International Journal of Modeling, Simulation,
and Scientific Computing
Vol. 7, No. 1 (2016) 1641001 (22 pages)

DOI: 10.1142/S1793962316410014

World Scientific
www.worldscientific.com

Formal methods, statistical debugging and exploratory analysis in support of system development: Towards a verification and validation calculator tool

Saikou Y. Diallo*, Ross Gore†, Christopher J. Lynch‡ and Jose J. Padilla§

Virginia Modeling, Analysis and Simulation Center
Old Dominion University
1030 University Blvd. Suffolk, VA 23435, USA
**sdiallo@odu.edu*
†*rgore@odu.edu*
‡*cjlynch@odu.edu*
§*jpadilla@odu.edu*

Received 30 March 2015
Accepted 20 January 2016
Published 10 March 2016

In this paper, we propose an approach to formally verify and rigorously validate a simulation system against the specification of the real system. We implement the approach in a verification and validation calculator tool that takes as input a set of statements that capture the requirements, internal conditions of the system and expected outputs of the real system and produces as output whether the simulation satisfies the requirements, faithfully represents the internal conditions of the system and produces the expected outputs. We provide a use case to show how subject matter experts can apply the tool.

Keywords: Verification and validation; formal methods; modeling and simulation.

1. Introduction

Modeling and Simulation (M&S) is an essential part of system development. According to M&S best practices, a subject matter expert (SME) identifies a system and a modeling question they wish to answer. The system is simplified into a conceptual model which is further simplified into a simulation that can answer the modeling question. The simulation is then verified to ensure that it is a fair representation of the conceptual model and the conceptual model is validated against the system to ensure that it is not overly simplified and accurately represent all of the key components that are essential in answering the modeling question. Unfortunately, because of the amount of simplification from system to conceptual model and from conceptual model to simulation, this process results in the loss of needed

*Corresponding author.

information and the addition and generation of extraneous information. The loss of information occurs because: (1) SMEs are unable to identify all of the requirements that define the context of their question and model[1]; and (2) the languages used to capture a SME's conceptual model offer more expressive semantics than the languages used to realize the simulation. Realizing the simulation also adds extraneous information to the representation of the system through the use of seeds for stochastic distributions and the sequentialization of simultaneous events. Finally, previously unidentified system information can be generated through the interactions of the underlying components of the model.[2]

These issues motivate the need for a framework that allows for multiple viewpoints of a system (modeling question, conceptual model and simulation) while maintaining a single consistent representation. The M&S system development framework (MS-SDF) addresses this need by tying together systems engineering process with M&S processes into a cohesive approach.[3] In the MS-SDF, multiple viewpoints of a system are realized while maintaining one consistent system representation through the application of verification and validation (V&V). Applying V&V in the MS-SDF framework enables SMEs to prove that each requirement is always satisfied, exactly met and that there is no undesired emergence due to the convolution of multiple viewpoints into a single simulation. While Ref. 3 provides the skeleton for such a framework they do not define which V&V procedures should be applied and how they should be performed in order to ensure consistency between the different viewpoints.

Within the MS-SDF: (1) a system is framed by a set of statements that are always TRUE (axioms) and capture the requirements of the SMEs in order to bound the system; (2) a system is a set of statements (independent or related) about the internal system conditions and outputs of the systems that are TRUE given the set of requirements. Together the set of requirements and the set of statements constitute the system specification; (3) verification is the process of ensuring that the simulation satisfies the system specification without contradiction and (4) validation is the process of ensuring that the entire specification is carried over from the system to the simulation. In other words, capturing the system specification in the form of statements reduces the V&V process to ensuring that: (1) statements that are always true about the system are also always true about the simulation, (2) statements that are satisfied in the system are also satisfied in the simulation and (3) every statement in the system is also a statement in the simulation.[3] Thus, the V&V challenges left unaddressed are: (1) how does a SME map simulation outputs to the necessary statements for V&V and (2) once mapped what V&V mechanisms does a SME employ to evaluate the statements.

In this paper, we address both these challenges. First, we define an approach that enables a SME to map system specifications to internal system conditions and outputs. Our approach is highly structured but entails a gentle learning curve for SMEs. Next, we introduce a V&V calculator for SMEs to evaluate the statements. Our V&V calculator provides access to a comprehensive set of rigorous V&V

procedures (formal verification methodology, statistical debugging and exploratory analysis) through a familiar look and feel encapsulated in a straightforward graphical interface to enable SMEs to get feedback related to the validity of their derived statements. Finally, we apply our approach and V&V calculator to a system use case to demonstrate and evaluate its efficacy. We combine these contributions to realize the V&V procedures specified in the MS-SDF through a SME-focused implementation with little to no learning curve.

2. Background

Recall, our approach guides SMEs through the process of mapping simulation outputs to the necessary statements for V&V. Then our V&V calculator enables SMEs to test each statement in an evaluation of the simulation's consistency with the system. Our V&V calculator encapsulates three different types of V&V procedures to comprehensively evaluate SME statements: (1) formal verification methodology, (2) statistical debugging and (3) exploratory analysis. Here, we present an overview of each of these procedures. Later, we introduce our V&V calculator and describe how it implements these procedures and the controls it provides to SMEs.

2.1. *Formal verification methods*

Formal methods are typically applied to verify that a simulation or conceptual model meets the requirements specified by SMEs. Applying this process requires: (1) unambiguously specifying the requirements and (2) verifying that those requirements are always upheld and never violated in the model and/or simulation.[4] We review these two components of the procedure.

2.1.1. *Formal specification*

Formal specification is the process of describing a system and its desired properties using a language with a mathematically defined syntax and semantics. Some formal specification languages such as Z,[5] VDM[6] and Larch[7] focus on specifying the behavior of sequential systems, where states are described in mathematical structures such as sets, relations and functions. Other methods such as CSP,[8] CCS,[9] Statecharts,[10] Temporal Logic[11] and I/O automata[12] focus on specifying system behaviors in terms of sequences, trees, or partial orders of events. Common to all these methods is the use of the mathematical concepts of abstraction and composition. Unfortunately, this syntax makes specification languages intimidating and unintuitive to most SMEs.

2.1.2. *Model checking*

Model checking is an automated means to check a formal specification for correctness. It requires a user to formulate a property as a predicate over variable values.

Then it automatically checks to see if the property holds in the specification. For example, a property stipulating that a variable x always be positive and that a variable y always be strictly smaller than x can be formulated as $x > 0$ && $y < x$.[13]

Two distinct approaches using properties are used to assess correctness: (1) pre/post condition and (2) invariant assertion. Pre/post condition approaches formulate the correctness problem as the relationship between a formula that is assumed to hold at the beginning of program execution, denoted ΦPRE, and a formula that should hold at the end of program execution, denoted ΦPOST. Assessing the correctness of the program involves determining whether the semantics of the program establishes ΦPOST given ΦPRE. Invariant assertion-based approaches define the correctness of a simulation by verifying that a user-specified Boolean formula, ΦINV, holds throughout simulation execution. While useful, both these approaches require that the entire possible state space of the model or simulation be explored. Unfortunately, this analysis can take hours and even days to complete their analysis, significantly limiting the utility of model checkers.[14]

Our V&V calculator enables SMEs to formally specify statements using a graphical property builder featuring drop-down lists of SME defined variables and natural language descriptions of the resulting property semantics. Then, the calculator performs model checking analysis on only the portion of the state space that the SME identifies to be of interest. This results in analysis that takes seconds as opposed to hours or days. These features ensure that it is accessible and responsive for SMEs across different domains.[15,16]

2.2. *Statistical debugging*

Statistical debugging is an automated fault localization technique used in computer science to aid human developers fix a faulty program. In order to elucidate its inner workings, we apply the process of statistical debugging to isolate the fault shown in Fig. 1. Figure 1 includes the program, mid() and its test suite. The program mid() takes three integers as input and is required to output the median value. The function fails to properly identify the median number for some inputs because there is a fault in Statement 7. Statement 7 should read $m = x$, however it reads $m = y$.

Figure 1 illustrates the process of employing statistical debugging to localize this fault. The debugger begins by executing mid() for each of the test inputs shown at the top of Fig. 1. The execution of mid for each test input is traced to record the statements that are executed. The columns below the test inputs reflect each execution trace: a black dot signifies that the statement was executed, the lack of a black dot signifies that the statement was not executed.

Once mid() is executed for a given test input, the actual output of the program is compared to the specified output. The actual and specified outputs for each test in-put are shown immediately below the execution trace. The actual output is written in italics while the specified output is underlined. These outputs determine if the corresponding execution trace is labeled as passing or failing. If the actual

Program Source Code		Test Inputs						Suspiciousness	Rank
		3, 3, 5	1, 2, 3	3, 2, 1	5, 5, 5	5, 3, 4	2, 1, 3		
`mid() {`		Execution Trace							
	`int x, y, z, m;`								
1	`read("Enter 3 numbers:", x, y, z);`	●	●	●	●	●	●	.16	7
2	`m = z;`	●	●	●	●	●	●	.16	7
3	`if (y < z)`	●	●	●	●	●	●	.16	7
4	`if (x < y)`	●	●			●	●	.25	3
5	`m = y;`		●					.00	13
6	`else if (x < z)`	●				●	●	.33	2
7	`m = y; // THIS IS A BUG!`	●					●	*.50*	*1*
8	`else`			●	●			.00	13
9	`if (x > y)`			●	●			.00	13
10	`m = y;`			●				.00	13
11	`else if ( x > z)`				●			.00	13
12	`m = x;`							.00	13
13	`print("Middle number is: ", m);`	●	●	●	●	●	●	.16	7
`}`		*Actual* vs. Specified							
		3 vs. 3	*2* vs. 2	*2* vs. 2	*5* vs. 5	*4* vs. 4	*1* vs. 2		
		Pass/Fail Label							
		P	P	P	P	P	F		

Fig. 1. Statistical debugging example.

output matches the specified output then the execution trace passes, otherwise it fails. The result of applying this labeling process to each execution trace of mid() is shown in the bottom row of Fig. 1.

Labeling each execution trace as passing or failing enables the suspiciousness and rank of each statement in the source code of mid() to be computed. Recall, suspiciousness measures how likely it is that a statement contains a fault. It is calculated by computing the ratio of the number of failing execution traces that include the statement to the number of total execution traces that include the statement. The suspiciousness of each statement is shown in second rightmost column of Fig. 1. The rightmost column of Fig. 1 shows the rank of each statement. The ranking column reflects the sorted order of the statements in mid() when they are returned to the developer. The rank is the maximum number of statements that would have to be examined by the developer. It is assumed that a developer examines all statements with the same suspiciousness together.

In Fig. 1, Statement 7 is identified as the most suspicious statement (0.50) because it is only included in two execution traces — one that passes and one that fails. Every other statement is included in at least two passing execution traces and no statement is included in more than one failing execution trace. As a result Statement 7 is returned first to a developer tasked with localizing the statement causing the failing output mid(). It is expected that by isolating the statement, the developer will quickly recognize the fault and correct the program. This is the appeal of statistical debugging. It is capable of automatically limiting the number of statements developers must sort through in the debugging process.[17]

In addition to profiling program statements, most statistical debuggers employ conditional propositions, or predicates, to record the values assigned to variables in an execution trace. A very simple example helps elucidate the value of predicates in the statistical debugging process: three predicates for every assignment statement in a program are used to test if a value being assigned to a variable is greater than, less than or equal to zero. In the context of Statement 7 in the program mid(), the three predicates used to test the value of m are: $(m > 0)$, $(m < 0)$ and $(m = 0)$. The suspiciousness of these predicates is calculated in the same manner as the suspiciousness of the statements in mid() computed in Fig. 1. The addition of predicates (including those that are complex than the example shown above) enables, statistical debuggers to analyze the coverage of statements and values in execution traces. In theory and practice this has been shown to improve effectiveness.[18] In Sec. 4.3, we present our V&V Calculator and the number of different predicates it enables a SME to employ in the exploration of simulation outputs.

2.3. *Exploratory analysis*

Sensitivity analysis has been proposed as a methodology to explore the robustness of an output in a simulation.[19] The principle behind sensitivity analysis is to vary the parameters of the model and rerun the simulation. This allows the SME to understand how sensitive the simulation is to the initial parameters. However, even with a small number of parameters, the number of combinations of parameter values quickly becomes large and the resources required to perform the analysis can be excessive. Sensitivity analysis has been refined to exploratory analysis. Exploratory analysis can be viewed as sensitivity analysis done efficiently.[20] Exploratory analysis relies on SME insight to significantly reduce the parameters that need to be explored. Given the reduced parameter set, SME hypothesizes about the extent to which an output will be manifested in the simulation. When the SMEs hypothesis is wrong, additional exploration is required to understand the difference between the expert opinion and the simulation output. Our V&V calculator enables SMEs to test hypotheses involving any combinations of variables represented as conditional propositions in order to explore how each conditional proposition affects the simulation output.

3. The Approach

Our approach assumes that a simulation of the system has been developed and the system specification is provided. We use the formal verification techniques described above to answer verification questions and apply exploratory analysis for validation. The remaining challenge is how to specify the verification requirements, expected conditions and outputs such that they can automatically be tested. In order to address this challenge, we propose a two-step iterative approach.

- **Setup**: Step A deals with specifying the system requirements, expected outputs and conditions of variables within the simulation. The system requirements are formally specified in the form of predicates that should ALWAYS be TRUE or FALSE in the simulation. We refer to testing that a simulation meets all system axioms as Level 1 testing and to axioms as Level 1 statements. We discuss how to construct Level 1 statements in the tool description section. The expected outputs are specified by labeling as TRUE (or 1) outputs that are expected for a given input set and range; outputs that are not TRUE are labeled FALSE (or 0). The expected condition is specified by providing a combination of system inputs that should result in TRUE or expected outputs. We refer to testing that a simulation produces the expected output for a given set of inputs as Level 2 testing and to statements that specify the relationship between inputs and expected outputs as Level 2 statements. We discuss how to create Level 2 statements in the tool section. In order to capture the outputs, we need to execute the simulation over time and replicate runs in the case of a stochastic simulation. In our approach, we recommend a formal design of experiment to specify the behavior space of the simulation and collect data. Specifically, we focus on trace experiments, optimization experiments and uncertainty experiments. Trace experiments provide data specific to entities within the simulation. They are useful if the system specification is at the entity level and the SME wishes to explore the aggregate behavior of the system. Optimization experiments attempt to optimize an objective function. These experiments are useful if the system specification is focused on a specific output and the SME wishes to explore the input and internal conditions of the system. Uncertainty experiments involve sampling the range of experimental values without bias. This type of experiment is useful if the SME is in full experimental mode and wants to test hypotheses by exploring the entire system solution space (input, internal conditions and output) of through the simulation. At the end of Step A, we have a specification of: (1) system requirements, (2) expected system output(s) and (3) expected contribution of how the internal conditions create the expect outputs and a experimental design to generate data. Figure 2 displays Step A.
- **Data Collection and Testing**: In Step B, we run the experiments and collect outputs that we label as TRUE or FALSE based on the specification of Step A. In order for the simulation to pass Level 1 testing, it must satisfy ALL of the Level 1 statements. If the simulation passes Level 1 testing, we proceed to Level 2. Otherwise, the simulation is either over-specified meaning the addition of information necessary to implement the simulation has led to it not being able to support a key requirement; or under-specified meaning the implementation process has resulted in too many simplifications being made. In either case, it must be redesigned and implemented using the same or a different platform or paradigm. At Level 2, we test that the combinations of inputs that lead to TRUE outputs match the Level 2 statements specified by the SME. If a system passes Level 2, it is considered verified and validated against the system specification.

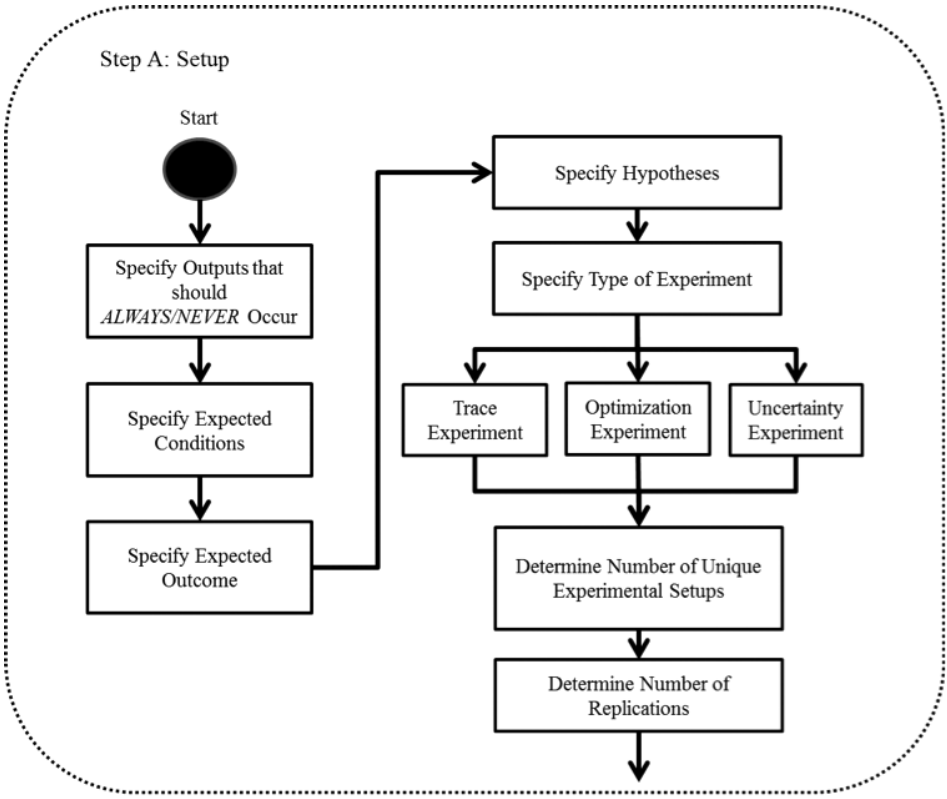

Fig. 2. Step A — Setup. This step of the approach defines the expected output and the single variable, scalar pair and compound predicates that determine if the simulation is valid. The completion of Step A results in a complete specification of the requirements, expected conditions and the expected output of the simulation as well as a design of experiment to ensure that the simulation is able to provide all of the necessary data for conducting Step B.

However, additional interactions of variables within the simulation can generate Level 2 statements that are not part of the specification. The SME can decide to (1) ignore these statements, (2) add controls to eliminate the statements, or (3) verify and validate them to ensure that they do not violate the system specification. If the SME decides to ignore the statements, testing is complete. This could be the case where the system is being designed for mission rehearsal and therefore the SME wants to ensure that the system behaves as expected but does not want to preclude additional outputs from occurring. If the SME decides to add controls, they need to redesign the simulation and add the restrictions that would eliminate the unwanted conditions and resulting outputs. They can use our approach to test the simulation iteratively until all unwanted statements are eliminated. This could be the case for a training system where the SME wants to ensure that only the specifications they wish to train are present. If the SME wishes to verify and validate the additional statements, they can add each statement into the

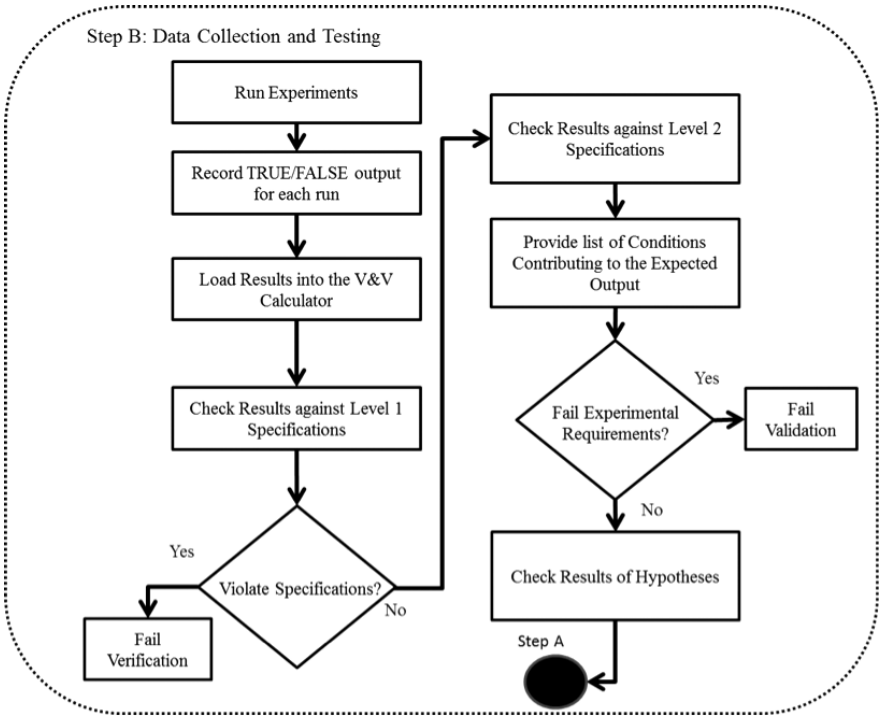

Fig. 3. Step B — Data collection and Testing. This step requires generating the outputs from the simulation and comparing the outputs against the specifications provided in Step A. The SME, as the user of the tool, makes the final decision as to whether the simulation passes the violate Specifications and the Fail Experimental Requirements checks.

system specification; update the list of requirements to ensure that the additional statements are satisfied under the new set of axioms and check that each Levels 1 and 2 statement are verified and validated against the updated specification. This could be the case for an experimental system where the SME wants to explore and improve system design. Figure 3 displays the steps and progression of Step B. In the next section, we discuss how the V&V calculator implements this approach.

4. The V&V Calculator

Given the statements generated by following our approach, the V&V calculator, shown in Fig. 4, allows a SME to verify the simulation requirements (Level 1) and validate the variable interactions causing an expected output (Level 2). Three components define the SME interaction with the calculator: the log file, verification of requirements and validation of expected outputs. Here, we describe each component and how it enables the SME to V&V the simulation. In Sec. 5, we apply our approach and the calculator to a use case to demonstrate its effectiveness.

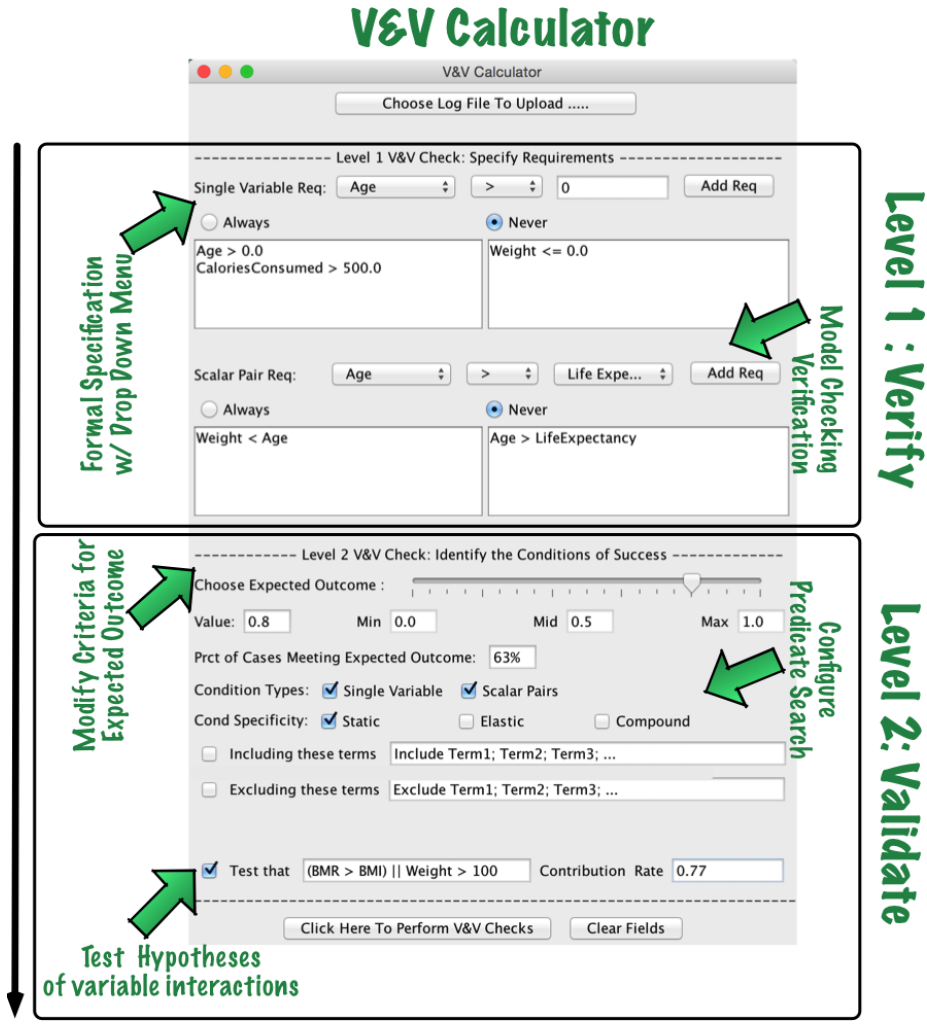

Fig. 4. The V&V calculator. The separation between the Levels 1 and 2 specifications of the tool are shown in the figure. Level 1 includes the specification of single variable and scalar predicates. Level 2 consists of specifying the expected output, the condition type, condition specificity, terms to include or exclude and the formation of hypotheses.

4.1. *The log file*

The V&V calculator requires the SME to create a comma separated value (CSV) log file containing the values of the variables and outputs within the simulation over a set of test cases that define the simulation's use. The CSV log file can reflect a standard experimental design where the values of simulation parameters and results are recorded for the test cases. However, it can also reflect a trace experiment where the file includes the values of variables collected over different times throughout the

execution of the simulation. In the general case, the CSV log file includes an initial row describing: (1) the name of every variable whose value will be logged and (2) the name of the simulation output that will be logged. Following the initial row, the file includes some number of rows containing the value of each variable and the output.

We do not view the generation of CSV log file as a significant burden for SMEs because: (1) a myriad of automated tools exist to create such a file and (2) this expectation is tantamount to requiring a record of input and output for the use cases that define the simulation. However, we do understand that one of the contributions of our work is a reduced burden for SMEs in the V&V process. As a result, we highlight this as the heaviest burden placed on a SME using the V&V calculator.

The SME's interaction with the V&V calculator begins when she/he loads the CSV generated log file into the calculator. This fills in all of the calculator menus with variable names included in the CSV log file. The instance of the V&V calculator shown in Fig. 4 has a log file loaded into it.

4.2. *Verification (Level 1) in the V&V calculator*

The specification and subsequent evaluation in the V&V calculator of the Level 1 requirements use a combination of conditional propositions for formal specification and model checking for evaluation analysis. We examine each in the next sections.

4.2.1. *Statement specification*

All requirements generated by our approach are specified as conditional propositions known as *predicates*. Each Level 1 requirement can either be expressed as a single variable or scalar pairs predicate. Single variable predicates evaluate if the value of a variable ever rises above, falls below or deviates from some SME specified threshold. In general, a single variable predicate takes the form: $x > \Phi, x \geq \Phi, x = \Phi, x \leq \Phi$ or $x < \Phi$, where x is a CSV log file variable and Φ is a SME specified threshold. In contrast, a scalar pairs predicate evaluates if the value of a variable ever rises above, falls below or deviates from another variable. In general, a scalar predicate takes the form: $x > y, x \geq y, x = y, x \leq y$ or $x < y$, where x and y are CSV log file variables. In support of our goal of reducing the burden placed on the SME during V&V the names of all the variables and operators (i.e., $>, \geq, =, \leq, <$) are loaded into the drop-down menus that enable SMEs to build single variable and scalar pairs predicates to evaluate for consistency in the most straightforward manner possible. For each predicate representing a requirement a SME chooses if the conditional proposition must ALWAYS be true in the simulation or if the conditional proposition must NEVER be true in the simulation. Recall from Sec. 3, these statements reflect the SME requirements of the system and the simulation.

4.2.2. *Limited scope model checking analysis*

Once all the requirements have been specified as predicates, the calculator evaluates them to see if they are upheld in every test case included in the CSV log file. Recall, a test case is reflected by one row of variable values in the CSV log file. In general, model checking analysis is intractable for most real-world simulations because it explores the entire space of possible states the simulation can reach. As a result, even for a trivial simulation, the analysis usually requires hours or days to determine if the specifications (like the predicates specified in our calculator) are upheld in a simulation. However, the model checking analysis performed by our V&V calculator limits its exploration to only those states represented in the test cases included in the CSV log file. As a result it is able to determine in seconds if the predicates specified by the SME are upheld in all of the logged test cases.

While our analysis does not reflect a full simulation state space exploration it does cover those conditions, which define how the SME will use the simulation to reflect the system. In this sense, our model checking analysis is exhaustive for the SME's use of the simulation. For each statement, the calculator applies the limited scope model checking analysis and provides feedback to the SME by recording the results in a plaintext file. The file includes the predicate the SME specified and the result — whether the predicate was UPHELD or the predicate was VIOLATED. The SME uses this feedback to adjust their modeling question, conceptual model, or simulation with the process described in Sec. 3.

4.3. *Validation of expected outputs*

The configuration and subsequent analysis performed in the V&V calculator to validate an expected output of the simulation is intimately related to the statistical debugging technique we presented in Sec. 2. Validating an expected output requires identifying those interactions of variables that contribute to it the most. As a means to this end, our V&V calculator adapts statistical debugging predicates and analysis for those variables included in the CSV log file.

4.3.1. *Defining the expected output*

Our V&V calculator enables SMEs to easily define an expected output through the use of a slider that spans the minimum and maximum value recorded for the simulation output in the CSV log file. Using the slider, the SME chooses a cutoff value Ω. All test cases in the CSV log file with a simulation output value greater than or equal to Ω achieve the expected output. By the same definition, all test cases with output values less than Ω do not achieve the expected output. Immediately following the SMEs specification of an expected output our V&V calculator provides the percent of the test cases that achieve the expected output defined by Ω. The output definition slider and the text field providing the SME with the number of test cases achieving the expected output are shown in Fig. 4.

4.3.2. *Configuring predicates*

Following the definition of an expected output, the SME configures the types of predicates, that will be used to explore variable interactions to identify those that contribute the most to expected outputs. The V&V calculator provides two different types of predicates (single variable, scalar pairs) at three different levels of specificity (static, elastic and compound) to evaluate the contribution of variable interactions to an expected output. The different specificity levels and types of predicates are established within the statistical debugging community where they have been shown to be effective for localizing software faults.[17]

The variables, as well as the values that they are checked against, that form the static predicates are specified by the SME before performing any analysis of the CSV log file. For example, static predicates include the single variable predicate $x > 0$ and the predicate $x > y$, where x and y are variables. This distinction is made because the decision to compare the value of x and the difference between the values of x and y to 0 is made without considering any of the values x and y obtain in the CSV log file. In general, the V&V calculator tests three single variable predicates for each variable $x(x > 0, x = 0$ and $x < 0)$ and three scalar pairs predicates for each pair of variables x and $y(x > y, x = y$ and $x < y)$.

While the variables captured in the elastic predicates used in the V&V calculator are determined before performing any analysis of the CSV log file, the values those variables are compared against are not. Elastic predicates require the V&V calculator to compute the mean and standard deviation of: (1) the distribution of each variable and (2) the mean and standard deviation of the distribution of the differences between each pair of variables. The mean and standard deviation of each variable is used to define the following three single variable predicates for a variable x, mean μ_x and standard deviation $\sigma_x : x > \mu_x + \sigma_x, \mu_x + \sigma_x > x > \mu_x - \sigma_x$ and $x < \mu_x - \sigma_x$. These three predicates reflect values of variable x that are well above their normal value, within their normal range of values and well below their normal value. The mean and standard deviation of the difference between two variables x and y are used to define the following three scalar pairs predicates for two variables x and y, with the mean of their difference μ_{x-y} and the standard deviation of their difference $\sigma_{x-y} : x - y > \mu_{x-y} + \sigma_{x-y}, \mu_{x-y} + \sigma_{x-y} > x - y > \mu_{x-y} - \sigma_{x-y}$ and $x - y < \mu_{x-y} - \sigma_{x-y}$. These three predicates reflect differences between variables x and y that are well above their normal value, within their normal range of values and well below their normal value.

Compound predicates reflect any combination of two static and elastic predicates that can be composed using the logical operators AND (&&) and OR ($\|$). In general, the V&V calculator tests two compound predicates for every two predicates P and Q $(P \&\& Q, P \,\|\, Q)$. Compound predicates provide an additional level of specificity for the purposes of identifying variable interactions that contribute to the expected output.

Recall from Sec. 2 that the suspiciousness of a statement is measured by computing the ratio of the number of failing test cases the statement appears into the total number of test cases the statement appears in. We adopt this metric to compute the contribution rate of each predicate configured by the SME. The contribution rate of a predicate is measured by computing the ratio of the number of times the predicate is true in test cases that produce an expected output to the total number of test cases the predicate is true in. Once the SME has configured the predicates they want to be explored, the V&V calculator uses the test cases within the log file to compute the contribution rate of each configured predicate. The predicates are sorted by contribution rate and provided in a plaintext file to the SME as feedback. The percent of test cases that the predicate is true in is also provided to enable the validation of the expected output.

It is important to note that the contribution rate of a predicate is not tied to any statistical confidence level. When sampling, different sets of individuals can be randomly selected from the same population; and each set of individuals can often produce a different confidence interval. A confidence level refers to the percentage of all possible samples that can be expected to include the true population parameter.[21] In our analysis, the SME provides the log file. This serves as the sample of the population of inputs possible to the simulation and the foundation for computing contribution rate of predicates. Unfortunately, we do not if the sample is random, biased, complete or partial. As a result, we cannot compute statistical confidence levels for the contribution rates of predicates. In future work, we will explore the use of importance sampling to address this deficiency.

4.3.3. *Testing SME hypotheses in the V&V calculator with custom predicates*

The final component of the V&V calculator is the specification and evaluation of SME hypotheses about variable interactions related to the expected output. As shown in Fig. 2, SME hypotheses take the form of a custom predicate and the calculator will determine the contribution rate of the custom predicate. The predicate can be composed of any number of single variable or scalar pairs predicates combined by the AND (&&) and OR (||) logical operators. While this capability allows for too many possibilities to enumerate in general, the following conditional propositions are all valid SME hypotheses where (1) a, b, c, d and e represent potential variables contained within a log file, (2) 70, 60 and 12 represent static comparison values specified by an SME and (3) the selected combination of logical operators are chosen arbitrarily for illustration purposes:

- $((a > b) \,\|\, (b < 70))\&\&((c < d) \,\|\, (d < e))$
- $((a > b) \,\|\, ((c < 70)\&\&(d > 60)))\&\&(e > 12)$
- $((a < b)\&\&(b < c)) \,\|\, ((c > d)\&\&(d > e))$.

Once the SME has specified the predicate portion of their hypothesis, they must specify the contribution rate they believe the variable interactions captured in the

predicate have the expected output. Together the predicate and the hypothesized contribution rate of the predicate form the SME's hypothesis. Finally, the V&V calculator computes the actual contribution rate of the predicate and provides the predicate, the SME's hypothesized contribution rate and the actual contribution rate of the predicate to the SME as feedback in a plaintext file. This capability enables SME to validate the expected output with respect to variable interactions that could not be constructed using the static, elastic, or compound predicates provided by the calculator. For the first example of hypothesis given above, the V&V calculator checks the log file for every occurrence where either predicate $(a > b)$ OR predicate $(b < 70)$ is true while simultaneously either predicate $(c < d)$ OR predicate $(d < e)$ is true. Remember that these values are obtained from the log file which contains static values for each variable every time that the simulation logs its values. Then, the V&V calculator measures the predicate's contribution to the output of interest by computing the ratio of the number of times that the predicate is true in cases that produce an expected output to the total number of cases that the predicate is true. The hypothesis can then be adjusted to test variations of the predicates (i.e., change $(b < 70)$ to $(b < 65)$ or $(c < d)$ to $(c < 50)$) to try and gain additional insight into how the predicate affects the output of interest.

In this sense the V&V calculator, serves as a means for SMEs to perform Turing tests on the variable interactions within their simulation. As originally proposed, the Turing test is a test of a machine's ability to exhibit intelligent behavior that is indistinguishable from that of a human.[22] The test was adapted for M&S validation by asking a SME if they can discriminate between the output of the simulation and the actual system.[23,24] If the SME cannot discriminate between the two outputs the simulation is considered valid. The SME hypothesis capability of the V&V calculator enables a similar validation but instead of focusing on output it targets differences between variable interactions in the simulation and in the system and/or conceptual model.

5. Use Case Evaluation

To illustrate the use of the V&V calculator and its applicability for verifying and validating simulations, we apply our tool to the following use case. We wish to construct a system that will allow us to predict the level of obesity in a population given physiological, environmental and behavioral conditions. Behavioral conditions include eating habits, physical activity levels, workplace physical activity requirements, travel methods and health consciousness. Environmental conditions include the types of food available, the size of the area and the distribution of restaurants, workplaces and homes. Physiological components include the height, weight, age, life expectancy and the gender of individuals. We also wish to use this system to study the impact of policy on reducing obesity levels in the future and therefore must ensure that it is a valid representation of how people gain and lose weight.

In order to help use design the system, we decide to apply M&S and first create a model and simulation of obesity. The obesity model tracks the weights of individuals over time and classifies them as severely underweight, underweight, normal, overweight, obese, severely obese, morbidly obese and super obese based on their body mass index (BMI) values. The gain or loss of weight occurs through each individual's weekly balance of calories. We use the basal metabolic rate (BMR) to determine the number of calories that an individual needs to remain at its current weight based on its current height, weight and age.[25,26] The BMR value minus the total calorie gain during the week produces the net change in calories for each individual. A positive net calorie gain results in weight gain, a negative net caloric gain results in weight loss and a zero net calorie gain results in a static weight. We represent each person within the simulation as a separate agent with behaviors and geographic distribution. The simulation explores the prevalence of obesity through the aggregate combination of all of the agents within the simulation.

In order to use the V&V calculator, we must provide: (1) Level 1 predicates that check if the requirements of the conceptual model are *ALWAYS* TRUE in the simulation and (2) Level 2 predicates that describe the extent to which the variables which create the expected output of the simulation match the SME's conceptual model. In the next sections, we present our use of the V&V calculator to verify and validate the obesity model.

5.1. *Defining the experiment and creating the CSV log file*

In order to capture data, we conduct a Latin hypercube sampling experiment to evenly sample the parameter space for possible combinations of input values in the obesity model. Since, our goal is to V&V the simulation of a system that will be used for prediction; we want to ensure that our sample of the parameter space is not biased. We design our experiment to generate 24 unique stochastic simulation test cases that are each run with 50 replications. This provides us with 1200 total test cases to construct our CSV log file. Table 1 provides a sample of a CSV log file generated by the Obesity Model.

5.2. *Specifying and verifying the representation of the conceptual model* (*Level* 1 *statements*)

Once our log file has been created we construct the list of requirements for conditions that must hold TRUE for all simulation runs of the obesity model. Since the system represents a population in a living environment, some initial input parameter values related to the population, workplaces, recreation facilities and the passage of time should ALWAYS be positive. Additionally, we require the simulation to initialize a population where all individuals are adults between the age of 18 and 65 and no births occur. All of these statements are specified as single variable predicates for the tool. Table 2 provides a sample list of predicates for the obesity model.

In addition, we have system specifications that relate components of the system to one another. For instance, the restaurants in the system offer dishes for the population from three different categories: (1) a *market* option that provides lower calorie meals; (2) a fast food option that provides high calorie meals and (3) a nonfast food option that provides medium calorie meals. Within each option there are higher and lower caloric choices where the exact calorie level is represented as distribution with a minimum, maximum and mode value. In every case, the minimum value must be less than the mode value which must be less than the maximum value. This results in 27 additional scalar predicates for the V&V calculator to check. Table 3 provides a sample list of scalar predicates to check with the tool.

The Level 1 statements captured in Tables 1 and 2 are input to calculator. The calculator completes its limited scope model checking analysis in less than 10 s for our model and provides the SME feedback for all the requirements. The next part of the setup process involves specifying Level 2 statements in the form of expected outputs for the SME's conceptual model and mapping them to interactions of variables.

5.3. *Specify the expected output and validate the contribution of the internal conditions*

Next, we specify the expected output of the obesity model. In this use case, we are interested in validating the obesity model for a population where no more than 30% of the individuals are obese. The expected output is specified using the slider shown in Fig. 4. Along with specifying the expected output, we also specify how the interactions of variables within the system will achieve the expected

Table 1. A sample of the CSV log file generated by the obesity model.

Population	Restaurants	Percent 18 to 65	Percent market	Total_percent_ BMI_levels	Area size	Percent BMI > 30: Year 10	Output
4870	91	100	79	100	37	0.143422	0
10,000	40	100	59	100	3	0.370453	1
1180	67	100	77	100	67	0.249375	0
9190	28	100	61	100	6	0.446734	1
3520	74	100	32	100	15	0.321787	1
7209	31	100	10	100	7	0.06875632	0

Note: The top row of the log file displays the parameter names from the obesity model. The first six columns provide the output data generated from five simulation test cases of the obesity model. Columns 1 and 2 are the total number of people and restaurants within the simulation. Column 3 presents the percentage of the population that has an age between 18 and 65 years old. Column 4 presents the percentage of the total restaurants that are markets. Column 5 provides an example of a compound predicate result that we preemptively convert to a single variable. Column 6 presents the size of the simulated environment in square miles. Column 7 displays the value of the simulation output and the final column presents the conversion of Column 7 into a 0 or 1 depending on if the value equals or exceeds 0.30 (30% obese).

Table 2. Sample of Axioms in the form of single variable predicates.

Population > 0 Work place > 0 Housing Units > 0 Rec Facilities > 0 Area Size > 0 Avg Travel Time to Work > 0 Max Travel Time to Rec Facility > 0	There is ALWAYS at least one person in the system and everyone has a place to live work and exercise. It takes time to move from place to place.
Percent Male ≤ 100 Percent Male ≥ 0 Percent Female ≥ 0 Percent Female ≤ 100	There is ALWAYS a mix of male and female in the system.
Restaurants > 0	People can eat at home or out and there is ALWAYS at least one place to eat out.
Percent Normal ≥ 0 Percent Underweight ≥ 0 Percent Obese ≥ 0 Percent Overweight ≥ 0 Percent Eating Habits Low Calorie ≥ 0 Percent Eating Habits Medium Calorie ≥ 0 Percent Eating Habits High Calorie ≥ 0	The system ALWAYS starts with a mixed population.
Percent 18 to 65 $= 100$ Annual Number of Births $= 0$	The system ONLY considers adult obesity.

Table 3. Simulation outputs that should ALWAYS occur in the form of scalar predicates.

Market_highCal_min < Market_highCal_max	FastFood_highCal_min < FastFood_highCal_max	NonFastFood_highCal_min < NonFastFood_highCal_max
Market_highCal_min <= Market_highCal_mode	FastFood_highCal_min <= FastFood_highCal_mode	NonFastFood_highCal_min <= NonFastFood_highCal_mode
Market_highCal_max $\geq$ Market_highCal_mode	FastFood_highCal_max $\geq$ FastFood_highCal_mode	NonFastFood_highCal_max $\geq$ NonFastFood_highCal_mode
Market_medCal_min < Market_medCal_max	FastFood_medCal_min < FastFood_medCal_max	NonFastFood_medCal_min < NonFastFood_medCal_max
Market_medCal_min <= Market_medCal_mode	FastFood_medCal_min <= FastFood_medCal_mode	NonFastFood_medCal_min <= NonFastFood_medCal_mode
Market_medCal_max $\geq$ Market_medCal_mode	FastFood_medCal_max $\geq$ FastFood_medCal_mode	NonFastFood_medCal_max $\geq$ NonFastFood_medCal_mode
Market_lowCal_min < Market_lowCal_max	FastFood_lowCal_min < FastFood_lowCal_max	NonFastFood_lowCal_min < NonFastFood_lowCal_max
Market_lowCal_min <= Market_lowCal_mode	FastFood_lowCal_min<= FastFood_lowCal_mode	NonFastFood_lowCal_min <= NonFastFood_lowCal_mode
Market_lowCal_max $\geq$ Market_lowCal_mode	FastFood_lowCal_max $\geq$ FastFood_lowCal_mode	NonFastFood_lowCal_max $\geq$ NonFastFood_lowCal_mode

output. This is done by configuring the V&V calculator to generate predicates and testing the contribution rate of the predicates against the SME expectation. Once the SME has specified predicates capturing the interactions of interest she/he: (1) employs the V&V calculator to score the contribution rate of each predicate

and (2) checks to see if the contribution rate of the predicate matches his/her expectation.

For the obesity model, where no more than 30% of the population is obese, the contribution rate of 0.82 for the following predicate did not match the expectation of the SME: *Percent underweight* >90&& *Percent eating habits low calorie* >90. The predicate states that to achieve a population where no more than 30% of the individuals are obese the simulation frequently employs a parametrization where almost all the agents are underweight and have low-calorie eating habits. The SME expected the combination of these conditions to result in a population where no more than 5% of the population was obese, not an obesity rate that included up to 30% of the population. As a result, they hypothesized a very low contribution rate (0.10) for the predicate.

Based on this mismatch of expectation, we performed further exploration and revealed a bug in the simulation that allowed agents to eat more than three meals a day. The additional meals created additional opportunities for weight gain, even for agents who initially were underweight and maintained low-calorie eating habits. Given this bug, simulation runs where the vast majority of the population was initially underweight and had low-calorie eating habits frequently resulted in 25–30% of the population being obese as opposed to <5%. While this bug appears straightforward once identified, it had been present and undiscovered in this simulation for over a month. Many bugs are obvious only once one knows where to look. The V&V calculator identified a condition, which led us to the lines of code where the expected output of the simulation was being generated by inner workings, which did not match the system or the conceptual model.

6. Related Work

Our proposed approach complements recent developments in verification, validation and testing (VV&T). Verification reflects confirming that a simulation accurately represents the developer's conceptual model. Validation reflects the degree to which a simulation is an accurate representation of the real-world from the perspective of intended use. Both definitions stress checking via comparison to a reference standard. However, verification checks that the model is correctly transformed into the simulation, while validation focuses on behavioral and representational accuracy. It measures the degree to which a model is an accurate representation of the real-world from the perspective of the intended use. Testing compares an expected result against model output and is inherent in simulation V&V.[27]

The goal of the V&V calculator is to provide one cohesive procedure for SMEs to meet their VV&T objectives as opposed to relying on their own *ad hoc* approaches and best estimates. In this sense, the tool provides a set of VV&T standards for SMEs to evaluate their models. Such a set of standards was proposed in recent research on a unified theory of validation.[28] Furthermore, as researchers continue to make progress in developing state of the art VV&T practices, we can

and will include them in our proposed approach to facilitate the VV&T efforts of SMEs.

Given this complementary relationship we review work related to both V&V. Simulation verification techniques are dependent on simulation implementation languages. Simulation-specific languages (Arena, Simulink) entail verification via structured walkthroughs of the code and program traces that are tested against expected results. These techniques check that the language, the pseudo-random number generator and the simulation are correct.[24] Implementing a simulation in a general-purpose programming language (Java, C/C++) allows for model checking to be employed. Model checking ensures that each requirement specified in the design of the simulation is satisfied in the implementation.[29] This analysis is powerful but model-checking tools often have a steep learning curve.

Simulation validation techniques are both subjective and objective. Subjective ones rely on the judgment of SMEs while objective techniques use statistical tests. Subjective techniques include face validity and Turing tests where SMEs assess if the simulation output is a reasonable facsimile of the real system.[22–24] While, objective tests include: (1) exploratory and sensitivity analysis to statistically test if the simulation inputs have the same effect on the outputs as they do in the real system,[30–32] and (2) stability analysis to quantify the variance of a simulation output.[33] Our proposed approach combines these two types of techniques by providing SMEs with an automated objective measure, contribution rate, to evaluate subjective hypotheses about variable interactions within their simulation.

7. Conclusion

M&S is an essential part of system development. Unfortunately, needed information about the system is frequently lost and additional extraneous information is added to the description of the system as it is conceptually modeled and implemented in a simulation. In this paper, we addressed this problem with an approach to formally verify and rigorously validate a simulation against the specification of the real system. We implemented the approach in a V&V calculator tool that takes as input a set of statements that capture the requirements, internal conditions of the system and expected outputs of the real system and produces as output whether the simulation satisfies the requirements, faithfully represents the internal conditions of the system and produces the expected outputs. We provided a use case to show how SMEs can apply the tool to their own simulations.

References

1. Spiegel M., Reynolds P. F. Jr., Brogan D. C., A case study of model context for simulation composability and reusability, *Proc. Winter Simulation Conf., 2005*,. IEEE, 2005.
2. Johnson C. W., What are emergent properties and how do they affect the engineering of complex systems? *Reliab. Eng. Sys. Saf.* **91**(12):1481, 2006.

3. Tolk A., Diallo S. Y., Padilla J. J., Herencia-Zapana H. Reference modelling in support of M&S—foundations and applications, *J. Simulation* **7**(2):69–82, 2013.
4. Bowen J., Stavridou V., Safety-critical systems, formal methods and standards, *Softw. Eng. J.* **8**(4):189–209, 1993.
5. Spivey J. M., *Understanding Z: A Specification Language and Its Formal Semantics*, Vol. 3 (New York: NY, Cambridge University Press, 1988).
6. Jones C. B., *Systematic Software Development Using VDM*, Vol. 66, Prentice Hall International, 1986.
7. Guttag J. V., Horning J. J., Garland S. J., Jones K. D., Modet A., Wing J. M., Larch: Languages and tools for formal specification. In *Texts and Monographs in Computer Science*, Springer-Verlag, 1993.
8. Hoare C. A. R., Communicating sequential processes, *Commun. ACM* **21**(8):666–677, 1978.
9. Milner R., *A Calculus of Communicating Systems*, Springer-Verlag, New York, 1982.
10. Harel D., Statecharts: A visual formalism for complex systems, *Sci. Comput. Program.* **8**(3):231–274, 1987.
11. Lamport L. The temporal logic of actions, *ACM Trans. Program. Lang. Syst.* (*TOPLAS*) **16**(3):872–923, 1994.
12. Lynch N. A., Tuttle M. R., Hierarchical correctness proofs for distributed algorithms, *Proc. 6th Annual ACM Symp. Principles of Distributed Computing*, ACM, pp. 137–151, 1987.
13. Clarke E. M., Grumberg O., Peled D. A., *Model Checking*, MIT Press, 2000.
14. Burch J. R., Clarke E. M., McMillan K. L., Dill D. L., Hwang L. J., Symbolic model checking: 1020 states and beyond, *Inf. Comput.* **98**(2):142–170, 1992.
15. Gore R., Diallo S., The need for usable formal methods in verification and validation, *Proc. 2013 Winter Simulation Conf.* (*WSC*), *IEEE*, pp. 1257–1268, 2013.
16. Gore R., Diallo S., Padilla J., Conce V. E., Conceptual modeling and formal validation for everyone, *ACM Trans. Model. Comput. Simul.* (*TOMACS*) **24**(2):12, 2014.
17. Gore R., Reynolds P. F. Jr., Kamensky D., Diallo S., Padilla J., Statistical debugging for simulations, *ACM Trans. Model. Comput. Simul.* (*TOMACS*) **25**(3):16, 2015.
18. Liblit B., Cooperative debugging with five hundred million test cases, *Proc. 2008 Int. Symp. Software Testing and Analysis*, *ISSTA '08*, ACM, New York, NY, USA, pp. 109–120, 2008.
19. Saltelli A., Annoni P., Azzini I., Campolongo F., Ratto M., Tarantola S., Variance based sensitivity analysis of model output: Design and estimator for the total sensitivity index, *Comput. Phys. Commun.* **181**(2):259–270, 2010.
20. Davis P. K., Dealing with complexity: Exploratory analysis enabled by multiresolution, multiperspective modeling, *Proc. 32nd Conf. Winter Simulation,* Society for Computer Simulation International, 2000.
21. Box G. E. P., Hunter W. G., Hunter J. S., Significance tests and confidence intervals for means, variances, proportions, and frequencies. In *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*, New York, NY: John Wiley & Sons, pp. 107–152, 1978.
22. Saygin A. P., Cicekli I., Akman V., Turing test: 50 years later, *The Turing Test*, Springer Netherlands, pp. 23–78, 2003.
23. Colby K. M., Hilf F. D., Weber S., Kraemer H. C., Turing-like indistinguishability tests for the validation of a computer simulation of paranoid processes, *Artificial Intelligence* **3**:199–221, 1972.
24. Sargent R. G., Verification and validation of simulation models, *Proc. 2005 Winter Simulation Conf.*, IEEE, pp. 130–143, 2005.

25. Harris J. A., Benedict F. G., A biometric study of human basal metabolism, *Proc. Nat. Acad. Sci. USA* **4**(12):370, 1918.
26. Roza A. M., Shizgal H. M., The Harris Benedict equation reevaluated: Resting energy requirements and the body cell mass, *Am. J. Clin. Nutr.* **40**(1):168–182, 1984.
27. Balci O., Verification validation and accreditation of simulation models, *Proc. 1997 Winter Simulation Conf.*, IEEE, pp. 135–141, 1997.
28. Bair L. J., Tolk A., Towards a unified theory of validation, *Proc. 2013 Winter Simulation Conf., Simulation Conf.* (*WSC*), IEEE, pp. 1245–1256, 2013.
29. Berard B., Bidoit M., Finkel A., Laroussinie F., Petit A., Petrucci L., Schnoebelen P., *Systems and Software Verification: Model-Checking Techniques and Tools*, Springer Publishing Company, 2010.
30. Saltelli A., Chan K., Scott E. M., *et al.*, *Sensitivity Analysis*, Vol. 134, Wiley, New York, 2000.
31. Barton P. I., Lee C. K., Modeling, simulation, sensitivity analysis, and optimization of hybrid systems, *ACM Trans. Model. Comput. Simul.* (*TOMACS*) **12**(4):256–289, 2002.
32. Birta L., Ozmizrak F., A knowledge-based approach for the validation of simulation models: The foundation, *ACM Trans. Model. Comput. Simul.* (*TOMACS*) **6**(1):76–98, 1996.
33. Highman D. J., An algorithmic introduction to numerical simulation of stochastic differential equations, *SIAM Rev.* **43**(3):525–546, 2001.