*To Be Presented at the Advanced Simulation Technology Symposium (ASTC), Washington DC, April 2004.*

# Building Simulation Modeling Environments Using Systems Theory and Software Architecture Principles

Hessam S. Sarjoughian and Ranjit K. Singh
Arizona Center for Integrative Modeling and Simulation
Dept. of Computer Science & Engineering
Fulton School of Engineering
Arizona State University
Tempe, Arizona, 85281-8809, USA
{Hessam.Sarjoughian | Ranjit.Singh}@asu.edu

**Keywords:** DEVSJAVA, MVC, Software Architecture

## Abstract

Use of simulations for design of systems requires assurance that the underlying modeling theories and simulation strategies are correctly realized. For example, not only it is crucial to employ modeling and simulation theories, but it is also critical to develop software architectures (environments) that guarantee correct implementation of simulation Model execution. In this paper, we will (i) examine key features of an object-oriented modeling and simulation (M&S) environment with respect to its software architecture traits and (ii) propose an approach capable of supporting essential functionalities advocated by system-theoretic modeling and simulation concepts and methods. We introduce a software architecture for discrete event M&S environments. We illustrate the software architecture using a prototype simulation tracking environment.

## 1. Introduction

Modeling and simulation environments (e.g., [1,10,16]) have been used as part of a repertoire of software engineering tools. Engineering of computer-based systems, particularly those that are complex and large-scale, increasingly require simulation modeling. Therefore, it is important that M&S tools support various quality attributes as is typically required of other software applications. Since there are numerous M&S tools for a variety of modeling methods and simulation techniques, it is useful to study the relationships between the realm of (i) model building and simulation execution in conjunction with (ii) software architecture, design, and development.

Looking at the vast range of domains M&S is applicable to, there are fundamental concepts of modeling and simulation independent of any one particular field. A general theory of M&S, founded on the fundamentals of system theory provides a basis toward creating software-based environments. With advances in computational sciences and engineering (e.g., design patterns [6]), we can create modeling and simulation environments that can support additional M&S capabilities based on well-defined M&S concepts and techniques (e.g., supporting verification and validation (e.g., [17])).

The conceptual view of the proposed approach is illustrated in Figure 1. Figure 1(a) shows a set of modeling & simulation artifacts and relationships among them. Figure 1(b) depicts a software architecture style towards the realization of Figure 1(a). The software architecture provides separation of concerns in terms of software design and implementation of M&S concepts and methods. For instance, the *Model* in 1(b) can be any of several alternative M&S approaches (e.g., parallel simulation of object-oriented models). Similarly, the *View* can present textual and/or graphical simulation behavior and the *Control* can be one of several strategies to mediate interactions between the View and the Model.

Software architecture plays a key role in developing robust simulation environments. For example, the High Level Architecture (HLA) [7] software specification offers concepts, methods, and processes for developing modeling and simulation environments based on the capabilities provided, for example, by the Run Time Infrastructure (RTI) [7]. Examination of HLA, however, illustrates the importance of supporting a systematic approach for development of simulation models and experimentations with them [13]. It is highly desirable to subject simulation Models to various experiments using well-defined, systematic techniques [16].

### 1.1. Modeling and Simulation

A variety of existing frameworks support modeling methodologies and simulation techniques (e.g., [4,5,10,16]). We briefly describe the Discrete Event System Specification framework which has three primary artifacts (source system, model, and simulator) and two relationships (validation and verification) [16]. To select among many alternative models, the framework employs the basic concept of experiments to reduce the scope of simulation complexity. An experimental frame specifies the conditions (i.e., experiments) under which the system (and/or its model) is experimented with. It specifies, in a computable

form, the objectives that motivate modeling and simulation studies.

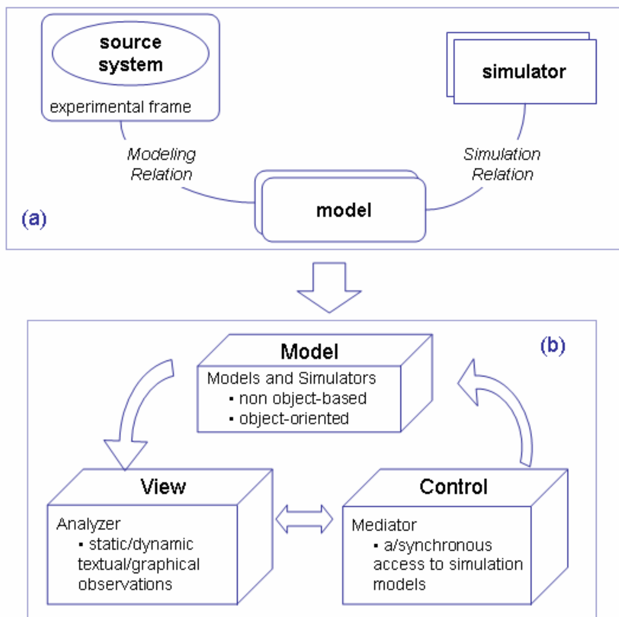


**Figure 1. Mapping of a M&S Framework onto a Software Architecture**

## 2. Modeling and Simulation Environments

The objective of this section is to illustrate important relationships that relate software design issues with M&S concepts and methods. A process-based, discrete-event simulation environment is called simjava [8]. In simjava, a simulation is defined by collections of entities that run in threads parallel to one another. A centralized class coordinates simulation activity by advancing simulation time and delivering events. The simjava interface is quite versatile in terms of its representation of entities in a simulation. As part of simjava's modeling process (which is done at the source code level), modelers may choose how an entity's dynamics will appear onscreen and what combination of graphics, text, or visual icons will best represent it. Although this provides a powerful level of customization for model presentation, there are issues such as scalability and modifiability. Beyond a few dozen models, it is difficult to conceptualize a system using such an interface. Additionally, since layout and visual representation is manually configured, adding or removing entities may prove to be error prone and time-consuming.

In addition to textual labels containing state information and the graphical animation of event passing, simjava provides a set of JavaBeans for its simulation output processing. An expandable set of components, the beans provide analysis capabilities in terms of a timing diagram (Gantt chart), graphical generator (2D chart), file saver (text file dump), etc. While this mode of analysis can support model validation, the approach inherently lacks scalability and Model reuse since no separation exists between a model and the means by which it is to be observed or acted upon. JDEVS is a visual M&S tool for discrete event systems [9]. Spanning across four modules, JDEVS consists of a DEVS-based simulation engine, a graphical user interface, and two visualization components for viewing simulations in two and three dimensions. The user interface provides a visual representation of models but does not support hierarchical models nor does it allow tracking of specific data of interest during simulation.

Although JDEVS is stated as a general purpose, collaborative environment for visual simulation model development and execution, it is more suited for natural systems as evidenced by the domain specificity of its third and fourth modules. Although modules one and two provide some degree of generic modeling, simulation, and debugging, the JDEVS environment does not describe advanced features, such as support for distributed simulation, or collaboration beyond simple XML interactions at the Graphical User Interface (GUI) level. Additionally, from an architectural standpoint, there is little in terms of a rigorous approach for enforcing the correctness of the simulation engine and the resulting data gathered from it. As an environment for modeling ecosystems, JDEVS's 2D and 3D panels offer a visually appealing perspective.

Another environment is Ptolemy II which supports a wide variety of modeling and design tasks with support for heterogeneous mixtures of Model computation types [12,14]. While Ptolemy II contains rich analysis facilities, similar to simjava, the process of connecting analysis components are done as a modeling activity as opposed to decoupling simulation scenarios as a separate and distinct concept.

### 2.1. DEVSJAVA Simulation Environment

To illustrate some details and issues pertaining to software architecture and design of modeling and simulation environments, we examine DEVSJAVA, a M&S tool implemented in the Java™ programming language. This environment supports characterizing models in the DEVS (Discrete Event System Specification) formalism [16]. DEVSJAVA software design is structurally split among four main packages. The partitions include a separation of M&S engines (genDEVS.modeling and genDEVS.simulation) from a user interface (simView). The genDEVS.modeling classes and their relationships represent a realization of the DEVS modeling artifacts. Similarly, the genDEVS.simulation package realizes the DEVS abstract simulator. The simulation and the modeling engines are designed for interpreting Models specified in DEVS. The classes in simView package are capable of accessing and displaying (via a user interface) information

about models and their executions. Aside from these packages, there exists GenCol which is used for organizing and manipulating objects [1,11].

The basic design approach to DEVSJAVA is the use of UML interfaces to interconnect objects across different packages. In terms of its runtime functionality, DEVSJAVA presents a model through a graphical block diagram capable of representing hierarchical compositions and couplings composed of atomic and coupled Models. Atomic models, which cannot be decomposed to other models, are depicted via a rectangular box with adjacent input and output ports that logically conceptualize a model's structure in a visual manner. Inside this box, textual data provides dynamics information (such as state variables). Coupled models are depicted via a clear rectangle that makes visible internal/child models, ports, and their connections (couplings). During simulation execution, messages that pass between coupled models are animated as they traverse from model to model.

There are two levels of control services available in DEVSJAVA, that of application control and that of simulation control. Application control includes the basic features of loading and closing the application, as well as more routine behaviors, like displaying a help window. Simulation control, on the other hand, entails setup and manipulation of the simulation engine. This includes logic to load a model and to step, run, or restart an execution cycle of a model.

While the graphical representation and simple controls are well suited for the easy conception and understanding of models, some key capabilities are absent in the DEVSJAVA environment. As an environment for running simulation experiments and conducting analysis, DEVSJAVA could have key benefits if it were to automatically track (or save) model states and input/output trajectories as they were animated on screen. This implies supporting an "on-the-fly" configuration of transducers for monitoring simulation behavior instead of relying on, for example, "print line statements" within simulation models. This is important for setting up experiments with many hundreds of models.

## 3. Approach

As suggested by Figure 1, the proposed software architecture is, at its core, a fusion between system theoretic concepts and the classic Model-View-Control (MVC) paradigm. In addition to the established benefits imparted from an MVC style of organization, the proposed architecture adds conceptual overlays that reify software components in MVC to have system theoretic connotations. The resultant is believed to be an underlying framework for M&S environments that not only supports a robust set of software-based quality attributes, but also purports additional traits that realize concepts in M&S methodology. In particular, the software architecture will make attempts to facilitate validation practices as well as basic concepts advocated by the experimental frame.

It is important to note that the architecture presented here puts into context the notion of modeling and simulation engines, but does not define lower-level details of these components or their relationships. The assumption is that the inherent complexities of typical M&S engines should be shielded. This assumption has key consequences on the design of the new architecture—the integrity of existing modeling and simulation relationships must be held, and as such, these relationships must remain invariant.

To ensure the integrity of these relationships, as well as the internals of the modeling and simulation engines, the following approach is taken in the design of the architecture. First, the interplay between Model representations and their interpretations are encapsulated with respect to the View and Control. That is, the Model generates data, under the directive of the Control, for consumption by the View. Second, the Control acts as a surrogate for the existing simulation protocol contained in the Model. This implies that Control must not introduce any side effects that influence how the Models are interpreted (the *simulation correctness* relationship between Model and simulator in the Model remains invariant). Third, the View does not introduce additional dynamics beyond what already exists in the model. In other words, the View will serve as the vehicle through which a simulation model can be exercised and observed. With these invariants, M&S engines, such as those in DEVSJAVA, can be included into the proposed approach without a loss of integrity.

### 3.1. Software Architecture

In this section, the software architecture as alluded to in Figure 1(b) is described. Using the MVC paradigm, along with the façade discussed above, Figure 2 presents the architecture in terms of its structural abstractions and interactions. The artifacts of the software architecture correspond to the elements of the general modeling and simulation framework described earlier (see Figure 1(a)). The following sections describe the roles, responsibilities, and collaborations of the architecture's components.

### 3.2. Model: Modeling & Simulation Subsystem

It is well recognized that M&S engines are complex in nature. For the purposes of developing a software architecture that utilizes such components, they will be treated as an integrated whole, whose details are abstracted out. The MVC decomposition, topological structure, components, and conceptual boundaries defined by the architecture being presented do not require an exhaustive specification of the Model in order to fulfill its purpose. In fact, imposing internal designs for these engines would greatly reduce the generality and applicability of the architecture while limiting the potential for different types

of M&S systems. Instead the Model will only be defined by its role, responsibilities, and boundaries with respect to the architecture, while the integrity of its internals (i.e., modeling and simulation relationships) will not be compromised by its integration within this framework.

As the composite of two important subsystems, the Model is responsible for encapsulating core M&S logic. The modeling and simulation engines within the Model define entities, behaviors, and relationships needed to realize software counterparts of modeling formalisms and associated simulation protocols. (Note that these engines depend on supporting containers classes.) Serving as the source of dynamic behavior of the architecture, the Model contains all the classes, interfaces, components, and subcomponents necessary to form a computational infrastructure for M&S activities.

Although the internals of the Model are not considered in the architecture, the external subset of static and dynamic properties exposed by the Model is key for interaction purposes with the Control and View. From *a software perspective*, there are important interface and impendence matching issues that must be addressed in order to facilitate connections and communication among the MVC parts. From a *conceptual perspective*, there are important levels of abstractions that need to be constructed which provide the M&S artifacts that are exchanged and understood within the architecture. With these points of interest in mind, two primary levels of concern can be established, that of interface, and that of coupling and communication.

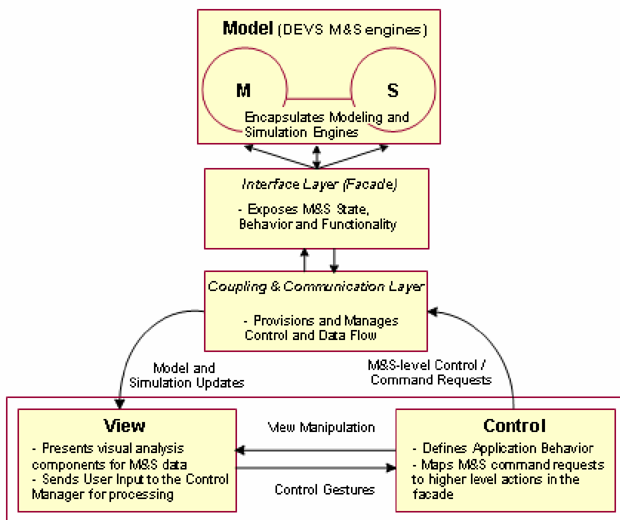


**Figure 2. New DEVSJAVA Software Architecture**

The interface level deals directly with structural and behavioral aspects that are exposed from the Model and are available to the View and Control (see Figure 2). The coupling and communication level complements the interface level in order to connect to the View and Control. The details of the interface layer and the coupling & communication layer are discussed next.

### 3.3. FAÇADE: Interface Layer

The role of the interface layer is that of a well-defined bridge capable of providing access and interaction with Model. The interface layer is implemented through a façade and exposes those mechanisms in the Model needed to support data and control services required by the View and Control (i.e., see Figure 2). It is the presence of this façade that allows the Model to maintain its 'black box' nature. Through the interface layer, the complexities of the Model are managed. As illustrated in Figure 2, only the interface layer façade is capable of interacting with the Model and reaching its inner components. With the proper implementation of the interface layer, the integrity of the Model (i.e., modeling and simulation relationships) can be maintained. This logically supports the addition of components outside of the Model such that they are known not to interfere with the correct operation of entities within the Model.

Fundamentally, the interface layer does not add any new functionality to the Model, but rather presents two conceptual sets of surrogate-views. The abstract modeling set defines the structure of simulation models and provides a means for accessing their attributes. The abstract simulator defines simulation functionality and behavior. The façade also defines an explicit set of simulation behaviors (e.g., run and step). In particular, it allows exposing higher level of abstraction behavior as compared with those possible with modeling and simulation engines inside the Model.

### 3.4. Coupling and Communication Layer

To avoid undesirable 'hard-wiring' between these components, we have introduced the coupling and communication (C&C) layer. It is responsible for establishing the mechanisms and protocols needed to connect the Model (through the façade) to View and Control. The Coupling and Communication layer supports communicating data and control commands between the Model and its View/Control – i.e., it handles timing, synchronization, concurrency, and data format. The existence and placement of the coupling and communication layer within the software architecture makes it an attractive candidate for supporting and facilitating such issues.

Models in MVC can be classified as either being passive or active. Passive Models do not make any provisions for their participation in the MVC [3]. Active Models such as the one considered shown in Figure 2, there is more than one control space and the Control is not always aware of changes that occur in the Model. Conventional solutions to this problem place additional logic (typically an Observer

[6] type of pattern) within the Model to facilitate update notification. The View registers itself as an observer to the Model, and the Model publishes changes as they occur.

Although the Model will need to provide some amenities for model updates (via the interface layer), the coupling and communication layer can subsume much of the logic needed for data flow services. The result of this decoupling is a more adaptable architecture whose communication protocols and logic to the View/Control can be enhanced or modified without alteration or influence to the Model. In addition, since the View and Control will connect only to the C&C layer, control flow will be specified within it. This has the effect of the C&C component acting as a sort of additional façade layer, capable of further refining or specializing the exposed services of the Model.

### 3.5. View: Observing Dynamics

In a traditional MVC design, the View's objective is generically defined as 'rendering the Model' in some fashion that applies to the context of the application. In this architecture, the View is further specialized by assigning context in terms of two key responsibilities: (i) providing a gateway between the end-user and application functionality, and (ii) facilitating experimentation with simulation Models.

The notions for (i) above are fairly straightforward. The View provides an access point for a user to interact with the system, particularly through a Graphical User Interface. The View supplies a visual representation of simulation Models and offers a means to inspect details or interact with those Models. In classic MVC designs, the View is Model-aware in the sense that it has explicit knowledge regarding the structure and semantics of artifacts exposed by the Model. Here the View is Model-aware with respect to the Interface Layer façade. The visual representation of Models and their level of interaction are based on the structural and behavioral abstractions defined at the interface layer façade.

The idea of (ii) is that of considering the View as a workspace for creating, conducting, and analyzing dynamic simulation scenarios. The extent to which the View can facilitate validation of Models, however, is dependent upon the functionality exposed via the façade. For instance, at minimum there should be a means to initialize a system and monitor simulation Model data at the I/O level. This, in turn, will allow mechanisms in the View such as on-the-fly View transducers that offer graphical perspectives on I/O data.

In its effort to support both (i) and (ii), the View must handle some interesting issues, including the preservation of logical correctness as well as more HCI types of concerns such as scalability and usability. The first topic deals with the accurate representation of data and semantics pushed down from the Model. Generally, MVC designs are susceptible to the liar-view bug [2] in which the challenges of synchronizing the current state of the Model with its visual depiction in the View can lead to erroneous results. In M&S this type of flaw can have detrimental effects on the outcome of results and will nullify attempts at validation. It is imperative for the data depicted by the View to at least maintain logical correctness. This implies that the timing and data values for all simulation data are accurate and both syntactically and semantically match that which is in the Model. It is not, however, mandatory for the View to maintain visual correctness at varying levels of resolutions. Unless explicitly accounted for within the Model and its collaboration protocols, there will always be some delay between the Model's state and its visual representation within the View. In addition, it is possible for some kinds of a simulation Model behavior not to be visible – for example due to concurrency or need for fast simulation. For M&S applications, in general, these kinds of discrepancies are acceptable. To maintain logical correctness then, these concerns must be addressed both internally to the View, Control, and C&C layers and well as externally between their couplings. The other concerns addressed by the View are HCI issues such as scalability and usability.

### 3.6. Control: Communicating Behavior

Within the architecture, there are two different echelons of 'Control': one is at the application level and the other at the Model level. Application-level logic resides with the View and Control and entails behavior local to these components. For instance, initialization and termination of the environment, window manipulation, and data interpretation all are encapsulated under application level logic. Model-level logic is encapsulated within the Model, and is exposed vertically through the façade and C&C layers. Model-level logic includes behaviors such as simulation and Model manipulation (e.g., run, inject input, etc). As the View exposes both echelons of Control, the Control defines application-level logic and acts as surrogate for Model-level logic.

Runtime logic within the Control focuses primarily on one task, that of processing and providing logic for 'Control gestures' that are sent from the View during runtime. Control gestures are high-level semantic events that encapsulate a control request (e.g., starting a simulation, or exiting the system). Control gestures are typically triggered in the View by a user request or action. The different types of control gestures available are enumerated within the Control. If the logic for processing a control request is not defined within the Control, then the Control simply maps the request to an appropriate corresponding call to the C&C layer. Control gestures provide a well-structured connection between the View and Control and present behavioral capabilities in a more formal manner than direct method

calls. Other traits of the Control are those characteristic of most Controls in MVC, for instance, changes in the Control are generally driven by alterations to application-level system behavior or cascade changes from the C&C or View.

## 4. Simulation Tracking Environment

To provide more depth to the concepts above and to illustrate feasibility, a simplistic tracking environment was developed using the presented architecture as a basis. This illustrative application allows setup and execution of simulation experiments from which simulation Model data sets (state variables, input/output ports, etc.) can be dynamically selected and observed for any number of Models during run-time. Model analysis and validation checking services include tabular data output that can be exported to other tools such as Microsoft Excel™ for further manipulation.

An important variation to note in the tracking environment is the lack of an explicit coupling and communication layer. Due to the relative simplicity on the requirements for collaborations, the needed behavioral and structural constructs of the C&C layer were easily pushed into the View and Control, providing testament to the flexibility of the architecture. A detailed explanation of the simulation tracking environment is presented in [15].

## 5. Conclusions

Concise and descriptive software architecture plays a central role in the development and evolution of software applications, especially those that are highly complex and large-scale. This paper introduced an approach to combine a theory of modeling and simulation with the principles of design patterns in order to create modeling and simulation environments that can support important features such as simulation model validation. We showed how combining the discrete event system specification approach with Model-View-Control architecture resulted in a software architecture that has a rigorous software design foundation. The proposed architecture adds a new level of specification to other specifications such as DEVS and HLA. It provides a software specification for the design and realization of simulation environments by formally allowing different views to be used with the same modeling and simulation engine and therefore enabling scalable simulation experiments.

## Acknowledgment

This research has been supported by NSF grant DMI-0122227 and Lockheed Martin in Sunnyvale, California.

## References

[1] ACIMS, Arizona Center for Integrative Modeling and Simulation, 2003, http://www.acims.arizona.edu/SOFTWARE.

[2] Allen, E., Diagnosing Java Code: The Liar View bug pattern, 2001, http://www-106.ibm.com/developerworks/java/library/jdiag5.html.

[3] Burbeck, S., 1992, Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC), http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

[4] Fishwick, P.A., 1995, *Simulation Model Design and Execution: Building Digital Worlds*, Prentice Hall.

[5] Fujimoto, R.M., 2000, *Parallel and Distributed Simulation Systems*, John Wiley and Sons, Inc.

[6] Gamma, E., R. Helm, R. Johnson, J. Vlissides, 1994, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley.

[7] HLA/RTI, https://www.dmso.mil/public/, visited Sept. 2003.

[8] Institute for Computing Systems Architecture, http://www.dcs.ed.ac.uk/home/hase/simjava/, visited Sept. 2003.

[9] JDEVS home, http://spe.univ-corse.fr/filippiweb/ debut.htm, visited Sept. 2003.

[10] Mesarovic, M.D., Y. Takahara, 1989, *Abstract Systems Theory*, Springer-Verlag, New York.

[11] Park, S., B.P. Zeigler, H.S. Sarjoughian, 2001, "Interface for Scalable DEVS and Distributed Container Object Specifications," IEEE SMC International Conference, October, Vol. 5, pp. 3075-80, Tucson, AZ, USA.

[12] Ptolemy II home, http://ptolemy.eecs.berkeley.edu/ptolemyII/, visited Sept 2003.

[13] Sarjoughian, H.S., B.P. Zeigler, 2000, "DEVS and HLA: Complementary Paradigms for M&S?," *Transactions of the Society for Computer Simulation*, Vol. 17, No. 4, pp. 167-197.

[14] Shuvra S. Bhattacharyya, et. al., 2002, "Heterogeneous Concurrent Modeling and Design in Java (Volume 2: Ptolemy II Software Architecture)," Memorandum, University of California, Berkeley< CA.

[15] Singh, R., H. S. Sarjoughian, 2003, "Software Architecture for Object-Oriented Simulation Modeling and Simulation Environments: Case Study and Approach," TR-009, Computer Science & Engineering Dept., Arizona State University, Tempe, AZ.

[16] Zeigler, B.P., H. Praehofer, and T.G. Kim, 2000, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Second Edition ed., Academic Press.

[17] Zeigler, B.P., H.S. Sarjoughian, 2002, "Implications of M&S Foundations for the V&V of Large Scale Complex Simulation Models," Foundations `02 Workshop, https://www.dmso.mil/public/transition/vva/.