Computers and M&S – Modeling and Simulation 101

David A. Cook, Ph.D.
Shim Enterprise Inc.

Software engineers have long viewed simulation as a tool to help them understand and develop complex systems. Unfortunately, some software practitioners focus solely on simulation failing to understand that modeling is a necessary foundation for a successful simulation. Before a simulation can be created and examined, the system being simulated must be mathematically modeled, then verified and validated. These two concepts, modeling and simulation, are both necessary and inherently inseparable.

One of my favorite definitions of simulation itself incorporates the concept of modeling and simulation as indivisible. Simulation is "the process of designing a computerized model of a system (or process) and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies for the operation of this system [1]." Simply put, a simulation allows you to develop a logical abstraction (an object), and then examine how an object behaves under differing stimulus.

There are several distinct purposes for simulation. One is to allow you to create a physical object such as a jet engine, as a logical entity in code. It is practical (and faster) to develop a code simulation for testing engine design changes. Changes to the engine can then be implemented, tested, and evaluated in the simulation. This is easier, cheaper, and faster than creating many different physical engines, each with only slightly different attributes.

Other uses of simulation are to create computer-based programs that can model complex systems such as an airport. Since it is impossible (or at least wildly impractical) to construct an actual working airport to test changes in operations, a computer simulation of an airport allows you to view the effect of changes in fueling, landing patterns, or take-off timing.

Unfortunately, before a simulation can be of benefit a model of the system must be developed that allows the simulation developer to construct the computer-based simulation. Modeling is the first step—the very foundation of a good simulation.

The Modeling Phase

You must have a model prior to creating a simulation. Modeling is an attempt to precisely characterize the essential components and interactions of a system. It is a "representation of an object, system, or idea in some form other than that of the entity itself [2]." In a perfect world, the object of a simulation (whether it be a physical object such as a jet engine or a complex system such as an airport) would have precise rules for the attributes, operations, and interactions. These rules could be stated in natural language, or (more preferably) in mathematical rules. In any case, a successful model is based on a concept known as *abstraction*, a technique widely used in object-oriented development. "The art of modeling is enhanced by an ability to abstract the essential features of a problem, to select and modify basic assumptions that characterize the system, and then to enrich and elaborate the model until a useful approximation results [3]."

However, we do not have a perfect world. Parts of the simulation might not have well known interactions. In this case, part of simulation's goal is to determine the real-world interactions. To make sure that only accurate interactions are captured, the best method is to start with a simple model, and ensure that it is correct and representative of the real world. Next, increase the interactions and complexity iteratively, validating the model after each increment. Continue adding interactions until an adequate model is created that meets your needs.

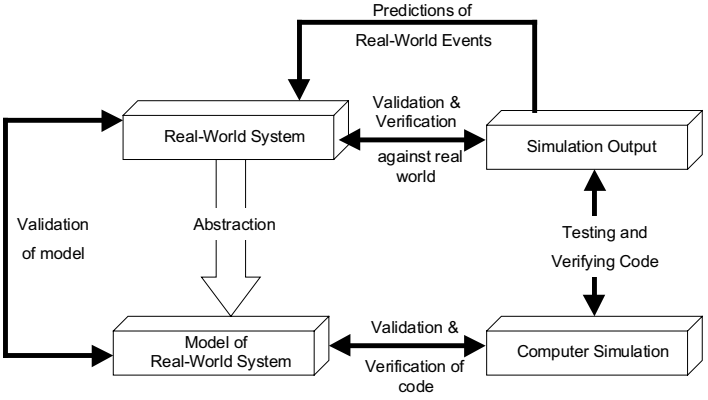
Unfortunately the previous description implies that you have clearly identified needs. This requires valid requirements. It also requires planning for validation of the model. As in creating any software product, requirements and needs must be collected, verified, and validated. These steps are just as important in a simulation as they are in any system. A system that is unvalidated has not been *field-tested* against the real world and could produce invalid results. Abstraction and validation are equally necessary to create a reliable model that correctly reflects the real world, and also contains all attributes necessary to make the model a useful tool for prediction.

The steps of abstraction and validation are in themselves, however, not totally sufficient to create a valid and usable model. Other steps are necessary to create a model that is of sufficient detail to be useful. These steps include planning, data acquisition, and model translation. Figure 1 shows how these steps are related. For a more detailed explanation, the reader may consult [4].

If the model is to be credible and a predictor of future behavior, it is critical that you validate it. This validation must take place at two different times. First of all, you must validate the model against the real world [5]. In addition, you must also validate the model again after the simulation code has been created, as Banks and Carson recommend. At this time, the simulation output will help you revalidate the model against the real world. However, my experience is that this second validation (after coding) tends to focus on validating the code against the model, with little emphasis on revalidating the model against the real world.

It is my opinion that valid software engineering practices dictate an emphasis on reviewing and validating the model prior to and also after coding. Even if sufficient tools and output artifacts

Figure 1. *Integration of Simulation and Software Engineering Processes*



are missing, the advantages of validation on the model prior to coding would outweigh the time and effort involved. A reevaluation of the validity of the model after coding is required. A. Davis looks at studies that show that the most effective way to find errors in software is to inspect it [6]. He further says that one study " … puts to rest the myth that we should not spend time analyzing nonexecutable forms of the software (for example, requirements and design) because the computer can more easily find errors when it executes the program." The model should be reviewed and inspected thoroughly—before *and* after coding.

The creation of a valid model requires domain experts who understand the workings of the physical system. Software engineers should note that domain expertise is a requirement for a valid model. The domain experts must be available not only to help create the model, but also for all phases of verification and validation. If domain expertise is absent during verification and validation, only verification will be performed. This will result in a system that is consistent and internally correct, but also one that might not actually correspond to (and therefore cannot be used as a reliable predictor of) the real world.

The Simulation Phase

The model is next transformed into code once it has been validated. Several methods exist to do this using either a specialized simulation language or a general-purpose language. Specialized simulation languages, such as GASP, SLAM, SIMSCRIPT, or SIMULA offer programming constructs designed to quickly convert a model into coding. Examples of such coding constructs are queues, random-number generation, event scheduling, event generation, built-in statistical analysis collection tools, and time management tools.

Drawbacks of the specialized language approach include limited availability of hardware and trained programmers, and general drawbacks associated with commercial off-the-shelf (COTS) software. In my opinion, the main drawback of specialized simulation languages is the general lack of flexibility in terms of modifying the system to operate on varying (dynamic) combinations of hardware. In general, special-purpose languages operate on a single hardware platform. Distributed simulations are not possible.

General-purpose languages, such as Ada, C, C++, or even Fortran are suitable for implementing simulations. With the use of predefined packages, templates, or subroutine libraries all of the constructs available in the specialized languages can be achieved in general-purpose languages. In addition, developers can take advantage of language features to distribute complex simulation across multiple platforms. Platforms can be homogenous (all similar hardware) or heterogeneous (dissimilar hardware). Distributing a simulation is important, because simulation can be an extremely time-consuming computational process [7]. Five methods exist for speeding up lengthy simulation runs [8]:

1. Using a compiler or compiler-based optimization technique to produce code that can be executed in parallel.
2. Running separate simulations on distinct systems in parallel, and then combining the results.
3. Using different systems for separate subroutines to support a larger distributed simulation.
4. Having a single computer act as a controller, and distributing simulation events to any available processor.
5. Having many processors, each of which only handles certain components (and events) of the overall simulation (object-oriented simulation).

Method one is useful when specialized hardware is available to permit parallel operations. Many specialized simulation hardware platforms contain multiple processors that permit this type of speedup. A different approach is used in method five. This method is useful for object-oriented distributed simulation.

Just as abstraction is used to create a valid model, abstraction is used to create computer-world objects that model real-world objects. Each object can then be distributed to a distinct processor. To complete the simulation, communications network and simulation protocols allow the objects to interact together. While this method of distributed simulation can be complex, the complexity is largely in the protocol and network. Once these have been established, the objects themselves interact relatively easily. Many protocols for distributed simulation exist and can be tailored to specific needs. See [9] for additional information.

In either case, you must select between general-purpose languages (utilizing libraries and subroutines) or specialized simulation languages. Also, you need to choose COTS hardware platforms or specialized simulation hardware. In any case, the following is a checklist for capabilities needed in choosing a simulation language (based on [10]). This list can also be used for comparing potential languages or language and hardware combinations with each other for their adequacy for your simulation needs.

- Well structured (easy to use and understand) data input and output mechanisms.
- Predetermined time-flow mechanisms.
- High quality random number generation routines (discrete and continuous).
- Clock routines to store, sequence, and select simulated events.
- Automatic statistical collection.
- Ease of use.
- Adequate documentation.

Simulation Output

The criteria of well structured data input and output mechanisms is particularly important is selecting a modern simulation platform. Many modern simulation languages and simulation platforms include real-time output using high-quality graphics. For example, having a constantly updated picture of the airflow around a jet engine during a simulated flight might be preferable to simply outputting airflow data. This visual output can make your simulation easier to verify; a picture is probably much easier to understand and evaluate compared to a large amount of numeric data. Because the output of the simulation is a picture of a real-world entity, domain experts can quickly examine and evaluate the picture without having to understand the output of a computer simulation program. In addition, visual data quickly show the effects of modifications to the model.

If graphical representation of output is important to your application, you must then compare hardware requirements (graphical quality and graphical speed) when selecting your simulation hardware and software. Options range from using the built-in display features of off-the-shelf systems (least expensive, but lowest in speed and quality) to using high-end specialized hardware.

Verification and Validation

First the model is created and validated. Next the simulation is coded, verified, and validated. Only then can meaningful results be obtained. The results of the simulation must be carefully examined against reality. If care is not taken, all the hard work and effort and modeling and simulation can create the computer equivalent of a self-licking ice cream cone. The purpose of a simulation according to Shannon is "understanding the behavior of the system or of evaluating various strategies for the operation of this system." The results have to be examined in light of the model and the physical world (see Figure 2). Tests must be performed to ensure that the model and simulation accurately represent the real world before the simulation can be used to predict behavior.

The tools and techniques necessary to validate a model fall in the realm of mathematics and include such methods as the chi-square goodness-of-fit test and the Kolmogorov-Smirnov test. Such phrases as "the results look quite good" [11] are meaningless. Statements such as "90 percent of the predicted values lie within 5 percent of the observed values" are better, but do not imply whether this is good (adequate) or bad (inadequate). Only a statistician working with a domain expert can make such judgements.

Finally, in situations where observed data cannot be used (for example, when making observations about a jet engine that does not actually exist), trained domain experts may have to extrapolate the quality of the simulation based upon data from a differing (but existing) system.

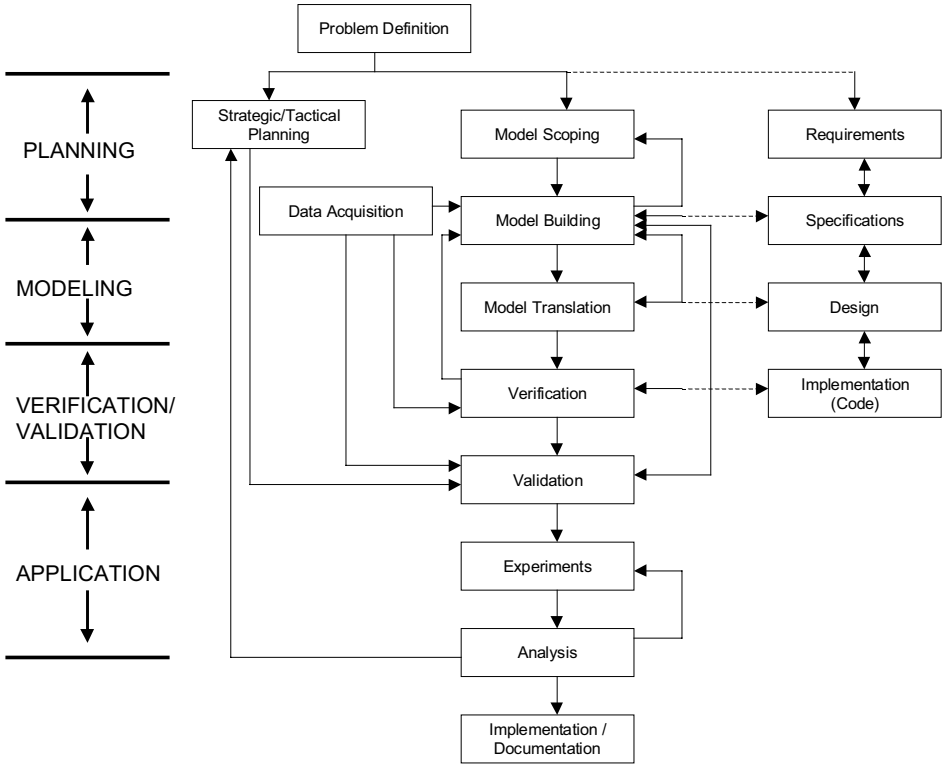
Summary

Simulation can be an effective tool to save money during development of complex systems. Once the problem has been identified, and a system analysis of the overall system has been accomplished, there are only four basic tasks that need to be performed in a simulation [12]:

1. Determine that the problem requires (or would benefit from) simulation. Crucial factors are the cost, feasibility of conducting real-world experiments, variability of the system, and the possibility of mathematical analysis.
2. Build a model of the problem.
3. Write a computer program that converts the model into a computer simulation.
4. Use the computer simulation to resolve the problem.

Although this list is simple, remember mathematical and domain experts must be involved to verify and validate the model. Care must be taken during each step to ensure that the model accurately reflects the real world, and that the code accurately reflects the model. Finally, analysts must examine the output of the simulation to make sure that the results are valid, and can be used to make accurate predictions.

Figure 2. The completed process (simplified)



References

1 Shannon, R.F., Simulation: A Survey with Research Suggestions, *AIIE Transactions*, Vol. 7, No. 3, Sept. 1975.

2. Shannon, R.F., *Systems Simulation*, Prentice-Hall, 1975.

3. Banks, J. and Carson, J., *Discrete-Event System Simulation*, Prentice-Hall, 1984.

4. Hamilton, J., Cook, D., and Pooch, U., A Software Engineering Perspective on Distributed Simulation, CROSSTALK, February 1996.

5. Law, A. and Kelton, W., *Simulation Modeling & Analysis*, McGraw Hill, 1991.

6. Davis, A., *Software Requirements: Objects, Functions and States*, Prentice-Hall, 1993

7. Cook, D., Accelerated Time Discrete Event Simulation in a Distributed Environment, *International Journal of System Science*, Vol. 24, No. 3, 1993.

8. Righter, R. and Walrand, J., Distributed Simulation of Discrete Event Systems, *Proceedings of the IEEE*, 1989.

9. Cook, D., Accelerated Time Discrete Event Simulation in a Distributed Environment, Ph.D. Dissertation, Department of Computer Science, Texas A&M University, August 1991.

10. Adkins, G. and Pooch, U., Computer Simulation: A Tutorial, *Computer*, Vol. 10, No. 4, April 1977.

11 Graybeal, W. and Pooch, U., *Simulation, Principles and Methods*, Little, Brown and Company, 1980.

12 Pritsker, A., *The GASP IV Simulation Language*, Wiley and Sons, 1974.

David A. Cook is the chief function officer of Shim Enterprise Inc. He is currently assigned as a software engineering consultant to the Software Technology Support Center, Hill AFB, Utah. He was formerly an associate professor of computer science at the U. S. Air Force Academy (where he was also the department research director), and also a former deputy department head of the Software Engineering Department at the Air Force Institute of Technology. He was a member of the Air Force Ada 9x Government Advisory Group, and has published numerous articles on software process improvement, software engineering, object-oriented software development, and requirements engineering. He has a bachelor's in computer science (University of Central Florida), a master's in teleprocessing from the University of Southern Mississippi, and a doctorate in computer science from Texas A&M University. Cook can be reached at david.cook@hill.af.mil◆