Int. J. Engng Ed. Vol. 12, No. 6, p. 457–466, 1996
Printed in Great Britain.

0949-149X/91 $3.00+0.00

New Approach to Teaching Discrete Event System Simulation*

A. M. ZIKIC[1] AND B. LJ. RADENKOVIC[2]
[1] *Department of Electronics Engineering and Physics, Electronics Division, University of Paisley, Paisley, PA1 2BE, Scotland, UK*
[2] *Faculty of Organisational Sciences, University of Belgrade, 11000 Belgrade, Jove Ilica 154, Yugoslavia*

The paper presents a different way of teaching discrete event system simulation. A completely new language, ISDS, based on GPSS concepts but with an interactive user interface and Pascal-like syntax with many other new features was developed as the teaching tool. The main purpose of this simulation system is to enable a cheap and easy learning and practising environment, particularly suited for students of engineering. In addition to the language definition, it's implementation and the user interface description, we also present a comparison of ISDS to GPSS and PASSIM and performance comparison between ISDS and GPSS/FON in teaching simulation. Finally, our experience in teaching simulation using ISDS is illustrated with an example of students' exercise that compares favourably with a previously published work.

SUMMARY OF EDUCATIONAL ASPECTS

1. The article discusses material for a novel course in modelling and simulation of discrete event systems, based on the ISDS language.
2. In order to fully asses the merits of the new approach and the ISDS language, a control group is taught the same syllabus under otherwise identical conditions, using GPSS/FON, [1].
3. Students of both groups (a minimum of 40 in each one) are selected such that they possess almost identical average capabilities. Additionally, the number of 'best' and 'worst' performers is equal in each group.
4. Students of mechanical, electrical/electronics and industrial engineering as well as those of computer science and management are taught.
5. Students of electrical and electronics engineering are taught at the postgraduate level while others are taught at undergraduate level.
6. Mode of presentation is through lectures, tutorials and laboratory course work. The latter consists of six compulsory laboratory and four compulsory homework exercises.
7. Material is presented in regular compulsory lectures.
8. The course is taught for sixteen weeks with two hours of lectures and two hours of tutorials each week. Laboratory and homework time is not strictly timetabled and varies from student to student. However, the lecturer is available twice a week for consultancy purposes.
9. The time needed to complete the ten compulsory laboratory/homework exercises varies from student to student and averages around 23 hours for ISDS users and 40 hours for GPSS/FON users. Students of either group that successfully complete the full course need little revision before passing their exam.
10. The novelty in our approach to teaching discrete event system modelling and simulation can be summarised in the following three statements:
    a) The new language ISDS with its modern concepts and syntax rules, described in the article, provides an easy, comfortable, affordable and yet powerful tool that enables faster learning and implementation of the modelling and simulation concepts.
    b) The integrated, interactive and user-friendly environment of both GPSS/FON and ISDS not only improve the rate at which students accept new concepts and improve their programming abilities but also makes the process of learning less frustrating than if other languages were used.
    c) The fact that both and especially ISDS are free of any proprietor fees and royalties, enables us to install as many copies as necessary and give students a copy of the appropriate language accompanied with manuals also free of any charge.

    Our experience shows that students' study is more relaxed if they can use their home computers in addition to the University facilities and that they gain better results spending fewer hours studying than if only the laboratory facilities were available.
11. The course notes issued to every student free of charge have a comprehensive coverage of the

* Accepted 27 February 1997

simulation concepts and respective language features and a complete coverage of its syntax and semantics. The short-form user-manual for the integrated environment and a set of worked 'no-nonsense' examples, that illustrate harder to grasp concepts, improve the student performance even further. Additionally, all terminals and microsystems allocated to this subject are equipped with complete GPSS/FON and ISDS manuals.

INTRODUCTION

THE DISCRETE event system theory, simulation and practice are covered in many books and articles in scientific journals and started appearing even in IEEE publications on control systems. The reader of this paper can find a brief summary of discrete event system theory in [1]. After many years of teaching the discrete event system simulation we monitored student progress in learning the concepts of GPSS and GPSS/FON [1–3] as well as the average time needed to accept an entirely new approach to programming. Our findings can be summarised as:

1. Students learn programming based on Pascal and C concepts that employ very strict and explicit typing of variables and objects.
2. On the other hand, GPSS does not have either the explicit declaration or strict typing of variables but uses a mixture of implicit and explicit of both.
3. A structured approach to programming developed by using Pascal as the main programming language becomes the second nature to students long before we attempt to teach them the discrete event simulation.
4. GPSS syntax is of assembler type with many labels and transfer block statements. At the time of its creation, the structured approach to programming was non existent and its structure is very similar to that of the contemporary languages.

It is obvious from above that our findings summarised in (1) and (3) are in an irreconcilable conflict with those in (2) and (4). There was, therefore, no doubt in our mind that the GPSS syntax and structure posed a great impediment in learning discrete event system simulation to all those who had any pre-knowledge of modern programming techniques and concepts. In order to improve the students' response to our teaching efforts, we decided to write an entirely new language that would satisfy the following requirements:

- cheap and affordable for an average student's home practising;
- language syntax with strictly divided specification of the model database, model structure and control of the simulation process;
- interactive environment with resident editor, processor and result analyser;
- fast and easy model debugging.

Before an attempt was made to write a rather pretentious package that would meet the above requirements, we decided to investigate simulation systems other than GPSS, available at the time. One of the affordable solutions was Passim, [4,5]. It embeds GPSS statements in a standard Pascal program and uses the pre-processor to generate the Pascal source. However, the main disadvantage of this solution is a necessity of the Pascal compiler. Second disadvantage is a fairly complicated debugging process for an inexperienced user. As even the most affordable package did not satisfy all our criteria we adventured into development of an Interactive System for Discrete-event System simulation (ISDS) as a joint research project of the Department of Organisational Sciences, University of Belgrade and the Department of Electronic Engineering and Physics, University of Paisley. The main requirement was to develop a simulation language with concepts similar to those of GPSS but with a small and concise kernel of syntax rules that would satisfy the following requirements:

- strict label, constants and entity declarations;
- model definitions with block statements, assign statements, wait-until and logical branch statements;
- input/output statements for interactive simulation and graphical result presentation;
- a set of control statement that supports interactive environment.

ISDS LANGUAGE SYNTAX AND SEMANTICS

When we started writing the syntax rules of the new language, it occurred to us that our graduates will have to use GPSS or some other commercially available system, rather then our ISDS. Thus whenever possible, we tried to retain the GPSS instruction names. Those instructions that are different in ISDS are either non-existent in GPSS or cause confusion in a novice's mind while using the latter. Modern concepts of compiler design 'demand' a single pass compilation which, on the other hand, means that the left recursion and backtracking must be strictly avoided. Thus, we adopted the recursive descent method for our syntax analyser which in return defined the global syntax structure of ISDS. The complete language definition in EBNF is given in [6]. We present here only a short description of the language syntax, given in Fig. 1. In addition, a number of pre-defined functions that define usual statistical distributions is readily available to the user. The semantics of typical ISDS statements is shown in Table 1.

```
<SIMULATION MODEL> ::=  MODEL <ModelName>
                            [<LABEL_definition>]
                            [<CONSTANT_definition>]
                            [<ENTITY_definition>]
                            [<FUNCTION_definition>]
                            [<MODEL_specification>]
                            [<CONTROL_specification>]
                            [<COMMENT>]
                        END_MODEL.

<ModelName> ::= <Identifier>

<LABEL_definition>  ::= LABEL <LabelName>

<CONSTANT_definition> ::= CONST <ConstantName> = <Value>

<ENTITY_definition> ::= ENT  <EntityName> : <EntityType>
    <EntityType>      ::= <SimpleType> | <ArrayType>
      <SimpleType>    ::= LOGIC| VAR | FACILITY | STORAGE | QUEUE | TABLE| CHAIN
      <ArrayType>     ::= ARRAY <Dimension> OF <BasicType>
        <BasicType>     ::= LOGIC | VAR

<FUNCTION_definition> ::= FUNCTION <FunctionName>[(<ListOfParametrers>)] := <ArithmeticExpression>

<MODEL_specification> ::= MOD_BEGIN
                              {<Statement_mod>}
                          MOD_END;
  <Statement_mod> ::= [<Label> : ] <CompoundStatement> | [<Label> : ] <SimpleStatement>
    <CompoundStatement> ::= BEGIN {<SimpleStatement>} END;
    <SimpleStatement>  ::= <BlockStatement> |<BranchStatement> | <AssignStatement>| <I/O_Statement>
      <BlockStatement> ::=  <ADVANCE> | <DEPART> | <ENTER> | <GENERATE> | <GATHER> | <LEAVE> |
                            <LINK> | <MARK> | <PRIORITY> |  <QUEUE> | <RELEASE> | <SEIZE> | <SPLIT> |
                            <TABULATE> | <TERMINATE> | <UNLINK>
      <BranchStatement> := <IF_statement> | <CASE_statement> | <TRANSFER_statement> | <WAIT_UNTIL>
      <AssignStatement> ::= <SimpleType> := <ArithmeticExpresion>
      <I/O_Statement> := <CLOSE> |<INPUT> | <OPEN> | <PRINT> | <REPORT> | <PLOT>


<CONTROL_specification> ::= CTRL_BEGIN
                                {<Statement_ctrl>}
                            CTRL_END
  <Statement_ctrl> ::= <RESET_statement> | <CLEAR_statement> | <START_statement>
      <RESET_statement> ::= RESET
      <CLEAR_statement> ::= CLEAR
      <START_statement> ::= <terminal_counter_value>
          <terminal_counter_value> ::= <numerical_operand>

<COMMENT> ::= <comment_start> {<symbol>} <comment_end>
      <comment_start> ::= {
      <comment_end>  ::= }
      <symbol>       ::= any ASCII character
```

N.B. **A comment can be placed anywhere between the MODEL <ModelName> and END_MODEL statements**

Fig. 1. Brief format of the ISDS language syntax.

Comparison of ISDS to GPSS, PASSIM and SIMAN/CINEMA

Our knowledge of and the experience in using SIMAN/CINEMA is fairly limited compared to that of PASSIM and various GPSS dialects, including our own [1], as neither of our institutions runs a copy of that language. Thus, our comparison of ISDS to SIMAN/CINEMA will be equally limited and is based on our experience through occasional access that we were granted by the licensed institutions.

Unlike GPSS, ISDS possesses a strictly formal

Table 1. Semantics of the most important ISDS instructions

INSTRUCTION	DESCRIPTION
ADVANCE	*Holds a transaction for a specified time interval* **Operand1** - Average Time ; **Operand2** - Deviation (1/2 Interval)
DEPART	*Leave a specified queue* **Operand1** - Queue Address
ENTER	*Enter a storage* **Operand1** - Storage Address
GENERATE	*Generates a transaction* **Operand1** - Average Time Between 2 Transaction ; **Operand2** - Deviation (1/2 Interval) **Operand3** - Time to Generate 1 Transaction ; **Operand4** - Total Number of Transactions **Operand5** - Priority
LEAVE	*Leave a storage* **Operand1** - Storage Address
LINK	*Admission to a user_queue* **Operand1** - User_queue Address ; **Operand2** - Type of Entry FIFO/LIFO
MARK	*Set the 'Marker' to the current value*
PRIORITY	*Transaction priority*
QUEUE	*Admission to a queue* **Operand1** - Queue Address
RELEASE	*Release a device* **Operand1** - Device Address
SPLIT	*Split a transaction* **Operand1** - Number of Transactions
GATHER	*Merge several transactions* **Operand1** - Number of Transactions
SEIZE	*Seize a device* **Operand1** - Device Address
TABULATE	*Save value for histogram* **Operand1** - Table Address ; **Operand2** - Table Attribute
TERMINATE	*Transaction leaves the model* **Operand1** - Decrement Value
UNLINK	*Transaction leaves a user_queue* **Operand1** - User_Queue Address ; **Operand2** - Address in the Queue **Operand3** - Number of Transactions Leaving ; **Operand4** - Destination Address
RESIT	*Reset the simulator*
CLEAR	*Delete the statistics and reset the clock*
START	*Start the simulator* **Operand1** - Terminal Time
OPEN	*Open a data base* **Operand1** - Add or Remove Data 1 - *INPUT* 2 - *OUTPUT* **Operand2** - Data Base Address
CLOSE	*Close data base* **Operand1** - Data Base Address
INPUT	*Read from a data base* **Operand1** - Data Base Address ; **Operand2** - List of Actual Arguments
INPUTLN	*Read a Line from a data base* **Operand1** - Data Base Address ; **Operand2** - List of Actual Arguments
PRINT	*Write to a data base* **Operand1** - Data Base Address ; **Operand2** - List of Actual Arguments
PRINTLN	*Write a line to a data base* **Operand1** - Data Base Address ; **Operand2** - List of Actual Arguments
PLOT	*Plots a table of data (Histogram)*
IF - THEN - ELSE	*Conditional branching*
CASE	*Computed conditional branching*
TRANSFER	*Unconditional branching*
WAIT_UNTIL	*Conditional temporary stop* **Operand1** - Condition for Restart

syntax definition and checking if operands are of permissible type for a given function. Also, a very useful feature is introduced through the conditional branching instruction set *IF-THEN-ELSE-CASE* that allow the nesting of program segments in a clear and easy-to-follow manner. Like GPSS and SIMAN/CINEMA, ISDS possesses the block oriented syntax structure in its processing of process interactions, but unlike the two it also has a facility of abstracting many instructions into a compound statement that makes the branching instructions much easier to use, follow and debug.

Unlike in GPSS, all static entities like *FACILITY, LOGIC SWITCH, SAVE VALUE,* etc. must be explicitly defined in a separate program segment and cannot be used unless previously defined. Thus, those errors due to a wrong implicit specification that may (as well as not) become obvious at the GPSS program execution are fully eliminated in ISDS and are reported at the compilation stage as non-defines. The ability of ISDS to compound statements and accept only explicitly defined static entities enables a trainee to make fewer mistakes in modelling a system than in either GPSS or SIMAN/CINEMA.

Although the PASSIM enables both Pascal and GPSS structures to appear in the same program, thus 'marrying the best of the two worlds', it still lacks several properties of ISDS:

1. Firstly, a PASSIM user should be reasonably fluent in both GPSS and Pascal which is not necessary in the case of ISDS; indeed, many of our re-training course attendants are not fluent in either and do not normally require any knowledge of Pascal for their present and/or future work.
2. Secondly, not all of the GPSS shortcomings are eliminated by the availability of Pascal code whereas practically all of them are in ISDS.
3. The user of PASSIM must have a Pascal compiler whereas an ISDS user does not require any additional piece of software other than the operating system.
4. An ISDS user has to debug only one language structure and not possibly two entirely different ones as a PASSIM user may.

ISDS PROCESSOR

The syntax analyser is designed by using a recursive descent method which translates the model into an internal representation, in a single phase. The internal representation of the model consists of two main parts.

1. Handling of transactions as dynamic entities. It includes:
   a) transactions
   b) transaction chains.
2. Static entities—blocks, storages, queues etc:
   a) tables of permanent entities;
   b) tables of statements;
   c) other global variables.

Transaction handling mechanism consist of two linked lists: Current Event Chain (CEC) and Future Event Chain (FEC) and procedures for transaction manipulation. In the execution phase, the processor prepares the simulation process as the 'next event strategy', by using a data base consisting of the model structure and its parameters [7]. The implementation was realised by using Turbo Pascal language and its object-oriented libraries.

STATISTICAL DATA ACQUISITION

The user must build his/her own set of control statements that depend on model structure. ISDS has facilities for collecting statistical data during run time, similar to those of GPSS. The standard numerical attributes, assign statements, input/output and control statements permit custom-designed data collection. The computation of the statistical parameters is performed by using standard recursive techniques. The main advantage in using the recursive technique is that only a simple database, consisting of two values, has to be updated in each pass. The statistical data related to permanent entities such as Storage, Facility, Queue and User Chain are obtained by using the well known statistical formulae.

The determination of the statistical values is performed when either a transaction and the entity meet or when a transaction leaves the entity. The actual evaluation is done by the appropriate block procedures as specified above. Some block procedures (Generate, Advance and Transfer) use three internal random number generators for computing their operand values. In this implementation of ISDS we use a linear multiplicative congruent pseudo random number generator [8].

USER INTERFACE

The user interface is built in accordance with prepositions given in [9], using an object-oriented approach. The communications among parts of the environment are realised through an event-driven mechanism. The latter, named 'dialogue acceptor', detects every change in the state of the environment and performs a task chosen by the user, such as the screen editor, operating system interface, simulator and the simulation result analyser.

ISDS interactive environment can be formally described by using the discrete event formalism:

$$IE = \langle IES, IM, OM, s, o \rangle$$

where

IE = integrated environment;

IES = integrated environment states comprising the finite set:

{Idle,
edit = ⟨file-1, file-2, . . . , file-n⟩,
compile ⟨file-1, file-2, . . . ,file-n⟩,
simulate,
analyse = ⟨statistical, graphical⟩,
utility = ⟨Chdir, Dos_shell⟩}

IM = input messages consisting of either keyboard or mouse motion sequences;

OM = output messages consisting of various comments, warnings and error messages;

s = state functions s: IES × IM ⇒ IES; (The event scheduler suspends the currently executed task several times per second and checks for input messages; if no input message is detected, the execution of the suspended task is resumed.)

o = Output functions o: IES × IM ⇒ OM; (The output messages scheduler is triggered by either an input message or the currently executed task.)

The number of files that can be edited or compiled simultaneously is limited by the available memory. The system operates as a quasi multitasking system; each state can be temporarily frozen in favour of some other one and reassumed latter. There is, however, no background execution of any suspended tasks.

The integrated environment provides a quick and comfortable means of system modelling, simulation and data analysis. In addition, the utility option enables a quick access to the operating system utilities. A typical control screen of the IE is shown in Fig. 2.

Example

A similar model to the one published [1] is used, for comparison reasons, to illustrate the language:

> Write the simulation model by using the following experimental data: Maximum number of customers in the shop is 40 at any time; currently, the manager runs the shop with 20 baskets only and virtually no queuing at the tills. There are two standard tills for any number of items and one express till for up to 5 items. Shoppers arrive at average time intervals of one minute with exponential distribution. A shopper buys 20 items in average. If baskets are not available, a shopper leaves the shop (a lost customer). The shop opens at 9.00h and closes at 18.00h.

Figure 3 presents one of the solutions submitted by students. Figure 4 shows the queue length at standard tills. Additional analysis identical to the one in [1], including 3D plots, can also be performed and is not included in this paper for obvious reasons.

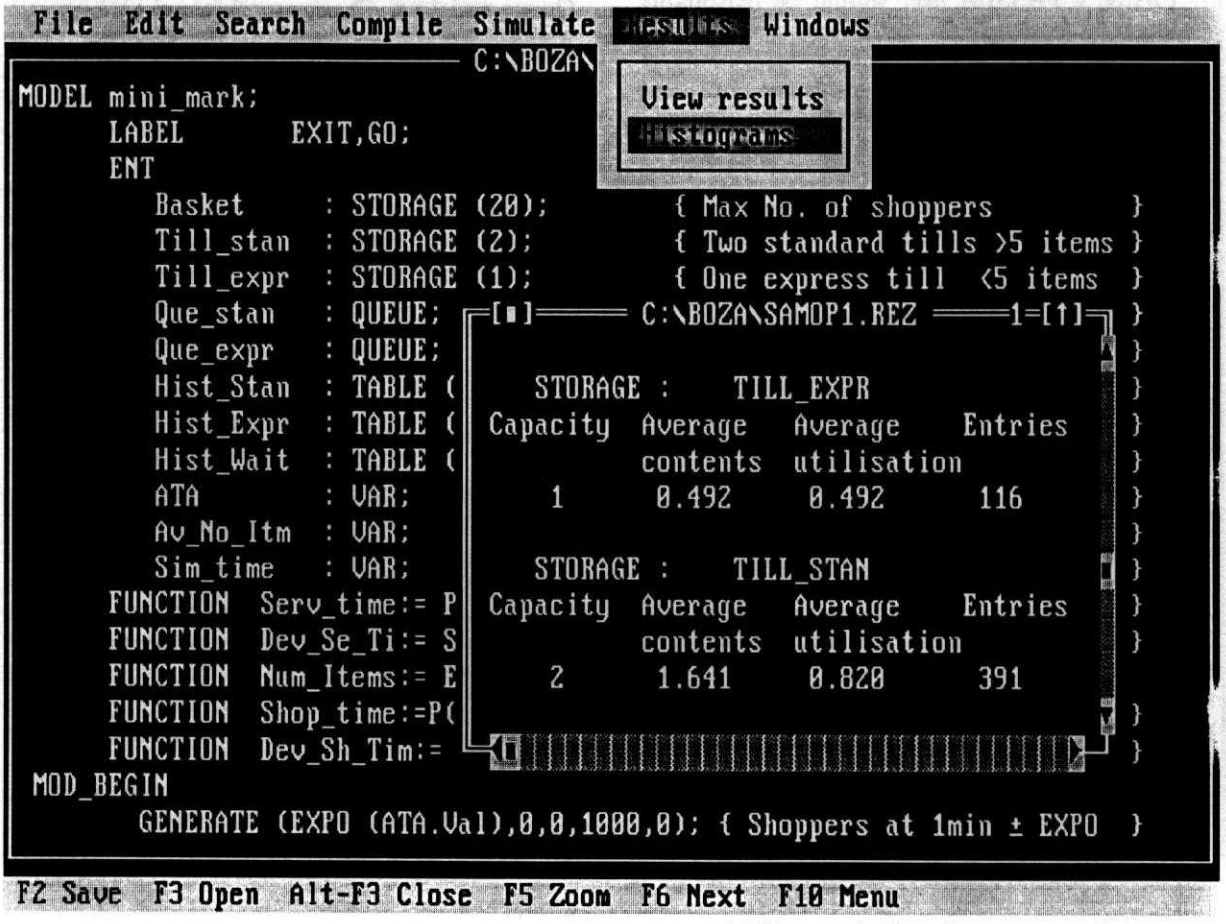

Fig. 2. A typical interactive environment screen of the ISDS system.

```
MODEL mini_mark;

      LABEL GO;

      ENT
            Basket          :STORAGE(20);      {Max. No. of shoppers        }
            Till_Stan       :STORAGE(20);      {2 standard tills >5 items   }
            Till_Expr       :STORAGE(1);       {1 express till <5 items     }
            Que_Stan        :QUEUE;            {Queue standard till         }
            Que_expr        :QUEUE;            {Queue express till          }
            Hist_Stan       :TABLE(1,1,10);    {Waiting at a standard till  }
            Hist_Expr       :TABLE(1,1,10);    {Waiting at the express till }
            Hist_wait       :TABLE(18,18,10);  {Time spent in the shop      }
            ATA             :VAR;              {Average arrival time        }
            Av_No_Itm       :VAR;              {Average number of items     }
            Sim_Time        :VAR;              {Simulation time             }

      FUNCTION Serv_Time    :=P(1)*40+120;     {Service time                }
      FUNCTION Dev_Se_Ti    :=Serv_Time/3;     {Deviation from service time }
      FUNCTION Num_Items    :=Expo(Av_No_Itm.val); {Number of items purchased }
      FUNCTION Shop_Time    :=P(1)*120+180;    {Time spent in the shop
      FUNCTION Dev_Sh_Tim   :=Shop_Time/4;     {Deviation of the above      }

MOD_BEGIN

      GENERATE (EXPO (ATA.Va l),0,0,1000,0);  {Shoppers at 1 min.±EXPO     }
      IF(Basket.S = Basket.R)                 {If no basket available      }
            THEN TRANSFER GO;                 {Leave the shop              }
      ENTER (Basket);                         {Take the basket             }
      ADVANCE (Shop_Tim,Dev_Sh_Tim);          {Do the shopping             }
      TABULATE (Hist_Wait,M1);                {Shopping time statistics    }
      IF Num_Items > 4 THEN
            BEGIN                             {Standard tills              }
                QUEUE (Que_Stan);             {Standard till statistics    }
                ENTER (Till_Stan);            {Come to a till              }
                DEPART (Que_stan);            {Pay and leave the queue     }
                TABULATE (Hist_Stan,Que_Stan.Q); {Save statistics for histogram }
                ADVANCE (Serv_Time, Dev_Se_Ti); {Time at a till            }
                LEAVE (Till_Stan);            {Leave the till              }
            END
          ELSE
            BEGIN                             {Express till                }
                QUEUE (Que_Expr);             {Express till statistics     }
                ENTER (Till_Expr);            {Come to the till            }
                DEPART (Que_Expr);            {Pay and leave the queue     }
                TABULATE (Hist_Expr,Que_Expr.Q); {Save statistics for histogram }
                ADVANCE (Serv_Time, Dev_Se_Ti); {Time at the till          }
                LEAVE (Till_Expr);            {Leave the till              }
            END;
      LEAVE (Basket);                         {Leave the basket            }
      TERMINATE (0);                          {Exit                        }
GO:   TERMINATE (0);
      GENERATE (Sim_Time.Val,0,0,100,0);
      TERMINATE (1);

MOD_END

CTRL_BEGIN

      Sim_Time.Val  :=28800;                  {Simulate 8 working hours    }
      ATA.Val       :=60;                     {Average arrival of 1 minute }
      Av_No_Itm.Val :=20;                     {Average purchase 20 items   }
      START (1);

CTRL_END

END_MODEL.
```

Fig. 3. A sample of ISDS program simulating a mini market (see text for details).

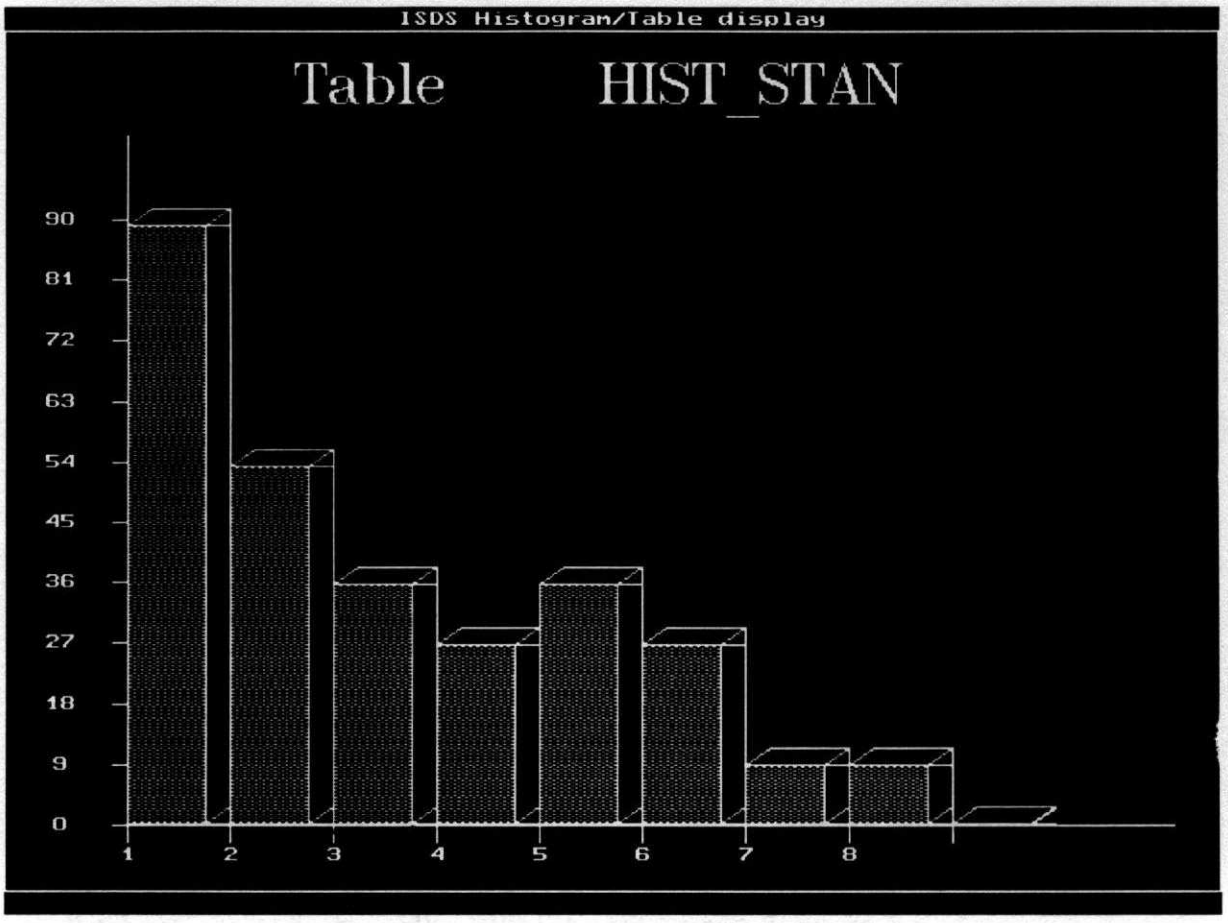

Fig. 4. A histogram produced by the ISDS graphical result-analysis tool.

ISDS PERFORMANCE METRICS

As soon as the 'zeroth' version of ISDS was finished and debugged back in 1991, as reported in [10], it was introduced in our teaching program. During the following academic year, we monitored the students' progress in accepting the new language and the merits of ISDS; both results were compared to those based on and accumulated over many years of GPSS practice. Encouraged by our preliminary findings we spent the following two years in accumulating relevant statistical data regarding the ISDS merits. The experiment was conducted by using two groups of students having similar learning and programming skills; the criterion used to form the groups was based on the past performance obtained from the Students' Record Department and a personal view of the relevant lecturers. The most 'talented' representatives from either group were used to form the "Pascal control" group. Each of the main groups consisted of at least forty students and the Pascal control group consisted of ten students irrespective of the main group sizes. One of the main groups was taught discrete event simulation by using ISDS and the other, the control group, covered the identical syllabus in GPSS/FON. GPSS/FON rather than the standard form was used in this assessment so as to eliminate the clear advantage that the ISDS's interactive environment would have over the standard implementation of GPSS. The Pascal control group was taught, in addition to its respective language, the event presentation, event chain management and other simulation concepts in Pascal.

We measured the following parameters:

- time needed to understand basic concepts;
- time needed to develop a simulation model;
- duration of the simulation process;
- ease of the simulation results analysis.

Evaluation criteria

A weighting factor, based on the following criteria was assigned to each student, irrespective of the group:

1. Average mark a_t taken over all previous subjects ($6 \leqslant n \leqslant 10$).
2. Average mark a_r taken separately over relevant subject like mathematics and programming ($6 \leqslant n \leqslant 10$).
3. Relevant subject lecturers' ranking a_l from 1 to 10.

A set of individual weights are assigned to each of the contributing factors; the assigned values are

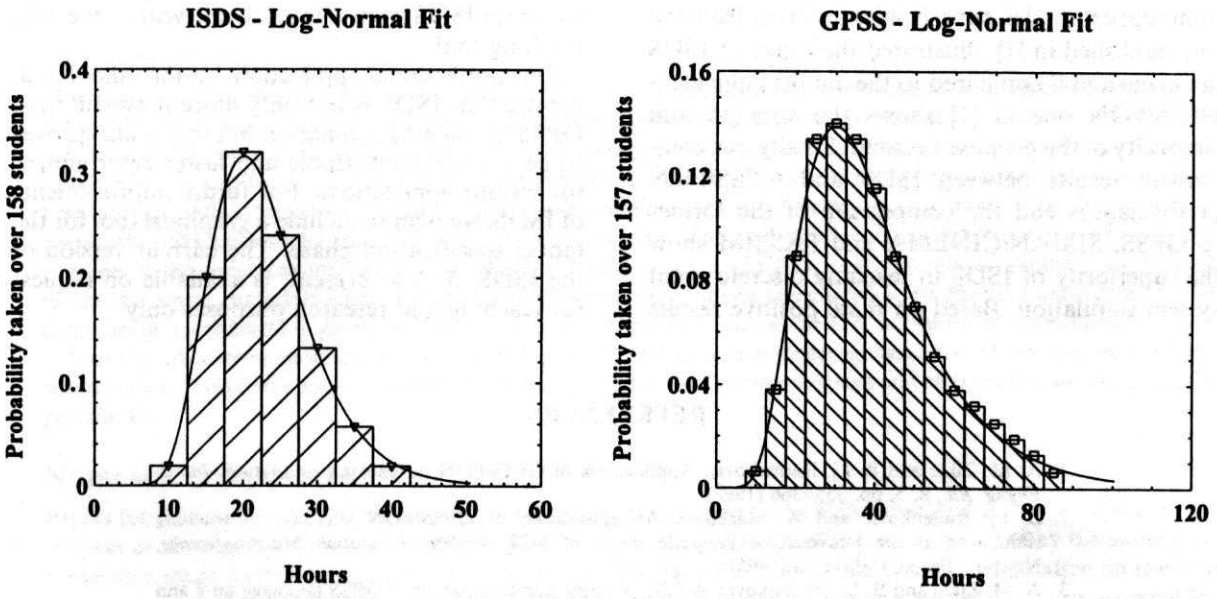

Fig. 5. Performance statistics of ISDS and GPSS groups at the compilation/debug stage.

based on the principle of importance and objectivity; for this purpose, the highest weight is assigned to relevant subjects and the lowest to the least objective, but still very important, lecturers' ranking. The weighting factor, w, is then calculated as:

$$w = 0.35a_t + 0.45a_r + 0.2a_l$$

It is assumed that a higher scoring individual needs less time to complete the same task then the lower scoring one. The time, in hours, needed to complete each of the 10 stages of the syllabus was multiplied by the weighting factor and the average time was then evaluated over the group size. The Pascal control group performance, based on the same criteria, was compared to the respective students' performance in their main groups. Our analysis shows that there is no significant difference in understanding the basic concepts of ISDS and GPSS. This result showed us that our criteria in the statistical analysis are very likely to be correct; as GPSS and ISDS share the same basic concepts, any other result would indicate a serious fault in our assessment strategy. As we hoped for, our further analysis showed that translation of the basic concepts into the respective codes took between 45–60% less time to the ISDS group than to the GPSS/FON group when writing models for identical exercises. Finally, the Pascal control group needed 300–400% more time to understand basic event scheduling concepts in Pascal.

The analysis shows that ISDS, compared to GPSS/FON, is:

- 30%–60% simpler at the model/specification stage;
- 40%–60% faster at the compilation/debug stage;
- 10%–20% faster at the execution stage;
- 3–5 times easier to use at the model testing stage.

The limiting values of the model-specification and compilation performance intervals (say 40%–60%) are determined so that 95% of all students' weighted performances fall below the average performance of the control group and that the mode and average values are both within the stated limits referenced to the control average.

The actual statistical distribution of performances, shown in Fig. 5, is of log-normal type and is fitted to the experimental data by using the strategy described in [11]. The ISDS performance distribution is described by the standard deviation of $\sigma = 0.3$ and the geometric mean of $b = 2$. The GPSS performance distribution is described by the standard deviation of 0.44 and the geometric mean of 36.44. By using these parameters, the average values are easily obtained, using $E[X] = b \times \exp(\sigma^2/2)$ [11], and are 23 and 39.66 for ISDS and GPSS distributions, respectively. The location parameters of both distributions are negative and negligibly different than zero. It is easily verifiable from Fig. 5 that more than 95% of all students in ISDS group fall below the 40 hour average of the control group and that the mode of 20 and average of 23 also fall between 40% (16) and 60% (24) of the same control average.

CONCLUSIONS

This paper has a goal to demonstrate the capabilities of ISDS language in teaching simulation of discrete event systems. ISDS has a number of features which help novices in a discrete event system 'environment' to start modelling very quickly. Our experience shows that once the new programming concepts are accepted it is a matter of days rather than weeks which is taken to become a GPSS (or in that matter any similar language) programmer. A typical example of the

mini-market model, superficially different from the one published in [1], illustrated the usage of ISDS language and if compared to the almost equivalent GPSS/FON one in [1] shows the strength and simplicity of the proposed system. Finally, the comparison results between ISDS and GPSS/FON performances and the comparison of the former to GPSS, SIMAN/CINEMA and PASSIM show the superiority of ISDS in teaching discrete event system simulation. Based on these positive results we adopt ISDS from this academic year as the only teaching tool.

Our professional application, on the other hand, proved that ISDS is not only more powerful than GPSS in teaching simulation but that it also proves to be a more comfortable and faster environment for serious applications. For future improvements of ISDS, we plan to include a graphical tool for the model specification phase. The current version of the ISDS, 3.15 at present, is available on request for teaching and research purposes only.

REFERENCES

1. A. M. Zikic and B. Lj. Radenkovic, Applications of GPSS/FON in teaching simulation, *Int. J. Engng. Ed.*, **8**, 5, pp. 355–366 (1992).
2. B. Lj. Radenkovic and A. Markovic, An application of GPSS/FON language in teaching simulation at the Universityof Belgrade, *Proc. of SCS Western Simulation Multiconference*, Newport Beach, California (1992).
3. A. M. Zikic and B. Lj. Radenkovic, A fully portable implementation of GPSS language on 8 and 16 bits microprocessors, *Proc. of the 7th System Engineering Conference*, Las Vegas, Nevada, pp. 773–779 (1990).
4. D. H. Ueno and W. Vaessen, PASSIM: A discrete-event simulation package for Pascal, *Simulation*, **35**, 6 (1980).
5. C. C. Barnett, Micro Passim: A discrete-event simulation package for a microcomputers using UCSD Pascal, in L. A. Leventhal(ed.), *Modelling and Simulation on Microcomputers*, The Society of Computer Simulation, La Jolla, California (1982).
6. B. Lj. Radenkovic, Interactive simulation system for discrete-stochastic simulation and it's implementation on mini and micro computers, Ph.D. Thesis, University of Belgrade, Yugoslavia (1989).
7. T. J. Schriber, *Simulation Using GPSS*, John Wiley and Sons, New York (1974).
8. G. S. Fishman, *Principles of Discrete Event Simulation*, Wiley Interscience, New York (1978).
9. J. O. Hernriksen, The integrated simulation environment, *Operations Research*, **31**, 6 (1983).
10. A. M. Zikic and B. Lj. Radenkovic, Interactive simulation system ISDS definition of a new language, *Proc 8th Int. Conf. on Sys. Engr.*, Coventry, pp. 260–266 (1991).
11. A. M. Zikic, R. I. Ristic and J. N. Sherwood, Three parameter distribution function fit to growth rate dispersion among small crystals, *Journal of Crystal Growth*, **158**, pp. 560–567 (1996).

Dr. Aleksandar Zikic is currently a lecturer and consultant at the University of Paisley, Scotland, at the Department of Electronics Engineering and Physics, a position he has held since 1986. Previous to this he was a lecturer, then consultant and lastly Head Designer. He started his career at the University of Belgrade, Yugoslavia, as a Junior Lecturer in 1974, having graduated from there with an equivalent to BSc (Hon) degree in Physics. He was granted his equivalent of MPhil in Scientific Instrumentation Design in 1978 and his PhD in Control in 1984, from the same University. Between 1980 and 1982 and during 1985 he was an invited researcher into self-tuning regulators and sigma-delta modulators at then Coventry Polytechnic. His interests are: real-time computer control of industrial processes and scientific experiments, mathematical modelling and computer simulation of analogue, discrete and discrete-event systems and development and implementation of numerical algorithms.

Dr. Bozidar Radenkovic is currently a Professor at the Faculty of Organisational Sciences, University of Belgrade, Yugoslavia. He was granted his equivalent of MPhil 1987 and his PhD in 1990 at the University of Belgrade in the area of computer simulation. From 1987 until 1990 he worked at the Faculty of Organisational Sciences as an assistant for computer simulation and simulation languages. Between 1990 and 1994, after becoming an Assistant Professor for computer simulation, he developed a version of the simulation system CSMP for continuous time system simulation and a portable version of GPSS for discrete event systems, both suitable for the minimum specification of 8-bit microprocessor-based computers. His primary areas of interests are implementation of simulation languages for discrete event systems simulation and optimisation and management of 'large structures' like industrial site harbours and steel and wood manufacturing/processing plants.