Proceedings of the 1985 Winter Simulation Conference
D. Gantz, G. Blais, S. Solomon (eds.)

A SIMULATION MODEL OF THE FAA'S FLIGHT SERVICE AUTOMATION SYSTEM

C. R. Spooner
A. Acampora
R. Regner

The MITRE Corporation
1820, Dolley Madison Boulevard
Mclean, VA, 22102

ABSTRACT

The Flight Service Stations of the Federal Aviation Administration are in process of being automated. To study performance, a simulation model has been built. To date, the model has been tentatively calibrated against the first version of the real system, and used to predict performance in a number of situations. The paper outlines the system being modeled, describes the model itself, and discusses some of the issues encountered during modeling. It then describes the calibration of the model, and the experience gained so far in its use. Future plans are outlined.

1. INTRODUCTION

The Flight Service Stations of the Federal Aviation Administration provide briefings to pilots, mostly over the telephone. Each call is answered by a flight service "specialist", who briefs the pilot on the weather for his proposed route of flight and then, depending on the intended flight discipline (visual or instrument flight rules), either records the flight plan lest a search and rescue become necessary, or forwards it to the en-route Air Traffic Control system.

Support to the specialist is currently being automated with what is termed the "Flight Service Automation System". A first version (officially entitled "Model 1" *) has been delivered; and operational site commissioning is due to begin in December of this year.

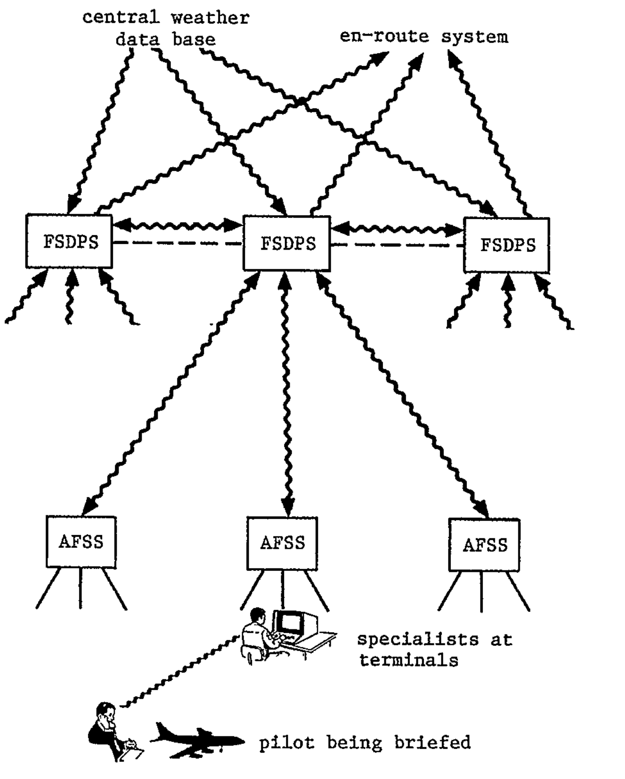
Because response times from the automated system had been a concern from the start, a simulation model was built of the hardware configuration and the proposed software design. This paper describes that modeling effort.

The simulation capability used for the modeling effort is based on a previous package developed at MITRE. It takes as input a model of the system being simulated and a specification for the load that is to be applied. Using this data, it performs a discrete-event simulation, then outputs statistics pertaining to resource utilization and response times. The system runs on a VAX 11/780 under the VMS operating system.

2. THE AUTOMATED SYSTEM

This section outlines the configuration for the automated flight service system, identifying features which are of interest for performance modeling. Figure 1 shows the overall configuration. Its general size and complexity is indicated by the fact that application software for model 1 alone exceeds 380,000 lines of source statements.

Computing capability will be concentrated in Flight Service Data Processing Systems -- FSDPSs for short. In model 1 there will be 13 FSDPSs, while in later models there will be 20 to serve all of the continental United States. Communication lines connect the FSDPSs to each other, to neighboring air-traffic control centers, and to central sources of weather. The weather sources will be the National Weather Service in model 1, and an FAA facility (a proposed Aviation Weather Processor) plus live radar data in later models.



FSDPS = Flight Service Data Processing System

AFSS = Automated Flight Service Station

Figure 1: Overall View of Flight Service Automation

----

* Note that the word "model" is used in two different senses throughout this paper.

Each FSDPS will remotely service a number of Automated Flight Service Stations (AFSSs for short) -- up to 6 each in model 1, up to 11 each in later models. Each AFSS will locally service a number of specialist terminals -- up to 35 each in model 1, up to 48 each in later models.

In model 1 the specialist's interaction with the automated system is alphanumeric only, and consists almost entirely of requests from the specialist followed by responses from the system. Much of the input is typed into skeleton formats placed on the screen in response to previous requests. Apart from these skeleton formats, the principal responses from the system are weather briefings and flight plan information.

The AFSS computing power is used mainly for multiplexing, store-and-forward and low-level terminal driving. There is, however, enough storage to hold several screen images per terminal. All specialist input is forwarded to the FSDPS for decoding and servicing. The FSDPS holds a national weather data base, plus a flight-plan data base for flights of local concern.

Figure 2 depicts the subject of the model, namely a single model-1 FSDPS and its AFSSs. It shows the hardware which is of interest for performance modeling. The specifications call for fail-safe operation, which implies redundancy and dynamic reconfiguration. The modeling, however, concentrates on stable states (both normal and fail-safe).

Each AFSS has a number of DEC LSI 11/23 processors. Two of the 11/23s are "communication processors". They handle communication with the FSDPS, and each has a disk for storing the screen images. The remainder are "terminal processors", with up to 8 terminals apiece in normal operation. The FSDPS consists of a 3-CPU Tandem Non-Stop-II system. Each CPU has two disks, but only one is modeled (the other being for back-up).

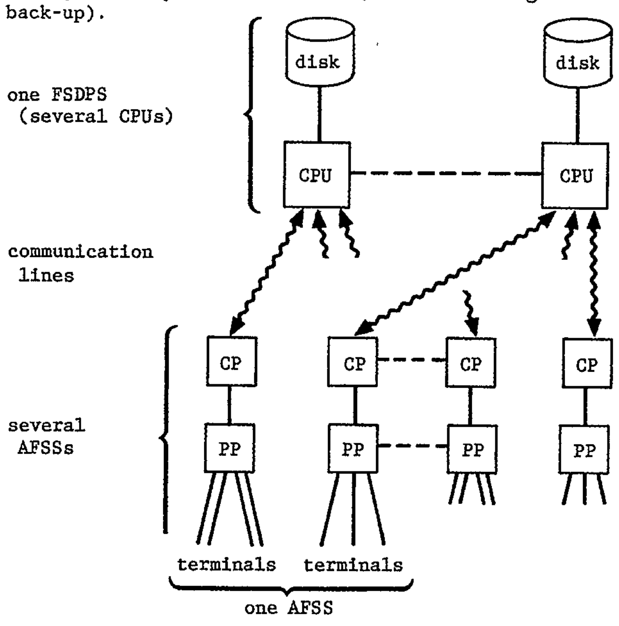


Figure 2: The Subject of Modeling

Connections between AFSS and FSDPS are of interest to performance. Each communication processor in an AFSS is connected to one CPU in the FSDPS, while each CPU in the FSDPS can be connected to several communication processors. On a CPU failure in the FSDPS, affected traffic is automatically redirected to a back-up CPU -- through the alternate communication processor in each AFSS, and hence over alternate lines. Other than that, the connections are permanent. Clearly the communication lines have to be configured to take into account various possible failure situations.

Software on both types of computer is organized as permanent processes (in other words, as a fixed number of processes). In the AFSS, a process serves one transaction at a time, in the main, to completion. In the FSDPS, the system provides each process with a data area. Application-level routines are available to partition the data area so that the process can serve multiple transactions; but only up to a fixed maximum. Such a process is said to have multiple "threads".

One further point is of interest to modeling. The contractor opted against dynamic load balancing. Instead, most of the processing of a transaction is performed in the CPU to which the communication line is attached. Moreover, much of the processing that is independent of a particular specialist terminal is performed on dedicated CPUs -- for example, flight data base update. Here again there is scope for optimizing, this time in the assigning of software to CPUs.

3. THE MODEL

3.1 Contents of the Model

An overview of the model is given first, to convey an idea of the level of detail that is modeled. Essentially, the model contains three descriptions. It describes:

1) the resources in the configuration being modeled

2) the load, or traffic, that is to be directed at that configuration. The traffic is composed of transactions which are referred to in the flight service context as "messages"

3) the paths, from resource to resource, along which the individual messages are to travel.

3.1.1 Resources. Hardware resources that are modeled are essentially those that were identified in the previous section (see figure 2), namely:

in the FSDPS:	3 CPUs
	3 disks
communication lines:	12 from AFSS to FSDPS
	12 from FSDPS to AFSS
in each of 6 AFSSs:	2 communication processors
	2 disks
	from 2 to 6 position processors
	up to 35 terminals

Apart from the configuration of the communication lines, the important hardware variables are the numbers of each type of element. Those given here (3 CPUs, 6 AFSSs, etc) are fpr a typical simulation run.

The throughput capacity of the actual terminals is not an item of interest. The reason why they are modeled as resources is explained later. Memory is not modeled; nor are communication lines other than to the AFSSs. Both were assumed to be adequate and hence not modeled.

Since the software processes can only service limited numbers of messages at a time, it is appropriate to model them as resources -- in the multi-threaded case as multi-server resources. In each AFSS processor, 6 processes apiece were modeled. In the FSDPS, the operating system was modeled with 1 process per disk, and 1 process per communication line; and the application was modeled with 6 single processes, and 8 which were repeated in each CPU.

To model interrupts would take the modeling to a finer level of detail, timewise, than was desired. Therefore the individual CPU requests were increased by a constant ratio to account for mid-execution interruptions -- with appropriate adjustments to the utilization statistics.

Figure 3 shows an edited extract from the model, showing the declaration of various resources. Text in quotes is comment, and has been added to explain material that may not otherwise be self-evident.

```
constants  #OfCommunicationLines = 12
           #OfCPUs = 3

resources  CPU (#OfCPUs)  EachServes 1

           "meaning a set of CPUs, which will be
           "referred to as 'CPU (1)', 'CPU (2)', etc

constants  FirstCPU = CPU (1)
           LastCPU = CPU (#OfCPUs)

resources  disk (FirstCPU, LastCPU)  EachServes 1

          "meaning a set of disks, one corresponding
          "to each CPU in the range FirstCPU to LastCPU

constants  FirstDisk = disk (FirstCPU)
           LastDisk = disk (LastCPU)

resources     "these resources are software processes"

   DiskProcess (FirstDisk, LastDisk)  EachServes 50

   Envoy (#OfCommunicationLines)  EachServes 7

   AFSIO (FirstCPU, LastCPU)   EachServes 15
   GDBED   serves 6
   ApplicationBackgroundProcess (FirstCPU, LastCPU)
                                             EachServes 1
```

Figure 3: An edited extract from the model

3.1.2 Load. The modeling package recognizes message types; and it recognizes "conversations", which are sequences of messages and responses on a particular topic between specialist and computer. For example, a weather briefing on behalf of an individual pilot represents such a conversation. The model describes both the message types, and the allowable sequences for conversations. The modeling package injects instances of each type of conversation into the simulation, at frequencies specified in the model; and as each message in a sequence completes, it injects the next message.

The modeling package also recognizes two classes of conversation: essentially a stable background load, plus a foreground load that can easily be altered from run to run. Conversations in the stable part have their inter-arrival times specified individually in the model (as constants, or as variable expressions which can include random functions). Conversations in the adjustable part of the load are specified as proportions of the total adjustable load. The proportions are stated in the model; the total is input as a parameter to each simulation run. From the parameter a mean frequency is derived for each message type; the mean is in turn used for generating the random distribution of arrival times for that message type.

The flight service model uses the adjustable part for the load from the AFSSs, and the stable part for the background of weather updates from the central weather source. This has made it easy to model the effect of increasing work loads in the flight service stations, while the background remains constant. The load from the AFSSs is modeled with 12 types of conversation, comprising 19 message types. The load from the central weather source is modeled as 7 types of degenerate conversation (that is, conversations containing a single message and no response).

An interesting problem arises when the system is overloaded. In real life, the throughput of a system is governed both by internal factors (the availability of resources) and by external factors (the number of terminals and the speed of their operators). The external factors place an upper limit on what the internal resources must handle: after a certain point, overloads queue for terminals rather than for internal resources. If we use a statistical algorithm for generating the load, without any restraining factor, then the load will have unrealistic temporary peaks; and these peaks can affect internal performance -- unrealistically. To model real life in this respect, the terminals are treated as resources. Conversations are generated statistically; but the first thing a conversation does is to acquire a terminal -- and during a peak in the load it queues for the next available terminal (hence tending to flatten out the peak, as in real life). While a conversation holds a terminal, the speed of the real-life operator is modeled by a forced delay between each pair of successive messages. The delay is achieved with a dummy "think-time" message type.

Two further problems may be of interest to other modelers. Firstly, incoming calls to the flight service stations automatically hunt for a free line. That is, the telephone system selects a terminal from a given set of terminals; if one is busy, then others will be tried. This raises the question of how, in the model, a terminal should be selected for each new conversation. The question is important because on the answer to it will depend which communication line, which CPU in the FSDPS, etc, are to serve the messages in the conversation. The solution is to model the terminals in sets, as multi-server resources corresponding to the sets of terminals from which the real telephone system makes its selections. The conversation selects and requests a terminal set (this is programmed in the model), then the modeling package (emulating the telephone system) selects the individual terminal within the set.

The second problem is more unusual, but may still be of interest to modelers. In the simple case we wish to select all terminals equally. To do this we make the choice of terminal set random, but weighted by the sizes of the sets. Sometimes, however, we do not want equal loading. This may be a reflection of a real-life situation, or it may be an artificial result of some modeling requirement.

As an example of the latter, the performance specifications call for a maximum AFSS (with a stated load), a maximum total load, and a maximum number of terminals. The three maxima are such that it is impossible to observe all three in the same simulation run, and at the same time to load the terminals equally. If a maximum AFSS is to be modeled in the same run as the maximum number of terminals, then the terminals outside the maximum AFSS must be run at a lower loading. Otherwise the maximum total load would be exceeded.

To provide for unequal loading, we also arrange the terminals in groups (each group being one or more multi-server sets). Within a group, terminals are selected equally, as described above; but each group is allocated a proportion of the total load, and the random choice of group is weighted by those proportions.

3.1.3 Message Paths. Each message type is associated in the model with a message path, which is a procedural description of the processing of messages of that type.

While each message type has a unique top-level path, much of the detail is delegated to common supporting procedures *. An example of this is seen in figure 4, which depicts an extract from the processing of one message type. Shaded areas represent software processes. Figure 5 shows the equivalent text as presented to the modeling package; here again the comments in quotes are for the benefit of the present reader. (The prefixes "FR." and "M." are explained later).

3.2 Organization of the Model

When the material for a model has been collected, it has to be organized as text which can be read by both man and machine. Importance was attached to this step because good structuring of the text can make a model easier to get right, easier to understand, and easier to adapt. In this section we discuss some of the issues that were encountered in attempting to achieve good organization.

3.2.1 Understandability. We have already observed that the material of the model divides naturally into three parts -- resources, load, and message paths. This suggests strongly that for ease of understanding it be structured into three separate areas -- and presented top-down inside each area. In particular, the separation of the external load from the internals

```
" ** extract from top level procedure ** "

StartInitialThreadIn (FR.AFSIO)

   TransmitAFSSToFSDPS (M.CharactersIn)
   UseCPU (.15)
   HistoryRecord

   call (FR.WXRTV)
      UseCPU (.2)
      DoTimes (3)
         UseDiskForUnstructuredFile
         EndDoTimes
      ReturnTo (FR.AFSIO)

   AFBAKCheckPoint
   EndThread

" ** typical supporting procedures ** "

HistoryRecord = WithChance (.15)
                      UseDiskForUnstructuredFile
                      EndChance
                   EndOfSequence

TransmitAFSSToFSDPS (CharactersIn) =
      ... etc ...
   call (FR.Envoy)
      UseCPU (EnvoyOverhead)
      ReturnTo (FR.AFSIO)
   EndOfSequence

AFBAKCheckPoint = call (FR.AFBAK)
                        UseBackUpCPU (.03)
                            ... etc ...
```

figure 5: A typical message path

of the system is healthy -- though not always easy to accomplish.

While organization along these lines was attempted and was in the main successful, it was frustrated by a language deficiency imposed by the modeling package, namely that words have to be defined before they are used. Because of this restriction, material from each topic has to be placed up-front, in the first section; and top-down reading within a section has to follow a zig-zag path which is bewildering to the first-time reader.

Nevertheless, in spite of these frustrations, it was possible to order the material in a reasonably intelligible way. The proof lies in the ease with which new members of the modeling project found their way around the model.

3.2.2 Adaptability. To achieve an adaptable model, one has to identify those features which are likely to

* At this level of description, we do not distinguish between procedure calling and macro substitution.

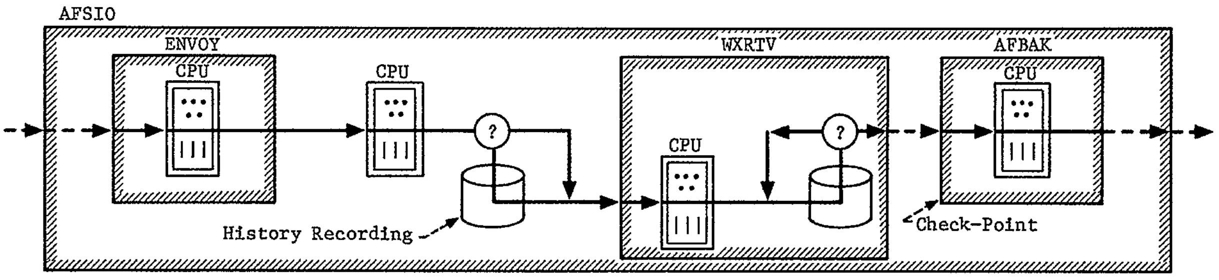


Figure 4: Message Paths--Excerpt from the Message Type "Area Forecast Request"

be altered between simulation runs, and program into the model the side effects of altering them. The number of CPUs in the FSDPS is a case in point. To change from three to two has ramifications throughout the real system, and hence throughout the model. However, the model is so organized that only the constant called "#OfCPUs" has to be changed, plus a small number of constants that define such things as allocation of processes among CPUs. Support procedures within the model, together with a cascade of constants defined in terms of previous constants, take care of the rest. Figure 3 offers a glimpse of this.

The extract in figure 5 illustrates another adaptability issue. Instead of "call (FR.WXRTV)", one could merely have used the resource-requesting primitive "get". During the modeling, however, it was not clear exactly what goes on in the real system when a process is entered. Therefore it was important that the model be easy to change in this respect. "Call" is a support procedure in which one can model process-acquisition centrally. The model abounds with examples of this kind of deliberate centralization.

The question of selecting which resource to acquire provides an example of both these issues. Many of the resources exist in sets (several CPUs, several copies of each process, etc); and for the individual message the question arises: which member of each set should it use? In the real system, the choices are all "hardwired" once the terminal is identified; but this is because real configurations are fixed -- whereas we want the model to be easily adaptable from configuration to configuration. In the model, therefore, multiple resources are declared generically, as arrays of resources.

This leaves it to the model to choose dynamically which element of each array to use. Rather than have it do so in a large number of places, the choice is made once for each conversation. To make the choice, each conversation starts with a dummy message type, whose message path takes it through a section of support procedures whose sole purpose is to select resources. In particular, all the complications concerning terminal groups and terminal sets, etc, are concentrated in this one section -- so that changing the configuration of terminals and communication lines (which could so easily have been an exceedingly complicated task) merely involves altering numbers in a small table.

Having chosen its selection of resources, a conversation must then carry this choice through all the component messages. Likewise, there are message-dependent values that have to be carried through the duration of a message. This raises a language issue. Textually scoped variables do not provide for this; global variables and parameters are both clumsy ways to do it. The solution is to have a type of variable which is textually global, but is visible only to a particular instance of a conversation or a message. Thus "FR.WXRTV", in figure 5, means "the copy of WXRTV for the current conversation". The particular lexical form is open to discussion; and there are other ways to provide such variables. The important point is that a neat solution to this language problem proved invaluable to good organization.

<u>3.2.3 Language.</u> Language and model were developed concurrently. The original language had few of the features usually found in programming languages. In so far as time allowed, features were added as they were found to be needed. The insight gained in the process may be of interest to those involved in designing or evaluating simulation languages.

Features which proved particularly helpful included:

1) treatment of resources as a data type, in particular:

    arrays of resources
    arrays indexed by resources
    constants of type "resource"

2) convenient constant definition, in particular:

    cascades of constants defined in terms of each other
    tables of constants

3) message-dependent and conversation-dependent variables (as discussed above)

4) the use of procedures to layer the material.

The two most serious points which could not be corrected in the time available are common to many programming languages. The first is a readability issue. Without keyword parameters it is hard to make the purpose of a parameter clear in the calling statement. For example, the meaning of "UseCPU (duration 10, priority 3)" is reasonably clear, whereas the meaning of "UseCPU (10, 3)" is by no means obvious to the uninitiated. The second weak point is the define-before-use restriction, whose devastating effect on structure we have already seen.

<u>3.2.4 Discussion.</u> Attempting to organize a model could easily lead one into a vicious circle. To make it intelligible, one layers the model in levels of abstraction. To make it easily adaptable, one builds extra logic into the supporting layers. These extra layers add to the size and complexity; if the model is to remain intelligible and easily adaptable, it will then have to be further layered, with more supporting logic; and so on.

The present project showed that this vicious circle can be broken -- by having a language in which sufficient attention has been paid to readability and organization. The model is some 5000 lines in length. It could have been shorter, at the expense of being less adaptable (ie with more of the decisions hard-coded). Much of the length is indeed in the lower-level supporting material; but it was possible to organize that material in a manner which was intelligible and which preserved the adaptability. The evidence on understandability has been mentioned. The evidence on adaptability will be presented in a later section.

<u>3.3 Calibrating the Model</u>

Advance documentation about the real system was of limited help for modeling. It suggested a general shape for the model, that is, the sequence of resource requests in each message path; but the associated numbers (CPU times and loop-iteration counts) were dubious. They were early estimates that, as one expected, proved in due course to be optimistic. Hence the model could not be useful for monitoring the design of the software.

Once the first version of the real system was available, however, it became possible (in theory) to take measurements on it, and hence to calibrate the model for studying subsequent problems. In practice, some limited measurements were available from the performance demonstration; and further measurements are planned in the expectation that a system will be available for measurement purposes.

Here we discuss calibration within the FSDPS, which is the area of greatest interest. Figure 6 shows part of a "calibration matrix" in which there is one column for each process, and one row for each message type. Each message type visits some but not all of the processes. Corresponding to each process that it visits is a cell in the table for which we need numbers. Such cells are marked with a "?" in the figure. The numbers that we need in that cell are CPU times and disk-access counts.

The Tandem operating system provides a measuring facility which will give these numbers. Unfortunately it was not possible, at the performance demonstration, to collect the statistics needed for modeling. However, a subset was obtained. With these a partial calibration was performed, in anticipation of a more comprehensive calibration in the near future. The simulation runs described in the next section were obtained from the partially calibrated model.

The statistics from the performance demonstration came in three categories. Corresponding to each row (ie to each message type), there was a mean response time; corresponding to each column (ie to each software process), there was a total CPU time across all relevent message types. These two sets of figures can be thought of, in a broad sense, as "cross totals" -- though their derivation from the individual numbers in the row or column is anything but linear. Disk-access counts were an even broader total -- by CPU.

The first task was to instrument the model to output numbers corresponding to those measurements. Some were already being output by the underlying modeling package; the output of the remainder had to be programmed into the model. Then followed a series of simulation runs in which the numbers output by the model were compared against the measured numbers, and the calibration adjusted. After some 30 runs, the numbers had converged (figure 7) to give close agreement (10% for disk counts, CPU times and the overall mean of response times; 30% for the worst-case message type).

A model that can be relied upon to give agreement as close as this in all cases is usually considered very good. For the flight service model, however,

message types \ processes	AFSIO	FLITE	etc	Response Time (Seconds)
Request Flight Plan	?			1.68
Enter Flight Plan	?	?		2.42
Detailed WX Request	?	?		1.98
Area Forecast Request	?			2.43
etc.				
% CPU used	40.75	7.06		

Figure 6: Calibration Matrix

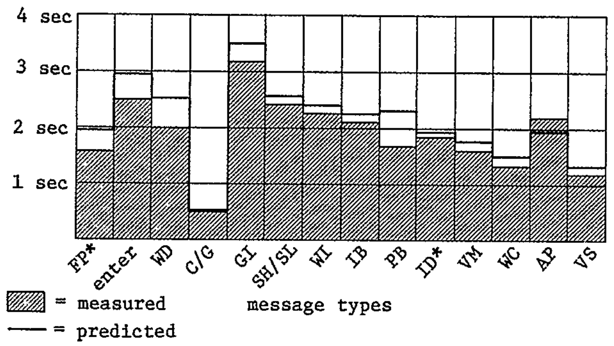


Figure 7: FSDPS Response Times

agreement was only proven in one case. Moreover, there were grounds for uneasiness about the calibration. In the exercise just described, an infinite number of solutions could fit the available data. While the "cross-totals" were in good agreement with measurement, there was no hard data for the component numbers in the individual cells. A large amount of intuitive judgment filled the gap.

The proper solution is, of course, to obtain the required measurements. Discussions for doing so are currently underway. However, it was of interest in several respects to test the partially calibrated model. If the partially calibrated model predicted well under a variety of circumstances, then prudent use of it could be beneficial. There was no guarantee as to when a full set of measurements would be possible. If they are destined to be never available, then we have a situation which is typical in the modeling world -- predictions are needed before any confirming real-life data is available. It is in the light of these considerations, then, that the simulation runs described in the next section are presented.

4. USING THE MODEL

This section describes how the credibility of the model was established, then discusses selected results obtained by using the model and the potential impact of those results on the FSAS program. Finally, observations concerning the model's ease of use are given. Note that these simulation runs were done after only a preliminary calibration had been completed (as described in section 3 above). AFSS portions of the model had not been calibrated at all.

4.1 Establishing Credibility

Establishing credibility in a simulation model includes ensuring that its behaviour is reasonable over a wide range of inputs. As with any other computer program, "bugs" may remain hidden beneath the surface, waiting for events such as array out of bounds, division by zero, etc. It is thus necessary to stress the software not only to test modeling assumptions, but also to uncover programming and logic errors.

4.1.1 Varying The Input Load. The preliminary calibration gave confidence at only one input workload: 2.25 transactions per second (tx/sec). A series of twenty experiments with loads ranging from 0.5 to 3.1 tx/sec were conducted. The results are shown on figure 8. Note that these are weighted mean

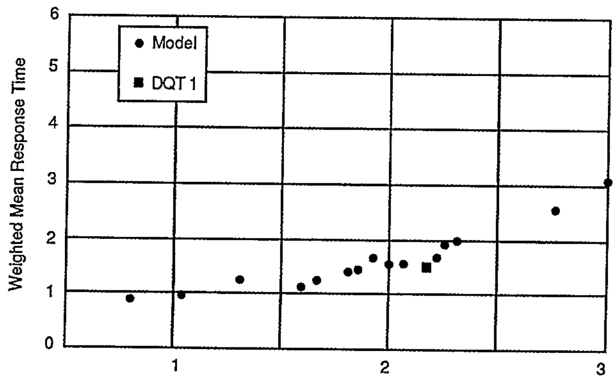


Figure 8: End-To-End Response Times Versus Load

response times at the FSDPS. All runs were of such a length that the total volume of specialist transactions was approximately 8000 (this sample size produced a standard deviation of 0.35 between respective runs at the same load).

The model predicts, for example, that if a 3-CPU FSDPS is subjected to a standard background load of weather network message processing and an average rate of 2 tx/sec from all specialist terminals connected to that FSDPS, then the average time for completing specialist transactions would be 1.6 seconds.

The model also predicts a response time lower limit of about 1.0 seconds. At the upper end of the range the mean response time is around three seconds, but one should not therefore conclude that all specialists will experience a mean 3-second response time at 3.0 tx/sec. Indeed, data from the same set of runs predicts widely varying response times for specialists at CPU 0 compared to those connected to CPU 2. (Refer to figure 9). Thus an acceptable overall mean can mask quite unacceptable response from a particular CPU. In the same way, a mean across a single CPU might mask intolerable means for individual terminals on that CPU -- so the model runs just quoted have to be seen as only the first step. (In passing, we note a specification issue which this raises: whether one should specify a maximum for the overall mean response time, or the mean response time which no terminal taken by itself shall exceed).

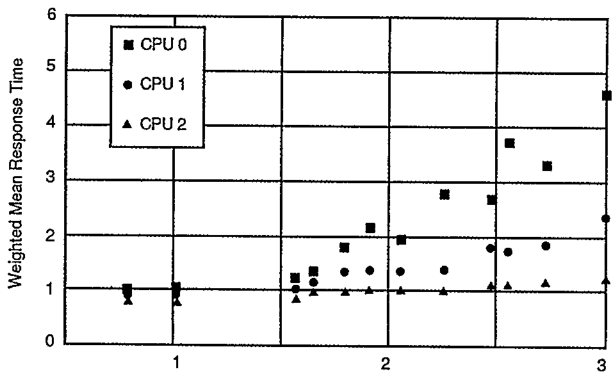


Figure 9: FSDPS CPU Response Times Versus Load

4.1.2 Varying the Configuration. The acceptance test on which the partial calibration was based was in fact the first of two such tests, performed in November 1984 and May 1985. They are referred to respectively as DQT1 and DQT2 (DQT stands for "Design Qualification Test"). Both the preliminary calibration and the runs discussed here were conducted in the 6-month interim between DQT1 and DQT2.

The runs described so far were based on the configuration used for DQT1. A series of simulation experiments then followed in which the configuration itself was varied. The primary purpose of these runs was to confirm that the model can be expected to provide reasonable results over a wide range of configurations. However, since the real-system configuration was in a state of flux (the contractor was meanwhile investigating alternative configurations for DQT2) it was of interest to see how well the model could suggest improvements. Clearly an opportunity existed, in theory at least, in which third-party simulationists could provide alternative system configurations to the design engineers and vice versa. While external constraints made such collaboration impossible, the exercise of using the model in this role was a valuable test of its potential usefulness.

An example of a performance improvement which was uncovered independently by the contractor's engineers on the test floor and by the simulation model was a process location swap. A particular process, Service B Transmit ($SVCBT), accounted for approximately 17% of CPU 0's total processing power. (Service B is a communication line to the other FSDPSs and En Route Air Traffic Control Centers.) Figure 10 shows response times when $SVCBT is relocated to CPU2. Response times for specialists at CPU 0 are reduced by over 30%, while those for specialists connected to CPU 2 remain reasonable.

This performance enhancement was based on optimizing the use of existing hardware/software. It was also of interest, however, to examine proposals for adding or replacing hardware. Whereas it would have been a major effort to experiment by upgrading the actual hardware, it was a relatively simple matter to construct models to simulate these alternatives.

For the real system the question had been raised as to whether to add a fourth TNSII CPU or to replace the three TNSIIs with three TXPs, the latest Tandem CPU. The versatility of the model was tested by applying it to that question.

Figure 11(a) shows the effect of adding a fourth CPU to the DQT1 baseline and moving eight terminals from CPU2 to CPU3. This gave only a slight improvement in response times, as compared to the DQT1 baseline. However, given a fourth CPU (CPU 3), swapping a line on CPU3 with a line on CPU0 reduces turnaround time for specialists connected to CPU0 by a factor of two (see figure 11(b)).

The model suggests, however, that replacing the TNSIIs with TXPs will yield even better performance results. Figure 12 shows a 40% further reduction in response time for CPU 0 when compared to the load-balanced 4-TNSII CPU configuration. The relatively flat slope of these curves indicates minimal utilization for all three CPUs at these input workloads. This was further verified by increasing the simulated TXP speed from 2.5 to 3.0 times that of TNSII: the reduction in response times was insignificant.

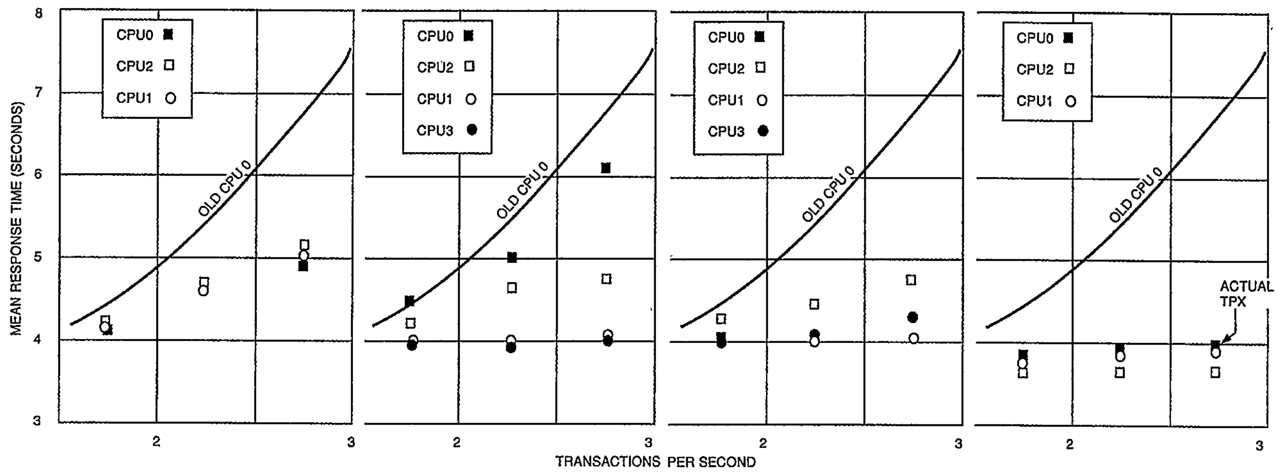


Figure 10: Performance Enhancement By Process Swap

Figure 11a: Four FSDPS CPUs With No Load Balancing

Figure 11b: Four FSDPS CPUs With Load Balancing

Figure 12: TNSIIs Replaced By TXPs

Along the way, the contractor obtained three TXPs on a trial basis, and provided a datapoint to the FAA. The circled point on figure 12 is an actual TXP measurement of the weighted mean response time for all three CPUs. The close agreement between measurement and model provides further evidence of the model's credibility over a range of configurations.

Other experiments for examining alternative configurations included:

- o reducing the number of AFSS-FSDPS communication lines from 12 to 6.
- o reducing the number of AFSS-FSDPS communication lines from 12 to 3. (The number of AFSSs was also reduced to 3 while maintaining the same number of specialist terminals).
- o examining the effect of keeping a frequently used FSDPS data structure (Flight Plan Mask) in main memory instead of on disk.

These experiments likewise yielded results that appeared reasonable, thereby increasing one's confidence in the model.

Meanwhile, the contractor experimented with connecting all AFSSs to two CPUs, reserving the third CPU for the periodic weather database updates and the transmitting of flight plans. A new configuration baseline was submitted and approved by the FAA for DQT2. A simulation model was then constructed and three simulation runs completed. The results shown on figure 13 show that response times for specialists connected to CPU 0 are much reduced when compared to the DQT1 arrangement (figure 9). However, simulation experiments with a failed communication line or an FSDPS CPU failure (figures 14 and 15 respectively) predicted unacceptable response times except at the lowest loads. These predictions were confirmed at the subsequent DQT.

4.1.3 Summary of Model Credibility. These "What

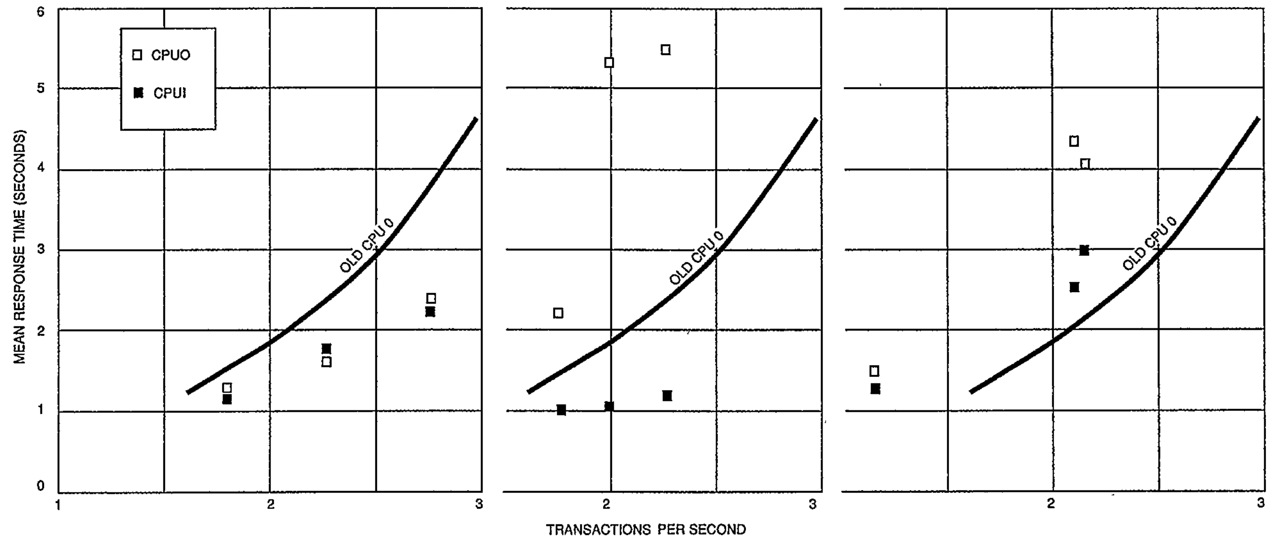


Figure 13: DQT2 Configuration

Figure 14: DQT2 Configuration With Comm. Line Failure

Figure 15: DQT2 Configuration With CPU2 Failed

if...?" runs showed that variations in predicted response times are "reasonable", as the configuration and/or load are varied -- that is, they follow curves that are of a kind and shape that one would intuitively expect. Moreover, at several points in this load/configuration space, real data has become available, and has matched well with predictions. While proper calibration on complete data is clearly still very desirable, the partially calibrated model emerged from the "What if...?" tests with considerable credit.

4.2 Practical Use Of The Model

In addition to the credibility-establishing experiments discussed in the preceding paragraphs, the model was used in a trial production mode to predict response times for the actual fielded system.

Model 1 FSAS is planned to have 13 FSDPS installations located throughout the continental U.S. Current plans are for the DQT2 hardware and software arrangement to be utilized, adapted as necessary for the number of AFSSs and terminals. Eleven of the thirteen sites will have three AFSSs, whereas the remaining two (Seattle and Miami) will have two.

The FSAS loads used in this study were projections for the year 1990 peak hour. These range from 1.05 tx/sec at Seattle to 2.24 tx/sec at Washington. The runs showed that all sites ,including the busiest (Washington, D.C.), met performance requirements when fully operational.

Experiments were then designed to investigate performance in the fail-safe mode. CPU 2 was failed at the following FSDPSs :

- o the one with highest projected load (Washington, D.C.)
- o the one serving the AFSS with the most terminals (Miami,Fl.)

Figure 16 shows the fail-safe mean response time results for the above two sites at three loads apiece (the 1990 peak hour load plus or minus 30%). Miami

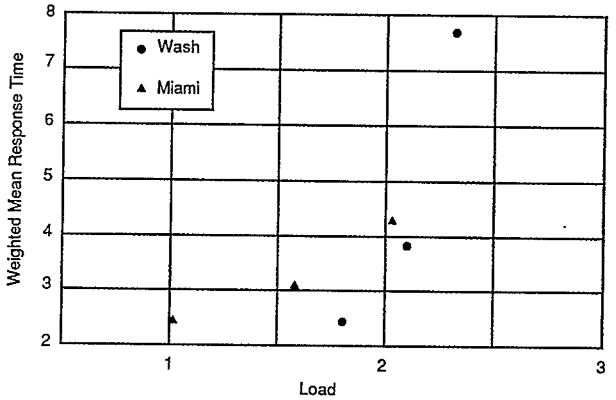


Figure 16: Fail-Safe End-To-End Response Times For Miami And Washington

FSDPS is lightly loaded and has no fail-safe response time problem; but Washington exhibited a significant response time increase when in the fail-safe mode. Experiments for other fail-safe configurations for Miami and Washington were also conducted, predicting end-to-end response times shown on figure 17. (In this figure, the X-axis shows the average utilizations of CPUs 0 and 1).

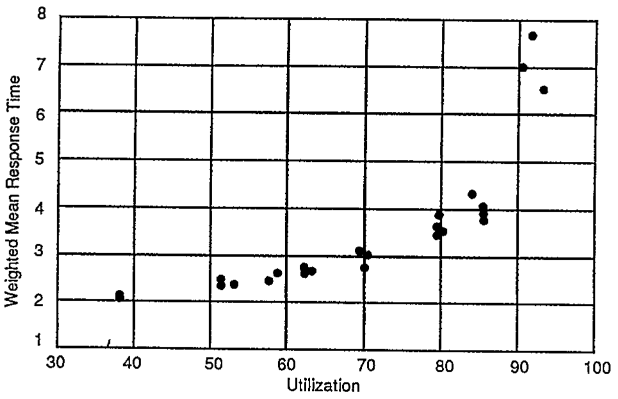


Figure 17: Fail-Safe End-To-End Response Times As Function Of CPU Utilization

The model was then used to investigate ways to improve the Washington site's performance. Figure 18 shows fail-safe response predictions when terminals on CPU 1 are gradually transferred to CPU 0 in an attempt to achieve even utilizations in the two CPUs. The upper curves show end-to-end response times for specialist transactions, whereas the lower curves represent the FSDPS's portion only. Figure 18a shows the unoptimized fail-safe response times (a 113% increase over normal operation). Moving 8 terminals from CPU 1 to CPU 0 (Fig 18b) gave an 8% improvement; moving only 4 terminals (Fig 18c) decreased response times 15%.

The difference between the upper and lower curves in figure 18(a,b,c) indicates queueing outside the period recorded as "FSDPS response time"; and inspection of Model output revealed excessive queueing for threads in the process which handles AFSS communication. (This process is in the FSDPS but for calibration reasons the first part of its operation was excluded from the FSDPS measured period.) By increasing the number of available threads from 15 to 20, queueing decreased significantly, as seen in figure 19, and end-to-end fail-safe response time was reduced 21%.

4.3 Impact Of Model On FSAS Program

DQT2 was conducted in May 1985. The performance tests at DQT2 showed that the FSAS equipment did meet the full operation requirements but did not meet the specified response times while in fail-safe mode. This was in agreement with the simulation model's predictions.

In regards to the six month interim between DQT1 and DQT2, a much greater impact could have been achieved had the simulation team been supplied with current-day resource utilization updates (this would of necessity have had to come from the design engineers).

What this series of runs demonstrated is how useful the model could be if two-way communication is open between designers and modelers. Instead of using the actual system with many SYSGENS and physical reconfigurations of cables and plugs, the simulation model could have provided many of the same results to within 10% accuracy. Cost savings for the project could have been realized had the simulation model been used to determine what eventually became the DQT2 baseline.

4.4 Effort Required To Adapt Model

Preparing models for different configurations required remarkably little effort. Figure 20 is a list of the twenty-seven configurations used thus far. Each

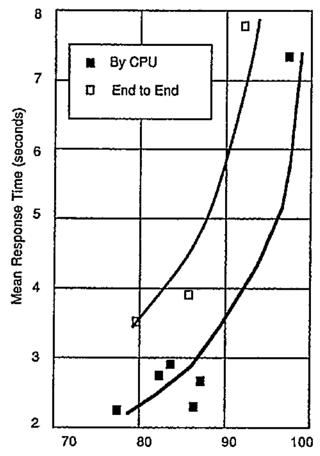


Figure 18a: Unoptimized Washington Fail-Safe Response Times

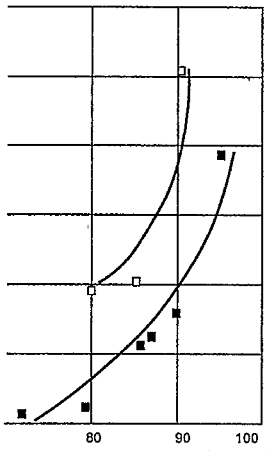


Figure 18b: Load balance I (Move 8 Terminals To CPU 0)

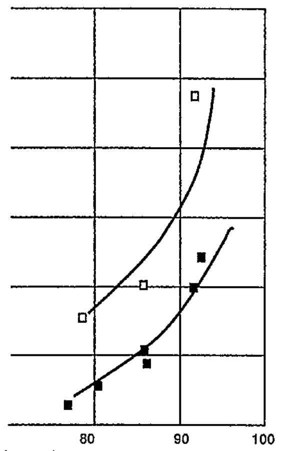


Figure 18c: Load Balance II (Move 4 Terminals To CPU 0)

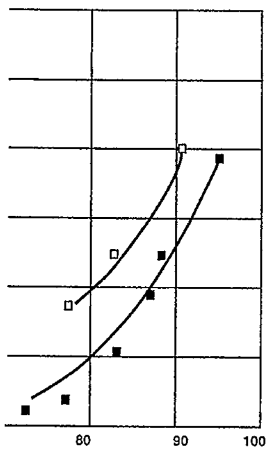


Figure 19: Load Balance I With Increased # of AFSS Communication Threads

configuration required a separate version of the model. Column two shows the amount of elapsed time used to modify the baseline model in assembling each new configuration. The average number of locations within the model that needed to be modified was 5. Note that these are not "code-changes" in the normal sense of altering program logic. They involve changes to literals (usually the values of constants). A hierarchy of system components exists, whereby an addition/deletion of a higher order element will imply an automatic adjustment in the appropriate array sizes for the lower order elements. It is this feature of the simulation support package which has allowed for a relatively small amount of effort to examine many alternative configurations.

It should be pointed out that these experiments were carried out by team members who at the start had no working experience with the model. That the model was easily learned by the newcomers, and that its code remained intact except for the "configuration choices", are a testament both to the model organization and the language it was written in.

Configuration Name	Elapsed Time For Modifications
DQT1 with SVCBT moved to CPU 2	5
DQT1 with 4 CPUs (no load bal)	20
DQT1 with 4 CPUs (swap commline 10,1)	6
TXP (2.5 times faster than TNSII)	15
TXP (3 times faster than TNSII)	5
DQT1 (remove disk access for FP mask)	5
DQT1 with 6 commlines	10
DQT1 with 3 commlines	10
DQT2 Basic Configuration	12
DQT2 with FP mask disk access removed	5
DQT2 with failed CPU2	45
DQT2 with failed COMMLine 5	10
CLEVELAND	5
INDIANAPOLIS	5
WASHINGTON	5
SALT LAKE CITY	5
HOUSTON	5
MIAMI	15
KANSAS CITY	5
CHICAGO	5
LOS ANGELES	5
NEW YORK	5
ATLANTA	5
SEATTLE	20
FORT WORTH	5

Figure 20: Model's Ease-Of-Use

5. FUTURE PLANS

Results from simulation runs to date are encouraging; nevertheless they were obtained from a model which is only partially calibrated. An important task will be to obtain more measurements from the real system, and so to place the calibration on firmer ground.

Clearly, with a system that is only just entering the field and is earmarked for continual evolution, there is a wealth of opportunities for the model to be of use. Possible applications include further configuration planning, design monitoring on the system enhancements that will be appearing in the years ahead, and feasability studies on enhancements that are still in the early planning stages.

6. SUMMARY

To study potential performance problems with the Flight Service Automation System, a model of one hub has been built. To date, only limited real-life measurements have been available for calibrating the model; so as yet the model is only partially calibr-ated, with intuitive judgment substituting for missing data. Nevertheless the partially calibrated model has proved capable of giving encouragingly realistic pred-ictions. Future plans include obtaining better data for a more complete calibration, followed by the oper-ational use of the model in a variety of situations.

Acknowledgements. The work described here was undertaken by the MITRE Corporation under contract to the Federal Aviation Administration, whose permission to publish is gratefully acknowledged.

C. R. SPOONER was educated at Cambridge, England. After working with ICT (later merged to become ICL), he emigrated to the US in 1968, where he has worked with Control Data Corporation and the MITRE Corporation. He has published papers on Operating System design, language design, and modeling and simulation. He is a fellow of BCS, and a member of ACM.

A. ACAMPORA Jr is a member of the technical staff at the MITRE Corporation. He received an MS in Computer Science from Stevens Institute of Technology in 1982. His current research interests include the ADA programming language, language design, real time systems, and artificial intelligence. He is a member of the ACM, IEEE Computer Society, and AAAI.

R. G. REGNER is a Member of the Technical Staff at the MITRE Corporation. Prior to that he held staff positions at Fairchild Industries and the Singer Company's Link Flight Simulation Division, where his primary contributions have been in the area of formal systems and software test. Mr Regner holds undergraduate degrees in Computer Science, General Science/Philosophy and Chemistry, and is completing requirements for an M.S. in Computer and Information Sciences from Hood college. Research interests include discrete-event simulation, experimental design and analysis, and the automated testing of software. He is a member of ACM SIGSIM, IEEE Computer Society, and International Test and Evaluation Association.

The MITRE Corporation,
1820, Dolley Madison Boulevard,
Mclean, VA, 22102
(703) 883-6000