Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

GENERAL PURPOSE SIMULATION WITH STROBOSCOPE

Julio C. Martinez
Photios G. Ioannou

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

ABSTRACT

Stroboscope is a programming language designed for the simulation of processes common to construction engineering. These processes are very complex and involve many different types of resources. As a result, Stroboscope is also an efficient and effective general purpose simulation system. This paper presents an overview of Stroboscope and illustrates how it can be used to model a complex example taken from classic simulation literature.

1 OVERVIEW

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

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

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

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

2 EXAMPLE — AIRPLANE SERVICE CENTER

The Stroboscope simulation language is best illustrated with an example. For the purposes of this discussion we have selected an example that should be easily understood by a wide audience. It is an airplane service center, and is a paraphrasing of problem 2.32 in (Law and Kelton, 1991). Obviously, our intent is to illustrate the capabilities of the language, not the operations of an airplane service facility.

2.1 EXAMPLE DESCRIPTION

Planes of seven different types i (see table 1) arrive at a service center for inspection and possible repair of their engines. The times between successive arrivals are exponentially distributed with means a_i days. The service

Table 1: Plane Types

i	Type	n_i	a_i	A_i	B_i	p_i	r_i	c_i
1	B707	4	8.1	0.7	2.1	0.30	2.1	2.1
2	B727	3	2.9	0.9	1.8	0.26	1.8	1.7
3	B737	2	3.6	0.8	1.6	0.18	1.6	1.0
4	B747*	4	8.4	1.9	2.8	0.12	3.1	3.9
5	DC8	4	10.9	0.7	2.2	0.36	2.2	1.4
6	DC9	2	6.7	0.9	1.7	0.14	1.7	1.1
7	DC10*	3	3.0	1.6	2.0	0.21	2.8	3.7

center maintains n service facilities, each of which can host one plane at a time. When a plane arrives it is immediately checked into a facility if one is available, otherwise it enters a queue common to all arriving planes. When a facility is available, an airplane is checked-in and its engines are serviced one by one. Service on one engine must be complete before work starts on another engine.

The service performed on each engine consists of a series of inspections and possible repairs. Engines are repaired if the result of the inspection indicates the need. The time required to perform an inspection is uniformly distributed between A_i and B_i days. The probability that an engine needs repair is p_i. The time required to repair an engine follows a 2-Erlang distribution with mean r_i days. Engines that have been repaired must be re-inspected and if necessary repaired again. Work continues on an engine until it eventually passes an inspection. The time required to inspect or repair an engine that has already been inspected or repaired at least once follow the same distributions, but with parameters that are 50% of the original. The probability of an engine failing an inspection more than once is also reduced by 50%. The reduced time distributions and probability of failure remain at the 50% of the original level for the second and all subsequent inspections and repairs.

When all engines in a plane are serviced, the plane is checked-out and the facility is available to other planes.

The downtime (waiting in queue or being serviced) cost in $/day for a plane of type i is c_i. The objective of this problem is to study the relationship between average daily downtime cost (over all types of planes) and the number of service facilities. The problem also requires an investigaion of the effects of two different queing disciplines: strict FIFO vs. giving priority to wide-body planes (these are denoted with a * in table 1) while retaining FIFO rule within the wide-body and regular classifications. The study period is 10 years (3650 days).

2.2 SOLUTION

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

2.2.1 THE NETWORK

Figure 1 shows the network for this model. At an abstract level, the Combis (rectangles with cut-offs in the top-left corner), Normals (rectangles), and Queues (large circles with a slash in the bottom right corner) shown in this figure are similar in appearance and function to CYCLONE modeling elements (Halpin and Riggs 1992, Ioannou 1989); the Forks (smaller circles with an inscribed triangle) resemble COOPS Routers (Liu 1991, Ioannou and Liu, 1992). The Links connecting elements are named, by convention, with 2 letters that abbreviate the type of resource that flows through them followed by a number. In this model *PL* stands for plane, *AS* for arrival scheduler, and *SF* for service facility.

Resources move from node to node in the direction of the Link arrows. Queues are storage locations for

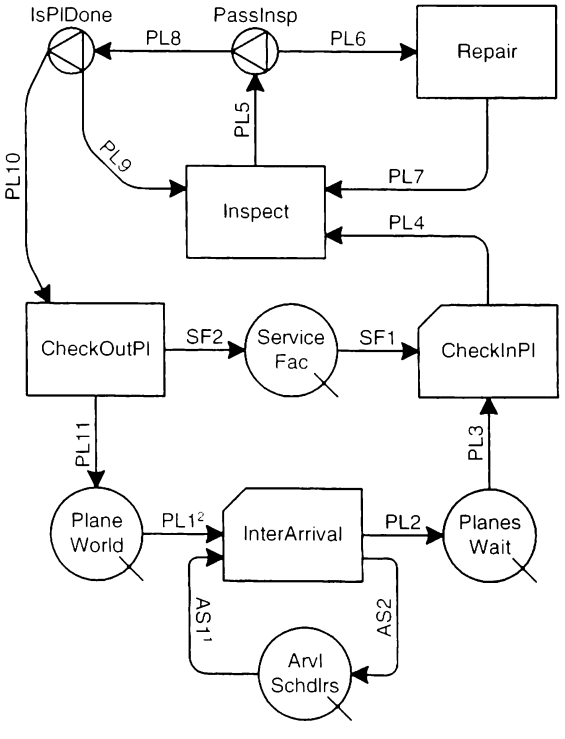

Figure 1: Plane Service Facility Network

inactive resources. Combis and Normals are classes of activities that require and take hold of resources for a certain amount of time. Combis activate themselves when enough resources are available in the Queues that precede them; upon activation they remove from their preceding Queues the resources needed to support the activity they represent. Normal activities start when activated by any of their preceding nodes and receive the resources released by the ending activity. A Fork is a probabilistic or decision-making node that selects which one of its successors to activate and route resources to.

Figure 1 shows planes going through a cycle. Initially a plane is in the *PlaneWorld* Queue until it is removed by the *InterArrival* Combi, which later releases the plane to the *PlanesWait* Queue. The plane remains in the *PlanesWait* Queue until the *CheckInPl* Combi removes it and immediately releases it to the *Inspect* Normal. After inspection, the *PassInsp* Fork decides probabilistically whether to activate the *Repair* Normal (inspection failed) or the *IsPlDone* Fork. If the *IsPlDone* Fork is activated it decides non-probabilistically whether to activate the *Inspect* Normal again (inspect another engine in the same plane) or the *CheckOutPl* Normal (plane is done and leaves the service facility). The *Inspect* Normal then releases the plane to whichever of the three possible successors: *Repair* via *PL5-PassInsp-PL6*; *CheckOutPL* via *PL5-PassInsp-PL8-IsPlDone-PL10*; or another instance of *Inspect* via *PL5-PassInsp-PL8-IsPlDone-PL9*. If the *Repair* Normal is activated, it takes hold of the plane for repairs and then releases it to a new instance of *Inspect* where the same engine is re-inspected. A plane may go through several *Inspect* - *Repair* cycles before it goes to service another engine or leaves the facility. Eventually all the plane's engines will be serviced and the *CheckOutPl* Normal will be activated. Upon activation, *ChckOutPl* will immediately release the plane to the world of operating planes, the *PlaneWorld* Queue.

Arrival schedulers are fictitious resources used to control the inter-arrivals of the different types of planes. Only one arrival scheduler resource exists for each of the seven plane types. An arrival scheduler starts out in the *ArvlSchdlrs* Queue until it is removed by the *InterArrival* Combi. Later, the *InterArrival* Combi releases the arrival scheduler back to the *ArvlSchdlrs* Queue, where the cycle starts again.

A service facility is initially in the *ServiceFac* Queue until the *CheckInPl* Combi removes it. The service facility is then implicitly attached to a plane until the plane to which it is attached enters the *CheckOutPl* Normal. *CheckOutPl* then explicitly releases the service facility to the *ServiceFac* Queue, where the cycle starts again.

The Combis control inter-arrivals and plane check-ins. The *InterArrival* Combi requires both an arrival scheduler and a plane for it to start. While the number of arrival schedulers is exactly seven, the number of planes in the world is possibly enormous. Because of this, the number of simultaneous instances of *InterArrival* is limited to the number of arrival schedulers. Once all available arrival schedulers are involved in instances of *InterArrival*, further instances of *InterArrival* can not occur because the *ArvlSchdlrs* Queue is empty. As denoted by the superscripts in links $AS1^1$ and $PL1^2$, the *InterArrival* Combi first removes an arrival scheduler and then a plane. By looking at the properties of the arrival scheduler, it can decide which type of plane to remove from the *PlaneWorld* Queue. When an instance of *InterArrival* ends, the arrival scheduler returns to the *ArvlSchdlrs* Queue. This allows a new instance of *InterArrival* to take place — another plane of the appropriate type is removed from *PlaneWorld*.

Similarly, the *CheckInPl* Combi creates the queuing effect in the *PlanesWait* Queue. Because only *n* service facilities exist, planes that enter the *PlanesWait* Queue may not find a service facility available in the *ServiceFac* Queue — the *CheckInPl* Combi will not remove the plane. It is not until the *CheckOutPl* Normal releases a service facility to the *ServiceFac* Queue, that *CheckInPl* will start and remove a waiting plane from the *PlanesWait* Queue. Note that the *CheckInPl* Combi also controls queing in the the *ServiceFac* Queue — a service facility is not removed (and *CheckInPl* will not start) if there is no plane to check-in.

2.2.2 SIMULATION MODEL SOURCE CODE

The network provides a high level representation of the operation. More specific details of the model are shown only in the source listing. The discussion that follows will go through the entire source-code listing for this model.

Stroboscope statements end in a semicolon. They can span several lines with arguments separated by white-space. Arguments that include white space must be enclosed in single quotations; text arguments must be enclosed in double quotations. In this paper, source code lines are shown using a fixed-pitched font; object names that have been defined explicitly for this model and are not part of the Stroboscope language are shown in italics (Stroboscope keywords and plain numbers are not).

2.2.2.1 DECISION VARIABLES

Variables that store model parameters are placed at the beginning of the source file so that they can be found

and changed easily when performing experiments with the model:

```
VARIABLE WideBodyFirst       1;
VARIABLE nServiceFacilities 11;
VARIABLE DaysToSimulate      3650;
```

WideBodyFirst is a flag used to specify the queuing discipline. It is set to zero to indicate strict FIFO, and to a non-zero value to indicate wide body - FIFO. *nServiceFacilities* holds the number of service stations in the service center. *DaysToSimulate* holds the number of days for which the simulation will run.

2.2.2.2 RESOURCE TYPES

The resource types used in this model need to be defined (arrival scheduler, plane, and service facility). Both 'arrival scheduler' and 'plane' are characterized resource types (i.e., they represent non-bulk resources that have properties attached to them). The resource type 'service facility' is generic (i.e., it represents bulk amounts of property-less resources). These three types are defined below:

```
/******* AS = ArvSchdlr **************
CHARTYPE ArvSchdlr          i  a;
/=====================================
SUBTYPE  ArvSchdlr ASB707   1  8.1;
SUBTYPE  ArvSchdlr ASB727   2  2.9;
SUBTYPE  ArvSchdlr ASB737   3  3.6;
SUBTYPE  ArvSchdlr ASB747   4  8.4;
SUBTYPE  ArvSchdlr ASDC8    5  10.9;
SUBTYPE  ArvSchdlr ASDC9    6  6.7;
SUBTYPE  ArvSchdlr ASDC10   7  3.0;
```

ArvSchdlr is the actual type name used for the arrival scheduler resource type. Resources of type arrival scheduler have properties that indicate the type of plane *i*; and the mean inter arrival time in days *a*. *ASB707*, *ASB727* ... *ASDC10* are subtypes of *ArvSchdlr* that have specific values assigned to the *i* and *a* properties.

```
/******* PL = Plane ****************
CHARTYPE Plane      i    nEng c    WB
                    A    B    p    r;
/=====================================
SUBTYPE Plane B707  1    4    2.1  0
                    0.7  2.1  30   2.1;
/-------------------------------------
SUBTYPE Plane B727  2    3    1.7  0
                    0.9  1.8  26   1.7;
/-------------------------------------
SUBTYPE Plane B737  3    2    1.0  0
                    0.8  1.6  18   1.0;
/-------------------------------------
SUBTYPE Plane B747  4    4    3.9  1
                    1.9  2.8  12   3.9;
/-------------------------------------
SUBTYPE Plane DC8   5    4    1.4  0
                    0.7  2.2  36   1.4;
/-------------------------------------
SUBTYPE Plane DC9   6    2    1.1  0
                    0.9  1.7  14   1.1;
/-------------------------------------
SUBTYPE Plane DC10  7    3    3.7  1
                    1.6  2.0  21   3.7;
```

Resources of type *Plane* have properties that indicate the type of plane *i*; the number of engines in the plane *nEng*; the daily downtime cost *c*; a flag to indicate their classification as regular (0) or Wide Body (1) *WB*; parameters for the duration of the inspection of one of its engines *A*, and *B*; probability of failing the first inspection *p*, and a parameter for the duration of the repair of one of its engines *r*. *B707*, *B727* ... *DC10* are subtypes of *Plane* that have specific values assigned to their properties.

Resources of type *Plane* additionally have the following read/write properties:

```
SAVEPROP Plane EngsToGo;
SAVEPROP Plane Repaired;
SAVEPROP Plane RepStartTm;
```

EngsToGo keeps track of how many engines in a plane are pending inspection/repair. *Repaired* is a flag that indicates if the current engine has been repaired already. *RepStartTm* stores the simulation time at which a plane checks-in.

```
/******* SF = ServFac ***************
GENTYPE ServFac;
```

ServFac is the actual type name used for the service facility resource type.

2.2.2.3 NETWORK DEFINITION

The network shown graphically in figure 1 is defined in the source file as follows:

```
/******* Queues *********************
QUEUE ArvlSchdlrs ArvSchdlr;
QUEUE PlaneWorld  Plane;
QUEUE PlanesWait  Plane;
QUEUE ServiceFac  ServFac;
```

```
/******* Combis **********************
COMBI InterArrival;
COMBI CheckInPl;

/******* Normals *********************
NORMAL Inspect;
NORMAL Repair;
NORMAL CheckOutPl;

/******* Forks ***********************
FORK PassInsp Plane;
FORK IsPlDone Plane;

/******* Links -- AS Cycle ***********
LINK AS1  ArvlSchdlrs  InterArrival;
LINK AS2  InterArrival ArvlSchdlrs;

/******* Links -- PL Cycle ***********
LINK PL1  PlaneWorld   InterArrival;
LINK PL2  InterArrival PlanesWait;
LINK PL3  PlanesWait   CheckInPl;
LINK PL4  CheckInPl    Inspect  Plane;
LINK PL5  Inspect      PassInsp;
LINK PL6  PassInsp     Repair;
LINK PL7  Repair       Inspect  Plane;
LINK PL8  PassInsp     IsPlDone;
LINK PL9  IsPlDone     Inspect;
LINK PL10 IsPlDone     CheckOutPl;
LINK PL11 CheckOutPl   PlaneWorld;

/******* Links -- SF Cycle ***********
LINK SF1  ServiceFac   CheckInPl;
LINK SF2  CheckOutPl   ServiceFac;
```

The preceding statements that define the graphical network can be inferred directly and unambiguously from figure 1. This section of a Stroboscope model is highly amenable to interactive graphics preprocessing tools.

2.2.2.4 MODELING ELEMENT ATTRIBUTES

Elements in the network have attributes that define the way they behave. Attributes have default values that make the network element to which they apply behave the natural or expected way. In the next part of the source file, attributes of modeling elements that are different from the default are set:

```
DRAWWHERE PL1
   i==InterArrival.ArvSchdlr.i;
```

Ordinarily, a Combi starts by removing the first available resource from each one of the Queues that preceed it. The statement above specifies that the plane to be removed by the *InterArrival* Combi from the *PlaneWorld* Queue through Link *PL1* must be such that its *i* property is the same as the *i* property of the arrival scheduler already acquired. Note that Link *AS1* is defined before Link PL1 in the network definition; this is indicated in the network drawing with the superscripts on the Link labels.

```
DURATION InterArrival
   'Exponential[
   InterArrival.ArvSchdlr.a]';
```

Unless otherwise specified, an activity (Combi or Normal) has no duration. This statement indicates that the duration of an instance of the *InterArrival* Combi is sampled from an exponential distribution with mean equal to the *a* property of the arrival scheduler acquired.

```
DISCIPLINE PlanesWait
   WideBodyFirst?-WB:1;
/i.e., IF(WideBodyFirst,-WB,1)
```

The default queuing discipline for Queues is FIFO. The above statement sets the discipline for Queue *PlanesWait* to give priority to wide-body planes if the *WideBodyFirst* variable is true (i.e., not zero). Otherwise the discipline remains FIFO. When resources tie in their values of *WB*, the FIFO rule takes effect. Similarly, in the case in which *WideBodyFirst* is false, the constant sub-expression '1' is the same for all resources and thus a strict FIFO rule takes effect again.

```
COLLECTOR WaitCost;
ONDRAW PL3 WaitCost
   (SimTime-TimeIn)*c;
```

WaitCost is a statistics collector that keeps track of costs incurred while planes are waiting. When the *CheckInPl* Combi draws a plane from the *PlanesWait* Queue, the cost incurred (the time spent in queue multiplied by the daily cost of the plane *c*) is added to the set of samples collected by *WaitCost* (SimTime is a system maintained variable that represents the current simulation time and TimeIn is a read-only property defined and updated by the system).

```
ONRELEASE PL4 EngsToGo   nEng;
ONRELEASE PL4 Repaired   0;
ONRELEASE PL4 RepStartTm SimTime;
```

When *CheckInPl* releases a plane through Link *PL4*, the *EngsToGo*, *Repaired*, and *RepStartTm* properties of the plane are initialized. Originally all the engines in the

plane are pending service (*EngsToGo=nEng*) and the current engine has not been repaired (*Repaired=0*). The plane is checked into the facility at the simulation time at which it flows through Link *PL4* (*RepStartTm* = SimTime).

```
VARIABLE RepairFactor
    1+Inspect.Plane.Repaired;
```

The time required to inspect an engine and the probability that the engine requires repair are reduced by a factor of 2 if the engine has already been repaired. *RepairFactor* is an auxiliary variable that is re-computed every time it is used. Its value is either 1 or 2, depending on the value of the *Repaired* property of the *Plane* currently in the *Inspect* Normal.

```
DURATION Inspect
    'Uniform[Inspect.Plane.A,
    Inspect.Plane.B]/RepairFactor';
```

The duration of an engine inspection is uniformly distributed between the *A* and *B* properties of the *Plane* to which it belongs. If the engine has been previously repaired, the time is reduced by 50%.

```
STRENGTH PL6
    Inspect.Plane.p/RepairFactor;
STRENGTH PL8
    100-Inspect.Plane.p/RepairFactor;
```

The probability that a Fork activates a particular successor depends on the strength of the Link that goes from the Fork to the successor. The strength of a post Fork Link is converted to a probability through division by the sum of the strengths of all the Links that originate from the Fork. The probability that the *PassInsp* Fork activates the *Repair* Normal is given by the *p* property of the *Plane* held by the *Inspect* Normal. If the engine has been previously repaired, the probability is reduced by 50%.

```
ONRELEASE PL5 EngsToGo
    EngsToGo-!Repair.InContext;
ONRELEASE PL5 Repaired
    !Repair.InContext?0:Repaired;
/ i.e., !x means IF(x<>0,0,1)
```

When the *Inspect* Normal releases a plane through Link *PL5*, Stroboscope updates the *EngsToGo* and *Repaired* plane properties. *EngsToGo* is decremented and the *Repaired* flag is cleared if the engine passes the inspection (!*Repair*.InContext returns 0 if *Repair* is the node selected and activated by the *PassInsp* Fork at the end of the *Inspect* Normal, otherwise it returns 1).

```
DURATION Repair
    'Erlang[2,Repair.Plane.r/
    (Repair.Plane.Repaired+1)]';
```

The duration of the *Repair* Normal follows a 2-Erlang distribution with mean equal to the *r* property of the acquired *Plane*. If the plane has been repaired, the mean is reduced by 50%.

```
ONRELEASE PL7 Repaired 1;
```

When repairs on an engine conclude, the *Repaired* flag for the plane is turned on.

```
STRENGTH PL9
    Inspect.Plane.EngsToGo!=1;
STRENGTH PL10
    Inspect.Plane.EngsToGo==1;
```

When the *PassInsp* Fork activates the *IsPlDone* Fork, *IsPlDone* decides whether to activate the *Inspect* Normal (go to the next engine in the same plane) or the *CheckOutPl* Normal (all engines have been serviced). The Boolean expressions that define the strengths of Links *PL9* and *PL10* force the probabilities to be 1 for one Link and 0 for the other. The Strength of *PL10* is 1 when all engines have been serviced (*EngsToGo* is 1 but will be decremented to 0 when the plane is released through Link *PL5*).

```
RELEASEAMT SF2 1;
```

When the *ChekOutPl* Normal finishes, it releases the corresponding service facility through Link *SF2* (the service facility had been removed by the *CheckInPl* Combi and was implicitly carried along with a plane).

```
COLLECTOR ServiceCost;
ONRELEASE PL11 ServiceCost
    (SimTime-RepStartTm)*c;
```

ServiceCost is a statistics collector that keeps track of costs incurred while planes are serviced. When the *CheckOutPl* Normal releases a plane to the *PlanesWorld* Queue, the cost incurred (the time spent between check-in and check-out multiplied by the daily cost of the plane *c*) is added to the set of samples collected by *ServiceCost*.

At this point the behavior of the model is completely defined.

2.2.2.5 QUEUE INITIALIZATION

The model's resources are created from the defined types and placed in the appropriate Queues:

```
INIT ArvlSchdlrs 1    ASB707;
INIT ArvlSchdlrs 1    ASB727;
INIT ArvlSchdlrs 1    ASB737;
INIT ArvlSchdlrs 1    ASB747;
INIT ArvlSchdlrs 1    ASDC8;
INIT ArvlSchdlrs 1    ASDC9;
INIT ArvlSchdlrs 1    ASDC10;
```

One Arrival Scheduler of each type is created and placed in the *ArvlSchdlrs* Queue.

```
INIT PlaneWorld 60 B707;
INIT PlaneWorld 60 B727;
INIT PlaneWorld 60 B737;
INIT PlaneWorld 60 B747;
INIT PlaneWorld 60 DC8;
INIT PlaneWorld 60 DC9;
INIT PlaneWorld 60 DC10;
```

A sufficiently large number of planes of each type are created and placed in the *PlaneWorld* Queue.

```
INIT ServiceFac nServiceFacilities;
```

The appropriate number of service facilities are made available by setting the initial content of the *ServiceFac* Queue.

2.2.2.6 SIMULATING & PRINTING RESULTS

Finally, the source file contains instructions that perform the simulation and display the results:

```
SIMULATEUNTIL SimTime>=DaysToSimulate;
```

The simulation will run until the value of SimTime becomes at least as large as the number of days to simulate.

```
PRINT StdOutput
"(FIFO=0,WB/FIFO=1)     : %.0f\n"
   !(!WideBodyFirst);

PRINT StdOutput
"Service facilities     : %.0f\n"
   nServiceFacilities;

PRINT StdOutput
"Av. downtime cost      : %.0f $/day\n"
   '(WaitCost.SumVal+
   ServiceCost.SumVal)/SimTime*10000';

PRINT StdOutput
"Av. delay in Queue     : %.2f days\n"
   PlanesWait.AveDur;

PRINT StdOutput
"Av. planes in Queue    : %.2f\n\n"
   PlanesWait.AveCount;

REPORT StdOutput;
```

Print to standard output the queuing discipline, number of service facilities, average daily downtime cost, average delay in queue, and average number of planes in queue. The last statement prints a report that shows detailed statistics about the modeling elements and the resources involved in the simulation.

2.3 RESULTS

The output from a run of this model (the detailed statistics report is omitted due to space considerations) is as follows:

```
(FIFO=0,WB/FIFO=1)     : 0
Service facilities     : 11
Av. downtime cost      : 323985 $/day
Av. delay in Queue     : 2.59 days
Av. planes in Queue    : 3.73
```

For each Queue, the detailed report presents the current content; lifetime content; average waiting time; and average, standard deviation, minimum and maximum queue length.

For each Activity, the detailed report presents the number of currently active instances; the lifetime number of instances; the time at which the first and last instances started; the average, standard deviation, minimum and maximum duration of the instances; and the average, standard deviation, minimum and maximum times between successive instantiations.

For all possible combinations of Subtypes and network nodes, the detailed report presents the average number of entries; average total time spent; and average, standard deviation, minimum and maximum visit time.

For each collector, the detailed report presents the number of samples and the average, standard deviation, minimum and maximum of the values collected.

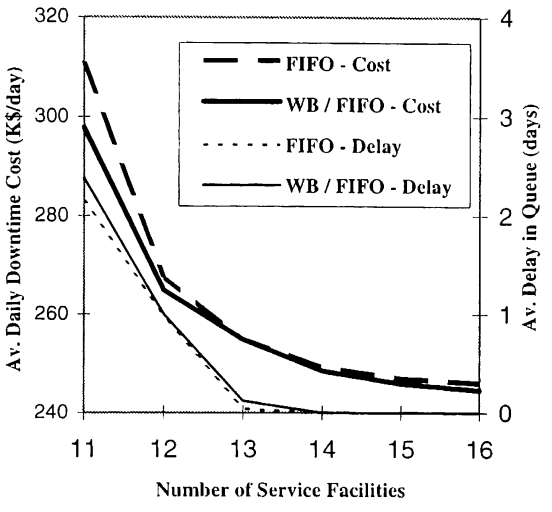

Figure 2: Downtime Cost and Average Delay

Figure 2 shows the average daily downtime cost and delay in queue plotted as functions of the number of service facilities; each point was obtained by averaging 50 replications.

3 IMPLEMENTATION

Stroboscope is implemented in two forms: a command-line compiler and an MS Windows integrated development environment (editor-debugger-compiler). Both are native MS Windows NT 32-bit programs developed in C++. The command-line compiler can run under DOS using the TNT DOS-extender. The integrated development environment can run under MS Windows 3.1, if the Win32s subsystem is installed.

4 CONCLUSION

The example presented in this paper illustrates Stroboscope's power as a general-purpose simulation system. Complex and realistic problems such as the example presented can be modeled, analyzed and optimized quite easily.

REFERENCES

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

Ioannou, P. G. 1989. UM-CYCLONE Discrete event simulation system reference manual. Technical Report UMCE 89-12, Department of Civil & Environmental Engineering, University of Michigan, Ann Arbor, Michigan.

Ioannou, P. G., and L. Y. Liu. 1992. Graphical object-oriented discrete-event simulation system. In *Proceedings of the 1992 Winter Simulation Conference*, Arlington, Virginia.

Law, A. M., and D. K. Kelton. 1991. *Simulation modeling and analysis*. 2nd ed. New York: McGraw-Hill.

Liu, L. Y. 1991. *COOPS - Construction object oriented process simulation system*. Doctoral dissertation, Civil & Environmental Engineering Department, University of Michigan, Ann Arbor, Michigan.

Martinez, J. C., Ioannou, P. G., and R. I. Carr. 1994. State and resource based construction process simulation. In Proceedings of the First Congress on Computing in Civil Engineering, Washington, D.C.

AUTHOR BIOGRAPHIES

JULIO C. MARTINEZ is a Horace H. Rackham Pre-Doctoral Fellow and a Doctoral Candidate in Civil Engineering at the University of Michigan. He designed and implemented the Stroboscope simulation language as partial fulfillment of the requirements for the degree of Doctor of Philosophy (Civil Engineering) at the University of Michigan. He received a Civil Engineer's degree from Universidad Catolica Madre y Maestra (Dominican Republic) in 1986, an M.S. degree in Civil Engineering from the University of Nebraska in 1987, and an M.S.E. degree in Construction Engineering and Management from the University of Michigan in 1993. His research interests are in computer applications to Construction Engineering and Management.

PHOTIOS G. IOANNOU is an Associate Professor in the Department of Civil and Environmental Engineering at the University of Michigan. He received a Diploma in Civil Engineering from the National Technical University, Athens, Greece, in 1979; and he received a SMCE and PhD from MIT in 1981 and 1984 respectively. He is currently serving as Chairman of the Computing in Construction Committee of the American Society of Civil Engineers. He co-developed the UM-CYCLONE construction process simulation system with R.I.Carr, supervised the design and development of COOPS by L.Y Liu, and is currently serving as chairman of J.C. Martinez's PhD Dissertation committee. His research interests are primarily focused on the areas of construction decision support systems and construction process modeling.