Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

THE ROLE OF SIMULATION FRAMEWORKS IN RELATION TO EXPERIMENTS

David W. King
Douglas D. Hodson
Gilbert L. Peterson

School of Electrical and Computer Engineering
Air Force Institute of Technology
Dayton, OH 45433, USA

ABSTRACT

The usefulness of software frameworks to support the development of military combat simulations is gaining attention. Using a framework increases model reuse and can avoid the duplication of infrastructure code used to support model and simulation application development. Simulation frameworks encourage defining abstractions for the domain of interest, which allow for multiple concrete (i.e., specific) implementations of models at varying levels of fidelity, resolution and/or detail to be produced and assembled. This flexibility leads to customized simulation applications that are focused and aligned to an envisioned conceptual model of a system of interest. However, the difference between a framework and a specific simulation application, and its relationship to experimentation is not always clear. This paper elaborates on these distinctions and addresses how software frameworks support experimental objectives.

1 INTRODUCTION

Experimentation supports the study of new technologies and ideas. In the defense industry, experiments range from low-cost software prototypes to multi-million dollar research efforts. Results support or refute a hypothesis and allow a researcher to understand relationships, inferences and draw conclusions about a phenomenon of interest. Simulations provide a simplified approximation of real world situations that can be used in experimentation. A simulations approximation is referred to as a model and is created by assembling a collection of domain objects that define functionality and behavior. However, the underlying architecture of the software system used to assemble object models is often misunderstood; specifically, the role of frameworks in supporting the development of simulation applications.

Frameworks define the relationships between domain objects through the use of abstractions, thereby, allowing complex models to be assembled from a collection of different implementations at various levels of detail. These features support the development of simulation applications better aligned with an envisioned conceptual model.

This paper highlights the role of frameworks as a tool to build simulation applications, and specifically addresses how this capability supports experimentation. We first present the problem context in terms of how the experimentation process drives simulation requirements. A short review on experimental processes and simulation frameworks provides a base terminology for understanding the roles and relationships between experimentation and simulation frameworks. Finally, we present two existing simulation frameworks with a notional example.

2 PROBLEM CONTEXT

The terms *model* and *simulation* are often conflated. To avoid this ambiguity, we define them as

- Model: physical, mathematical or otherwise logical representation of a system, entity, phenomenon or process.
- Simulation: the execution or processing of a model over time with an assumed modeling paradigm.

The execution of a simulation assumes a paradigm that defines how models are conceptualized and created. For example, we are particularly interested in event-stepped (i.e., event-driven or discrete-event) and time-stepped (i.e., frame-based) model execution which constitute two popular paradigms. Event-stepped simulation differs from time-stepped in that time advances by irregular increments, whereas, time-stepped advances time by fixed increments (Fujimoto 2000). Depending upon simulation purpose, one paradigm can execute model code more efficiently than the other.

The role of software-based simulation applications to support experiment execution is well understood; they *create* a representation of a system of interest. In the experimental process, these applications are executed to generate data for analysis. As experiments are usually iterative, expansion of simulation applications to incorporate more models or behaviors is expected (Montgomery 2013).

Based on our experience, many simulation applications follow a common life cycle. Initially, an envisioned conceptual model of interest is defined and a simulation application to support its study through experimentation is developed. Because experimentation is often an iterative exercise, extension to the originally developed application are made that include more and/or different aspects to account for other phenomenon of interest. At some point, the simulation application becomes complex enough, that some way of improving or simplifying the User Interface (UI) is pursued. Example UI improvements include the creation of application configuration files, the development of a supporting language (i.e., scripts) to configure and describe behaviors or the creation of Graphical User Interfaces (GUI).

At some point in time, even with these usability improvements, if the application continues to grow, the desire for more examples, documentation and training material is sought to communicate and understand the complexity. Beyond that, often, the effort to maintain the existing application ceases in favor of starting over with a simpler, more understandable solution.

It can be argued that within this life cycle, opportunities to refactor (i.e., restructure, reorganize and/or improve) the code to reduce complexity should have been considered. Failure to recognize this leads to large applications difficult to "setup and run," but more importantly, really understand and extend. Software frameworks, specifically simulation frameworks, are designed to mitigate this complexity.

3 EXPERIMENTAL PROCESS

In 2006, The Technical Cooperation Program (TTCP), an international organization that collaborates in defense scientific and technical information exchange, published the "Guide for Understanding and Implementing Defense Experimentation (GUIDEx)" (Bowely et al. 2006). GUIDEx defines defense experimentation as a series of tasks as shown in Figure 1.

We are particularly interested in the relationship between the "Determine the required fidelity of representations" and the "Develop models and systems representations" activities of the "Experiment Development" task. The activity of determining the fidelity, resolution and/or detail of a model is challenging - and the process of doing so is probably more art than science. But there are some guidelines, for example, the prevailing view has been to keep models "as simple as possible." Cautioning against model complexity, the following quote succinctly states the concern:

> "It has long been understood by operational researchers that, in dealing with complicated situations, simple models that provide useful insights are very often to be preferred to models that get so close to the real world that the mysteries of the world they intend to unravel are repeated in the model and remain mysteries." (Bowen and McNaught 1996)

In other words, the main point of modeling is to rationalize the complexity of real life by simplifying it (Bowely et al. 2006).

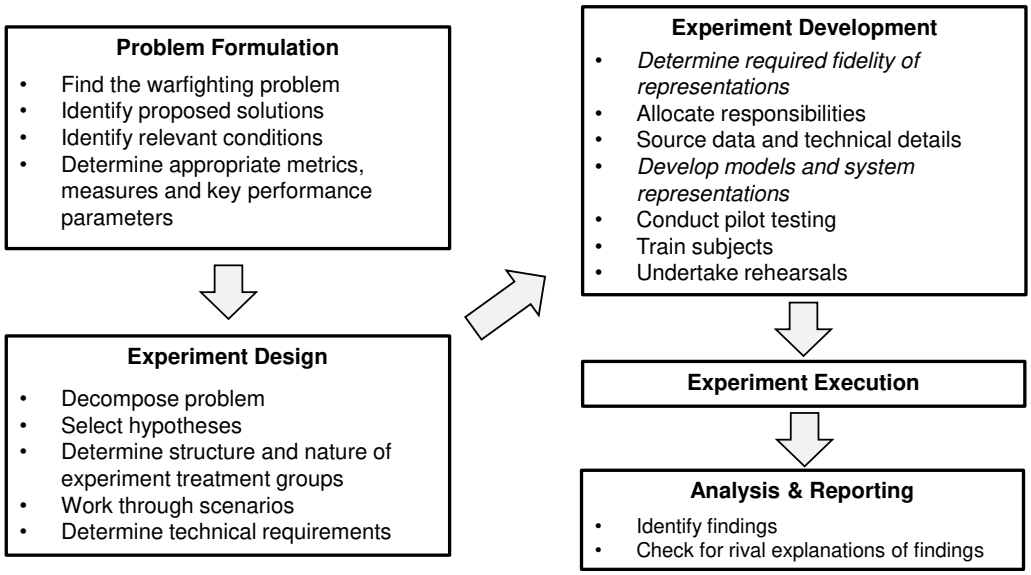

Figure 1: Defense Experimentation Planning Flowchart (Bowely et al. 2006).

The activity of determining the required fidelity of representations (or abstracting a model from the real world) is a mental process, called *conceptual modeling*. Conceptual modeling is the abstraction of a simulation model from the real world system that is being modeled; in other words, choosing what to model and what not to model (Robinson 2013) - its relationship to the problem domain is shown in Figure 2. As shown, the conceptual model defines the "required fidelity of representations" activity of the GUIDEx "Experiment Development" task and the "develop models and system representations" relates to "Model Design" (with an assumed paradigm) and the coding of a "Computer Model" (i.e., a simulation application).

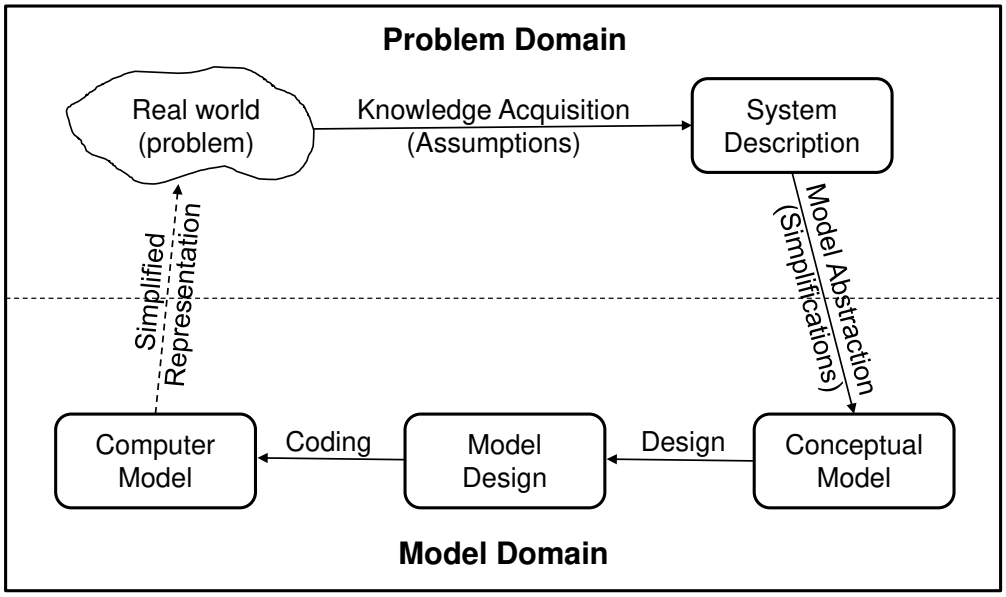

Figure 2: Conceptual Modeling (Robinson 2011).

In summary, the "Experiment Development" task encompasses the creation of a conceptual model from a system description using acquired knowledge about a real-world problem of interest to the full development of software code that defines it. The remaining tasks include the execution of the experiment (i.e., "runs" or "tests" from a test matrix) to generate data for analysis, understanding, reporting and possible decision making.

4 SIMULATION FRAMEWORKS

The experiment development stage is where simulation software development begins; the creation of executable models with an assumed paradigm. If software is written to implement a specific conceptual model, there is usually good alignment between the model being created and the one envisioned. Because experiments are usually iterative, very often, the next experiment will require changes to the simulation application itself to include more and/or different models to account for other aspects of interest. Software frameworks can mitigate this growth in model complexity by separating core "infrastructure" code from its specific use within a given application.

A general purpose *software framework* is a collection of abstract and concrete classes and the relationships between them (Sommerville 2007). From a process point of view, creating the abstractions often results from the refactoring of existing concrete classes to create "black-box" points from which developers can extend to build applications (Gamma et al. 1995). For example, given a particular modeling paradigm, say event-stepped, software code that defines how events are stored, selected and processed (i.e., executed) can be separated from the specific models that uses it. This same separation can occur at the model level, where abstractions of what things are (called domain objects) and the functionality they define, can be separated from specific implementations of it. Separating these two aspects alone is often the foundation for a general purpose *simulation framework*.

Since abstractions allow for multiple concrete instances, frameworks provide the best method for object-oriented code reuse; they provide a one to many mapping of models and behaviors to multiple applications of interest (Sommerville 2007). This is counter to simulation applications that are more focused, and often provide one to one mappings of specific models and behaviors for the intended experiment.

5 ROLES AND RELATIONSHIPS

Figure 3 shows the relationship between experimental design activities and software development. As this figure shows, the experimentation process starts with a problem or question, which manifests into an experiment to be run with a system of interest that, ideally, matches the envisioned conceptual model.

The role of the simulation application is to create and execute this representation by instantiating and assembling the specific domain objects (i.e., concrete models) from the problem domain as shown in Figure 2. The role of the framework is to define the abstractions so that specific models can be built, and often provide the means to execute it.

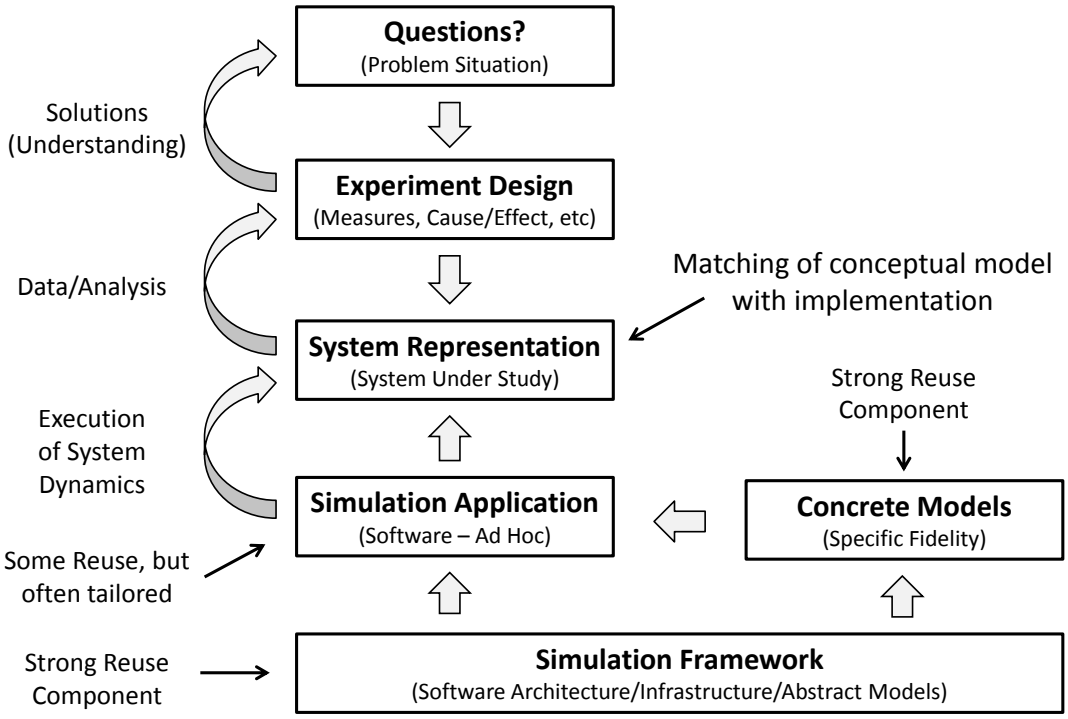

Figure 3: Experimental Design/Software Development Relationship.

Very frequently, certain aspects of model code can be "reused" given the ability to adjust some parametric data. For example, the development of a radar model with a high degree of resolution or detail might be a significant development effort all on its own, but if designed with some flexibility (e.g., externally specified parameters), the laws of physics this model captures could possibly be reused in other experiments. For these concrete models, there may be a strong reuse component. For others, specifically models that are narrowly focused on a particular aspect of the problem domain, they are often included, or bundled, with the simulation application.

Because of this, we consider *simulation applications* to be "ad-hoc," meaning they are used for a particular purpose (e.g., a study), then often no longer needed. This is not to imply or dismiss their importance, but rather to contrast their narrower purpose and shorter lifespan in relation to the framework.

6 EXAMPLE FRAMEWORKS

To illustrate the advantage of leveraging software frameworks to support simulation application development, we present two established frameworks; one embraces the event-stepped modeling paradigm, the other is time-stepped.

The Advanced Framework for Simulation, Integration and Modeling (AFSIM) (Clive et al. 2015) is an example of an event-stepped (i.e., discrete-event) software framework. AFSIM is written in C++ and supports simulation development by extending an application through software plug-ins, which are selected points to add functionality. These points allow for the creation of specific concrete models that represent system components of interest.

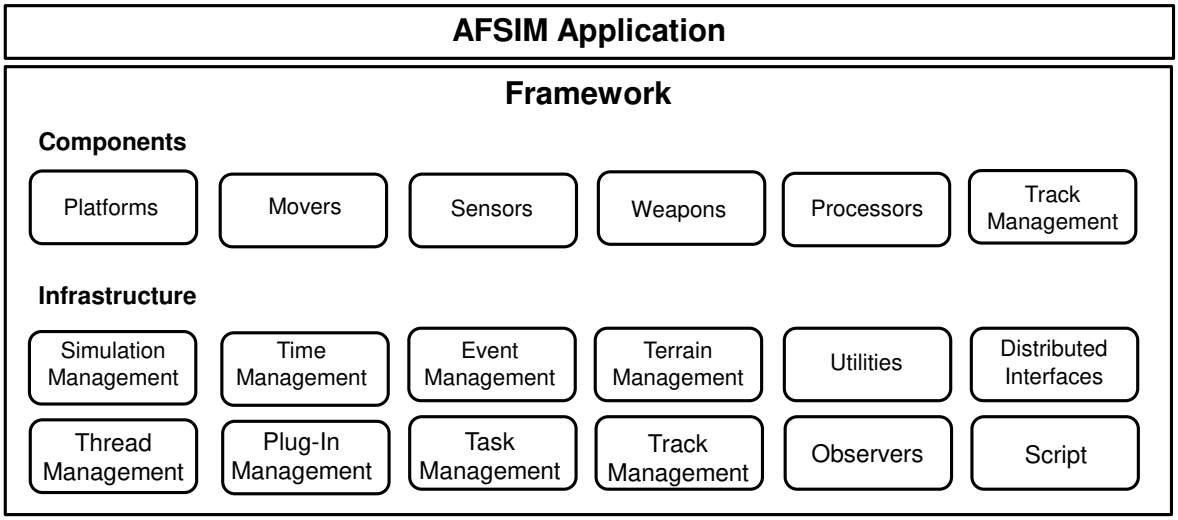

Figure 4: Advanced Framework for Simulation, Integration and Modeling (AFSIM) (Clive et al. 2015).

As shown in Figure 4, AFSIM defines an underlying infrastructure to support model and simulation development, and another layer of components for model building. The components are abstract domain objects of interest to the military combat simulation community. AFSIM defines both those abstractions, and provides a collection or library of specific models (i.e., concrete models) tunable through user defined input parameters to align a system representation with an envisioned conceptual model. AFSIM comes with a number of supporting tools to facilitate model development, scenario definition, and simulation execution.

The Mixed Reality Simulation Platform (MIXR), previously named OpenEaagles (Hodson et al. 2006), is an example of an time-stepped simulation framework. Like AFSIM, MIXR is written in C++, but the functionalities and purposes of intended applications are usually different, namely the inclusion of people and/or other real-world devices into the simulated system representation (i.e., environment, world or situation). In other words, MIXR is a platform to build a simulation with a set of connected controls (i.e., a *simulator*) to provide a realistic interface to vehicles, such as an aircraft, or other complex systems. For example, MIXR can be used to build a flight simulator that a person can "fly" within a so-called virtual world. Advancing simulation time in fixed discrete-time intervals (i.e., time-stepped) aligns the execution of models with real-time software scheduling structures (e.g., cyclic) - which enables the system to meet response time requirements. Execution in this fashion also supports reliable processing of external driving functions (or signals from input devices) (Cellier and Kofman 2006).

Figure 5 shows MIXR which includes an underlying infrastructure to support the development of models, simulations and simulator applications. While what is modeled in AFSIM and MIXR (the abstractions, the domain objects represented as composable software components) is similar to each other, differences in available concrete models in terms of how they are conceptualized (i.e., modeling paradigm) and fidelity and/or detail do exist. This is a reflection of intended use and purpose.

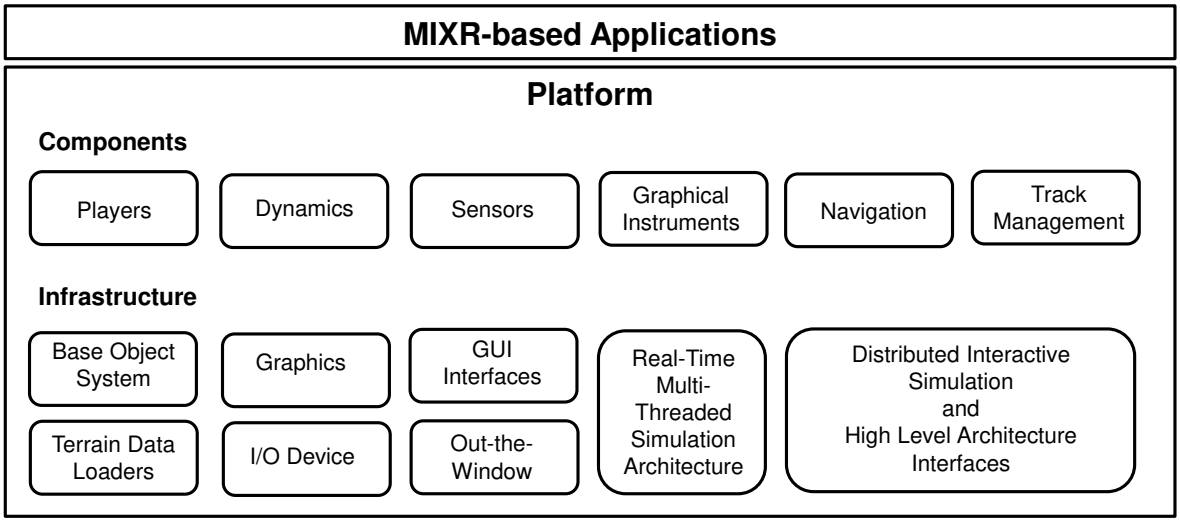

Figure 5: Mixed Reality Simulation Platform (MIXR) (Hodson et al. 2006).

For both frameworks, the key relationship to recognize is the layering (or relationship) of the application (or applications) on top of the framework. The application doesn't just use the framework as a generic library of functionality (i.e., a toolbox), it extends, molds and shapes available abstractions so that there is a tight alignment between the conceptual model and the representation it's designed to create.

6.1 Application

As a specific example, if the goal of an experiment is to understand the value of carrying a given number of missiles on an aircraft against an adversary who has a radar capable of detecting presence within a given range, then a model that indicates or produces detections based on range might be sufficient. But, if the goal of an experiment is to understand how to fly an aircraft in such a way to avoid detection by an adversary, then a more detailed radar model might be necessary. For example, a radar model that produces detections of an adversary accounting for their signature, atmospheric attenuation, terrain occulting and the accumulation of several correlated electromagnetic refections might be more correct and better aligned with the envisioned conceptual model.

6.2 A Note on Credibility

In the first case, it's easy to make an argument that radar detection is greatly affected by the concept of operations in play to deploy the missile, thus the experiment is invalid and not creditable. In the second, it could be equally argued that the radar model used was too detailed, as the modeled terrain occulting effects might confound results with the setting in the simulated world. Both criticisms are valid, hence the reason to execute a campaign of experiments (Bowely et al. 2006) to gain a fuller or more complete understanding of the system dynamics and relationships.

6.3 Usefulness of Concrete Models

As this simple example demonstrates, the value or usefulness of concrete models should not be judged solely on the fidelity or detail represented; usefulness depends on how well it aligns with the conceptual model defined for a particular experiment. A simple model might "fit" or align with experimental objectives much better, then say, another which defines so much detail that it might confound the results (i.e., introduce too much reality!).

Framework abstractions provide the means to select and use available concrete models, refine existing ones, and/or create completely new representations. Ideally, from a reusability and/or capability perspective, having multiple models at different fidelities, resolution and/or detail that represent the same conceptual thing would be ideal. Frameworks provide the means to enable this capability.

7 CONCLUSIONS

As frameworks become more prevalent with the military combat simulation community, it is essential to understand their role in relation to applications. The usefulness of software frameworks to support the development of military combat simulations is clear; they offer a modeling infrastructure and a means to craft and assemble specific models to represent a system of interest, for a given purpose. That purpose is to align with an envisioned conceptual model.

In the experiment design process, frameworks provide a supporting role to developing simulation applications. Applications provide the means to execute the model, which creates the envisioned system representation, which may include facets of the real-world, such as people and/or devices, over time, to generate data for subsequent analysis, understanding and decision making.

REFERENCES

Bowely, D., P. Comeau, R. Edwards, P. Hiniker, G. Howes, R. Kass, P. Labbe, C. Morris, R. Nunes-Vaz, J. Vaughan, S. Villeneuve, M. Wahl, K. Wheaton, and M. Wilmer. 2006. *Guide for Understanding and Implementing Defense Experimentation*. The Technical Cooperation Program.

Bowen, K., and K. McNaught. 1996. "Mathematics in Warfare: Lanchester Theory". In *The Lanchester Legacy - A Celebration of Genius*, edited by J. Fletcher.

Cellier, F. E., and E. Kofman. 2006. *Continuous System Simulation*. New York, NY: Springer.

Clive, P. D., J. A. Johnson, M. J. Moss, J. M. Zeh, B. M. Birkmire, and D. D. Hodson. 2015. "Advanced Framework for Simulation, Integration and Modeling (AFSIM)". In *International Conference on Scientific Computing*, 73–77.

Fujimoto, R. M. 2000. *Parallel and Distributed Simulation Systems*. New York, NY: Wiley-Interscience.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Hodson, D. D., D. P. Gehl, and R. O. Baldwin. 2006. "Building Distributed Simluation Utilizing the EAAGLES Framework". In *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*.

Montgomery, D. C. 2013. *Design and Analysis of Experiments*. Eighth ed. John Wiley and Sons.

Robinson, S. 2011. "Conceptual Modeling for Simulation". In *Encyclopedia of Operations Research and Management Science*, edited by J. Cochran. New York: Wiley Press.

Robinson, S. 2013. "Conceptual Modeling for Simulation". In *2013 Winter Simulation Conference (WSC)*, edited by R. Hill, M. Kuhl, R. Pasupathy, S. Kim, and A. Tolk, 377–388.

Sommerville, I. 2007. *Software Engineering*. 8th ed. Addison-Wesley.

AUTHOR BIOGRAPHIES

DAVID W. KING is a PhD Student at the Air Force Institute of Technology. He holds a B.S. in Computer Science from the University of Maryland and a M.S. in Cyber Operations from the Air Force Institute of Technology. His email address is david.king@afit.edu.

DOUGLAS D. HODSON is an Associate Professor of Software Engineering with the Air Force Institute of Technology. He received a B.S. in Physics from Wright State University in 1985, and both an M.S. in Electro-Optics in 1987 and an M.B.A. in 1999 from the University of Dayton. He completed his Ph.D. at the Air Force Institute of Technology in 2009. He has over 30 years of experience in the domain of

modeling and simulation and has a research interest in characterizing the consistency of shared simulation state data in terms of its temporal properties to estimate Live-Virtual-Constructive and Distributed Virtual Simulations performance, cloud computing and modeling quantum key distribution systems. He is the lead technical developer and project manager for the open-source Mixed Reality Simulation Platform (MIXR) which has been used to develop a wide variety of standalone and distributed simulation applications. His email address is douglas.hodson@afit.edu.

GILBERT L. PETERSON is a Professor of Computer Science at the Air Force Institute of Technology, and Chair of the IFIP Working Group 11.9 Digital Forensics. Dr. Peterson received a BS degree in Architecture, and an M.S and Ph.D in Computer Science at the University of Texas at Arlington. He teaches and conducts research in digital forensics, statistical machine learning, and autonomous robots. His research has been sponsored by the NSF, DARPA, AFOSR, AFRL, and JIEDDO. He has over 90 peer reviewed publication, and 5 edited books. In 2008, he received the Air Force Junior Scientist of the Year Category I award. His e-mail address is gilbert.peterson@afit.edu.