PASSIM: a discrete-event simulation package for PASCAL

DEAN H. UYENO holds a BS degree in mechanical engineering, an MSE from the University of Washington, and a PhD in industrial engineering and management science from Northwestern University. He joined the Management Science Division of the Faculty of Commerce at the University of British Columbia in 1971 and is currently an associate professor there. His interests include computer modeling of public systems, particularly health care systems.

WILLEM VAESSEN is a graduate student in computer science at the University of British Columbia. His academic interests lie primarily in the development of management science tools and the social impact of the application of such tools.

ABSTRACT
This paper describes a set of procedures that can be used to construct discrete-event simulation models in PASCAL. The set of procedures is called PASSIM for PASCAL-based SIMulation. The approach combines scheduler and entity concepts from GPSS with pointer-based data structures and control structures from PASCAL. The procedures employ only standard PASCAL statements to ensure portability.

INTRODUCTION
PASCAL has become a popular high-level language. Its ability to support data structures, structured programming, and top-down design is a good reason to use it for simulation modeling. We describe a set of procedures, written in standard PASCAL as defined by Wirth, to guarantee portability, which facilitate the implementation of discrete-event models. The set of procedures is called PASSIM for PASCAL-based SIMulation.

MOTIVATION
Despite the availability of sophisticated simulation languages, many analysts implement models in general-purpose, high-level languages. Franta suggests that up to 70% of all simulations are programmed in FORTRAN. A major reason is that modelers are usually familiar with FORTRAN and, of course, with simulation concepts, but are unwilling to invest the time to learn a special-purpose simulation language. Unfortunately, FORTRAN does not provide built-in procedures for such basic requirements of discrete-event modeling as describing entities and events, scheduling events, and formatting and analyzing results.

To overcome this deficiency several FORTRAN-based simulation packages such as SPURT and GASP have been created. (Indeed, SIMSCRIPT at one time was dependent on FORTRAN.) GASP initially was a small set of procedures to assist in data analysis, scheduling, queuing, random deviate generation, and statistical analysis. With use, GASP has been continually expanded and now has impressive capabilities.

However, in the years since the invention of FORTRAN, there have been improvements in the design of general-purpose, high-level languages. FORTRAN may well lose its position of leadership with the rapid increase in popularity of PASCAL. Users often cite significant reduction of software development costs as the major reason for switching to PASCAL. Furthermore, PASCAL’s popularity in academic circles has led to its use as the main language in introductory computer science courses. A discrete-event modeling package for PASCAL would therefore simplify the integration of simulation into the curriculum.

In view of these facts, we developed a set of procedures to help in implementing discrete-event models in PASCAL. The procedures assume an event-oriented approach like that taken by GPSS. In this paper, we provide a broad outline of this approach to simulation software development for PASCAL. Other approaches such as the process-oriented approach of SIMULA are certainly also worthy of investigation.

THE EVENT-ORIENTED VIEWPOINT
PASSIM forces the modeler to adopt an event-oriented viewpoint. The modeler must define the kinds of events that occur in the system and specify how those events change the state of the system.
To illustrate the event-oriented viewpoint, consider a single-queue, single-server system. A PASSIM model of this system will be shown later. As illustrated in Figure 1, the server is a worker in a repair shop who repairs defective parts.

Figure 1 - A single-server single-queue system

Parts arrive at the shop and join the queue in a first-come, first-served order. The worker works on one part at a time, stopping only when all work is done. If a part arrives while the worker is idle, work is begun immediately. Later, we will assume some probability distributions for interarrival and service times.

The length of the queue of defective parts and the status of the worker characterize the system. A state of the system is an ordered pair \((Q, BUSY)\), where the integer \(Q\) denotes the length of the queue, and the Boolean variable \(BUSY\) is the status of the worker. \(BUSY\) is true if the worker is busy and false otherwise.

The queue and the worker are referred to as system entities. The defective parts are referred to as dynamic entities. Events (i.e., state transitions) cause parts to move through the system. This example involves three kinds of events, as shown in Figure 2.

The arrival of a part increases the length of the queue by 1, but does not alter the status of the worker. The start of service for a part changes \(BUSY\) to true and decreases the length of the queue by 1. The completion of service changes \(BUSY\) to false.

It is useful to distinguish between primary and secondary events. An event is secondary if it occurs at time \(t\) because of another event occurring at the same time. An event is primary if it is not secondary. Primary events must be scheduled in advance.

In this example, event A (arrival of a part) triggers event S (start of service) if \(BUSY\) is false. Event C (completion of service) triggers event S if \(Q > 0\) (i.e., if there is another part in the queue). Thus S is a secondary event and A and C are primary events.

In such a discrete-event model, among the questions of interest are:

- What is the average time required to repair a part?
- What is the average time parts spend in the queue?
- What are the average and maximum lengths of the queue?
- How busy is the worker?

Answering these questions requires a more detailed description of the entities. For instance, statistics associated with the queue must accumulate the number of entering parts and the time they spend in the queue. These statistics must be updated when events occur.

**EVENT SCHEDULING**

A discrete-event model should produce a realistic sequence of events; it must update the state of the system as events occur. Therefore, an essential part of a model is a scheduling procedure which maintains the simulated clock and schedules events. Events must be ordered according to the times at which they are scheduled to occur, called event-times.

Lists are the natural vehicle with which to implement ordering.

Primary events must be scheduled in advance of their event-times. In practice, the scheduling procedure maintains two lists: one for events scheduled to occur at the current simulated clock time and one for events scheduled to occur in the future.

The occurrence of the start of the service event triggers the generation of the service completion event. This latter event is scheduled by placing it on the future event chain. Its event-time is the sum of the current clock time and the service time.

**ADVANTAGES OF A PASCAL IMPLEMENTATION**

PASCAL allows the modeler to represent the various entities clearly and concisely. The type declaration allows the modeler to define new types that combine the structured types array and record and the simple variable types-integer, real, Boolean, and char. The record type is composed of simpler components called fields which do not have to be of the same type. For example, the user could define a new type VECTOR to be an array of type integer. Or the user could define a type AMBULANCE to be a record with fields AGE, VELOCITY, and BUSY of types integer, real, and Boolean, respectively. The type FLEET could be an array of type AMBULANCE.

Furthermore, PASCAL provides dynamic variables. The program allocates storage for a dynamic variable at execution time by calling a standard PASCAL procedure NEW which returns a pointer to the assigned space. The programmer can combine dynamic record variables and pointers to develop the data structures required for efficient manipulation and scanning of lists and the effective utilization of memory.

**OVERVIEW OF PASSIM**

PASSIM includes a dynamic entity type and three system entity types. The dynamic entity is the transac-
tion. The system entities are the facility, storage, and chain. The facility and storage represent unit and multiple-unit capacity equipment, respectively. The chain represents lists of temporarily inactive transactions.

The scheduler also uses the chain type for its lists. Current events are stored on the current event chain and future events on the future event chain.

PASSIM provides the modeler with a basic set of procedures, called block-procedures, that represent events. The execution of a block-procedure signifies the occurrence of an event and modifies the values of attributes of system entities.

A pointer variable that contains the location of a transaction is a major parameter of most of the block-procedures. If a block-procedure is executed, the transaction to which that variable points is said to enter the block-procedure. The sequence of block-procedures that a transaction enters is called its trace.

The core of the simulation model consists of one or more sets of labelled procedure calls or segments that determine the flow of dynamic entities through the model. The core specifies the possible traces of transactions. The scheduler produces a sequence of events, ordered by their event-times, that merges the traces of the transactions that are active during the simulation.

ENTITIES

The transaction is the single dynamic entity in PASSIM. It has many fields. For the modeler, the most important fields are those that determine the flow of the transaction through the model and those used to collect statistics. Some fields serve more than one purpose. A transaction is always a member of exactly one chain. Some of the fields of the transaction record are:

PREV Pointer to the previous transaction in the chain
NEXT Pointer to the next transaction in the chain
SORT Integer determining the position of the transaction in the chain
CIT Time at which the transaction was placed in the chain
PRIORITY Priority level of the transaction
CRTIME Creation time, the time at which the transaction entered the core of the model
EVTIME Event-time, the time at which the transaction will enter the next block-procedure
NXBLOCK Label associated with this block-procedure
SI Status indicator—false if the transaction is blocked and true otherwise (if SI is false, the scheduler will not allow the transaction to enter the next block-procedure)

The facility is the simplest system entity defined in PASSIM. It also is a PASCAL record. Some of the fields of the facility are:

NAME Name of the facility
FREE True if the facility is available; false otherwise
SEIZING Pointer to the transaction seizing the facility

MAX-ST Maximum service time actually spent by any one transaction
TOT-SERVED Total number of transactions served by the facility
ACCUM-TIME Sum of the service times of all transactions served by the facility
CHANGE Clock time when the service began for the transaction seizing the facility

The storage entity is similar to the facility but can serve more than one transaction at the same time.

The chain is the third system entity available in PASSIM. It is defined indirectly since a chain is simply a pointer to a chain-head record. Some of the fields of the chain-head are:

NAME Name of the chain
FIRST Pointer to the first transaction in the chain
LAST Pointer to the last transaction in the chain
TOT-WAIT Total time spent in the chain by all transactions which have left the chain
TOT-EXITS Total number of transactions that left the chain
MAX-WAIT Maximum time spent in the chain by any transaction
CURRENT Number of transactions in the chain at the current clock time
MAX-CONT Maximum value of CURRENT at any time

Figure 3 contains a diagram of a chain and illustrates the use of pointers.

Figure 3 - Structure of a chain

Note: A pointer contains the address of an item rather than the item itself.

BLOCK-PROCEDURES

PASSIM's block-procedures represent events. The execution of a block-procedure changes the state of the system by updating the attributes of the appropriate entities. Block-procedures also schedule other events.

PASSIM provides a basic set of block-procedures. The user must define additional block-procedures for most applications. For example, user-defined block-procedures could set and test status variables to control the flow of transactions through the core of the model. Such procedures could allow a transaction to choose the shortest queue. The model in Appendix A includes a user-defined block-procedure, TEST-SPARE, which checks to see if a spare part is immediately available, and installs it if it is available or changes the machine's status to Blocked if it is not.
The remainder of this section describes the block-procedures that we used to construct the model of the repair shop. Appendix B provides a list of the block-procedures available in PASSIM.

The execution of the block-procedure GENERATE enters a transaction into the model. GENERATE has a parameter NEXT which is the label of the next block-procedure to which the transaction will be sent. GENERATE also must schedule the next transaction to be created. To this end it acquires space for that transaction, samples the interarrival time distribution to determine the event-time at which that transaction will enter the model, sets NXBLOCK to the label of the GENERATE block-procedure, and places the transaction on the future events chain.

The execution of the block-procedure TERMINATE removes a transaction from the system. TERMINATE dynamically releases the space for reuse.

The QUEUE, SEIZE, and RELEASE block-procedures specify the interaction of transactions with the facility.

The QUEUE block-procedure with parameters NEXT, SERVER, and FIELD, checks the status of the facility SERVER that the entering transaction wants to seize. If the facility is busy, the transaction is placed on the chain SERVER-QUEUE. FIELD determines the position of the transaction on the chain. If the facility is free, the transaction proceeds to the block-procedure SEIZE.

The execution of the SEIZE block-procedure, with parameters NEXT, SERVER, and ELAPSE, signifies the start of service for the entering transaction. SEIZE sets the status of facility SERVER to BUSY by setting FREE to false. It uses the time ELAPSE sampled from the appropriate distribution of service times to determine when service will be completed. Then it transfers the transaction from the current event chain to the future event chain, where it waits until the service time elapses.

The execution of the RELEASE block-procedure, with parameters NEXT, SERVER, and SERVER-QUEUE, signifies the completion of service for the entering transaction. RELEASE frees the facility SERVER by setting FREE to true. If possible, it next transfers a waiting transaction from the chain SERVER-QUEUE to the current event chain. Finally, RELEASE forces the scheduler to rescan the current event chain if a transaction is waiting so that service for this transaction can start at the current simulated clock time.

Like the QUEUE and SEIZE block-procedures, RELEASE updates the statistics of the facility and chain entities.

SCHEDULING

The procedure SCHEDULER alternately calls two procedures LOAD-CEC and EXHAUST-CEC until the simulation terminates.

LOAD-CEC advances the simulated clock in integer units to the event-time of the next event on the future event chain and transfers all transactions with that event-time from the future event chain to the current event chain. The future event chain is ordered by the event-times of the transactions. The first transaction of the chain represents the most imminent event. The current event chain is a PASCAL array of chains with one chain for each priority class of transactions.

EXHAUST-CEC moves transactions on the current event chain as far as possible through the model. It starts with the first transaction in the highest priority class. The movement of a transaction consists of the sequence of block-procedures that the transaction enters. It is a partial trace. A transaction moves through the model until it is either explicitly blocked, placed on the future event chain, placed on a user-defined chain, or terminated.

If the RASCAN flag has been set (see the paragraph on the RELEASE block-procedure) then EXHAUST-CEC will rescan the current event chain. This feature of the scheduler handles concurrent events.

HOW TO USE PASSIM

To construct a simulation model, the programmer must follow these five steps:

1. Declare the entities
2. Initialize system entities
3. Create the initial events for each segment of the model
4. Implement additional procedures and block-procedures
5. Define the core of the model.

PASSIM provides a number of procedures to assist the programmer. Procedures ON-CHAIN and OFF-CHAIN manipulate lists and can be combined to transfer a transaction from one chain to another. Procedures GETSPACE and FREESPACE are employed to minimize storage requirements. GETSPACE dynamically acquires space for transactions; FREESPACE releases space when a TERMINATE occurs. Other procedures initialize entities, display the accumulated statistics, and provide basic probability distributions.

THE REPAIR SHOP EXAMPLE

To complete the discussion of PASSIM, we implement the repair shop example following the five steps outlined in the previous section.

1. The variables LINE and SERVER represent the queue of parts and the repair shop. LINE is declared to be the type chain and SERVER is declared to be of type facility. The PASCAL declarations are:

   VAR LINE : CHAIN;
   SERVER : FACILITY;

2. LINE and SERVER are initialized and named by calling the appropriate supporting procedures. INIT-CHAIN initializes the queue of parts as empty and gives it the name LINEUP. INIT-FAC initializes the facility to an empty state and gives it the name REPAIR-SHOP.

   INIT-CHAIN (LINE, 'LINE-UP');
   INIT-FAC (SERVER, 'REPAIR-SHOP');
The initial event is the arrival of a part. Space is created for the first part by calling the supporting procedure GETSPACE. The parameter NXBLOCK is set to the label of the GENERATE block-procedure. The arrival time is set to 0 by setting the parameter EVTIME to 0. The procedure ON-CHAIN places the first part on the future event chain (abbreviated FEC).

FIRST-PART := GETSPACE(PART);
FIRST-PART.NXBLOCK := ARRIVAL;
FIRST-PART.EVTI14E 0;
ON-CHAIN (FIRST-PART,O,FEC);

No additional block-procedures are required.

The core of the model is a single segment consisting of the following five labelled block-procedure calls.

<table>
<thead>
<tr>
<th>Block-Procedures</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRIVAL: GENERATE</td>
<td>(WAIT, EXPONENTIAL(MEAN));</td>
</tr>
<tr>
<td>WAIT: QUEUE</td>
<td>(SERVICE, LINE, SERVER, EVTIME);</td>
</tr>
<tr>
<td>SERVICE: SEIZE</td>
<td>(COMPLETION, SERVER, NORMAL(MEAN,DEV));</td>
</tr>
<tr>
<td>COMPLETION: RELEASE</td>
<td>(FINISH, LINE, SERVER)</td>
</tr>
<tr>
<td>FINISH: TERMINATE</td>
<td></td>
</tr>
</tbody>
</table>

A transaction entering GENERATE triggers the next arrival event. The function EXPONENTIAL returns a sample from an exponential distribution of inter-arrival times. After leaving GENERATE, a transaction enters QUEUE. If the SERVER is free, the transaction immediately enters SEIZE; otherwise, it waits until the server becomes available. A transaction entering SEIZE is placed on the future event chain where it waits until its service time elapses. (In this example, we assume a normal distribution of service times.) A transaction entering RELEASE frees the SERVER so that a waiting transaction can proceed to SEIZE. The transaction leaving RELEASE enters TERMINATE and is removed from the model.

The reader who is familiar with GPSS should see that PASSIM block-procedures are similar to GPSS blocks.

EVALUATION OF PASSIM

PASSIM is a compact set of standard PASCAL procedures which greatly facilitate the implementation of discrete-event models. It has more capabilities than the FORTRAN-based SPURT package, but less than GASP. A few medium-sized test models have been constructed without difficulty. Although results indicate that PASSIM can be used to construct relatively efficient models, it is too early to draw definite conclusions. (See Appendix A for a comparison with GPSS.)

No simulation language or package guarantees a working and well-designed model. Much depends on the experience and skill of the modeler.

To use PASSIM the modeler should be a competent PASCAL programmer, understand basic modeling concepts, and understand the PASSIM scheduling procedure.

For most applications the modeler will want to add block-procedures to supplement the ones in PASSIM. Frequently used block-procedures could be integrated into PASSIM as usage expands.

AVAILABILITY

A documented program, a dictionary describing variables, and one or two test models are available on request.

ACKNOWLEDGEMENTS

The National Science and Engineering Research Council of Canada provided partial support for this work. The project reflects the ideas of Tony Solecki, who was one of the original participants.

REFERENCES

4. KIVIAT, P.J., VILLANEUVA, R., MARKOWITZ, H.M. SIMSCRIPT II.6 Consolidated Analysis Centers Los Angeles, California 1973
7. SPURT: a FORTRAN Simulation Package for University Research and Training University of British Columbia Computing Centre Vancouver, Canada September 1969

ADDITIONAL READING

SCHNEIDER, G.M., WEINGART, S.W., PERLMAN, D.M. An Introduction to Programming and Problem-Solving with PASCAL Wiley New York 1979
GROGONO, P. Programming in PASCAL Addison-Wesley Reading, Massachusetts 1978

(See pages 188-190 for Appendix)
APPENDIX A
COMPARISON OF PASSIM AND GPSS-V

To evaluate PASSIM, we implemented case study 6c from Reference 6 (pp. 351-356) in GPSS-V and in PASSIM.

Statement of the problem

A certain machine uses a type of part which is subject to periodic failure. Whenever the in-use part fails, the machine must be turned off. The failed part is then removed, a good spare part is installed if available, or as soon as one becomes available and the machine is turned on again. Failed parts can be repaired and reused.

The lifetime of a part is normally distributed, with a mean of 350 hours and a standard deviation of 70 hours. It takes 4 hours to remove a failed part from the machine. The time required to install a replacement part is 6 hours. Repair time for a failed part is normally distributed, with mean and standard deviation of 8 and 0.5 hours, respectively.

The machine operator himself is responsible for removing a failed part from the machine and installing a replacement part in its place.

There is a repairman who is responsible for repairing failed parts. The repairman's duties also include repair of items routed to him from another source. These other items arrive in a Poisson stream with a mean interarrival time of 9 hours. Their service-time requirement is 8 ± 4 hours. These other items have a higher repair priority than the failed parts used in the machine of interest and use the model to estimate the fractional utilization of the machine as a function of the number of spare parts provided in the system. Study the system under the alternative assumptions that zero, one, and two spare parts are provided. Run each simulation for the equivalent of 5 years, assuming 40-hour work weeks.

Both solutions were executed on an Amdahl 470 V/6 Model II computer under the Michigan Terminal System. The PASSIM version required 1.2s of CPU time while the GPSS-V version required almost twice as much time.

Only the excerpts of the PASSIM version that correspond to the divisions of the How to Use section have been included.

GPSS solution

SIMULATE
* * NON-STANDARD RANDOM NUMBER SEQUENCE INITIALIZATION(S)
* *
RMULT 121,17 SET RANDOM SEQUENCES FOR 1ST RUN
* *
FUNCTION DEFINITION(S)
* *
SNORM FUNCTION RN2,C25 STANDARD NORMAL DISTRIBUTION FUNCTION
XPDIS FUNCTION RN3,C24 EXPONENTIAL DISTRIBUTION FUNCTION
* *
VARIABLE DEFINITION(S)
* *
1 FVARIABLE 700*FN$SNORM+3500 LIFETIME FOR THE TYPE OF PART
FIX FVARIABLE 5*FN$SNORM+80 REPAIR TIME FOR THE TYPE OF PART
* *
MODEL SEGMENT 1
* GENERATE ,,,1 FIRST WORKER ARRIVES
AGAIN SEIZE MAC TURN THE MACHINE ON
ADVANCE VI PART'S LIFETIME ELAPSES
RELEASE MAC TURN THE MACHINE OFF
ADVANCE 40 REMOVE THE FAILED PART
SPLIT 1,FETCH SEIZE FIXER CAPTURE THE REPAIRMAN
SEIZE FIXER REPAIR PROCEEDS
RELEASE FIXER FREE THE REPAIRMAN
SAVEVALUE 1+,1 UPDATE THE NUMBER OF GOOD SPARES
TERMINATE LEAVE CO-WORKER WILL CARRY ON
FETCH TEST G X1,0 WAIT (IF NECESSARY) FOR A GOOD SPARE
SAVEVALUE 1,-,1 UPDATE THE NUMBER OF GOOD SPARES
ADVANCE 60 INSTALL THE PART
TRANSFER ,AGAIN GO TURN ON THE MACHINE
* *
MODEL SEGMENT 2
* GENERATE 90,FN$XPDIS,,1 "OTHER ITEMS" ARRIVE AT REPAIR SHOP
ADVANCE DUMMY ADVANCE BLOCK (SEE WRITEUP)