Proceedings of the 2009 Winter Simulation Conference
M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls, eds.

PROGRAM SLICE DISTRIBUTION FUNCTIONS

Ross Gore
Paul F. Reynolds, Jr.

Dept. of Computer Science

151 Engineer's Way, P.O. Box 400740
Charlottesville, VA 22901, USA

ABSTRACT

Unexpected behaviors in simulations require explanation, so that decision makers and subject matter experts can separate valid behaviors from design or coding errors. Validation of unexpected behaviors requires accumulation of insight into the behavior and the conditions under which it arises. Stochastic simulations are known for unexpected behaviors that can be difficult to recreate and explain. To facilitate exploration, analysis and understanding of unexpected behaviors in stochastic simulations we have developed a novel approach, called Program Slice Distribution Functions (PSDFs), for quantifying the uncertainty of the dynamic program slices (simulation executions) causing unexpected behaviors. Our use of PSDFs is the first approach to quantifying the uncertainty in program slices for stochastic simulations and extends the state of the art in analysis and informed decision making based on simulation outcomes. We apply PSDFs to a published epidemic simulation and describe how users can apply PSDFs to their own stochastic simulations.

1 INTRODUCTION

Exploratory simulations have entered the mainstream of critical public policy and research decision-making practices (Cha 2005, Whipple 1996, Hooke 2000, Elderd 2006, National Science Foundation 2006, Arthur 1999). Public policy-makers and scientists look to these simulations for insight, trends and likely outcomes. Unfortunately, methods for gaining insight into unexpected outcomes with uncertain validity, as related to model design and simulation implementation and use, have not kept pace. We present a new approach to automating and quantifying some of the insight-gathering process for identifying, analyzing and understanding sources of unexpected behaviors in exploratory stochastic simulations.

The specifications of exploratory simulations are often incomplete because the application domain is poorly understood; the purpose of the simulation is to explore the domain of interest (Trenouth 1991). Writing the simulation becomes a theory construction task where the software is the expression of the theory (Wielinga 1978). Trusted simulations in the same application domain, datasets from physical experiments, and subject matter expert opinions are used to test the theory. This is what gives exploratory simulations their experimental nature. Those behaviors that are not defined in the specification and do not match the behavior of other trusted simulations, data sets from physical experiments or subject matter expert opinions are *unexpected behaviors*. Unexpected behaviors require understanding and explanation to determine if the behavior is an error or new knowledge in the application domain.

The daunting nature of quantifying, analyzing and understanding uncertainty in model design and simulation outcomes is evident in the results of epidemiology studies conducted this century. Epidemiologists have addressed the question of government level actions and reactions regarding the spread of infectious diseases such as smallpox and bird flu. Should a comprehensive vaccination program be initiated? How and to what degree should infected individuals be isolated, and for how long? The range of answers to these questions is broad and full of conflict. Recently, Elderd (Elderd 2006) has shown analytically that just four of the potentially hundreds of critical independent variables in these studies induce extreme sensitivity in model predictions, leading to serious conflict regarding remedial approaches involving billions of dollars and millions of people. Subject matter experts must be given additional capabilities to understand the behavior of their models so that model results can be used effectively and with confidence. According to a February 2006 report of the NSF Blue Ribbon Panel on Simulation-Based Engineering Science (SBES): "The development of reliable methodologies, algorithms, data acquisition and management procedures, software, and theory for quantifying uncertainty in computer predictions stands as one of the most important and daunting challenges in advancing SBES (National Science Foundation 2006)."

When an unexpected behavior is first observed in a stochastic simulation most users apply classic debugging techniques to identify program statements that lead to the unexpected behavior. This process is time-consuming and frustrating due to the uncertainty within stochastic simulations. The techniques require users to manually modify the source code and do not offer any capabilities that quantify the uncertainty within the unexpected behavior. We propose Program Slice Distribution Functions (PSDFs) to automate the collection of and quantify the uncertainty of dynamic program slices (simulation executions) in stochastic simulations. PSDFs represent the first attempt to quantify the uncertainty in program slices for stochastic simulations. The approach automatically samples, according to monte-carlo methods, the dynamic program slices and the values for each variable state within each dynamic program slice of a stochastic simulation, for a given dynamic slicing criterion. The monte-carlo method sampling captures the probability distribution of the dynamic program slices and the values for each variable state within each dynamic program slice. These probability distributions form a PSDF and provide users with means to explore the correlation between unexpected behaviors, program slices, and variable states.

PSDFs are part of our INSIGHT methodology (Gore and Reynolds 2009b), which enables user understanding and validation or rejection of unexpected behaviors. Within INSIGHT, PSDFs are used to improve the Causal Program Slicing (CPS) component. CPS combines program slicing and causal inference to automatically identify the statements in a simulation's source code that have the strongest influence on the unexpected behavior. In previous work we showed how PSDFs enable CPS, and thus INSIGHT, to be applied to stochastic simulations (Gore and Reynolds 2009b). Here, unlike our previous work, we highlight the capabilities of PSDFs to facilitate understanding and analysis of unexpected behavior outside of INSIGHT. Despite the use of PSDFs outside of INSIGHT, they are still used in a role that supports the use of our methodology.

Stochastic programs with uncertain inputs and structure used with an emphasis on insight are most likely to benefit from PSDFs. Epidemic models e.g. (Elderd 2006) are examples. Here, the goal is not to predict an exact number of infections and deaths but to determine the sensitivity of the predictions due to data and model uncertainties and the effectiveness of different vaccination scenarios. The use of simulations to gather insight separates them from software built without the intention of gathering insight, making simulations the ideal software domain to employ and evaluate PSDFs. Other software domains are less likely to benefit from PSDFs, for example, device drivers which are built to enable computer hardware to interact with software implement a specified interface (Rubini and Corbet 2001).

There is a difference between validating a simulation and validating an unexpected behavior that arises. The former represents an effort to demonstrate that the simulation exhibits expected behavior(s) (Boehm 1984). The latter is a demonstration of the validity of behavior that was unexpected for a given set of conditions, or experimental frames (Zeigler et al. 2000). The program comprehension and simulation communities have determined that understanding and subsequently validating simulation behavior requires hypothesis testing to accumulate insight into the behavior and the conditions under which it arises (Zeller 2002, Cleve and Zeller 2005, Storey 2006, Ruthruff et al. 2006). Then the problem can be reframed so that the unexpected behavior becomes part of the set of behaviors considered valid.

Next, we review work related to and employed by PSDFs. Then we present PSDFs and describe how we applied them to a case study model exhibiting unexpected behavior. Finally, we summarize our contributions and discuss future work.

2 RELATED WORK

PSDFs draw on the areas of program slicing, monte-carlo methods, and other applications of program slicing. In this section we review work in each of these areas and describe how it relates to PSDFs. We also review the role of PSDFs within our INSIGHT methodology.

2.1 Program Slicing

Program slicing is a decomposition technique that extracts program statements relevant to a particular computation within the program (Weiser 1984). A program slice provides the answer to the question, "What program statements affect the computation of variable v at statement s?" (Binkley and Gallagher 1996). An important distinction is that between static and dynamic slices. Figure 1(a) shows an example program that reads an integer input n, and computes the sum and the average of the first n positive numbers. If the sum of the first n integers is evenly divisible by n the program assigns sum to x. Otherwise the program assigns -1 to x. The criterion for a static slice is a 2-tuple consisting of {line number of statement s, the name of variable v}, where v is the variable of interest and s is the statement of interest. Figure 1(b) shows a static slice of this program using criterion {13, x}. As shown in Figure 1(b), all computations not relevant to the final value of variable x have been "sliced away". Slices are computed by identifying consecutive sets of transitively relevant statements, according to data flow and control flow dependences (Tip 1995). Since only statically available information was used to compute these slices they are static.

In the case of dynamic program slicing, only the dependences that occur in a specific execution of the program are taken into account. A dynamic slicing criterion specifies the input, and distinguishes between different occurrences of a statement in the execution history; it consists of {input, line number of statement s, name of variable v}. The difference between static and dynamic slicing is that dynamic slicing assumes fixed input for a program, whereas static slicing does not make assumptions regarding the input. Figure 1(c) shows a dynamic slice of the program in Figure 1(a) using the criterion {n = 4, 13, x}. Note that for input n = 4, the assignment x := -1 is executed, and the assignment x := sum is not executed. The "if (sum mod n == 0)" branch of statement 10, and statement 11 in Figure 1(a) may be omitted from the dynamic slice because the assignment of x := sum is not executed. The resulting dynamic program slice of the program in Figure 1(a) is shown in Figure 1(c).

```
1    read(n);
2    i := 1;
3    x := 0;
4    sum := 0;
5    average := 0;
6    while i<= n
7       sum := sum + i;
8       i := i + 1;
9    end
10   if (sum mod n == 0)
11      x := sum;
     else
12      x :=-1;
13   print (x);
14   average := sum/n;
15   print (average);
```

(a)

```
1    read(n);
2    i := 1;
3    x := 0;
4    sum := 0;

6    while i<= n
7       sum := sum + i;
8       i := i + 1;
9    end
10   if (sum mod n == 0)
11      x := sum;
     else
12      x :=-1;
13   print (x);
```

(b)

```
1    read(n);
2    i := 1;
3    x := 0;
4    sum := 0;

6    while i<= n
7       sum := sum + i;
8       i := i + 1;
9    end



12      x :=-1;
13   print (x);
```

(c)

Figure 1: (a) An example program. (b) A static slice of the program using criterion {`13 ,x`}. (c) A dynamic slice of the program using criterion {`n = 4, 13, x`}.

One shortcoming of the use of a single dynamic program slice is the inability to precisely analyze stochastic programs. Precision is measured by the number of dynamic slices for a fixed input provided by the analysis divided by the number of possible dynamic slices for a fixed input (Van der Walt and Barnard 2006). For stochastic models PSDFs provide users with a means to capture more dynamic program slices than any existing analysis technique. Furthermore, for each captured slice, users are provided with the likelihood of the slice being executed.

2.2 Monte-carlo Methods

Monte Carlo methods are useful for modeling phenomena with uncertainty. They represent a class of computational algorithms that operate on a series of independent samples from a specified probability distribution for uncertainty in models to approximate the probability distribution of the model's prediction for a specified input or set of inputs (Berg 2004). This process is shown in Figure 2 which is adapted from (Warren-Hicks et al. 2002).

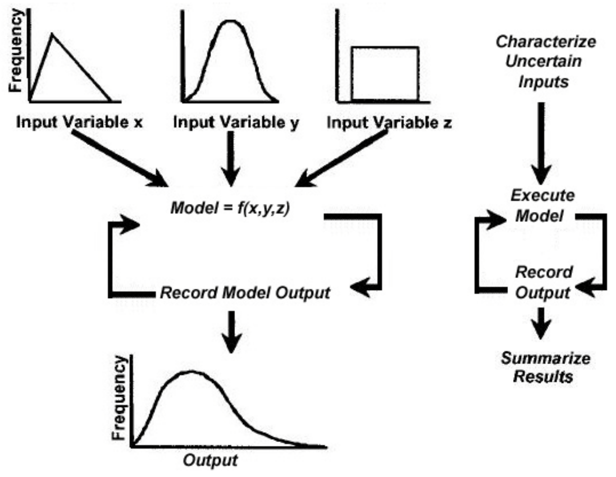

Figure 2: An outline of the Monte-Carlo method adapted from (Warren-Hicks et al. 2002)

Our intent is to apply Monte Carlo methods to collect the different possible dynamic program slices of a simulation with uncertain inputs for a given dynamic program slicing criterion. The approach captures the probability distribution of the dynamic program slices and the values for each variable state within each dynamic program slice. These probability distributions provide users with means to explore the correlation between unexpected behaviors, program slices, and variable states. The construction and application of PSDFs employing monte-carlo method sampling is the focus of our work here.

2.3 INSIGHT

INSIGHT combines semi-automated hypothesis testing, program slicing, causal analysis and generation of slice distribution functions to define an end-to-end process for discovering, analyzing and understanding sources of unexpected behaviors in simulations (Gore and Reynolds 2009b).

A primary component of INSIGHT is Causal Program Slicing (CPS). CPS combines program slicing and causal inference to provide insight into the interactions of simulation variables and source code statements that cause unexpected behavior. CPS provides: 1) automatic identification of all variables in a simulation that may influence the computation of the unexpected behavior, 2) capture of state changes throughout simulation execution for each of the identified variables, 3) quantification of the influence each state change in a variable has on the unexpected behavior and 4) a mapping of each state change for each variable to the statement in the simulation's source code that caused the state change (Gore and Reynolds 2009a). CPS precision suffers when a simulation includes stochastics, as most program analyses do. As a remedy PSDFs are employed to quantify the uncertainty of dynamic program slices in stochastic simulations. This use of PSDFs greatly increases precision in CPS analysis for stochastic simulations (Gore and Reynolds 2009b).

Our interest in PSDFs was originally sparked by the need to improve the precision of CPS analysis. However, we realized that PSDFs could be used outside of INSIGHT to facilitate understanding and analysis of unexpected behavior. Outside of INSIGHT, PSDFs support user generation of hypotheses about an unexpected behavior by automating the exploration of the correlation between unexpected behaviors, program slices, and variable states. Then, these hypotheses can be tested by applying INSIGHT to identify the program statements within the dynamic program slices that have the strongest influence on the unexpected behavior. Using PSDFs in this manner, as a supportive technology to INSIGHT is one of the contributions of our work.

2.4 Applications of Program Slicing

Previous researchers have used static and dynamic program slicing separately and in combination to enable program debugging. In debugging, one is often interested in a specific execution of a program that exhibits anomalous behavior, which in part matches our goal of understanding all possible model behaviors. Dynamic slices are particularly useful here, because they only reflect the actual dependences of that execution, resulting in smaller slices than static ones (Harman 1997, Korel and Rilling 1997). Pan and Spafford present a number of heuristics for fault localization using dynamic slices to select a set of statements likely to contain a bug (Pan and Spafford 1994). Agrawal et al. combined static and dynamic program slicing to propose an approach for semi-automated debugging of programs (Agrawal et al. 1993).

Most of the work related to program slicing and thus PSDFs is based in software engineering. While program slicing is rooted in the software engineering community our goal is to apply PSDFs to stochastic programs with uncertain inputs and structure used with an emphasis on insight gathering. Most stochastic simulations are built to gather insight into a system, instead of being built to meet a set of requirements. These simulations generally never reach the stage of software deployment, instead they are iteratively modified to gain increasing insight into the modeled system. PSDFs enhance the capabilities available to users for gathering insight into unexpected behaviors in these types of stochastic programs. As a result simulations are an ideal software domain to employ and evaluate PSDFs.

3 PROGRAM SLICE DISTRIBUTION FUNCTIONS

When an unexpected behavior is first observed in a stochastic simulation, the prospect of explaining and then either validating or eradicating that behavior can be daunting. Most users apply classic debugging techniques to identify program statements that lead to the unexpected behavior. Subsequently an explanation for the behavior is formed, code is modified and the user iterates this process until satisfied. The process is manual, time-consuming and complicated due to stochastic behavior of the simulation. We realized that the process could benefit from a significant degree of automation not currently employed. Thus program slice distribution functions (PSDFs) were born.

Our introduction and use of PSDFs represents the first attempt to quantify the distribution of dynamic program slices in stochastic simulations. Our approach automatically samples, according to monte-carlo methods, the dynamic program slices and the values for each variable state within each dynamic program slice of a stochastic simulation, for a given dynamic slicing criterion. The monte-carlo method sampling captures the probability distribution of the dynamic program slices and the values for each variable state within each dynamic program slice. These probability distributions form a PSDF and provide a means to explore the correlation between unexpected behaviors, program slices, and variable states, thus enhancing the ability to analyze and understand the unexpected behavior. The environment in which we envision the use of PSDFs is given in Section 3.4. First, we introduce the particulars of our PSDF technology.

3.1 Imprecision Through the Use of a Single Dynamic Program Slice

For a stochastic simulation a single dynamic program slice does not necessarily capture all of the program slices for a given input. Recall, precision is measured by the number of dynamic slices for a fixed input provided by the analysis divided by the number of possible dynamic slices for a fixed input (Van der Walt and Barnard 2006).

Figures 3(a), 3(b) and 3(c) elucidate the imprecision in the use of a single dynamic program slice. From the user's point of view the behavior of the example program in Figure 3(a) is stochastic. Figures 3(b) and 3(c) show the two possible dynamic program slices using criterion {n = 13, 7, x} for the program in Figure 3(a). Figure 3(b) shows the dynamic program slice, when the random number generator does not generate a number between .998 and .999. Figure 3(c) shows the dynamic program slice when the random number generator does generate a random number between .998 and .999. Assuming a uniform random number generator the dynamic program slice for the program in Figure 3(a) is the program slice shown in Figure 3(b) approximately 99.9% of the time and the program slice shown in Figure 3(c) 0.1% of the time. Without additional analysis capabilities a user employing a single dynamic program slice of the program in Figure 3(a) cannot capture all the possible behaviors.

Ideally, a user would be given all possible slices and would know the likelihood of each slice for a given dynamic slicing criterion. The goal of PSDFs is to provide this analysis. For the example in Figure 3(a) PSDFs will capture the two possible program slices and show that the program slice in Figure 3(b) is executed 99.9% of the time, and the program slice in Figure 3(c) is executed 0.1% of the time.

```
1     read(n);
2     x = 0;
3     rand := randomNumber(0, 1);
4     if (rand >= .998 &&
          rand <= .999)
5        x := rand + n;
      else
6        x := n;
7     print (x);
```

(a)

```
1     read(n);
2     x = 0;

// (rand >= .998 &&
//  rand <= .999) == false


6        x := n;
7     print (x);
```

(b)

```
1     read(n);
2     x = 0;
3     rand := randomNumber(0, 1);
//    (rand >= .998 &&
//     rand <= .999) == true
5        x := rand + n;


7     print (x);
```

(c)

Figure 3: (a) A stochastic program. (b) One possible dynamic slice of the program using criterion {n=13,7,x}. (c) Another possible dynamic slice of the program using the same criterion.

3.2 Generating a PSDF

The method for generating PSDFs is straightforward but computationally intensive. Generating a PSDF assumes a program, most likely a stochastic simulation, a dynamic program slicing criterion, and an integer *n*, which specifies the number of times to execute the simulation. Next, the stochastic simulation is executed *n* times for the specified dynamic slicing criterion. For each execution, the dynamic program slice (or execution path) for the given slicing criterion is stored. Along with the dynamic program slice the variable states and program statements causing the variables to change state are stored. Once all executions are complete the stored dynamic program slices and their respective variable state data are grouped as follows. Each dynamic program slice is grouped with the dynamic program slice (or execution path) it matches exactly. If no such dynamic program slice exists, a new group is formed. Two dynamic program slices A and B match exactly if and only if every statement within program slice A is executed in order, in program slice B, and every statement in program slice B is executed in order, in program slice A. Variable state data that accompanies a dynamic program slice plays no role in the matching process but is stored along with its respective dynamic program slice. Probability distributions of the dynamic program slices and variable states within the dynamic program slices for a stochastic simulation given a slicing criterion result from the matching process. The distributions can be sampled to determine the likelihood of 1) a dynamic program slice is executed for a stochastic simulation given a slicing criterion, and 2) a value of a particular variable state within a specified dynamic program slice. Our tool for automating this process is called PSDF Generator.

3.3 Applying PSDFs to an Example Program

We elucidate the process of generating and employing a PSDF by applying it to the simple program in Figure 3(a). The process transpires between the user and PSDF Generator.

1. The user specifies the unexpected behavior with the static program slicing criterion that captures the program state of the unexpected behavior. In this case the unexpected behavior is the value of `x` at line 7; the user specifies{7, `x`}.
2. PSDF Generator applies the static slicing criterion to preprocess the program to collect variable states and the execution paths. The details of the preprocessor are described in (Gore and Reynolds 2009a).
3. The user specifies the input of interest for analysis of the unexpected behavior. Next, the user specifies, *n*, the number of times to execute the simulation for the input. In this case the input of interest is 13 and *n* = 100,000.
4. PSDF Generator combines each specified input with the static slicing criterion to form a dynamic slicing criterion. The dynamic slicing criterion formed is {13, 7, `x`}.
5. PSDF Generator executes the dynamic slicing criterion 100,000 times.
    a. For each execution the dynamic program slice (or execution path) is stored.
    b. The variable states within the dynamic program slice are stored by the inserted preprocessing code.

    The probability distribution of the dynamic program slices of the program in Figure 3(a) is shown in Figure 4.
6. The user specifies any variable states of interest, using a dynamic slicing criterion, for which to generate a probability distribution. In this case the specified variable state of interest is the state of `x` at line 7, {13, 7, `x`}. Next, the user specifies the number of samples, *k,* to form the probability distribution for the variable state. In this case *k* = 10,000.
7. PSDF Generator forms a probability distribution of 10,000 samples for each specified variable state. Each sample requires two steps.
    a. PSDF Generator chooses with uniform random probability a dynamic program slice group within the probability distribution of dynamic program slices.
    b. Within the chosen dynamic program slice group, PSDF Generator chooses with uniform random probability a sample of the specified variable state.

    The probability distribution function for the specified variable state, {13, 7, `x`}, is shown in Figure 5.

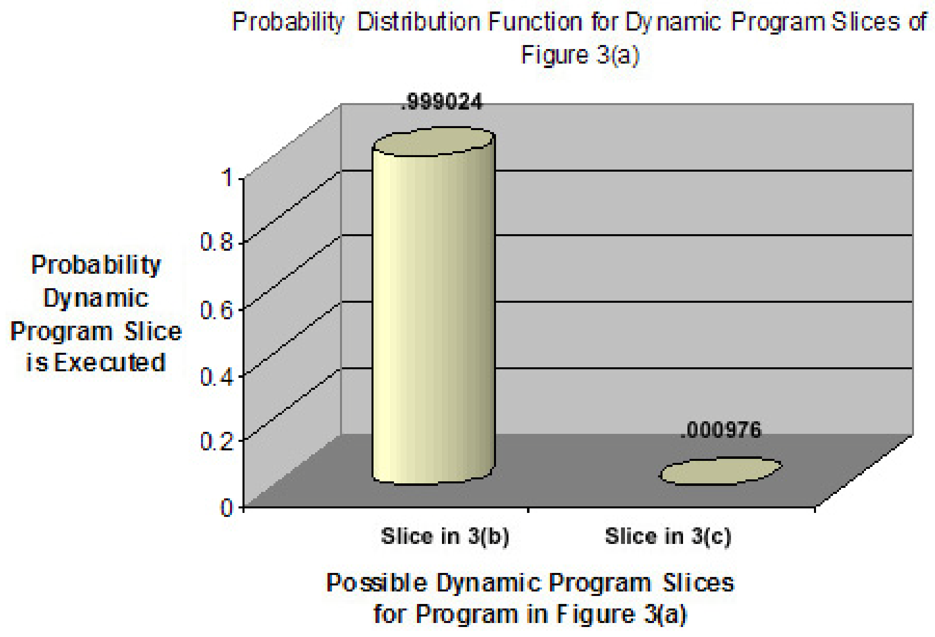

Figure 4: The probability distribution of the dynamic program slices of the program shown in Figure 3(a)

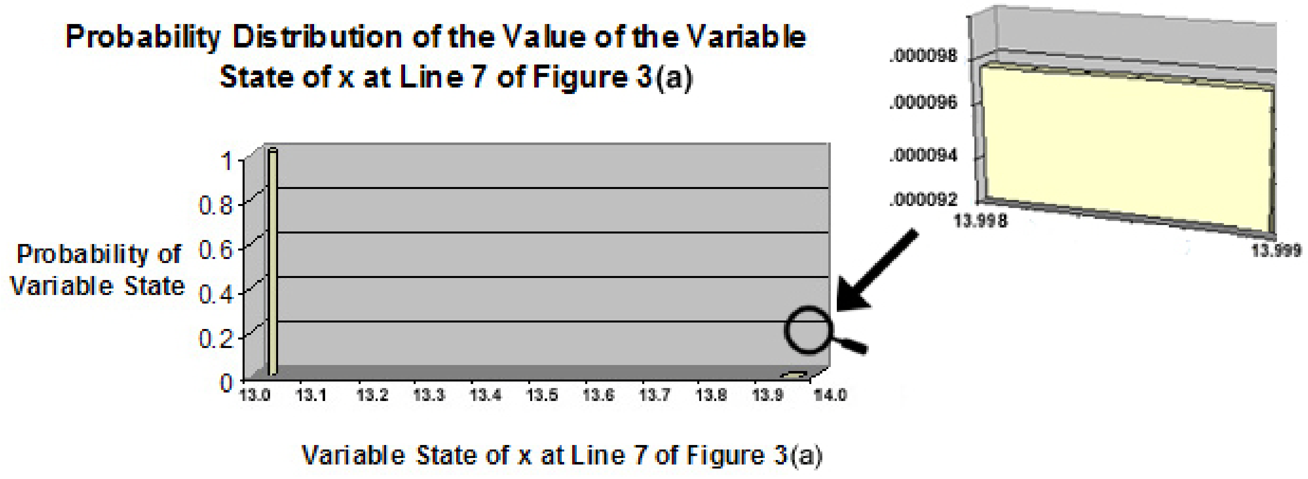

Figure 5: The probability distribution for the state of x at line 7 of the program shown in Figure 3(a). Note the area between 13.998 and 13.999 on the x-axis that has been magnified for visual clarity.

The PSDF shown in Figures 4 and 5 reveals the correlated relationship between the dynamic program slice executed and the value of x at line 7. The probability that the program slice in 3(b) is executed is the same as the probability that the value of x at line 7 is 13. Conversely, the probability that the program slice in 3(c) is executed is the same as the probability the value of the x at line 7 is not 13. Based on this correlated relationship one could hypothesize that value of x at line 7 is determined by which program slice is executed. Then our INSIGHT methodology can be employed to verify this hypothesis and determine the program statements in these dynamic slices that have the strongest causal influence on the value of x at line 7. The use of INSIGHT and PSDF analysis together is discussed further in Section 3.4.

The example shown here is not meant to be representative of an actual program with unexpected behavior, but to illustrate how PSDFs are generated and applied. In Section 4, we present a case study where PSDFs are rigorously applied to a published stochastic epidemic simulation and valuable insight is gained from the process.

3.4 Using PSDFs

When an unexpected behavior is first observed in a simulation, PSDFs can be applied in several ways to facilitate formation of explanations for the behavior. A user can choose to apply our methodology INSIGHT, to identify program statements that cause the unexpected behavior. However, applying INSIGHT requires the user to create hypotheses about the unexpected behavior to test. PSDFs can be used as a supporting technology, outside of INSIGHT, to help users generate hypotheses about an unexpected behavior to test by automating the exploration of the correlation between unexpected behaviors, program slices, and variable states.

Often users of a stochastic simulation observe unexpected behaviors that do not occur each time the model is executed for a given input. In this case the user can specify the input and static program slicing criterion representing the unexpected behavior and generate a PSDF. A PSDF captures the distribution of the state information for each variable in a slice including the state of the variable representing the unexpected behavior. This distribution makes evident the likelihood of an unexpected behavior for a given input. Then, the user can compare the likelihood of an unexpected behavior with the probability distribution of the dynamic program slices, which is also provided by a PSDF. Often a strict subset of the dynamic program slices is responsible for causing the unexpected behavior. A user can examine the PSDF to determine if the likelihood of one or more dynamic program slices is similar to the likelihood of an unexpected behavior. If a subset of dynamic program slices seems as likely as the unexpected behavior, the user can hypothesize that the identified dynamic program slices cause the behavior. Then, the user can test her hypotheses and gain additional insight into these relationships by applying INSIGHT to identify the program statements within the dynamic program slice that have the strongest influence on the unexpected behavior.

Besides providing insight to enable an explanation of unexpected behavior, PSDFs can be used for any problem where a quantification of the distribution of each variable state or the different possible dynamic program slices within a stochastic program is needed.

4 CASE STUDY: DUNHAM MODEL

In order to explore the effectiveness of the insight provided by PSDFs we performed a case study using the Dunham agent-based epidemic model (Dunham 2005). The Dunham model (Dunham ABM) predicts disease spread by modeling interactions on a 2-D torus. At each time step, infectious individuals in proximity to susceptible individuals within a specified radius spread their infection with a given probability. Dunham claims, "the curves (Susceptible, Exposed, Infected, Removed) created are a qualitative match to real-world epidemic data. With proper parameterization, this model could be used for realistic simulations (Dunham 2005)". Figure 6 shows the result for a single run of Dunham's ABM with the author's suggested parameterization for an SEIR epidemic for 100 days for a population of 100. This prediction will be referred to as the initial prediction.

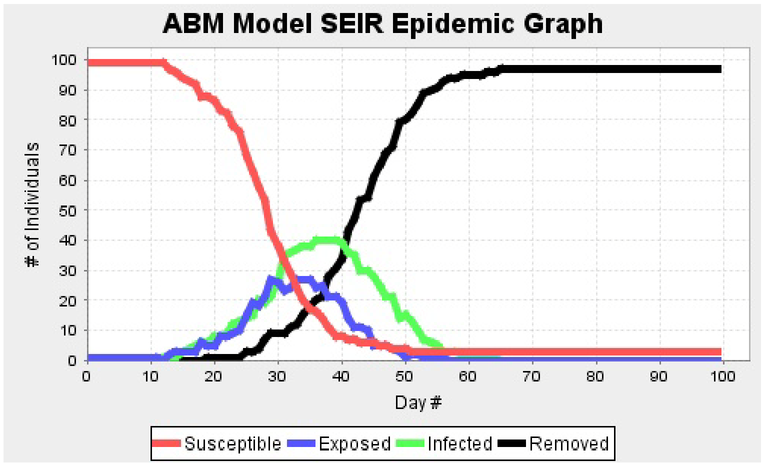

Figure 6: The initial prediction of Dunham's ABM using the suggested parameterization for a population of 100 for 100 days

The Dunham model is stochastic with uncertain inputs. We performed a monte-carlo method sampling to determine the range of possible predictions for the model given the author's suggested parameterization. For each prediction we recorded the arithmetic mean of the standard deviations of the predicted four curves (S,E,I,R) from the four curves in the initial prediction shown in Figure 6. The monte-carlo method sampling of the arithmetic mean of the standard deviation of the four curves from the four curves in the initial prediction over 10,000 executions is shown in Figure 7.

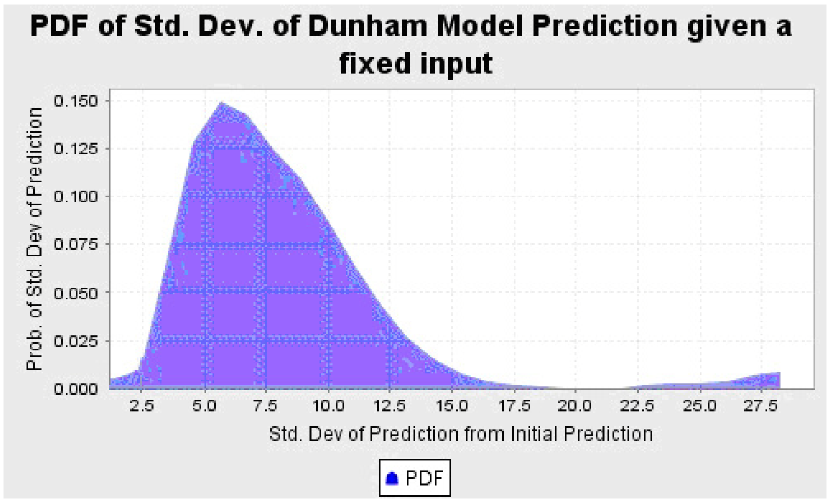

Figure 7: The monte-carlo method sampling of the probability distribution function of the arithmetic mean of the standard deviation of the four curves of a trial from the four curves in the initial prediction

Figure 7 reveals an unexpected behavior, 2.5% of the time the Dunham model makes a prediction that has a standard deviation that is more than 25% of the size of the population (far right end of the curve in Figure 7). We found this unexpected behavior to present an opportunity to use a PSDF to gain insight. To generate the PSDF for the Dunham model, we ran the model under the author's suggested parameterization for 10,000 trials. For each trial we recorded the dynamic program slice that was executed, the output, and the changes in variable state in each dynamic program slice.

The generated PSDF for the Dunham model contained 10,000 different executions resulting in 10,000 different program slices! Due to modeling epidemic disease on an individual by individual basis and the amount of uncertainty in the inputs of the Dunham model, all of the groups of program slices had a size of one. In other words, no two dynamic program slices were the same, preventing any dynamic program slices from being grouped together, following our original evaluation criterion which required program slices to be identical. This unexpected outcome for our methodology led us to reexamine our evaluation criteria for establishing identity.

We modified our PSDF evaluation criterion to measure the *similarity* of the program slice forming the initial prediction and the program slices from the 10,000 trials. Recall, the Dunham ABM executes in time steps. At each time step, each agent executes a schedule of actions. Our modified evaluation criterion, which measures the *similarity* of program slices is measured by the percentage of program statements within each time step that match program statements in the same time step of the initial prediction. Similar program statements are required to be within the same time step of the Dunham ABM and execute the same program statement. However, similar program statements are not required to be in the same program order within a time step. This relaxed evaluation criterion allows us to quantify the relationship between the 10,000 different dynamic program slices in a meaningful manner. The modified PSDF in Figure 8 shows the percentage of program statements in the 10,000 dynamic program slices that are dissimilar, by our evaluation criterion, from the initial prediction's dynamic program slice.

Our adaptation of PSDFs to compute the similarity of dynamic program slices employs an analogous strategy to delta change algorithms (Hunt et al. 1998). Our current approach is not optimized and employs a $\Theta(n^2)$ algorithm to compute the similarity between *n* dynamic program slices and the initial prediction slice. For large values of *n* this leads to prohibitively long compute times. For example, it took us five days to run the similarity algorithm for the 10,000 dynamic program slices collected. However, due to the progress made in improving time and space efficiency in delta change algorithms, progress that we did not exploit in this experiment, we are optimistic the cost of generating modified PSDFs can be reduced.

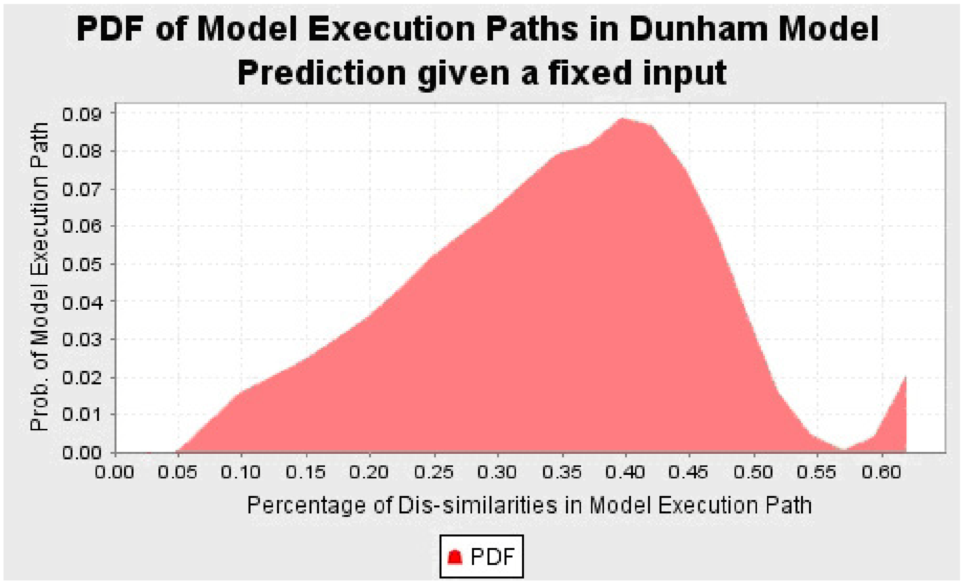

Figure 8: The modified PSDF for the Dunham Model prediction with the author's suggested parameterization. The modified PSDF shows the percentage of dissimilar program statements in a trial program slice from the program slice in the initial prediction.

The modified PSDF in Figure 8 offers a user insight into the predictions with a standard deviation of 25% of the population from the initial prediction. Dynamic program slices with dissimilarity greater than 60% occur with the same frequency as the predictions with a standard deviation greater than 25%. To investigate the correlation between the standard deviations of a prediction from the initial prediction and the dissimilarity of a dynamic program slice we plot their relationship in Figure 9.

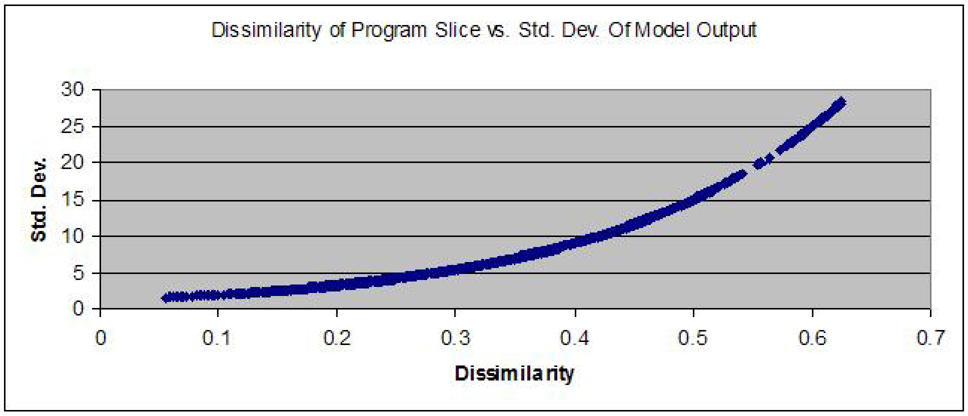

Figure 9: The plot of the standard deviation of the prediction from the initial prediction vs. the dissimilarity of the prediction's program slice from the program slice of the initial prediction

Figures 8 and 9 contain useful information for developing insight. Figure 8 demonstrates the dissimilarity of the dynamic program slices in the Dunham Model for the author's suggested parameterization and suggests a correlated relationship between the standard deviation of predictions and the dissimilarity of the dynamic program slices. Figure 9 confirms and quantifies the strength of the nonlinear correlated relationship with a correlation ratio of .79 using (Kelley 1935). Because most of the PSDF construction process can be automated, a user can gain the valuable insight we've demonstrated here with little effort.

Our acquired insight led us to hypothesize that the most dissimilar program slices are causing the unexpected behavior, the predictions with the largest standard deviation. Since the correlation shown in Figure 9 does not imply causality, we applied our INSIGHT methodology, which employs CPS to determine if and how the dissimilar program slices cause the unexpected behavior. We envision PSDFs and INSIGHT being applied to stochastic simulations in this manner. PSDFs are used to explore correlated relationships between unexpected behavior, program slices and variable states, and then to facilitate generation of hypotheses about the unexpected behavior. Then, INSIGHT is applied to test the hypothesis and identify the causal connections among program statements that have the strongest influence on the unexpected behavior.

Through our application of INSIGHT in this analysis we discovered in the Dunham Model that there are far more program statements computing the probability that an individual is exposed or infected with the disease in the initial prediction program slice than the most dissimilar program slices. In the predictions with a large standard deviation from the initial prediction very few individuals become exposed or infected. As a result, the dynamic program slices for these trials are very different from the program slice of the initial prediction where every agent becomes exposed and infected. Users equipped with an understanding of why some predictions of the Dunham model with the suggested parameterization have a much larger standard deviation from the initial prediction possess greater insight into the simulation's behavior. The combined use of PSDFs and INSIGHT enabled this automated analysis for the Dunham model. Now, the user must determine if the behavior is valid.

5 CONCLUSION

Simulation has become the tool of scientific analysis under circumstances where it is infeasible or impractical to study a system directly (Whipple 1996, Hooke 2000, Arthur 1999, National Science Foundation 2006). Everyday policy debates involving simulations such as Dunham's ABM (Dunham 2005), raise the perfectly legitimate question of whether decision makers can use the simulation-based predictions with confidence. How can policy makers make informed decisions involving billions of dollars and millions of people in confidence when poorly understood unexpected program behaviors are pervasive? With impact on this scale a methodology for improving simulation understanding and supporting the validation of unexpected behaviors in stochastic simulations is needed; this need has motivated the design and development of PSDFs.

PSDFs automate the collection of and quantify the uncertainty of dynamic program slices (simulation executions) in stochastic simulations. The approach automatically samples, according to monte-carlo methods, the dynamic program slices and the values for each variable state within each dynamic program slice of a stochastic simulation, for a given dynamic slicing criterion. The monte-carlo method sampling captures the probability distribution of the dynamic program slices and the values for each variable state within each dynamic program slice. These probability distributions form a PSDF and provide users with means to explore the correlation between unexpected behaviors, program slices, and variable states. PSDFs represent the first attempt to quantify the uncertainty in program slices for stochastic simulations. Coupled with causal analysis tools, such as our INSIGHT methodology (Gore and Reynolds 2009b), users now possess a richer suite of semi-automated analysis methods for explaining unexpected simulation outcomes. We have shown how a PSDF is used to enhance understanding of an unexpected behavior in the published SEIR epidemic Dunham ABM. In future work we expect to improve the efficiency of the process to generate PSDFs.

ACKNOWLEDGMENTS

We gratefully acknowledge support from our colleagues in the Modeling and Simulation Technology Research Initiative (MaSTRI) at the University of Virginia and funding from the Science Applications International Corporation (SAIC).

REFERENCES

Arthur, W. 1999. Complexity and the Economy. *Science* 284:107-109.

Agrawal, H., R. A. DeMillo and E. H. Spafford, 1993. Debugging with dynamic slicing and backtracking. *Software-Practice & Experience* 23(6): 589-616.

Berg, A. 2004. *Markov Chain Monte Carlo Simulations and Their Statistical Analysis: With Web-based Fortran Code*. World Scientific Press: London.

Binkley, D., 2002. An empirical study of the effect of semantic differences on programmer comprehension, In *Proceedings of the IEEE 8th International Workshop on Program Comprehension*, 97-106. Piscataway, New Jersey. Institute of Electrical and Electronics Engineers, Inc.

Binkley, D. and K. Gallagher. 1996. *Advances in Computers: Program Slicing*. Academic Press: Burlington.

Boehm, B. 1984.Verifying and validating software requirements and design specifications. *IEEE Software* 1 (1): 7-88.

Cha, A. 2005. Computers Simulate Terrorism's Extremes. In *Washington Post*, ed. P. Bennett, M. Coleman, and L. Downie, A1. Washington Post: Washington, D. C.

Cleve, H. and A. Zeller. 2005. Locating Causes of Program Failures. In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, 342 - 351.

Duesterwald, E., R. Gupta, and M.L. Soffa. 1992. Rigorous data flow testing through output influences. In *Proceedings of the Second Irvine Software Symposium (ISS'92)*, 131-145.

Dunham, J. 2005. An Agent-Based Spatially Explicit Epidemiological Model in MASON. *Journal of Artificial Societies and Social Simulation* 9:(1).

Elderd, B, V. Dukic and G. Dwyer. 2006. Uncertainty in predictions of disease spread and public health responses to bioterrorism and emerging diseases. *Proceedings of the National Academy of Sciences* 103(42): 15693-15697.

Gore, R and P. Reynolds. 2009. Causal Program Slicing. In *Proceedings of the 23rd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2009)*, 19-26.

Gore, R. and P. Reynolds. 2009. INSIGHT: Understanding Unexpected Behaviors in Agent-Based Models. Submitted to *Journal of Simulation*.

Hooke, W and R. Pielke. 2000. *Prediction: Science, decision making, and the future of nature*. Island Press: Washington, D.C.

Fox, C., M. Harman, S. Danicic and R. Hierons. 2000. ConSIT: A conditioned program slicer. In *IEEE International Conference on Software Maintenance (ICSM).* 216-226. Piscataway, New Jersey. Institute of Electrical and Electronics Engineers, Inc.

Gupta, R., M. J. Harrold and M. L. Soffa. 1992. An approach to regression testing using slicing. In *Proceedings of the 1992 Conference on Software Maintenance*, 299-308.

Harman, M. 1997. Cleaving Together - program cohesion with slices. *EXE Software Developer's Magazine* 11(8): 35-42.

Hierons, R. M. and M. Harman. 2000. Program analysis and test hypotheses complement. In *IEEE ICSE International Workshop on Automated Program Analysis, Testing and Verification,* 32-39. Piscataway, New Jersey. Institute of Electrical and Electronics Engineers, Inc.

Hunt, J., K. Vo. and W. Tichy. 1998. Delta Algorithms: An Empirical Analysis. *ACM Transactions on Software Engineering and Methodology* 7(2): 192-214.

Kamkar, M. 1993. Interprocedural Dynamic Slicing with Applications to Debugging and Testing, PhD thesis, Linkoping University.

Kelley, T. 1935. An Unbiased Correlation Ratio Measure. *Proceedings of the National Academy of Sciences* 21: 554-559.

Korel, B. and J. Rilling. 1997. Application of dynamic slicing in program debugging. In *Proceedings of the 3rd International Workshop on Automated Debugging*, ed. M. Kamkar, 43-57. Linköping: Linköping University Electronic Press.

Mund, G.B., R. Mall and S. Sarkar. 2003. Computation of intraprocedural dynamic program slices. *Information and Software Technology* 45(8): 499-512.

National Science Foundation 2006. Simulation-based engineering science: Revolutionizing engineering science through simulation. Report of the NSF Blue Ribbon Panel on Simulation-Based Engineering Science.

Pan, H. and E. H. Spafford. 1994. Fault localization methods for software debugging. *Journal of Computer and Software Engineering* 2: 302-324.

Rubini, A. and J. Corbet. 2001. *Linux Device Drivers.* 2nd Ed. Sebastopol: O'Reilly.

Ruthruff, J. R., S. Elbaum and G. Rothermel. 2006. Experimental program analysis: a new program analysis paradigm. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, 49-60.

Storey, M. A. 2006. Theories, Methods and Tools in Program Comprehension: Past, Present, and Future. In *Proceedings of the 13th International Workshop on Program Comprehension*, 181-191.

Tip, F. 1995. A Survey of Program Slicing Techniques. *Journal of Programming Languages* 3(3): 121-189.

Trenouth, J. 1991. A survey of exploratory software. *The Computer Journal* 34(2) : 153-163.

Van der Walt C and Barnard E. 2006. Data characteristics that determine classifier performance. In *Proceedings of the 17th Annual Symposium of the Pattern Recognition Association of South Africa,* 160- 165.

Warren-Hicks, W., J. P. Carbone, and P.L. Havens. 2002. Using monte carlo techniques to judge model prediction accuracy. *Environmental Toxicology and Chemistry* 21(8): 1570–1577.

Weiser, M. 1984. Program Slicing. *IEEE Transactions on Software Engineering* 10(4): 352-357.

Whipple, C. 1996. Can nuclear waste be stored safely at yucca mountain? *Scientific American* 274(6): 72-79.

Wielinga, B. 1978. AI programming methodology. In *Proceedings of the Artificial Intelligence and Simulation of Behavior Conference*, 355-374.

Zeller, A. 2002. Isolating Cause-Effect Chains from Computer Programs. In *Proceedings of ACM SIGSOFT 10th International Symposium on the Foundations of Software Engineering (FSE-10)*, 1-10.

Zeigler, B. P., H. Praehofer, and T.G. Kim. 2000. *Theory of Modeling and Simulation*. 2nd Ed. Burlington: Academic Press.

Zhang, X., and R. Gupta. 2004. Cost Effective dynamic program slicing. *ACM SIGPLAN Notices* 39(6): 94-106.

AUTHOR BIOGRAPHIES

ROSS GORE is a Ph.D. Candidate in Computer Science and a member of MaSTRI at the University of Virginia. His email address is <rjg7v@virginia.edu>.

PAUL F. REYNOLDS, JR. is a Professor of Computer Science and member of MaSTRI at the University of Virginia. He has conducted research in Modeling and Simulation for over 30 years, and has published on a variety of M&S topics, including parallel and distributed simulation, multi-resolution modeling and simulation. His email address is <reynolds@virginia.edu>.