than Queue 30, the priority rule implies that Activity 10 will never start. Activity 5 will always be examined first and as long as Queue 30 is not empty it will be able to start. When Queue 30 becomes empty, Activity 5 will not able to start and thus Activity 10 will be examined. However, it cannot start since Queue 30 is now empty. In order for Activity 10 to be examined and to have a chance to start, it is necessary for Activity 5 to be preceded by another Queue, for example Queue 35. If Queue 35 becomes empty before Queue 30 does, then Activity 5 cannot start and thus Activity 10 will be able to start.

The Generate Phase

A simulation enters the Generate Phase as soon as it finds one Activity that is able to start. The primary purpose of this phase is to generate a future event time to which the simulation clock will be advanced during a subsequent Advance Phase. (Do not confuse the Generate Phase with the GEN function of a Queue.)

The first step during the Generate Phase is to "*Move Units to Combi Activity*". This decreases the contents of the Queue(s) directly preceding the Combi by one unit.

The next step is to "*Generate the Combi Activity Duration, DUR*". Every time a Combi or Normal Activity starts, UM-CYCLONE determines its duration. If the Activity has a constant duration then DUR is set equal to that value. Otherwise, the Activity duration is determined by sampling from the associated probability distribution.

Given the Activity's duration, it is now possible to determine when the Activity will finish: "*Calculate End Event Time, EET = TNOW + DUR*". *EET* is the finish time for the Activity, *TNOW* is the current value of the simulation clock (the start time), and *DUR* is the sampled Activity duration.

The final step in the Generate Phase is to "*Record EET in Future Event List*". The *Future Event List* is a list of future finish times for Activities that have already started, sorted in

ascending chronological order. This list is used for advancing the simulation clock (TNOW) as explained below.

It is important to note that the simulation clock does not advance during the Generate Phase: *i.e.,* the value of the system variable TNOW, does not change. Furthermore, the Generate Phase is repeated continuously, at the same TNOW, until no Combi Activity can start. This means that if Combi Activity 5 is preceded only by Queue 30, and the number of resource units in Queue 30 is 2, then Activity 5 will be able to start twice (in parallel). Both start times will be the same and equal to TNOW. As a result, Activity 5 will post two finish times EET in the Future Event List. Whether the two EET are the same or different depends on whether Activity 5 has a deterministic or probabilistic duration.

The Advance Phase

A simulation enters the Advance Phase when no more Combi Activities can start at the current value of the simulation clock. Since no Combi can start, the purpose of the Advance Phase is to advance the simulation clock to a future point in time at which an Activity (Combi or Normal) will finish, release the resources it holds, and thus another Activity (Combi or Normal) might be able to start. The Advance Phase also updates the state of the system to reflect changes that occur between now and the new clock value.

Changes in the system occur only when an Activity finishes: the contents of Queues are increased, Combi and Normal Activities may start, and Consolidation Nodes may increment their *start counters*. Points in time between the current simulation time and the next earliest finish of an Activity are of no interest because no changes in the state of the system can occur.

Thus, instead of advancing the simulation clock by a small *dT* and examining the system for possible changes, it is much more efficient to advance the clock directly to that point in time where the next change in the system will indeed occur: *i.e.,* the next earliest EET. The fact that the simulation clock is advanced to next earliest event time is what makes UM-CYCLONE a *discrete event* simulation system.

Advancing the clock is accomplished by the first two steps in the Advance Phase: "*Find next EET in Future Event List*", and "*Advance the simulation clock: TNOW=EET*". In addition

to advancing the simulation clock, these procedures also identify which Activity is finishing at the new value of TNOW.

The next step "*Release units to following nodes*" is a complex procedure that updates the state of the system. The actions taken by this procedure depend on whether the links from the finishing Activity to its successors are deterministic or whether they form a probabilistic fork. In the former case, the procedure activates all directly succeeding links and updates the state of all direct successors. In the case of a probabilistic fork, the procedure activates only one succeeding link and updates the state of only one direct successor. This selection is accomplished by sampling based on the specified branch probabilities.

In either case, the actions of the updating procedure depend on the type of the direct successor:

- If the following node is a Queue then its contents are increased by GEN units. If no GEN is specified then the Queue contents are increased by one unit.

- If the following node is a Normal Activity then it is allowed to start immediately. This process is similar to that for a Combi. An Activity duration, DUR, is sampled and a new End Event Time, EET, is computed and inserted in Future Event List.

- If the following node is a Consolidation Node, then its *start signal* counter is increased by one. If this makes the counter equal to the value CON, then the Consolidation Node is *also* considered *finished*. Its *start signal* counter is reset to zero and the procedure "*Release units to following nodes*" is repeated for its own set of direct successors.

The final step in the Advance Phase is to examine whether the finishing Activity is the *Flagged Activity* and, if so, to increment the counter ITER: "*If Flagged Activity finishes at TNOW then ITER=ITER+1*".

Stopping the Simulation

A UM-CYCLONE simulation continues until either the time limit (*SimTimeLimit*) or the production limit (*ProdLimit*) are reached or exceeded. When this happens the program produces an output file containing the simulation results and stops.

A simulation will also stop if none of the Activities can start and the Future Event List does not contain any event times EET that are greater than the current value of the simulation clock. If this happens before the above limits are exceeded, the first step of the Advance Phase, "*Find next EET in Future Event List*", will fail. In this case, UM-CYCLONE advances the simulation clock to *SimTimeLimit* directly, and the simulation stops by reaching its time limit.

This case is of interest because it allows controlling the simulation duration by specifying the initial contents of a Queue. In order for this method to work, however, the network must be structured so that no Activity will be able to start when this Queue becomes empty. Furthermore, the time limit and the production limit must be set sufficiently high so that the Queue will become empty before either of these limits is reached.

As a simple example, consider a linear network with three nodes, one Combi Activity and two Queues. Combi Activity 10 is preceded by Queue 5 and followed by Queue 15. Activity 10 has a constant duration of 20 time units. At the start of simulation Queue 5 contains 1 unit. The simulation time limit is 100, and the production limit for Activity 10 is 50. At time 0, Activity 10 will be able to start once and will finish at time 20. When the simulation clock is advanced to 20, Activity 10 will not be able to start because Queue 5 is now empty. Since neither of the control limits has been exceeded, and since the Future Event List contains no EET greater than 20, UM-CYCLONE will simply advance the clock to the time limit 100, and the simulation will stop.

A more interesting example is the case of an earth-moving operation where a certain quantity of soil must be moved from one location to another using various kinds of excavating and hauling equipment. The simulation should stop when all the soil has been transported. The total amount of soil to be moved is specified by the initial contents of the Queue "*Soil*". This Queue is a common predecessor to several Combi Activities, each succeeded by a path that represents excavating, loading, hauling, and placing the soil by the different types of equipment. If all activities in this network depend on the existence of soil to excavate and transport, it is obvious that the simulation will stop when all the soil has been moved and the contents of the Queue "*Soil*" become zero. Thus, it is possible to stop the simulation without having to specify the appropriate (and unknown) production limit for any one of the excavation or hauling activities.

Simulation Output

The output of a UM-CYCLONE simulation consists of two parts. The first part is a complete duplicate of the model description as input by the user, so that the model may be fully reconstructed from a printout of the output file. This part includes:

- Model Title
- Sim Time Limit
- Production Limit
- Flagged Activity
- Whether an internal data map is desired
- Whether a listing of event times (EET) is desired
- Random Number Generator Seed and whether it is internal or external
- Activity Data List
- Queue Data List
- Consolidation Node Data List

The second part of the output includes the simulation results and the collected statistics.

The simulation results first report the final values of the simulation clock and the iteration counter for the *Flagged Activity*, along with their respective limits. This information indicates which limit was reached first, causing the simulation to stop. In the following description of the system statistics, the final value of the simulation clock is represented by the symbol *TotalSimTime*.

The rest of the output provides useful statistics about each node in the simulated network. The statistics section consists of three tables, one for each node type, in the following order: Queues, Activities, and Consolidation Nodes. Each line in these tables shows statistics for one node. Nodes within each table are sorted in ascending order of their ID numbers. The node's

description, if any, is shown on the preceding line. The various system statistics are described below.

Queue Statistics (*)

QUEUE: The Queue Number, *QueNo*.

UNITS @START: Number of resource units in Queue at the start of simulation.

UNITS @END: Number of resource units in Queue at the end of simulation.

OUTPUT: Number of resource units that departed the Queue.

AVERAGE: Time average of the number of resource units stored in the Queue during the simulation.

STD DEV: Standard deviation of the number of resource units stored in the Queue during the simulation.

Q>q: P[$Q > q$]; fraction of *TotalSimTime* during which the number of units in the Queue exceeded the value q.

(*) Note: The number of units in the above Queue statistics are given in terms of *incoming* units and not GENerated (*outgoing*) units. *I.e.*, units to which the GEN function has not been applied yet.

Activity Statistics

ACTIVITY: The Activity Number, *ActNo*.

OUTPUT: Number of times the Activity has finished.

GROSS_RATE: *Output* / *TotalSimTime*

NET_RATE: *Output* / (*LastFinishTime* - *FirstStartTime*)

AVGDUR: Average Activity duration.

SDDUR: Standard deviation of Activity duration

START1: Time of Activity's first start.

STARTN: Time of Activity's last start.

AVGINT: Average of the time intervals between successive Activity starts.

SDINT: Standard deviation of the time intervals between successive Activity starts.

Consolidation Node Statistics

CONSOL: The Consolidation Node Number, *ConNo*.

OUTPUT: Number of times the Consolidation Node *finished* (*i.e., consolidated*).

GROSS_RATE: *Output* / *TotalSimTime*

NET_RATE: *Output* / (*LastFinishTime* - *FirstStartTime*)

AVGDELAY: Average Consolidation Node delay. (*)

SDDLY: Standard deviation of Consolidation Node delay. (*)

START1: Time the Consolidation Node received the first *start signal*.

STARTN: Time the Consolidation Node received the last *start signal*.

AVGINT: Average of time intervals between the Consolidation Node's successive finish events.

SDINT: Standard deviation of time intervals between the Consolidation Node's successive finish events.

(*) Note: The *delay* in a Consolidation Node is the time span from the time the internal *start signal* counter becomes 1 (first *start signal*) to the next time the *start signal* counter reaches the value CON (next *finish*).

Other Queue Statistics

Most of the statistics reported by UM-CYCLONE are straightforward and can be used directly. Some useful Queue statistics, however, are not part of the output and must be derived from other simulation results. This section explains how to compute the *average waiting time per visit in a Queue*, and the *average of the total waiting time that resources spend in a Queue.*

Consider a Queue with at least one incoming link and one outgoing link. Resources enter and exit the queue during simulation. At any point in time, the number of units q in the Queue might be 0, 1, 2,..., N. Let $t(q)$ be the simulated time during which the Queue contained exactly q units, and T be the total duration of the simulation. Obviously,

$$T = \sum_{q=0}^{N} t(q)$$

The time average of the number of units stored in the Queue, reported by UM-CYCLONE under the heading "AVERAGE", is given by:

$$\bar{q} = \sum_{q=0}^{N} [q \frac{t(q)}{T}] = \sum_{q=0}^{N} [q \cdot f(q)]$$

where $f(q)$ is the fraction of the total simulation time during which the Queue contained exactly q units.

Every time a resource unit enters the Queue it remains there (waiting) until it is removed by the start of a following Combi. Thus, resource units *visit* the Queue (enter and exit), and each visit is associated with a certain waiting time w_v. The sum of all the waiting times over all visits equals

$$\sum_{v=1}^{V} w_v = \sum_{q=0}^{N} q \cdot t(q) = \bar{q} \cdot T$$

where V is the number of resource units that have exited the Queue plus the number of units that remain in the Queue at the end of the simulation:

$$V = \text{OUTPUT} + \text{UNITS @END.}$$

Then, the average waiting time per visit is:

$$\bar{w} = \frac{\bar{q} \cdot T}{V} = \frac{(\text{AVERAGE queue contents})(\text{TotalSimTime})}{(\text{OUTPUT} + \text{UNITS@END})}$$

Notice that $\bar{w}$ is the average duration of a *generic* visit, and not the average waiting time for a *particular* resource that may have visited the Queue. Since several resource units may enter the Queue, and each resource unit may visit the Queue a different number of times (each having a

different duration), it is impossible to compute the average waiting time for a particular resource without tracking the flow of individual resource units through the simulation network. As a result, $\overline{w}$ must be interpreted as the average waiting time per visit of the *generic* (or *average*) resource unit.

Suppose that the number of *different* resource units that have visited the Queue at various points in time is known to be N. Obviously, N must be less than or equal to the number of visits V to account for multiple visits by the same resource unit. The total waiting time (total visiting time) for the i^{th} of the N resources is W_i. Each W_i is the sum of nv_i separate waiting times, where nv_i is the number of visits to the Queue by the i^{th} resource unit. If W_i and nv_i were known, then it would be possible to compute the average waiting time per visit for the i^{th} resource unit: W_i / nv_i.

Even though the individual W_i and nv_i are not reported by the simulation, it is possible to compute their sums and averages over all N resources.

The sum of the individual number of visits over all resources equals the total number of visits:

$$\sum_{i=1}^{N} nv_i = V$$

The average number of visits per resource is:

$$\overline{nv} = \frac{1}{N}\sum_{i-1}^{N} nv_i = \frac{V}{N}$$

The sum of the individual total waiting times over all N resources equals the sum of the waiting times for all visits V:

$$\sum_{i=1}^{N} W_i = \sum_{v=1}^{V} w_v = \sum_{q=o}^{N} q \cdot t(q) = \overline{q} \cdot T$$

The average total waiting time over all N resources is:

$$\overline{W} = \frac{1}{N}\sum_{i=1}^{N} W_i = \frac{\overline{q} \cdot T}{N} = \frac{\text{(AVERAGE queue contents)(TotalSimTime)}}{\text{(Number of different resources that visited the queue)}}$$

Notice that $\overline{W}$ can also be interpreted as the total waiting time for the *average* resource. As a result, this quantity can also be derived by multiplying the average waiting time per visit $\overline{w}$ times the average number of visits per resource $\overline{nv}$:

$$\overline{W} = \overline{w} \cdot \overline{nv} = \frac{\overline{q} \cdot T}{V} \cdot \frac{V}{N} = \frac{\overline{q} \cdot T}{N}$$

Other Queue statistics can be computed by similar calculations.

Conclusion

The modeling elements of UM-CYCLONE can be combined to produce simulation networks for almost any discrete system. Portions of these networks, especially those that deal with daily work clocks, lunch and overnight breaks, resource scheduling, resource arrivals, etc., can be standardized and reused from one network to the next. Thus, it is possible to develop network "mechanisms" that serve as ready to use simulation network IC's. A good example of a standard mechanism is a "universal clock" that measures simulated time independently of the interactions between the simulated system's Activities and Queues, and which controls the availability of resources based on shifts, lunch breaks, and evening breaks. This and other examples of standard mechanisms appear in the demonstration problems distributed with the UM-CYCLONE programs.

References

A description of the original CYCLONE modeling system, as well as example network models, can be found in:

Halpin, D.W., and Woodhead, R.W. (1976). "Design of Construction and Process Operations." John Wiley and Sons, Inc., New York, N.Y.

A description of the UM-CYCLONE programs can be found in:

Ioannou, P.G. (1989). *UM-CYCLONE Discrete Event Simulation System, User's Guide*, Report UMCE 89 12, Dept. of Civil Engin., Univ. of Michigan, Ann Arbor, MI.