Implications of M&S Foundations for the V&V of Large Scale Complex Simulation Models[1]

Bernard P. Zeigler[2]

and

Hessam S. Sarjoughian[3]

Arizona Center for Integrative Modeling and Simulation

www.acims.arizona.edu

An Invited Paper for Session A6 of the Foundations for V&V in the 21st Century Workshop (Foundations '02) held at the Johns Hopkins University Applied Physics Laboratory

Laurel, Maryland (USA)

October 22-24, 2002

[1] This research has been supported in part by NSF/MES under grant #DMI-0075557, NSF/NGS under grant #EIA-997505, and DOE SciDAC grant.
[2] Electrical & Computer Engineering Dept, University of Arizona, Tucson, AZ, USA; Zeigler@ece.arizona.edu
[3] Computer Science & Engineering Dept., Arizona State University, Tempe, AZ, USA; Sarjoughian@asu.edu

Table of Content

Abstract

While there are no general tools that would ease the V&V process for large scale complex models, there is a theory and framework to support the development of such tools. In this review of the theory and framework, we discuss the possibility of universal V&V tools that are entirely domain-independent in the sense that they operate with concepts that are applicable to any model independently of its domain semantics. As background for this discussion, we review the major formalisms that are relevant to M&S, particularly from the V&V perspective. We find that a major trend in software development, object-orientation and the Unified Modeling Language, is not adequate for the specific requirements of M&S, which supercede those of general software development. This motivates our review of systems theory and the theory of modeling and simulation based on it. As one application, we show how the theory-based concepts enable a systematic guideline to the role of V&V in the various key steps of the High Level Architecture Federation Development and Execution Process (FEDEP). The approach incrementally introduces appropriate levels/kinds of verification and validation activities throughout the FEDEP, and thus making it possible to handle the complexity inherent in V&V of distributed, legacy-based simulations. We also discuss the theory's support for V&V within the model composability problem and in component-based model development and reuse. Finally, we close with recommendations for further research in the development of universal tools for V&V of very large scale simulation models. We show how such tools can be based on computationally feasible domain-independent spaces that are conceivable within the spirit of general systems theory and that offer advantageous comparison of complex dynamical behaviors.

It is also appropriate to warn the reader on what this review does not attempt to do. There is an abundance of concepts, methods, techniques, and tools to relating verification, validation, and accreditation in the various literatures of engineering, science and defense. Despite some attempts at categorization, these various techniques are only loosely related to one another and fail to display any obvious commonality. While we believe that eventually all such techniques could well be integrated into the framework we discuss here, we do not make any attempt in this paper. We do believe that such integration would benefit from research in the future.

1. Introduction

As the scale of a simulation modeling effort increasing, the V&V required is increasingly difficult. For large enough model size and complexity, human intuition fails and provides little guidance as to what to expect and what constitutes correct behavior. This is especially true of models containing rules whose interaction is not amenable to easy analysis and where phenomena are expected to emerge that can not be anticipated apriori. Currently, to our knowledge, there are *no general tools* that would ease the V&V process for such cases. We have not included "general theories and concepts" in the foregoing statement, since we believe that a theory and framework *is* available to support the development of such tools. There are two kinds of general tools that can be considered. In the first case, a tool can accept information about the model's application domain and employ this information as input to otherwise generic routines (just as a domain-independent rule-based expert system engine can accept domain specific rules and thereby become tuned to the domain in question). In the second case, the tool can be entirely domain-independent in the sense that it only operates with concepts that are applicable to any model independently of its domain semantics. This distinction can be viewed in terms of "generic" vs. "domain-dependent" worldviews where the former offers certain advantages over the latter.

For modeling and simulation, we believe the time is ripe for the domain-independent (generic) variety to be developed for application to the large-scale complex modeling and simulation cases just mentioned. When familiarity with the domain is not a guide, only generally applicable concepts can be relied upon and that is where the theory of modeling and simulation can be most valuable. The burning question then becomes – which generic concepts can be useful and to which issues in V&V do they usefully apply? In this paper we address this question in some depth. We hope to demonstrate the necessity for continued development of the theory of modeling and simulation and its potential for supporting the development of both generic and domain-dependent tools in V&V.

1.1. General vs Specialized Systems

George Klir provided an excellent starting point for our discussion of generic versus domain-dependent in his characterization of systems science [1]. Summarized in Figure 1, if a system is taken to be a set of things connected through some relations, then we can make distinctions among systems based on their constituent *things* or on their *relations*. In the first case, the study leads to traditional sciences such as physics and chemistry; in the second, we get system science, a non-traditional science with its own domain of inquiry -- general systems and problems, body of knowledge – systematization of relational properties, and methodology – collection of methods for solving general systems problems. Solution to general systems problems are relevant for the particular sciences to the extent that the abstraction process, through which a general system is taken to represent many particular systems, preserves the essence of a particular problem and its solution can be effectively re-interpreted in the context of a particular application. For example, control theory has provided generic algorithms that when interpreted in a wide variety of engineering disciplines provide effective control mechanisms.

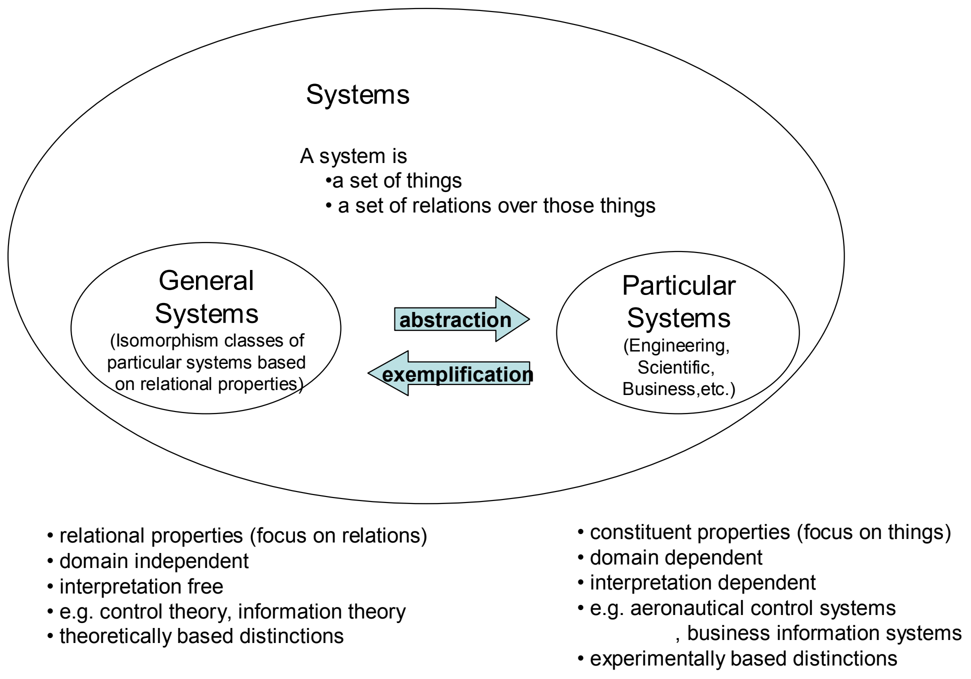

Systems
A system is
•a set of things
• a set of relations over those things
General Systems
(Isomorphism classes of particular systems based on relational properties)
abstraction
exemplification
Particular Systems
(Engineering, Scientific, Business,etc.)
• relational properties (focus on relations)
• domain independent
• interpretation free
• e.g. control theory, information theory
• theoretically based distinctions
• constituent properties (focus on things)
• domain dependent
• interpretation dependent
• e.g. aeronautical control systems , business information systems
• experimentally based distinctions

Figure 1:

The theory of modeling and simulation is a general systems theory in the spirit of Klir's characterization. The theory formulates a basic set of entities and relationships that are abstracted from real-world modeling and simulation enterprises and formulates relations among them that are asserted to be the critical ones for working in M&S. Within this framework, it formulates classes of problems that are phrased in the abstract terms of its entities and relations – among which are the problems of verification and validation under focus here.

2. Approaches to Characterization of Systems

There are many kinds of systems (biological, cosmological, artificial, etc.) that have been characterized in mathematical, computational or other forms of description. These systems have been characterized at different levels of abstraction using many approaches founded on principles such as physics, chemistry, biology, etc.. For the purposes of this paper, we focus on two broad approaches which apply to general (domain-independent) system-level characterizations.

2.1. Unified Modeling Language

Software engineering promotes systematic, disciplined, and quantifiable approaches to the development, operation, and maintenance of software-intensive systems. By applying engineering principles to software, it strives to bring together methods, processes, and tools in a unified fashion. While fundamentally different approaches to software engineering have emerged in recent years, the object-orientated approach has become widely accepted and practiced [2-4]. In the object-oriented worldview, the software development process includes conceptualization, analysis, design, and evolution (e.g., [2]) and supports the architecture-driven paradigm based on a hybrid of spiral and concurrent software development processes. Adherents of the object-oriented approach consider it superior to other software development approaches such as the functional and procedural methods. Furthermore, the modular architecture-driven approach can strongly support incremental, stepwise, iterative specification, design, and development of hardware and software components concurrently. Other advantages of the object-oriented approach include: support for scalable high-performance execution and model development; dynamic reconfiguration; systematic and incremental verification and testing; and team-oriented development. The adaptation of object orientation to software engineering has become increasingly indispensable for systems exhibiting heterogeneity and demanding flexibility in terms of both software and interoperability with multiple hardware components.

The unified modeling language (UML) has been managed by the vendor-neutral Object Management Group (OMG) since 1997. UML originated as a combination of approaches to software modeling developed by J. Rumbaugh, I. Jacobson, and G. Booch, but has evolved into a public standard with contributions by many other researchers and practitioners [5]. OMG committees are defining ways in which the next version of UML can facilitate activities such as: the design of web applications; enterprise application integration; real-time systems; and distributed platforms. UML attempts to support a higher-level view of design and coding in terms of diagramming. However, the majority of developers still build in source code, working with linguistic rather than spatial intelligence. UML vendors are attempting to educate programmers to pay attention to design views, allowing users to decide which of them they want to see at any given time. UML definition is still in a state of flux. For example, many proponents believe that its features should be reduced to a small core or kernel (see Figure 2). One proposal for such a kernel would include use cases, class diagrams, and interaction diagrams but would exclude state charts and activity graphs that provide some of the richest semantics in UML.

UML is aimed at general software development, e.g., for business applications, and is not simulation-aware. UML is the union of a number of diagramming notations based on informal and in some case formal concepts and theories. However, in the realm of software engineering, and in particular, software development for models and simulations, it is increasingly imperative to rely on rigorous methods and approaches and beyond informal practices. In addition to the factors relating to all software, which include software design principles, exploiting patterns, and scalable architecture, the M&S developer must understand the particular characteristics of dynamic systems, the error properties of numerical algorithms, and the intricacies of parallel and distributed simulation protocols. However, while these considerations are considered in software engineering, their treatments are rather informal. For example, although state and state chart diagrams are included in UML, they are inadequate to handle the variety of dynamic systems of interest in M&S. UML does not support model construction from dynamic system components or from reusable model components as required for SBA [6]. Fundamentally, UML should be applied to the development of software to support modeling and simulation, but not to the construction of dynamic system models.

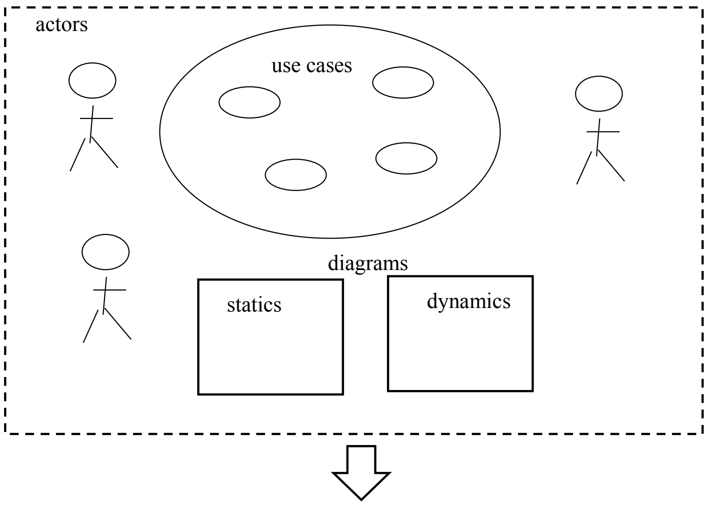


Figure 2: Specifying User Requirements in UML

2.2. Systems Theory

Mathematical systems theory, first developed in the nineteen sixties [1, 7-11], provides a fundamental, rigorous mathematical formalism for representing dynamical systems. There are two main, and orthogonal, aspects to the theory:

- **levels of system specification** – these are the levels at which we can describe how systems behave and the mechanisms that make them work the way they do.
- **systems specification formalisms –** these are the types of modeling styles, such continuous or discrete, that modelers can use to build system models.

System theory distinguishes between system *structure* (the inner constitution of a system) and *behavior* (its outer manifestation). Viewed as a black box (Figure 3) the external *behavior* of a system is the relationship it imposes between its input time histories and output time histories. The system's input/output behavior consists of the pairs of data records (input time segments paired with output time segments) gathered from a real system or model. The internal *structure* of a system includes its state and state transition mechanism (dictating how inputs transform current states into successor states) as well as the state-to-output mapping. Knowing the system structure allows us to deduce (analyze, simulate) its behavior. Usually, the other direction (inferring structure from behavior) is not univalent – indeed, discovering a valid representation of an observed behavior is one of the key concerns of the modeling and simulation enterprise.

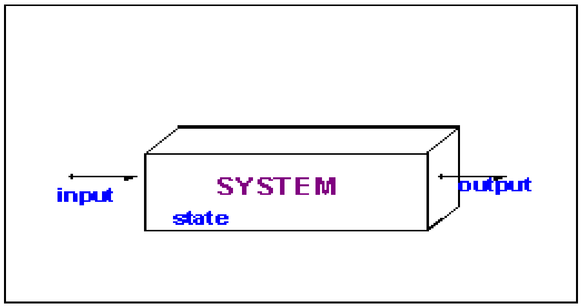


Figure 3: Basic System Concepts

An important structure concept is that of *decomposition* namely, how a system may be broken down into component systems (see Figure 4). A second concept is that of *composition,* i.e., how component systems may be coupled together to form a larger system. Systems theory is *closed under composition* in that the structure and behavior of a composition of systems can be expressed in the original system theory terms. The ability to continue to compose larger and larger systems from previously constructed components leads *to hierarchical construction.* Closure under composition guarantees that such a composition results in a system, called its *resultant*, with well-defined structure and behavior. *Modular* systems have recognized input and output ports through which all interaction with the environment occurs. They can be *coupled* together by coupling output ports to input ports and can have hierarchical structure in which component systems are coupled together to form larger ones. The difference between a decomposed systems, as in Figure 3 and undecomposed systems, as in Figure 4, suggests that there are distinct *levels* of systems specification – the former are at a higher level of specification than the latter since they provide more information about the structure of the system.

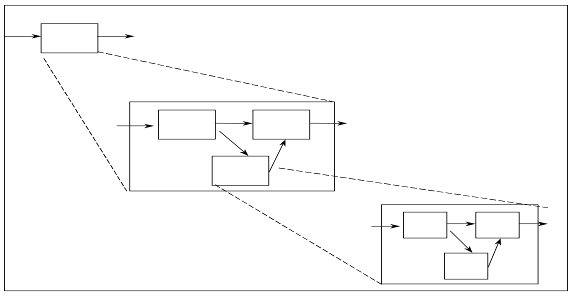


Figure 4: Hierarchical System Decomposition

The hierarchy of system specification levels [12, 13] is summarized in Table 1. As the level in the table increases, we move in the direction of increasing structural specificity (i.e., from behavior to structure).

0	I/O Frame	(T, X, Y)
1	I/O relation observation	(T, X, Ω, Y, R)
2	I/O function observation	(T, X, Ω, Y, F)
3	I/O system	$(T, X, \Omega, Y, Q, \Delta, \Lambda)$
4	Iterative specification	$(\mathrm{T}, \mathrm{X}, \Omega_{\mathrm{G}}, \mathrm{Y}, \mathrm{Q}, \delta, \lambda)$
5	structured system specification	$(T, X, \Omega, Y, Q, \Delta, \Lambda)$ with X, Y, Q, Δ, and Λ are structured
6	non-modular coupled multi-component system	$(T, X, \Omega, Y, \mathrm{D}, \{M_d = (Q_d, E_d, I_d, \Delta_d, \Lambda_d)\})$
7	modular coupled network of systems	$N = (T, X_N, Y_N, D, \{M_d \mid d \in D\},$ $\{I_d \mid d \in D \cup \{N\}\}, \{Z_d \mid d \in D \cup \{N\}\})$

Table 1 Hierarchy of System Specifications

2.3. Continuous and Discrete-event Worldviews

Figure 5 depicts the basic systems modeling formalisms considered as shorthand means of delineating a particular system within a subclass of all systems. For the traditional formalisms, mathematical representation had proceeded their computerized incarnations. However, the reverse was true for the third class, the Discrete Event System Specifications (DEVS) [14]. Discrete event models were largely prisoners of their simulation language implementations or algorithmic code expressions. Indeed, there was a prevalent belief that discrete event “worldviews” constituted new mutant forms of simulation, unrelated to the traditional mainstream paradigms. Fortunately, that situation has changed as the benefits of discrete event abstractions in control and design became clear (e.g., [15-17]) and discrete event distributed simulation continues to become integral development of all systems. With respect to such alternative discrete event formalisms, DEVS has been shown to be universal in the sense that any discrete event behavior can be expressed with it [2].

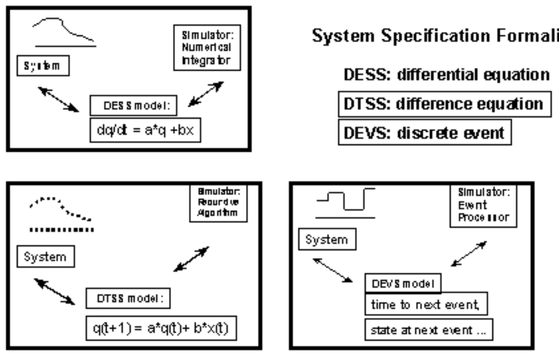

System
Simulator: Numerical Integrator
DESS model:
dq/dt = a*q +bx
System Specification Formalism:
DESS: differential equation
DTSS: difference equation
DEVS: discrete event
Simulator: Recursive Algorithm
System
DTSS model:
q(t+1) = a*q(t)+ b*x(t)
Simulator: Event Processor
System
DEVS model
time to next event,
state at next event ...

Figure 5: Basic Systems Specification Formalisms

System theory, and its mathematical instantiations, provides a rigorous framework for M&S but it lacks a computational and conceptual infrastructures. Object orientation (OO) provides a strong computational basis for M&S. Currently, OO is being given commonly accepted conceptual interfaces such as UML for general software development.

2.4. Relation To Object Orientation

Models developed in a system theory paradigm bear a resemblance to concepts of object-oriented programming. Both *objects* and system models share a concept of internal state. However, mathematical systems are formal structures that operate on a time base while programming objects typically do not have an associated temporal semantics. Objects in typical object-oriented paradigms are not hierarchical or modular in the sense just described. The coupling concept in modular systems provides a level of delayed binding – a system model can place a value on one of its ports but the actual destination of this output is not determined until the model becomes a component in a larger system and a coupling scheme is specified. It can therefore: **a)** be developed and tested as a stand alone unit, **b)** be placed in a model repository and reactivated at will and **c)** reused in any applications context in which its behavior is appropriate and coupling to other components makes sense.

While coupling establishes output-to-input pathways, the systems modeler is completely free to specify how data flows along such channels. Information flow is one of many interactions that may be represented. Other interactions include physical forces and fields, material flows, monetary flows, and social transactions. The systems concept is broad enough to include the representation of any of these and supports the development of M&S environments that can make including many within the same large-scale model.

Although systems models have formal temporal and coupling features not shared by conventional objects, object-orientation does provide a supporting computational mechanism for system modeling. Indeed, there have been many object-oriented implementations of hierarchical, modular modeling systems. These demonstrate that object-oriented paradigms, particularly for distributed computing, can serve as a strong foundation to implement the modular systems paradigm.

The Unified Modeling Language is inadequate when solely applied to M&S since it is not intended to handle intrinsic M&S elements such as characterization of continuous and discrete dynamics, model composability, abstraction-related families of models, efficient simulation infrastructure, etc. A more encompassing M&S theory is needed to address these requirements. Such a theory built on theory of modeling and simulation, systems theory, and UML combined with a computational basis to support execution of large-scale simulation models.

3. Framework for Modeling and Simulation

The *Framework for M&S* in [12] establishes *entities* and their *relationships* that are central to the M&S enterprise (see Figure 6). Terms such as "model" and "simulator" are often loosely used in current practice but have a very sharp meanings in the framework we will discuss. Based on this framework, the basic issues and problems encountered in performing M&S activities can be better understood and coherent solutions developed Therefore, it is important to understand what is included and excluded by the definitions.

The entities of the Framework are*: source system, model, simulator* and *experimental frame*; they are related by the *modeling* and the *simulation* relationships. Each entity is formally characterized as a system at an appropriate level of specification of a generic dynamic system. Likewise each relationship is characterized as an appropriate morphism between corresponding dynamic systems. Presentation of the systems theory background is beyond the scope of this paper but can be found in [3] and is briefly reviewed in another paper in these proceedings [4]. Assuming this background, the following briefly reviews the concepts of the M&S Framework.

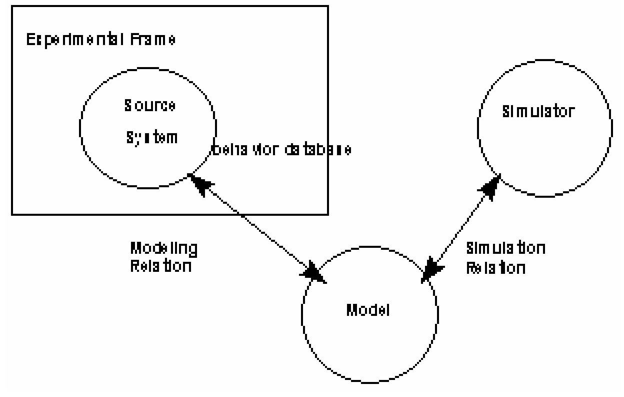


Figure 6: Framework Entities and Relationships

3.1. Source System

The source system is the real or virtual environment that we are interested in modeling. It is viewed as a source of observable data, in the form of time-indexed trajectories of variables. The data that has been gathered from observing or otherwise experimenting with a system is called the system behavior database. This data is viewed or acquired through experimental frames of interest to the modeler.

3.2. Experimental Frame

An experimental frame is a specification of the conditions under which the system is observed or experimented with. An experimental frame is the operational formulation of the objectives that motivate a modeling and simulation project. A frame is realized as a system that interacts with the system of interest to obtain the data of interest under specified conditions (Figure 7).

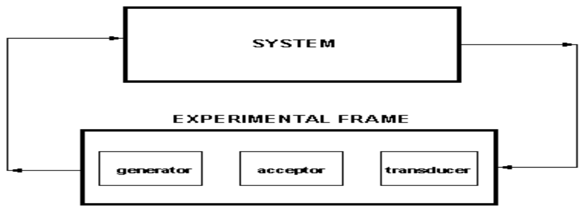


Figure 7: Experimental Frame

For example, experimental frames that are developed for contrasting objectives, such as interdiction and prevention, are quite different. The first (interdiction) calls for experimenting with a model in which all known prevailing fuel, wind and topographic conditions are entered to establish its initial state. The output desired is a detailed map of fire spread after say five hours within the region of interest. The second experimental frame (prevention) calls for a wider scope, lower resolution representation of the landscape in which a range of expected lightning strike, wind, rain and temperature regimes may be injected as input trajectories. The model may then be placed into different states corresponding to prevention/mitigation alternatives, e.g., different fire break spatial regions. The output for a particular run might be as simple as a binary variable indicating whether or not the residential area was engulfed by fire. The output summarized over all runs, might be presented as a rank ordering of alternatives according to their effectiveness in preventing fire spreading to the residential area (e.g., the percent of experiments in which the residential area was not engulfed by flame).

3.3. Model

In its most general guise, a model is a system specification at any level in the behavior-structure hierarchy. However, in the traditional context of M&S, the system specification is usually at a high level, such as a set of instructions, rules, equations, or constraints for generating input/output behavior. In other words, we write a model with state transition and output generation mechanisms to accept input trajectories and generate output trajectories depending on its initial state setting. Such models form the basic components in more complex models that are constructed by coupling them together to form models at a higher level specification. Models may be expressed in a variety of formalisms which may be understood as means for specifying subclasses of dynamic systems. The DEVS formalism delineates the subclass of discrete event systems and, through its representational ability, it can express the systems specified within traditional formalisms such as differential (continuous) and difference (discrete time) equations [3].

There are many meanings that are ascribed to the word "model". For example, a model is conceived as any physical, mathematical, or logical representation of a system, entity, phenomenon, or process. The definition in terms of system specification has the advantages that it has a sound mathematical foundation and has a definite semantics that everyone can understand in unambiguous fashion.

3.4. Simulator

A simulator is any computation system (such as a single processor, or a processor network, or more abstractly an algorithm), capable of executing a model to generate its behavior. The more general purpose a simulator is the greater the extent to which it can be configured to execute a variety of model types. In order of increasing capability, simulators can be:

- dedicated to a particular model or small class of similar models
- capable of accepting all (practical) models from a wide class, such as an application domain (e.g., communication systems)
- restricted to models expressed in a particular modeling formalism, such as continuous differential equation models
- capable of accepting multi-formalism models (having components from several formalism classes, such as continuous and discrete event).
- Validation and Verification Relationships

The entities – *system, experimental frame, model, simulator* – take on real importance only when properly related to each other. For example, we build a model of a particular system for some objective – only some models, and not others, are suitable. Thus, it is critical to the success of a simulation modeling effort that certain relationships hold. Two of the most important are *validity* and *simulator correctness*.

4. M&S-based VV&A Concepts and Approaches

The basic concepts of verification, validation, and accreditation have been described in different settings, levels of details, and points of views (e.g., see [18]). These concepts have been studied by a variety of scientific and engineering disciplines and various flavors of validation and verification concepts and techniques have emerged from a modeling and simulation perspective. The validation and verification concepts are themselves founded on more primitive concepts such as multi-level system specifications and homomorphism (e.g., [8, 19, 20]).

Within the modeling and simulation community, a variety of methodologies for V&V have been suggested by authors such as Knepell and Aragno [21], Law and Kelton [22] and Sargent [23]. Balci [24] offers a collection of 77 verification, validation and testing techniques along with 15 principles to guide the application of these techniques for a M&S lifecycle [25]. These techniques, however, vary extensively – e.g., alpha testing, induction, cause and effect graphing, inference, predicate calculus, proof of correctness, and user interface testing. For example, in terms of verification, these techniques can be categorized as informal, static, dynamic, symbolic, constraint, and formal [26]. However, despite the categorization of these techniques, it is evident that they are loosely related to one another and therefore serve as informal guidelines for VV&A. For example, even though these techniques are presented and discussed with the Conical simulation study lifecycle [27], it is easy to see that the relationship between the techniques and methodology is rather informal. As a consequence, the plethora of VV&A techniques does not lend itself as mathematically-based framework for VV&A.

Others such as Wymore [19] and Zeigler [12, 13] have proposed and described mathematically-based concepts and frameworks to support formal verification and validation from the system-theoretic point of view (see Section 6.6 for other V&V formal methods). The system-theoretic approach offers a rigorous foundation to express a variety of relationships (e.g., modeling and simulation relations) that are necessary to move from model specifications to their interpretations. It is, of course, important to note that the formal and informal approaches complement one another toward addressing VV&A challenges.

Under the DMSO VV&A Program [18], verification, validation, and accreditation has been studied extensively from the M&S perspective[4]. The DMSO VV&A Program has developed a Recommended Practices Guide containing a set of terms divided into generic (established) and special (adopted) categories [33]. The general category consists of 45 terms including verification, validation, and accreditation. The special category adopts a set of terms (such as data, data quality, exercise, joint exercise, joint training, multinational exercises, and service training) to support exercise and training needs. The basic definitions provided are:

Verification: The process of determining that a model implementation and its associated data accurately represents the developer's conceptual description and specifications.

Validation: The process of determining the degree to which a model and its associated data are an accurate representation of the real-world from the perspective of the intended uses of the model.

Accreditation: The official certification that a model, simulation, or federation of models and simulations and its associated data are acceptable for use for a specific purpose.

4.1. Validation

The basic modeling relation, *validity,* refers to the relation between a model, a system and an *experimental frame*. Validity is often thought of as the degree to which a model faithfully represents its system counterpart. However, it makes much more practical sense to require that the model faithfully captures the system behavior only to the extent demanded by the objectives of the simulation study. In our formulation, the concept of validity answers the question of whether it is impossible to distinguish the model and system in the experimental frame of interest. The most basic concept, *replicative validity*, is affirmed if, for all the experiments possible within the experimental frame, the behavior of the model and system agree within acceptable tolerance. Thus replicative validity requires that the model and system agree at the input/output (I/O) relation level of the system specification hierarchy.

Stronger forms of validity are *predictive validity* and *structural validity*. In predictive validity we require not only replicative validity, but also the ability to predict as yet unseen system behavior. To do this the model needs to be set in a state corresponding to that of the system. Thus predictive validity requires agreement at the next level of the system hierarchy (see Section 2.2), that of the I/O function. Structural validity requires agreement at level 3 (state transition) or higher (coupled component). This means that the model not only is capable of replicating the data observed from the system but also mimics in step-by-step, component-by-component fashion, the way that the system does its transitions.

The term *accuracy* is often used in place of validity. Another often used term, *fidelity*, is often used for a combination of both validity and detail. Thus, a high fidelity model may refer to a model that is both highly detailed and valid (in some understood experimental frame). However when used this way, the assumption seems to be that high detail alone is needed for high fidelity, as if validity is a necessary consequence of high detail. In fact, it is possible to have a very detailed model that is nevertheless very much in error, simply because some of the highly resolved components function in a different manner than their real system counterparts.

[4] "Several studies discuss the importance of these for M&S research and development ([28-31])" [32].

4.2. Verification

The basic *simulation* relation, simulator correctness, is a relation between a *simulator* and a *model.* A *simulator correctly simulates a model* if it is guaranteed to faithfully generate the model's output trajectory given its initial state and its input trajectory. Thus, simulator correctness requires agreement at the I/O function level. In practice, as suggested above, simulators are constructed to execute not just one model but a family of possible models. In such cases, we must establish that a simulator will correctly execute a particular class of models. Since the structures of both the simulator and the model are at hand, it may be possible to prove correctness by showing that a *homomorphism* relation holds. As will be discussed in a moment, a homomorphism is a correspondence between simulator and model states that is preserved under transitions and outputs.

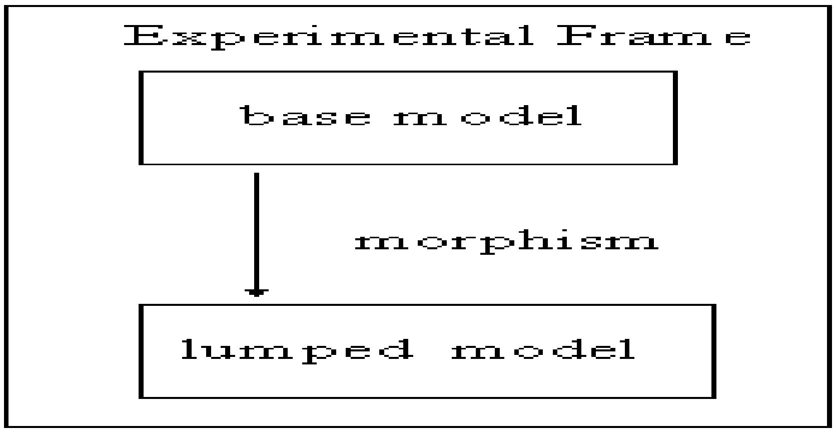


Figure 8: Base/Lumped Model Equivalence in Experimental Frame

4.3. Model Abstraction and Morphism

Besides the two fundamental relationships, there are others that are important for understanding V&V within a modeling and simulation framework. These relations have to do with the ordering of models and experimental frames. We have already mentioned the homomorphism concept which is a relation between two systems described at a high level of system specification.

4.4. Modeling Relation

It is often critical to know whether it makes sense to apply a given experimental frame to a model. The relation that determines if such an application is possible is called *applicability* and its converse, is called *accommodation*. Validity of a model in a particular experimental frame, requires, as a precondition, that the model accommodates the frame. For useful model repositories it is critical to have an ability to ask whether there are any experimental frames that meet our current objectives and whether there are models that can accommodate this frame. Only those models have a chance of providing valid answers to our current questions.

The degree to which one experimental frame is more restrictive in its conditions than another is formulated in the *derivability* relation. A more restrictive frame leaves less room for experimentation or observation than one from which it is derivable. So, as illustrated in Figure 9, it is easier to find a model that is valid in a restrictive frame for a given system. The applicability, accommodation, and derivability relationships, therefore, are important specific steps toward handling scope of validation.

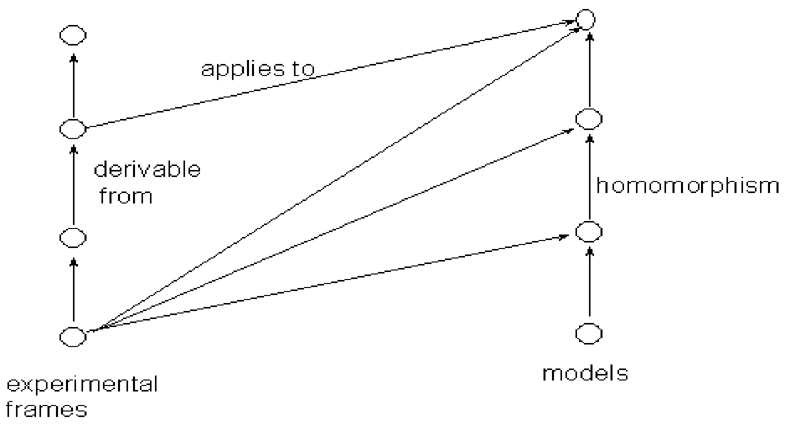


Figure 9: Illustrating Important M&S Relations.

4.5. Simulation Relation

As mentioned earlier, verification attempts to establish that the simulator correctness relation holds between a simulator and a model. Actually, as we have seen, the simulator may be capable of executing a whole class of models. In this case, we wish to guarantee that it can execute any of these models correctly. The basic concept in establishing correctness is that of *homomorphism*. When a model should go through a state sequence such as *a, b, c, d*, the simulator should go through a corresponding state sequence say *A, B, C, D*. Typically, a simulator has a lot of apparatus, represented in its states, necessary to accommodate the whole class of models rather than a single one. Thus we don't assume that simulator states and models states are identical – only that there is a predefined correspondence between them illustrated by the shaded connecting lines in the Figure 10.

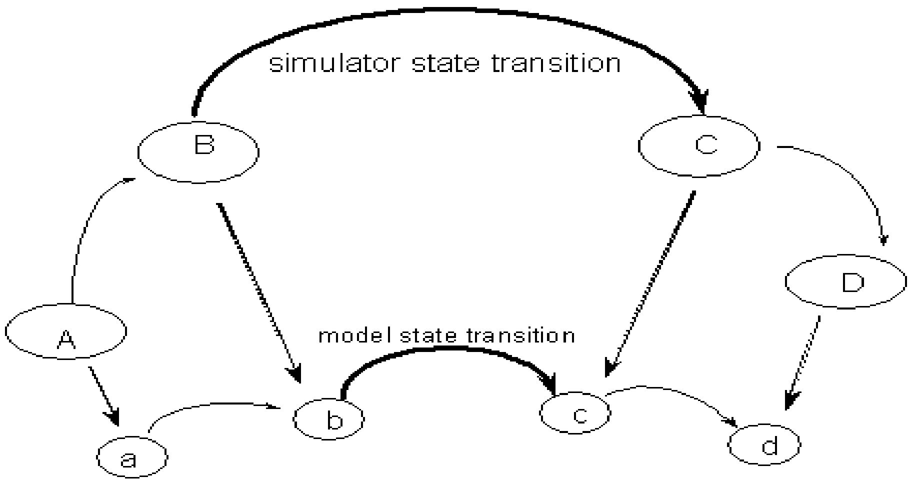


Figure 10: Homomorphism Concept

Now to establish that this correspondence is a homomorphism requires that whenever the model specifies a transition, such as from state *b* to state *c*, then the simulator actually makes the transition involving corresponding states *B* and *C*. Typically, the simulator is designed to take a number of microstate transitions to make the *macrostate* transition from *B* to *C*. These are computation steps necessary to achieve the desired end result. It is not hard to see that if a simulator designer can show that such a homomorphism holds then any state trajectory in the model will be properly reproduced by the simulator.

There are two general approaches to verification:

- formal proofs of correctness
- extensive testing

Formal proofs employ mathematical and logical formalisms underpinning the concepts in the systems specification hierarchy to rigorously establish the requisite homomorphism. Unfortunately, such proofs are difficult or impossible to carry out for large, complex systems. Moreover, they may also be prone to error since ultimately humans have to understand the symbols and carry out their manipulations. On the positive side, more automated tools are becoming available to relieve some of the burden.

In the absence of once-and-for-all proofs, extensive testing must be done to assure that all conditions that could arise in simulator operation have been covered by test cases. Time and other resources limit the amount of testing that can be done. However, even though formal proofs may not be employed, the homomorphism concept still offers a framework for laying out the combinations of inputs and states that have to be tested for thorough, yet efficient, testing.

4.6. VV&A Processes

To effectively relate specific techniques and methods for validation, verification, and accreditation, it is imperative to have a set of encompassing processes as described in DoD's VV&A Recommended Practices Guide [34]. These processes (e.g., New and Legacy M&S Development) offer roadmaps which may be used to determine satisfiability of VV&A objectives. Software tools supporting qualitative and quantitative measurements using techniques such as fuzzy arithmetic and rule-based expert knowledge with input from domain experts has been reported [24]. One key benefit of such processes is that they are independent of explicit techniques to be used, for example whether to use systems-theoretic or knowledge-based approaches for determining verification relation between a model and its implementation. In Section 6.4, we describe our approach within the HLA/FEDEP [35, 36] which itself is founded on the software engineering processes such as classical and spiral [2].

5. Key benefactors of V&V-based M&S Framework

The M&S architecture [37] shown in Figure 11 offers a high-level view of verification and validation using a layered separation of concerns. This architecture consists of six layers where the lower layers provide services for the upper layers: (1) Network (the lowest layer), (2) middleware, (3) simulation, (4) modeling, (5) search/decision, and (6) collaboration. All layers are required to support the full enterprise of M&S though some may be absent in particular settings. For example, while all the layers are necessary to support Simulation Based Acquisition, fewer layers are needed to support Collaborative Model Engineering.

It is intuitively evident that the architecture provides a holistic view of VV&A especially the middleware, simulation, and modeling layers. Multi-level system specifications based on abstraction and aggregation are principal schemes toward disciplined V&V given these three layers. Within this architectural framework, we describe the well-known issues of model composability, simulation interoperability and computational complexity.

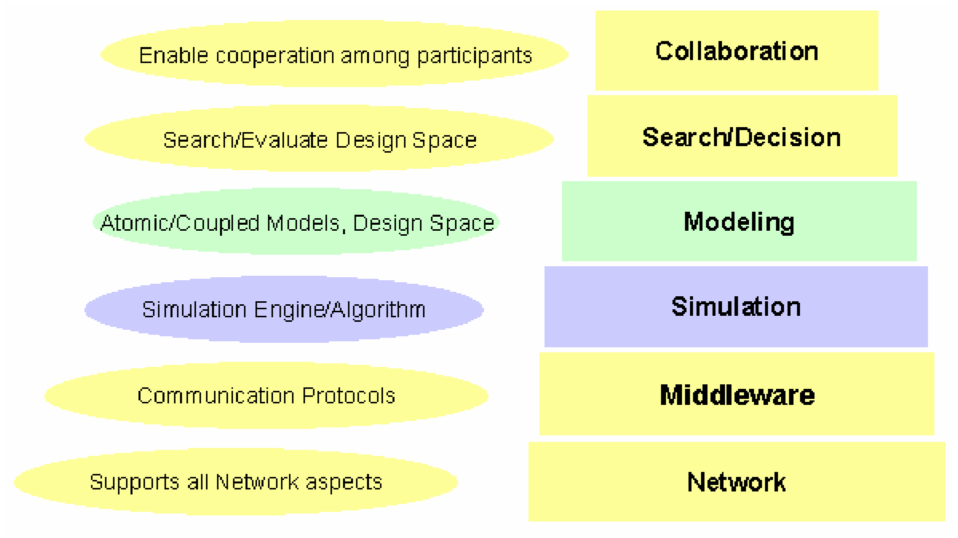

Enable cooperation among participants
Collaboration
Search/Evaluate Design Space
Search/Decision
Atomic/Coupled Models, Design Space
Modeling
Simulation Engine/Algorithm
Simulation
Communication Protocols
Middleware
Supports all Network aspects
Network

Figure 11: An Architecture for Modeling & Simulation

5.1. Model Composability

Aggregate models are increasingly constructed by combining off-the-shelf and new models – these models must of course satisfy requirements such as functionality and performance (e.g., [38-40]). Model composability (or integrability) refers to the synthesis of a collection of existing or new models in such a way that the resultant model behavior is the composition or "aggregation" of component model behaviors [41-45]. To be successful, this approach to synthesizing a new aggregate model from reusable and new models relies on a number of prerequisites:

- the behaviors of the putative components must be sufficiently well characterized or understood – this prerequisite will be satisfied to the extent that abstract specifications of the these behaviors exist and to the extent that V&V has achieved some confidence that the actual behaviors meet these specifications.

- putative components can be coupled together in a manner that the likelihood of achieving the desire composite behavior is high – this prerequisite will be satisfied to the extent that a composition framework exists to facilitate the coupling and a theory/methodology exists to guide the choice of components and coupling in plausible ways. V&V must be an integral part of the latter theory/methodology, since even assuming a high degree of confidence has been achieved in validation of the component behaviors, their coupling can produce unanticipated, emergent, or otherwise unpredicted behaviors that have to be characterized through a V&V process as either in conformance, or not, with the desired behavior.

We conclude that V&V must be integrated in an intimate manner within any proposed model composability approach since successful reuse of existing component models will depend heavily on the confidence one has on their "advertised" behaviors and in the ability to ascertain that their composition displays the desired behavior. To our knowledge, there is not as yet a working example of a model-composition framework with a robust range of reusable components, although there are examples of model composition and reuse within restricted domains ([46], [47]).

Currently, we lack of a robust theory on which to base selection of the size and content of modules, a theory/methodology for announcing/determining interdependencies between modules, and a means to constrain the potentially exponential number of compositions to those that are plausible given component behaviors and their interdependencies ([32, 48]).

5.2. Modeling and Simulation: Distinct Levels of Interoperability

Just as VV&A is critical to composability, it is also essential to simulation interoperability. We have asserted above that the separation between modeling and simulation is an essential concept in simulation model development. This separation leads naturally to a distinction between interoperability at the modeling and at the simulation levels. Simulation interoperability is concerned with multiple simulators interacting correctly with another. Specifically, interoperability at the simulation level refers to ability of different simulations to interoperate with one another using concepts such as common interface definitions. Interoperability at the modeling level requires a common formalism in which different type of models can be interconnected and interoperated. Interoperability at this level can be distinguished from interoperability at the simulation level which is the primary concern of current interoperability efforts surrounding the High Level Architecture standard [49], More specifically, interoperability at the simulation level is concerned with interoperating simulators that are described by interfaces that encapsulate and hide their underlying models. In contrast, interoperability at the modeling level is concerned more fundamentally with integrating models that are usually developed in different paradigms and expressed in different formalisms, such as analog/differential equations or digital/state machines. For more discussion on the different levels of interoperability see [49-51]. To see how V&V plays different roles in the levels of interoperability consider that verification of simulator correctness is dependent on ensuring the correct handling of a host of (primarily syntactic) issues such as data exchange compatibility (e.g., C++ vs. Java data types), time management in heterogeneous simulation protocols, and calls to a data repository. At the modeling level, often there is a need for co-existence of models at multiple levels of abstraction [52]. V&V is needed to assure that models residing at these abstraction levels form a coherent whole and relate to each other in a predictable and verifiable manner. The concept of homomorphism, and in particular, of parameter morphism, is essential to operationalizing such coherency verification. For more discussion of the parameter morphism concept see [13, 14].

6. Component-based M&S Framework

6.1. Discrete Event System Specification

In this section, we focus on the DEVS (Discrete Event System Specification) formalism's support for hierarchical modular model construction and component reuse. DEVS [14] has well defined concept of system modularity and component coupling to form composite models. It enjoys the property of closure under coupling which justifies treating coupled models as components and enables hierarchical model composition constructs. This considerable simplifies management of component models. The underlying structure of hierarchical models is captured in the DEVS concept called the system entity structure (SES) (e.g., [13]). We review this concept and discuss its application to construction of combinatorially-based families of models – as most large scale model development projects are likely to generate.

The DEVS modeling approach supports capturing a system's structure in terms of *atomic* and/or *coupled* models where coupled models are hierarchical satisfying closure under coupling [13]. While a part of a system can be represented as an atomic model with well-defined input/output interfaces, a system represented as a DEVS coupled model designates how systems can be coupled together to form system of systems. Such coupled models also have the same input/output interfaces. Given atomic models, DEVS coupled models can be formed in a straightforward manner. Both atomic and coupled models can be simulated using sequential and/or various forms of parallel computational techniques [53, 54]. A DEVS atomic model represents a component's dynamic behavior and has a well-defined mathematical structure as follows:

$$\textit{Atomic Model} = < X, Y, S, \delta, \lambda, ta > \text{ where}$$

X	set of *input events*,
S	set of *sequential states*,
Y	set of *output events,*
δ	*internal/external transition functions* specifying state transitions due to internal/external events,
λ	*output function* generating external events as outputs,
ta	*time advance* function.

Sets X, Y, and S can represent visible data of an atomic model. The remaining elements of the atomic model structure represent its dynamics. In terms of distributed modeling, internal functionality of an atomic model is of interest only to the extent of its states and input/output events (data) associated with the input/output ports. For analyzing DEVS modeling features, we may set aside the atomic model's dynamics and focus solely on its interface. That is, with respect to OMT, data available on input/output ports map into *interaction class parameters* and states map into *object class attributes*.

$$CM = < D, \{M_i\}, \{N_i\}, \{Z_{i,j}\} > \text{ where}$$

D	set of *component names*;
M_i	set of basic components for each i in D;
N_i	set of *influences* for each i in D;
$Z_{i,j}$	i to j output translation for each i in D and each j in N.

Given a coupled model CM, its representation is concise and reusable since any coupled model has a corresponding basic model due to the closure-under-coupling property [13]. The couplings among components can be systematically captured using output to input mappings (i.e., $Z_{i,j}$). In particular, there exist three different types of coupling: internal coupling, external input coupling, and external output coupling. Internal coupling interconnects components of a coupled model. External input coupling interconnects input ports of a coupled model to input ports of its components. Similarly, external output coupling interconnect component output ports of a coupled model to the output ports of the coupled model itself. Note that the ports provide a generic pipe through which a variety of messages (data/objects) can be transferred from one component to another.

6.1.1. Modularity and Coupling in DEVS

We provide an informal exposition of the DEVS formalism, focusing on the concepts of modularity and component coupling. Implicit in the presentation will also be the levels of system specification. More detail is available in many references (e.g., [2] also see URL: www.acims.arizona.edu). A DEVS model is a modular system receiving inputs, changing states, and generating outputs over a time base. Input and output streams are time-indexed series of events. Figure 12 illustrates an input event segment containing inputs x_0 x_1 x_2 arriving at times t_0 t_1 t_2 respectively; and a output event segment with outputs y_0 y_1 departing at times t'_0 t'_1 respectively. There are no constraints on the spacing between events; however, only a finite number of events are allowed in a finite time interval.

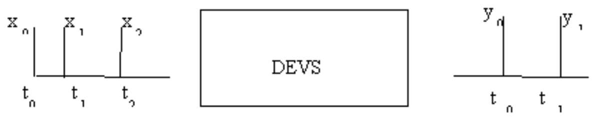


Figure 12: Input and Output Event Streams

DEVS models have input and output ports through which all interaction with the external world takes place. By coupling together output ports of one system to input ports of another, outputs are transmitted as inputs and acted upon by the receiving system. Thus, there are two types of DEVS models, *atomic* and *coupled*. An atomic model directly specifies the system's response to events on its input ports, state transitions, and generation of events on its output ports. A coupled model is a composition of DEVS models that presents the same external interfaces as do atomic models. For example, in Figure 13, CM is a coupled model with four components. A coupled model specifies three types of coupling:

- *external input* – from the input ports of the coupled model to the input ports of the components (e.g., from start of CM to start of counter)
- *internal* – from the output ports of components to input ports of other components (e.g., from explosion of bomb to strike of target), and
- *external output* – from the output ports of components to output ports of the coupled model (e.g.. from damage of target to damage of CM.

Although arbitrary fan-out and fan-in of coupling is allowed, no self-loops are permitted. DEVS is closed under coupling, which means that a coupled model can itself be a component within a higher level coupled model, leading to hierarchical, modular model construction.

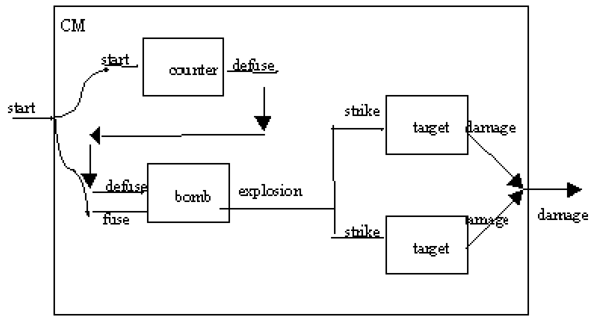


Figure 13: DEVS Coupled Model

To illustrate the DEVS modeling process, consider a scenario in which a bomb can threaten some objects in its vicinity called targets. If the bomb is left undisturbed over a long period it will lose its potency. If its fuse is ignited, it will in a very much shorter time, explode and impact the targets. However, if during this period, it is deactivated, the bomb will not go off but return to its dormant state. Many processes have similar *conditionally timed behavior*. For example, a grain can be dormant until picked up by a bee and transported to pollinate other flowers; but the bee may be killed before arriving at its destination thereby aborting the attempt at plant reproduction. A pre-emptive processor can be leave the job currently being done if interrupted by one of higher priority. A reliable network packet protocol waits for a fixed time for all packets in a message to be acknowledged by the receiver; otherwise it retransmits them (often called a timeout). The modular, state-based concept of DEVS enables it to respond such external inputs based on its current state and the time that has elapsed in that state.

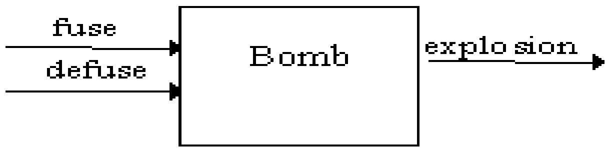


Figure 14: DEVS Atomic Model

To model this type of behavior in DEVS we define a coupled model, as in Figure 13, containing components for the bomb, some targets, and a defuser capable of deactivating the bomb. Each component is specified as a modular atomic model with input and output ports. In particular, the bomb has input ports for fusing (activating) and defusing (deactivating) it and an output port for the explosion event (Figure 14).

Due to its modularity, DEVS distinguishes between internal and *external* events. While internal events represent state changes that are self induced, *external input events*, on the other hand, are impressed from outside the model and are inputs to which the system must respond. For example, the arrival of a fuse event causes the phase to change from dormant to live. *As part of the effect of an external input event, the timing of the next internal event may be changed*. In this case, if the time advance associated with phase live is *t*, which means that in *t* units the phase is scheduled to change to explode (whereas before the external event, the dormant-to-dead transition was scheduled at some much later time). *Output external events* are generated by a system at its internal events. For example, the transition from explode to dead generates the explosion output event. Note that from the bomb's point of view, the arrival of the defuse input is unpredictable, it could arrive in time (i.e., before the time elapses in phase live) or after this time has elapsed (including never). The DEVS formalism, which modular concept of external events, requires and allows explicit specification of the dynamics of such conditionally timed events.

Figure 15a illustrates a *phase transition diagram* that portrays the dynamics of an atomic model. The two kinds of transitions are shown – *external*, an arrow labeled by an internal port, and *internal* – a dashed arrow labeled with a time advance. For example, the transition from dormant to live is externally induced by the fuse external event. The transition from live to explode is an internal transition as is the transition from explode to dead.

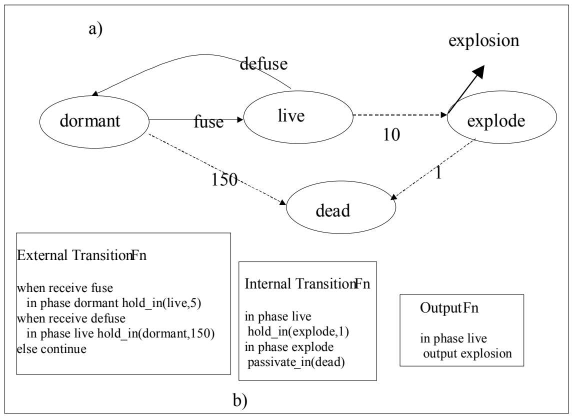


Figure 15: Atomic Model Specification

Such state diagrams are good for giving a graphical abstract representation of the dynamics but they have difficulty conveying their details. The authoritative specification of an atomic model is given in its internal transition, external transition, and output functions as in Figure 15b.

- *The External Transition Function* specifies how the system responds to external events. In general, this response is a change in state conditioned on the nature of the external event (input port of arrival and parameters of the received message), the current state, and the time elapsed since the last event. For example, an arrival on port fuse changes the phase from dormant to live. But the time that the system has been dormant might degrade the power of the explosion. This might be modeled by including a state variable, potency, that would be reset by the external transition function to a decaying exponential function of its original value and the elapsed time in phase dormant.
- *The Internal Transition Function* specifies the change in state that occurs upon an internal event. It is given as a function that maps the state before the event to the state that is to be in effect immediately after it.
- *The Output Function* specifies the output port and value that will be generated just before an internal event. It is given as a function that maps the state before the event to the output space (set of possible outputs). Not all states need to generate outputs. For example, transitioning from live to explode generates the explosion output, while no other transitions generate output events.

One characteristic DEVS feature that Figure 15b) does not show is *the time advance function*. This assigns to every state the time in which the system will stay in this state before an internal event occurs. Rather then give this function explicitly, we allow a state variable, called *sigma,* to hold its value for the current state, and to have this value set by the external and internal transition functions. The command "hold_in(live, 10)" means set the phase to live and sigma to 10. The command "passivate_in(dead) means to set phase to dead and sigma to infinity. In other words, the system is scheduled to remain in dead forever[5]. It is also possible for a state to have a zero time advance. For example, we might change the time to go from explode to dead (the explosion time) from 1 to 0.

[5] unless an external event changes this), but an event capable doing this is not shown in Figure 15.

6.1.2. Closure Under Coupling

An important feature of the DEVS formalism is that, like its systems superset, it closed under coupling. This means that the resultant of a coupled model consisting of DEVS components is itself expressible within the DEVS formalism. Such closure supports the ability to "componentize" a coupled model, i.e., to encapsulate it in a form that enables it to be treated as an atomic model. This closure property further guarantees that coupled models can be included as components in more inclusive coupled models thus supporting stagewise hierarchical construction. This simplifies development, verification and management of component models.

6.2. System Entity Structure

A System Entity Structure provides the means to represent a family of models as a labeled tree [13, 55]. Two of its key features are support for decomposition and specialization. The former allows decomposing a large system into smaller systems. The latter supports representation of alternative choices. Specialization enables representing a generic model (e.g., a computer display model) as one of its specialized variations (e.g., a flat panel display or a CRT display.) Based on SES axiomatic specifications, a family of models (design-space) can be represented and further automatically pruned to generate a simulation model. Such models can be systematically studied and experimented based on alternative design choices. An important, salient feature of SES is its ability to represent models not only in terms of their decomposition and specialization, but also their aspects (SES represents alternative decompositions via aspects.) With respect to Figure 11, modeling services of DEVS and SES primarily match up with the Modeling layer and the model design space matches up with the Search/Decision layer.

The system entity structure (SES) formalism provides an operational language for specifying such hierarchical structures. An SES is a structural knowledge representation scheme that systematically organizes a family of possible structures of a system. Such a family characterizes decomposition, coupling, and taxonomic relationships among entities. An *entity* represents a real world object. The decomposition of an entity concerns how it may be broken down into sub-entities. As already mentioned, coupling specifications tell how sub-entities may be coupled together to reconstitute the entity. The *taxonomic* relationship concerns admissible variants of an entity.

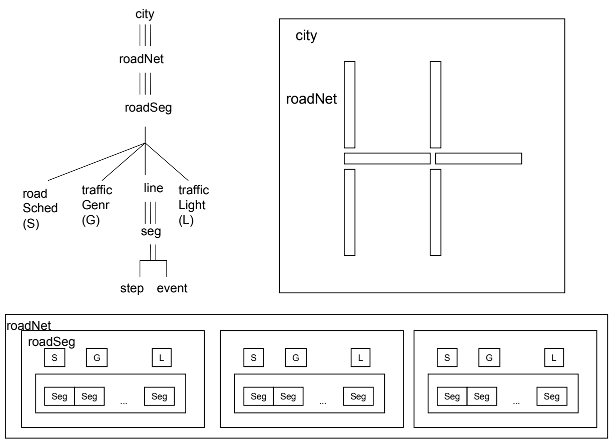


Figure 16: A System Entity Structure

The SES/MB framework [3-5] is a powerful means to support the plan-generate-evaluate paradigm in systems design. Within the framework, entity structures organize models in model base. Thus, modeling activity within the framework consists of three sub-activities: specification of model composition structure, specification of model behavior, and synthesis of a simulation model.

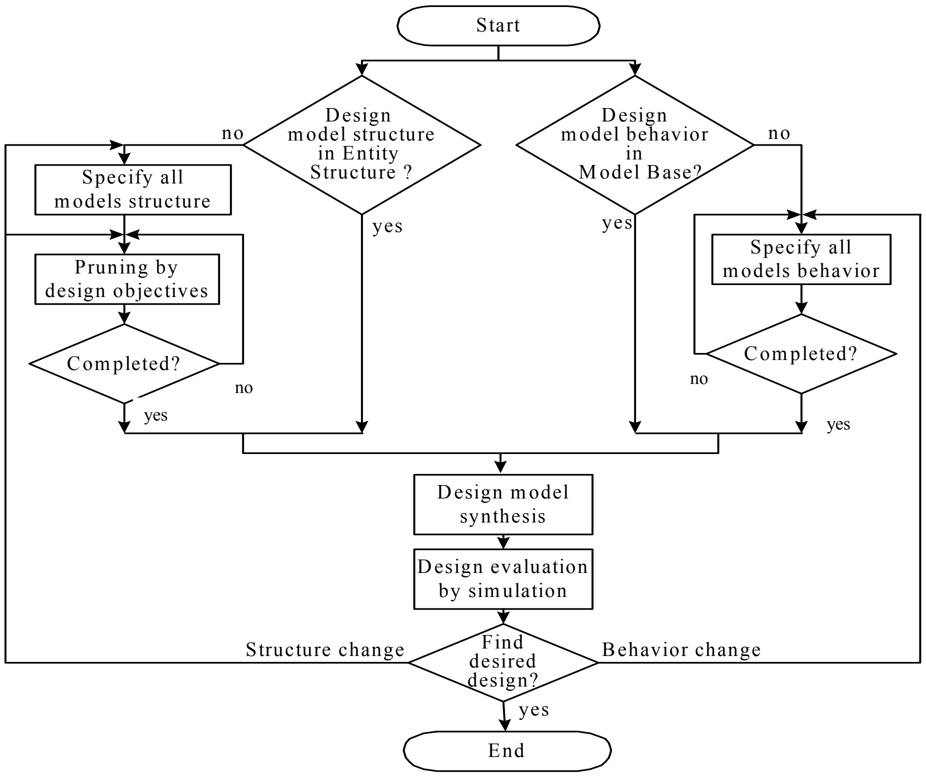

Start
Design model structure in Entity Structure ?
Design model behavior in Model Base?
no
no
yes
yes
Specify all models structure
Pruning by design objectives
Completed?
no
yes
Specify all models behavior
Completed?
no
yes
Design model synthesis
Design evaluation by simulation
Structure change
Find desired design?
Behavior change
yes
End

Figure 17: Design Methodology Using SES/MB Framework

Figure 17 shows a modeling and simulation methodology based on the framework in the process of iterative systems design. In the figure, the generation phase consists of two sub-phases: *pruning* and *model synthesis*. The structure specification and/or the behavior specification may already exist in the entity structure base and/or model base. However, if the structure specification is not in the entity structure base, we need to specify it by building an SES which represents a family of possible model structures. Likewise, if the desired model components are not in the model base, we need to develop them and store them in the model base for later use. In the pruning phase, we select a sub-structure by pruning the SES with respect to design objectives. A simulation model is automatically synthesized from such a pruned entity structure. Simulation experiments may require changes of structure and/or behavior of the design model. The pruning-synthesis-evaluation process is repeated until a desired design is found. Once simulation experiments are completed, the designer can save structure and behavior specifications in the system entity structure base and the model base, respectively for later use.

6.2.1. Automatic Pruning of an SES

Instead of having the user prune an SES for a desired model structure, we can provide an automatic means of iterating through all prunings [13, 55]. Provided that the number of prunings is not too large, this provides an automated search capability for finding a best design. If the number of design alternatives is too large for an exhaustive search, we have to turn to more natural, and artificial, intelligence to constrain the search space. To provide such search capability we need an algorithm that, given an SES, is capable of computing its total number of prunings and iterating through them one-at-a-time, each time synthesizing the associated hierarchical simulation model from the model base and evaluating it. The design of such an algorithm is based on the fact that an SES is built recursively with alternating specializations and decompositions. Likewise, the number of alternative prunings can be enumerated and generated recursively. The number of alternatives under an aspect is the *product* of the alternatives under its entities. The number of alternatives under a specialization is the *sum* of those under its entities. This recursion stops when leaf entities are encountered. The number of alternatives represented by a leaf entity is just 1 (itself). Finally, the number of prunings of the SES is the number of alternatives under the root node.

In DEVSJAVA [56], the basic DEVS formalism and the coupled model formalism are encoded in respective classes. The SES capability is obtained by extending the coupled class to a class which treats members as alternatives in a specialization rather than simultaneously coexisting components as in its parent class. For example, in Figure 18 there are two specializations of a processor, one with, and the other without, a buffer to queue the arriving jobs. Class procQ (with queue) is derived from class proc and they both have the same I/O ports. Thus they are proper alternative choices in a processor specialization.

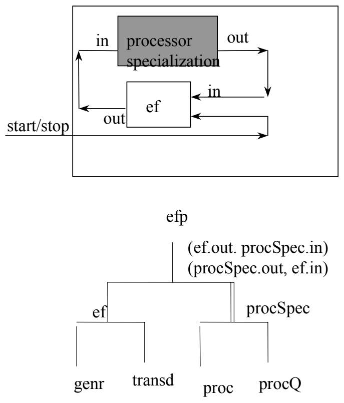


Figure 18: Experimental frame coupled to processor specialization

To be of use, such a specialization must be able to replace a component in another coupled model. For example, we could have it coupled to an experimental frame as in Figure 18. We call class efp *prunable* since it has within it (at least) one specialization. By selecting one of the two choices in procSpec, we get a new model that is pruned and simulateable. After pruning and transforming, the modeler can immediately experiment with the resulting model. Alternatively, s/he can issue a command to write the model description into an automatically created class file. This enables reuse of the pruned structure as an executable simulation model and obviates the need to prune the master structure again to get the same model. Reuse of pruned structures adds a second layer of reuse to that of initially constructed components.

6.2.2. Families of Models: Constraints on Coupling

Constraints on coupling assure that any source-generated value can be handled by the receiving component. In DEVSJAVA, an object-oriented implementation of DEVS formalism, such values are instances of classes derived from the base class. Since such objects must be manipulated by the receiver, the constraints translate into the requirement that that the receiver should be able to apply any method it needs to the instances. As illustrated in Figure 19, classes may be associated with ports. Associating a class c with output port p, indicates that values in contents of messages sent on port p are instances of class c. Similarly, associating a class with an input port means that the receiver expects to treat values arriving on this port as instances of this class.

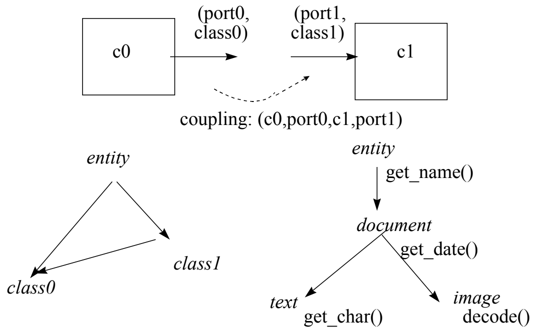


Figure 19: Constraints on coupling

Thus, we can couple an output port port0 with class0 to an input port port1 with class1 if every method in class1 is also in class0. Typically this is guaranteed by inheritance from class1 to class0 (yes, this is the right direction!). For example in Java, this is satisfied if class0 is derived from class1 and the methods are public. As an important special case, the requirements are satisfied if class0 and class1 are actually the same class. Table 2 lists all admissible combinations for the inheritance hierarchy in Figure 19. For example, if the receiver expects an `entity` (base class instance) than any of the derived class instances can be sent to it. If class1 is `document` than class0 can't be `entity`, but can be `document` or its derived classes. Finally, if the receiver expects a `text` document then only a `text` instance can be sent to it. This form of type checking (i.e., coupling) is necessary to assure meaningful coupling of such components and thus important toward V&V.

Sender class0	Receiver class1	some applicable methods at receiver	some methods **not** applicable at receiver
Entity	entity	get_name()	
document	entity	get_name()	get_date()
Image	entity	get_name()	decode()
Text	entity	get_name()	get_char()
document	document	get_date()	get_char(),decode()
Image	document	get_date()	decode()
Text	document	get_date()	get_char()
Image	image	decode()	get_char()
Text	text	get_char()	decode()

Table 2: Illustrating constraints on coupling

6.3. High Level Architecture

The HLA represents a major advance in M&S methodology as an IEEE standard mandated by U.S. Department of Defense (DoD) to promote simulation interoperability and reuse. The HLA standardization, however, standardizes only certain aspects of the M&S process while leaving others unspecified. HLA defines Object Model Template (OMT) for Simulation Object Model (SOM) and Federation Object Model (FOM) interface specification and Run Time Infrastructure (RTI) for run-time data exchange. It also defines Federation Development and Execution Process (FEDEP) as the methodology of choice for simulation software development.

6.3.1. HLA/OMT

The concept of an HLA Object Model Template (OMT) plays a central role in building HLA-compliant simulations as viewed by HLA standard[6]. The primary role of HLA/OMT is to *document* HLA-relevant information about federates and federations. The HLA/OMT specifies two object models: *Federation Object Model* (FOM) and *Simulation Object Model* (SOM). Specifically, there are two main technical objectives for the HLA/OMT 1.3 Specification [36]. The first objective is aimed at FOM and thus it is to provide a common specification for *exchange of data* and *general coordination* among members of a federation. Correspondingly, the second objective is aimed at SOMs. It is to provide a common, *standardized mechanism* for describing the capabilities of *potential* federation members. To achieve these objectives, the HLA/OMT is devised as a collection of three components: (1) object classes, (2) interaction classes, and (3) routing spaces and their dimensions. (More precisely, the HLA/OMT is divided into seven components: object model identification, object class structure, interaction class structure, attributes, parameters, routing spaces, and FOM/SOM lexicon.) Object and interaction classes represent the data that can be exchanged among federates. Object classes contain *attributes* and interaction classes contain *parameters*. Each object class has a lifetime associated with it in contrast to an interaction class (event) which can only exist instantaneously. The remaining element of the HLA/OMT specification is to facilitate efficient data distribution (routing) among federates.

A FOM deals with the issues surrounding the decomposition of a federation into its federates. In particular, it specifies a federation's object and interaction classes along with their attributes and parameters which collectively establish a necessary but insufficient contract supporting interoperability among federates. The FOM can also optionally contain *supplementary* modeling constructs. Unlike FOM, a SOM deals with intrinsic capabilities of federates such as their dynamic behavior (operations). We can associate the above technical merits one and two with FOM and SOM, respectively.

We can view modeling concerns/issues contained in the HLA/OMT (FOM and SOM) in terms of federation rules and interface specifications [57]. The HLA standard rules are divided into federation and federate rules where each set of rules specify the constraint under which federates and federations and interface specification work with one another. The federation rules are mainly concerned with execution and therefore with HLA RTI. The modeling aspect of these rules are tied to the FOM and SOM where in a federation execution, all data exchanges must be in accordance with FOM data, and each SOM can be owned by at most one federate.

We can examine HLA rules 7 through 9 which relate to the federates. Rule 7 states that federate shall be able to (a) update and/or reflect attributes and (b) send and/or receive interactions. Dynamic transfer and acceptance of attribute ownership is captured via Rule 8. Rule 9 addresses dynamic conditions (e.g., threshold setting) under which attributes are updated. Thus, these rules capture chiefly *interface modeling* *constructs* that are essential in modular model representation and the constraints by which models interoperate. Note that these rules *do not* take into account the internals of federates participating in a federation. The specifics of what might comprise a federate's behavior are not specified by HLA rules and OMT. As mentioned earlier, such knowledge is considered supplementary as far as building HLA-compliant simulations are concerned. Consequently, the interactions among the components of a modular, hierarchical model (federate) neither can be captured by OMT nor do they have to comply with any of HLA rules. The exclusion of each federate's internals is due to the fact that a federate can be anything represented by HLA object models.

6.3.2. FEDEP

The HLA Federation Development and Execution Process (FEDEP) model proposes six steps in an "iterative waterfall software process" to support development of HLA-compliant simulations [58] (see Figure 1). The Six-Step-Process of the Federation Development and Execution Process (FEDEP) Model are.

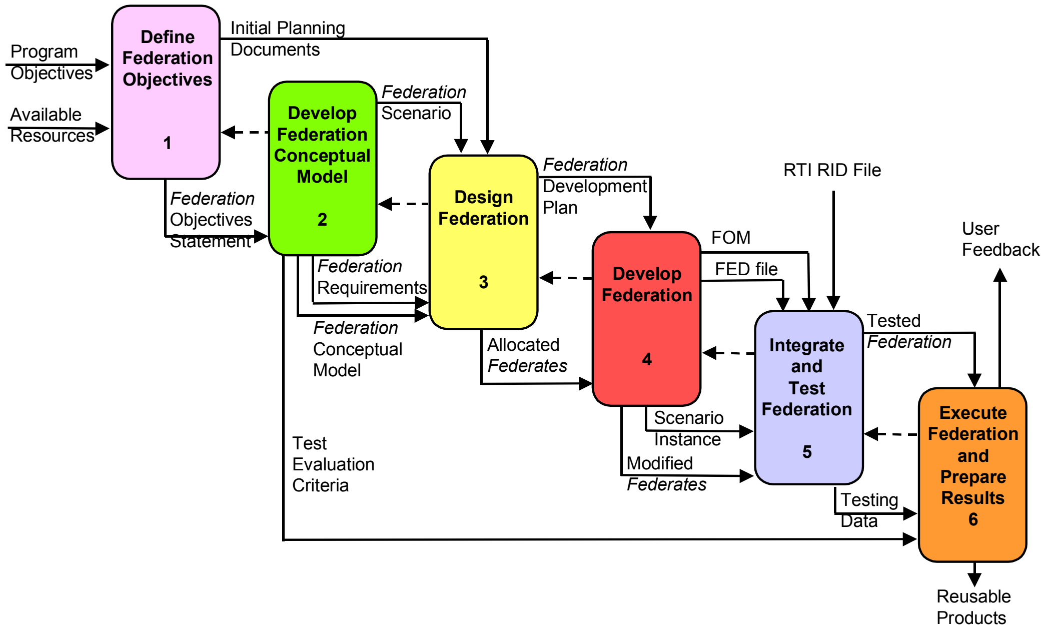


Figure 20: Modes of development for the FEDEP Model [36]

- *Step1*: The federation user and federation development team define and agree on a set of objectives, and document what must be accomplished to achieve those objectives.
- *Step 2*: A representation of the real world domain that applies to the problem space is developed.
- *Step 3*: Federation participants (federates) are determined, and required functionalities are allocated to the federates.
- *Step 4*: The Federation Object Model (FOM) is developed, federate agreements on consistent databases/algorithms are established, and modifications to federates are implemented (as required).

- *Step 5*: All necessary federation implementation activities are performed, and testing is conducted to ensure interoperability requirements are being met.
- *Step 6*: The federation is executed, outputs are generated, and results provided.

Given these steps, steps 2 (develop federation conceptual model), step 3 (design federation), and step 4 (develop federation) are essential in representing the federation and simulation object models. Knowledge such as the *data* to be shared among federates and to a lesser extent the *abstract intrinsic capabilities* of each simulation represent the *model* of an HLA-compliant distributed simulation. The outcomes of these steps, in conjunction with the HLA, is aimed at ensuring simulation interoperability while supporting reuse. The remaining steps in the FEDEP process, of course, are constitutive in the ultimate realization of the "reusable products" such as federate and federations models that can be verified, validated, and accredited.

6.3.3. Interoperability

One of the two central promises of HLA is interoperability – "The High Level Architecture is an architecture for reuse and *interoperation* of simulations" [59]. HLA is claimed to provide a foundation for "technical" interoperability. This is to be distinguished from "substantive" interoperability which is left to federation developers to achieve [49][7]. Technical interoperability is focused at interoperation among different simulators, live systems, and simulation support tools. That is, interoperability among federates and federations is dependent upon Run Time Infrastructure (RTI), the definition of which is not part of the HLA specifications (RTI internal report [60]). While HLA standardizes FOM for single federation execution, inter-federation interoperability (i.e., how two or more federations can be composed to form a "super-federation") is unspecified. Substantive interoperability is intended to support interoperation of disparate modeling approaches in contrast to simulation interoperability[8]. Within the HLA framework, substantive interoperability focuses on a federate's internals. The internal (dynamic) specification of a federate specifies how the federate's behavior is governed during its lifetime. Substantive interoperability of a federate in a given federation depends on its level of abstraction/representation, the types and number of attributes, behavioral computation, and temporal/spatial resolution. Clearly, substantive interoperability must satisfy the objectives of a given federation above and beyond what each of its federates can achieve. This implies interdependency among the internals of federates (models) and not merely their data types and "exchange contracts" as captured in HLA FOM and SOM.

Another related issue with interoperability is "contextual" dependency [49]. Its purpose is to deliberately account for contextual requirements of a federation proposed behavior according to its objectives. Such contextual dependencies can be modeled explicitly as a single federate or spread among a set of federates of a given federation. For example, environmental phenomena provide contextual dimension. They can be represented as ground truth data in a central database (denoted federate) or alternatively have its model dispersed among several federates.

The desire to support interoperability only at the technical level can be traced to HLA's objective to enable interoperability of *simulators* not of models. The distinction is critical because, if viewed only as simulators, federates must be able to interoperate *independently of their underlying modeling paradigms* no matter how incongruent these might be. Recognition of this crucial requirement for "substantive"

[7] Another way to make this distinction is that HLA supports interoperability at the "syntactic" level (federates can converse with a common grammar) but not at the "semantic" level (assuring the meaningfulness of what they say to each other is not supported).
[8] Demand for model interoperability is not unique to HLA; it is a necessity for any modeling methodology conceived or proposed to support model interoperability.

interoperability points to the fact that HLA (or its future extensions) must require one – or in multiplicity, convergent – federate modeling paradigm(s). Adoption of one (or a suite of) modeling methodology(ies) can (collectively) support technical and substantive interoperability[9] which are necessary to support verification and validation.

6.4. A Systems Approach to V&V

The V&V methodology is strongly based on systems theory modeling and simulation methodology [61]. We first give a brief review of systems theory modeling and simulation concepts. At one level of abstraction, M&S involves three types of entities: real system, model, and simulator. Among these entities there are two fundamental relationships: the modeling and the simulation relation. The simulation relation is concerned with ensuring that the simulator carries out the model instructions correctly. The process of assessing the correctness of the simulator, with respect to the model, is called *verification*. The modeling relation, deals with the relationships between real systems and models and is concerned with how well model-generated behavior agrees with observed system behavior.

The goal of the introduced FEDEP_{EF} Model is to provide support for V&V throughout the entire M&S process [34]. We pointed out the advantages of an experimental frame (EF) within this process and give a sketch how to construct an EF via an example.

The presented system theory based V&V methodology is objectives-driven. The EF represents an operational formulation of the objectives. Systems theory helps to build EF and therefore the model as an optimal fit for our objectives. For this optimal fit validation has to be shown relative to the EF. In general it is easy to proof validation for the EF, because it is built from simple, well-known atomic models G, T, A. To show relative validity for certain objectives makes more sense than to ask for absolute validity which can be too strong of a demand for complex simulation systems. The introduced FEDEP_{EF} Model is inherently iterative and evolutionary and systematic. It takes into account that knowledge is induced horizontally and vertically in the specification hierarchy. The structure of EF and Model are enriched throughout the steps. This approach provides the appropriate means for V&V along the process by augmented the steps by introducing formal characterization for validation and verification structures.

The process of ascertaining this degree of agreement is called *validation*. Fundamental for the validation process is the experimental frame (EF). The EF is an operational formulation of the objectives that motivate a M&S project. Validation always has to be carried out with respect to an experimental frame of interest. The primary question is if the real system and the model are equivalent with respect to the experimental frame.

The EF drives the model and observes and analyzes the model output segments. Any gathering/reduction (statistics, performance, measurement, etc.) or behavioral control (initialization, termination, etc.) that is conceptually not carried out in the real system should not be placed within its model but rather formulated as part of the EF. The separation of model and EF allows applying the EF to both the model and the real system.

6.4.1. V&V methodology

The idea of the proposed V&V methodology [62] is based on the Six-Step-Process of the Federation Development and Execution Process (FEDEP) Model by the U.S. Department of Defense (DoD)

[9] While substantive and contextual dependent modeling can be conceptually simple to describe and to some extent adequately understood, its realization in a formal setting (mathematical or otherwise) requires basic research in areas such as multi-resolution modeling [43].

described above. The steps in this process are augmented by introducing formal characterization for validation and verification structures which can guide stepwise accountability of verification and validation at appropriate places along the FEDEP process.

Figure 21 shows each of the six steps of the M&S process populated with experimental frames. The FEDEP process activities are refined to explicitly represent lumped models (M1), base models (M2), and the real system (RS). The thin horizontal arrows stand for the back and forth development of the EF and the models (base models and lumped models). The thick vertical lines emphasize the necessity for V&V for each step. Since the experimental frame plays a significant role in support of V&V, we call the extended process FEDEP$_{EF}$ Model.

In the following subsections, a brief informal description of the V&V in each step is given.

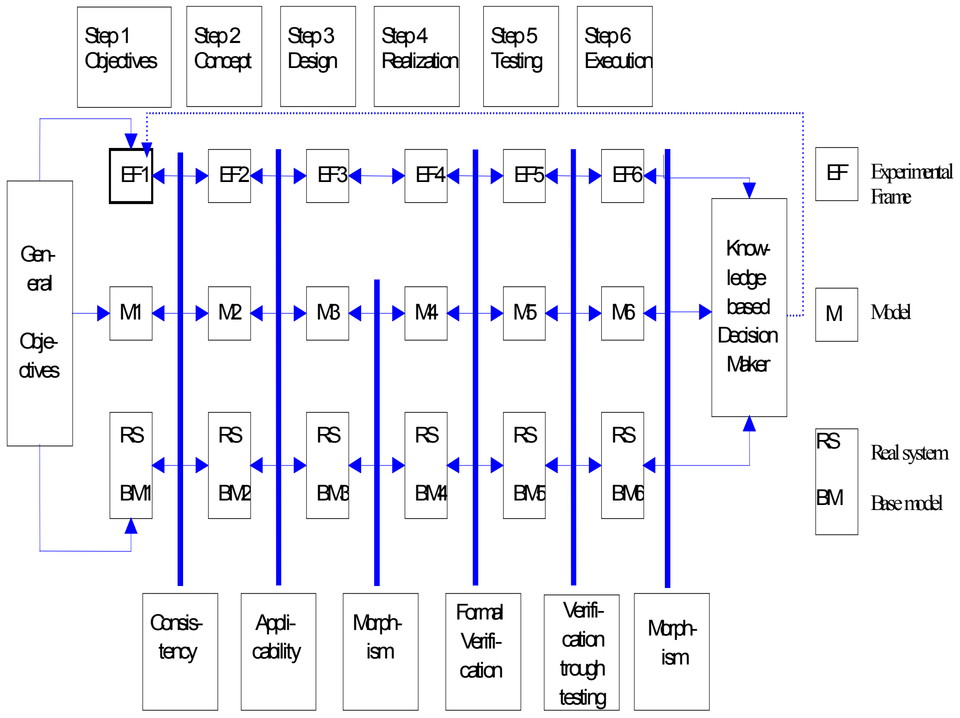


Figure 21: 6-Step-Process and V&V

Step 1: Consistency

After having defined informally the general objectives for EF, models and real system, an informal, yet systematic, consistency check should be made to ensure that the objectives are accounted for.

Step 2: Applicability

A very important relationship has to hold between EFs and (lumped respective base) models (M1 and M2): the *applicability relation*, which guarantees that the EFs can be applied to the models ([13] gives a formal definition of the applicability relation.) The conditions on experimentation required by the frame can be enforced for the model.

Step 3: Morphism

At this level, the concept of *morphisms* comes into play. In general, the concept of morphism tries to capture similarity between pairs of systems specifications. In [13] a system specification hierarchy together with a hierarchy of morphism relation is formulated which allows to compares pairs of systems in a formal way. In this step, employment of system morphism concepts can have a great impact in showing replicative, predictive, and structural validity [14] of models with respect to experimental frames.

Step 4: Formal Verification

Formal proofs of correctness employ mathematical and logical formalisms to establish morphisms at different levels of the system specification hierarchy between model and simulator. These proofs are difficult or impossible to carry out for large, complex systems, although more automated tools are becoming available. This leads to Step 5: Extensive testing.

Step 5: Extensive Testing

Extensive testing should ensure that all conditions that could arise in simulator operation have been covered by test cases. Time and other resources limit the amount of testing. Here the morphism concept can point to a thorough and efficient testing since it offers a framework for laying out the combinations of inputs and states that have to be tested.

Step 6: Morphism

In step 6 we have to deal with the replicative validity relationship. This relation is affirmed if, for all the experiments possible within the EF, the behavior of the model and system agree within acceptable tolerance. The results can be the input for a knowledge based decision-maker. This decision can halt the process if the results are satisfying or return to Step 1-6.

6.4.2. FEDEP$_{EF}$ Specification

In this section, we introduce six levels of EF specifications for the FEDEP process (see Table 1) [62]. In Figure 1 specific names are given to the six EF boxes to emphasize different specifications of the experimental frame for specific uses. In Subsections 3.1–3.4, we provide a sketch of the experimental frame specifications toward verification and validation for the FEDEP$_{EF}$ process. Each subsection defines a definition of the experimental frame concept based on the suggested 6-Step-Process shown in Figure 1. These definitions are developed based on the system specification hierarchy [13] in order to succinctly arrive at successively higher levels of details and elaboration for verification and validation. For example, EF$_1$ specifies the most elementary elements (e.g., the experimentation period) required to arrive at verification and validation in Step 6 depicted in Figure 1.

Table 3 depicts an overview of the specifications for EF and model or real system[10]. Our discussion on the EF specifications applies to a simulation model as well as a real system provided there exists a well-defined relationship (e.g., homomorphism) between them.

Step	EF Specifications	Model Specifications
1	EF$_1$: Requirements level	M$_1$: Observation system
2	EF$_2$: Conceptual level	M$_2$: IORO and IOFO systems

10 By extending Table 1 entries for the base model, lumped model, and the real-system, we can carry out verification and validation at the end step of the FEDEP process based on its associated experimental frame.

3	EF_3: Design/realization level	M_3: IO system
4	EF_4: Design/realization level	M_4: Coupled system
5	EF_5: Parameterized experimentation level	M_5: Parameterized coupled system
6	EF_6: Parameterized experimentation level	M_6: Parameterized coupled system'

Table 3: Experimental Frame and Model Specifications for the FEDEP Process Model

Later, we consider the connection of EF to the Model and takes into account the constraints that must be used to induce knowledge both horizontally and vertically in the specification hierarchy.

Step 1: Requirements Level

In this step, the EF structure contains four elements, three of which have direct counterparts in the Observation Frame specification (see Table 1). An EF_1 is a structure is defined as

$\langle T, I, C, O\rangle$ where

T is a time base
I is a set of variables, the input variables
C is a set of variables, the run control variables
O is a set of variables, the output variables

Step 2: Conceptual Level

The experimental frame in Step 2 contains additional elements to define admissible input and control segments as well as summary mappings. The definition of the EF_2 structure is as follows [14]:

$\langle \mathbf{T, I, C, O, \Omega_I, \Omega_C, SU}\rangle$ where

$\Omega_I \subset (X, T)$, $X = i_1 \times \ldots \times i_n$, $i_i \in I$, admissible input segments
$\Omega_C \subset (L, T)$, $L = c_1 \times \ldots \times c_m$, $c_j \in C$, operation/execution control segments
$D_E = \{(\omega,\rho)\}| \ \omega \in \Omega_I$, $\rho \in (Y,T)$, $dom(\omega) = dom(\rho)$, subject to $\Omega_C\}$
SU: a set of summary mappings defining consolidated I/O observations

Details such as specification, generation, and acceptance of Ω_I, Ω_C, SU and their relationships with the remainder of the above structure are detailed in [14]. Ω_I, Ω_C, and SU form the basis for defining generator (**G**), transducer (**T**) and acceptor (**A**) for the experimental frame (EF_2).

Steps 3-4: Design Level

We define the experimental frame EF_3 structure as $\langle \mathbf{T, I, C, O, \Omega_{I,N}, \Omega_{C,N}, SU_N}\rangle$. Similar to EF_2, $\Omega_{I,N}$ and $\Omega_{C,N}$ are defined to be specifications of admissible input/control segments for I/O System and Coupled System specification. $\Omega_{I,N}$ is defined to be cross product of the input segments to the simulation model or real system. Similarly, $\Omega_{C,N}$ is a cross product of control segments. SU_N is the resultant behavior space (input/output pairs) given the components of the EF. Generally, for simple I/O systems, EF2 as defined above is necessary and sufficient. That is, at the I/O system level, EF_3 typically has three types of components. Generator(s) produce input segments to the model or system. Acceptor(s) control and monitor experiments under specified conditions. Transducer(s) collect, observe, and analyze the model or system output segments.

Consider $\Omega_{I,\ N}$. This can be defined in terms of all appropriate inputs specified by the EF generator components. The components of EF (one or more generators, transducers, and acceptors) may be identified/determined based on the elements of the EF from Step 2. For example, one or more generators may be devised to generate input segments for an atomic or couple model (i.e., input values specified by EF).

However, for elaborate simulation models or real systems, experimental frame as defined here are believed to be more suitable. Consider Coupled Systems. In view of such systems, the experimental frame (EF_4) can then be considered to be composed of generators, transducers and acceptors as opposed to an experimental frame consisting of a generator, transducer, and an acceptor. Of course, a set of generators (or transducers or acceptors) may be mapped into a single generator (transducer or acceptors). However, using the divide and conquer principle, we defined $\Omega_{I,\ N}$ ($\Omega_{C,\ N}$ and SU_N) to account for multiple generators, transducers, and acceptors and thus allow multiple generators to be associated with different models of a coupled system.

Specification of experimental frame in Steps 4 entails identification of the couplings among **G**, **T**, and **A** as well as (hierarchical) couplings among the individual components. Additionally, realization specifics as defined by HLA services such as Data and Time management may be incorporated into the EF and used for formal verification.

Steps 5 – 6: Parameterized System Levels

An experimental frame for parameterized coupled system is a structure $\langle \mathbf{T}^{\mathbf{p}}, \mathbf{I}^{\mathbf{p}}, \mathbf{C}^{\mathbf{p}}, \mathbf{O}^{\mathbf{p}}, \mathbf{\Omega}^{\mathbf{p}}_{\mathbf{I,\ N}}, \mathbf{\Omega}^{\mathbf{p}}_{\mathbf{C,\ N}}, \mathbf{SU}^{\mathbf{p}}_{\mathbf{N}}\rangle$**.** A parameterized EF is defined to support experimentation based on a set of simulation models or real systems depending on the lower and upper bounds placed on input values, output values, and control parameters. These two specifications may be employed to account for testing and execution by suitably specifying bounds on the elements of **G**, **T**, and **A**. Based on desired distributed execution (e.g., logical-time) mode, we may further specify this experimental frame structure. With parameterized experimental frames, we will have a suitable basis for Testing and Execution steps of the $FEDEP_{EF}$ model.

6.4.3. Application of $FEDEP_{EF}$

This section provides a sketch as how to apply the $FEDEP_{EF}$ Model in a real-world setting. As case study model we chose Regozon, a mesoscale ozone forecast model [63]. Regozon was implemented at the GMD Research Institute for Computer Architecture and Software Technology (GMD FIRST). In cooperation with GMD FIRST our department at the Johannes Kepler University simulated a realistic summer smog episode in the Greater Linz area [64]. The model area has 100x100 km extension. This project was realized by order of the environmental departments of the state government Upper Austria and of the town council Linz. Figure 2 shows a graphical representation of the simulated ozone values in the Greater Linz area. In this example you see the ozone plume in the rural area in the lee of the city. The simulation results made sense for the experts of the environmental departments (=qualitative validation). To compare the simulation results with values of the ozone measuring stations quantitatively, we used different statistical methods like calculating difference plots, trends, averages, and maximum values.

In the following, we describe how to verify and validate the Regozon Model by the $FEDEP_{EF}$ Model. An EF, which plays a central part in the $FEDEP_{EF}$ Model, can be seen as the operational formulation of the objectives that motivate a modeling and simulation project. Thus, first of all, there must be an objective. Which questions do we have that our model should answer? A typical question from responsible persons of environmental departments, who have to deal with ozone alarm plans, is:

“Do m (out of n) measuring stations within a certain time segment (e.g. 3 hours) exceed an ozone threshold value?”

For example the ozone alarm plan for Upper Austria looks like in Table 2.

Level Description	**Ozone Threshold Value**
European Union Informative Level	180 µg/m^3 within 1 h.
Precaution Level	200 µg/m^3 within 3 h.
Warning Level I	300 µg/m^3 within 3 h.
Warning Level II	400 µg/m^3 within 3 h.

Table 4: Ozone Alarm Plan for Upper Austria.

In step 1 we determine that the Time Base T is the I (integers). Then we have to consider the variables we need for a proper representation of the real world situation. As input variables we choose e.g wind (direction, velocity), diurnal variation of emissions (traffic, industry) and cloud cover. An output variable for our proposed example can be a variable which indicates the number of the Alarm Plan Level. The criteria for a good ozone prediction model for our purpose could be that it predicts with a probability of 90% the Alarm Plan Level of the following day. In step 1 we also determine the duration time, e.g. 20 hours, and the starting time, e.g. 3 a.m. It is crucial to know from the beginning whether we want to simulate within one day or for a week.

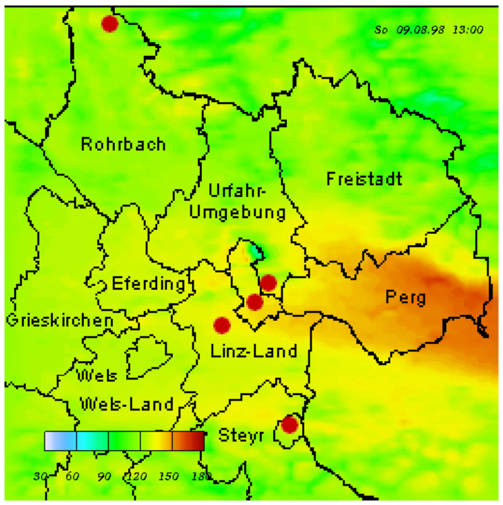


Figure 2: **Simulated Ozone-Concentrations in µg/m^3 in the Greater Linz area, August 9, 1998, 1 pm**

In step 2, we determine the I/O data space, the set of all pairs of I/O segments. From 3.2 we see that the EF selects out a subset of I/O pairs for consideration from all those conceivable. This subset constitutes a focus of attention determined by the objectives which motivate the modeling effort.

In steps 3 and 4, we design generators, acceptors and transducers and their couplings. Straightforward from step 1 we need generators for wind, diurnal variation of emissions and cloud cover. Transducers observe and analyze the model and system output segments. So for our objective the transducer will observe n stations and analyzes, if m stations of the model or system exceeded within a time segment a certain threshold and results in a number of the Alarm Plan Level. Different acceptors are conceivable. For this example we decided for an initial subset acceptor. This common EF component accepts an input segment if its initial value lies in the specified subset. We specially are looking for the initial ozone-value. This value is computed as an average value from the measuring stations from the day before our simulation starts. A second acceptor only accepts input segments, where wind is faster than 1 m/sec and/or cloud cover is less than 70%. Without these preconditions a model will not reach one of the alarm plan levels, because it needs a sunny day to produce ozone and without a significant wind speed the produced ozone is destroyed again. The simulation terminates if more than 3 subsequent input segments were not accepted, otherwise it will end after the duration time was exceeded. Steps 5 and 6 demand to specify bounds on the elements of G, T, and A.

6.5. A roadmap for V&V Using HLA and DEVS

In order to provide a roadmap based on combining HLA/OMT and DEVS/SES, we adopt a set of six management services (i.e., federation, object, data, declaration, ownership, and time) that are suggested by HLA [35].

Category	Functionality
Federation Management	▪ Create and delete federation executions ▪ Join and reusing federation executions ▪ Control checkpoint, synchronization, restart
Declaration Management	▪ Establish intent to publish and subscribe to object attributes and interactions
Object Management	▪ Create and delete object instances ▪ Control attribute and interaction publication ▪ Create and delete object reflections
Ownership Management	▪ Transfer ownership of object attributes
Time Management	▪ Coordinate the advance of logical time and its relationship to real time
Data Distribution Management	▪ Supports efficient routing of data

Table 5: HLA Management Services

We are chiefly interested in the types of services offered and their management. Specifically, we view federation and object services as the basis for modeling and simulation be it distributed or not. The data distribution and declaration services are key to enabling distributed simulation. In the modern era of multidisciplinary joint simulation studies (e.g., One Semi-Automated Forces (OneSAF) [65].), ownership services are required to control proprietary issues in a timely and systematic fashion. The remaining set of services (i.e., time management) provide the foundation to enable various types of execution (conservative/optimistic) of logical models and real-time entities.

Further, in order to support representing family of models and synthesis of models, we include "design-spaces" as supported by SES. The design spaces provide the remaining fundamental capability that must be offered by a simulation framework supporting distributed modeling and simulation. Before we proceed further, we summarize important features of HLA/OMT and DEVS/SES.

	CAPABILITIES
DEVS	Captures attributes, values and dynamics Well defined atomic and coupled model semantics Closed under coupling Well defined subclass of systems with well known relation to other subclasses
System Entity Structure	Captures family of models Supports automation of search exploration of model space

Table 6: Strengths of DEVS and SES

	CAPABILITIES
Federation object model	Supports exchange and coordination of data
Simulation object model	Supports representation of attributes and values
Interface Specification	Supports publish/subscribe protocol Supports Routing Space Supports simulated and live federates

Table 7: Strengths of HLA/OMT

6.5.1. HLA/OMT and DEVS/SES

In this section, we discuss HLA/OMT, DEVS/SES with respect to the services just described [50]. Each row of the table reveals the capability offered or promised by HLA/OMT, DEVS, and SES. Some of the capabilities listed for HLA/OMT and DEVS have been assigned "partially supported" or "can support" descriptors with respect to the full set of capabilities listed under "Distributed Simulation and Modeling Services". The use of these terms can be traced to descriptions of HLA/OMT and DEVS/SES[11]. It is noteworthy to state that our use of the terminology is rather imperfect. Our goal is to informally show the strengths and weaknesses of a given methodology or framework with respect to the above services. Furthermore, the comparison is intended to illustrate the complimentarity of HLA/OMT and DEVS/SES.

We begin with the federation management service. This service is divided along the lines of a model as a *standalone* entity or as a *collection of cooperating models*. Clearly, the simulation object model (SOM) and atomic model represent dynamic behavior. Similarly, the federation object model (FOM) and coupled model represent distributed model – i.e., decomposition of a federation (coupled model) in terms of its federates (components.) Therefore, the first and second rows (federation management service) show that both HLA/OMT and DEVS can be used to model a single model as well as a set of interacting models. Note that DEVS coupled model can be either a federation or federate since based on the HLA standard, a federation, in essence, is a single-layer hierarchy.

[11] A thorough analysis and evaluation of such capabilities and more refined terms instead of "partially supported" and "can support" require an in-depth exposition of DEVS and HLA, something that is beyond the scope of this writing.

The modeling features offered by HLA/OMT and DEVS can be categorized as *Interface Modeling* and *Dynamical Modeling*, respectively. Interface Modeling is defined as modeling of composite systems where its dynamics are either unspecified or at most specified as input and output values. Dynamic modeling subsumes interface modeling by not only supporting interface specification but also specifying behavioral dynamics (e.g., formula to compute a projectile's velocity). For example, DEVS/SES modeling primitives and constructs support interface modeling at the level of first-class objects that have full representation and functionality. The HLA standard, on the hand, is intended to support *lowest common denominator* (e.g., support non-object-oriented models) modeling primitives. Through this lowest common denominator one declares knowledge about a federation (e.g., FOM vocabulary) while allowing details of models to be encapsulated inside respective SOMs that are essentially hidden to their FOM.

The row associated with the declaration management shows that HLA supports data exchange through user-declared, flexible protocols. OMT supports *updates* via "publish" and "subscribe" services and *interactions* via "initiate", "sense", and "react" services. DEVS uses the concepts of *ports* and *couplings* to support declaration of information flow via message-passing.

The row associated with the object management service shows that DEVS and HLA/OMT support the concept of object management, albeit from different and complementary perspectives. HLA/OMT supports data exchange as either updates and/or interactions via an intermediary known as FED file [35]. DEVS, on the other hand, supports data exchange via ports/coupling among components which is succinctly represented in a coupled model.

The ownership management row shows HLA offers some degree of "ownership" which is limited to which federate or set of federates may be assigned the responsibility of generating (simulating) the desired data. The concept of ownership in HLA/OMT is confined to determining which federate or a set of federates can update attributes. That is, ownership is used to ensure that at most one federate at any given time can update the value of an attribute. The ownership can be set at initialization and altered dynamically (updating of an attribute can be transferred from one federate to another). Currently, DEVS does not support ownership. However, ongoing research effort is investigating a formal ownership representation for atomic and hierarchical coupled models to support the usual connotations of ownership including control of execution, sharing and property rights [51, 66].

The data distribution management row indicates that the HLA/OMT Routing Space [36] and DEVS-based Predictive Contracts [67, 68] are complementary to one another. The routing space enables "as needed" data exchange based on federates' proximity to one another (in routing space), attribute updates, and parameter interactions. The predictive contract concept, applicable to discrete and continuous systems, provides additional filtering based on the content of data as well as spatial encounter prediction techniques [51] to minimize unnecessary data exchange.

The time management row shows that simulation approaches supported by both HLA/RTI and DEVS rely on a centralized time management scheme. HLA/RTI supports distributed simulation by assigning time stamps to messages and ensuring adherence to a very strict causality principle of physical systems. HLA/RTI also supports, in principle, receive order messaging as well as its use with time-stamped order messaging. The Parallel DEVS protocol is based on message passing alone without requiring messages to be time stamped and ordered to ensure satisfying causality. Furthermore, unlike HLA/RTI, DEVS offers the unique capability of generating outputs in zero simulation time. From DEVS theory, any legitimate DEVS model [14] is ensured to be correctly simulated using the Parallel DEVS protocol. Both HLA/RTI and DEVS are capable of supporting concurrently logical and real-time federates. Neither HLA/RTI nor DEVS has been shown to fully support interoperability among heterogeneous simulation strategies (conservative, optimistic, and their variations). Given the unique advantage of the Parallel DEVS simulation protocol and DEVS wide range of modeling capabilities, we believe the Parallel DEVS simulation protocol might provide a more suitable basis for enabling interoperability among disparate simulation strategies than the current HLA specification.

The last row in Table 5 suggests that neither HLA nor DEVS can support design-spaces. OMT supports model-bases within the confined setting of object and interaction object classes for a single federation. Using DEVS and SES, a model-base can offer not only a federation and its federates, but also the concept of specialization and aspect. In comparing HLA/OMT and DEVS with SES, the use of N/A indicates that either the particular service cannot be supported by SES or is inappropriate given what DEVS and/or HLA offer. For example, it is pointless for an SES to have a single entity (i.e., atomic model).

Distributed Modeling Services		**HLA/OMT**	**DEVS**	**SES**
Federation management: membership & operations	Singular, standalone model	SOM (modeling is partially supported)	Atomic model	N/A
	Coupled, distributed models	FOM (modeling is partially supported)	Coupled model (execution management is partially supported)	Model decomposition is supported
Declaration management		Publish/Subscribe Updates and interaction	Supports interaction	N/A
Object management		Object class instantiation, send/receive interactions are supported	Object class instantiation, messaging through port and coupling is supported	N/A
Ownership management		One or more federates may be declared to be responsible for computing a set of SOM attributes (partially supported)	Can support ownership in its usual connotations	N/A
Data distribution management		Routing Space (partially supported)	Predictive Contracts (partially supported)	N/A
Time management		Centralized time management; time-stamped order and receive order message passing	Centralized time management; messaging follows output/input port couplings	N/A
Design Spaces		not supported	not supported	Alternative decompositions, specialization + Pruning and Synthesis

Table 8: HLA/OMT & DEVS/SES Modeling and Simulation in Spotlight

The claim that HLA/OMT supports neither specialization nor aspect features requires justification. Our reasoning is based on the belief that while HLA standard asserts that SOMs can be written in such a way that they can be reused in alternative federations, there exists no *systematic* support for representing design spaces (family of models). In the HLA six-step FEDEP FOMs are purported to be reusable during the design phase either through their modifications or their off-the-shelf reuse. Unfortunately, a framework possessing a well-defined scheme (syntax and semantics) and capable of supporting FOM reusability is not provided by HLA standard. The consequence is that such knowledge cannot readily lend itself to automation where it is most desirable.

From the standpoint of DEVS and HLA capabilities (see Table 5), we can inquire whether they truly provide a sound basis to support VV&A. Clearly, the existing DEVS/SES and HLA/OMT duo suggests a sufficient and complete set of services for modeling and simulation of systems from a higher-level perspective. Realization of lower-level processes and assuring their interoperability (at the same layer or between two layers) is an open problem. Specifically, "substantive" interoperability [49] becomes particularly challenging as other kinds of systems (e.g., AI-based) are to be accounted for or interoperated with. This suggests, HLA/OMT and DEVS/SES supported the types of systems that are known to be (can be) modeled with DEVS and HLA.

6.6. VV&A from Software Engineering Perspective

There exists an extensive body of research (e.g., [69, 70]) in the field of formal software specification which deals with general computer program correctness and specific topics such as software component reuse, storage, and retrieval. Some of these techniques employ partial-orders to establish equivalence relationships among "functionalities" of components. Such relationship, for example, may be asserted for replacing one software component with another and proved correct using varieties of the predicate calculus. However, the use of such formal techniques has been limited due to the difficulty in justifying the substantial resources required, lack of trained people and availability of support tools.

A critical pitfall of such logic-based techniques for M&S use, in our view, is their failure to make distinctions between a "model" and a "simulation". As we have described above, separation of models from their simulation embodiments plays an essential role in determining the V&V concepts that are appropriate for the situation at hand. Furthermore, the model/simulator distinction impacts the choice of specific V&V techniques and how V&V might be applied throughout a modeling and simulation lifecycle process. We have discussed above how the roles and methods of verification and validation may be substantially different at the modeling and the simulation levels (see sections on Model Composability and Modeling and Simulation Interoperability). Thus, as we assert below (Section 7.1), it is possible to verify that a simulator (such as the HLA RTI) has certain correctness properties in its time management services while failing to show that it can support the complete class of (discrete event) models for which it is intended [71].

Clearly, concepts, methods, and techniques from formal software specification can play a major role in V&V as required by M&S community. However, a framework for M&S is needed that places the applicability of such methods into perspective so that comprehensive approach to V&V can emerge.

7. General Recommendations and Suggestions

We have reviewed the support of the DEVS formalism for hierarchical modular model construction and component reuse. We have shown that the basic concept of modularity for dynamic systems is more demanding than that of computational objects and is supported in the basic DEVS formalism, especially through the external transition function. We further showed that closure under coupling supports the ability to "componentize" a coupled model. This in turn, justifies and enables the application of, hierarchical model composition constructs. The underlying structure of hierarchical models is captured in the system entity structure (SES) which supports application to construction, management, and reuse of combinatorially-based families of models – as most large scale model development projects are likely to generate. We discuss the implementation the SES in DEVS/HLA with illustrations from examples developed in the DEVS/HLA (JAVA) environment.

This paper exposed the strengths and limitations of DEVS/SES and HLA/OMT. In the course of our analysis, we noted the while OMT focuses primarily on modeling a federate in terms of its attributes and parameters and a single federation, DEVS provides a comprehensive set of dynamic modeling constructs. Based on the HLA FOM/SOM and DEVS atomic/coupled model relationships, we concluded that the HLA object and interaction classes can be derived from DEVS atomic and coupled components. The key advantage of using DEVS is that the derived object and interaction class hierarchies have a sound basis due to (1) the theoretical underpinning of DEVS and (2) its realization in a fully object-oriented language. In contrast, in HLA/OMT, object and interaction classes are to a large extent arbitrary in terms of their structure and consequently can provide limited support for model (substantive) interoperability.

As an example of capabilities that are not currently supported, under the proposed unified M&S framework, automated mapping and generation of FOMs and SOMs from models developed using the Collaborative DEVS Modeler environment can be supported. Among other products, this would provide OMT documentation required by any HLA-compliant simulation. Similarly, support for model design space, reuse, and management using SES concepts can enable not only systematic DEVS model development, but also, as a consequential benefit, FOM and SOM specifications as well.

We argued a significant impediment to successful use of HLA/OMT is its inability to support HLA FOM and SOM model reuse. While facilitating comprehensive model reuse across multiple representation scheme remains to be an open problem, DEVS/SES provides the basis to support model reuse for a large class of systems which includes well-known continuous, and all discrete-time and discrete-event systems. Finally, we suggested that HLA/OMT and DEVS/SES collective services provide a set of collective powerful capabilities (e.g., object management and design-spaces) that transcends significantly what each of DEVS/SES and HLA/OMT can provide in isolation.

7.1. Verification and Validation within DEVS/HLA:

To deal with the correctness of these implementations we assume that the underlying RTIs are correctly implemented with respect to the HLA specification. While verification based on formal proof is feasible, we have not yet attempted that, but instead have employed an extensive suite of test cases. The suite consists of DEVS models and is to designed to cover various independent aspects of the DEVS simulation protocol. Since the correct behaviors of the models are known, an implementation of the protocol can be shown to be incorrect if comparison of any of the generated behaviors to their known counterpart standards reveals a discrepancy. Of course passing, without error, a finite suite of tests cannot conclusively establish the correctness of an implementation. However, the comprehensive coverage of all aspects of DEVS functionality in a test suite can afford significant levels of confidence in correctness.

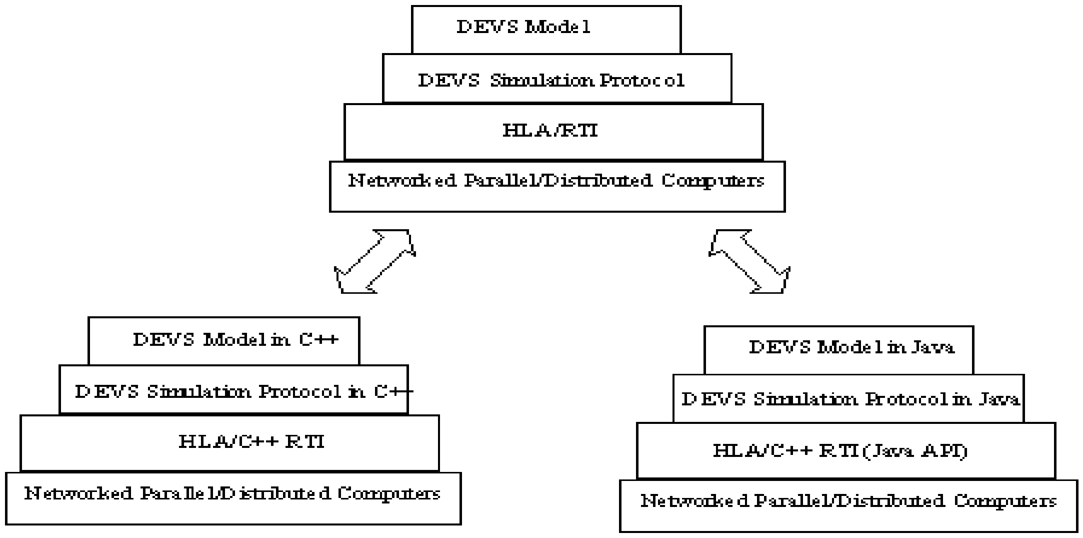

DEVS Model
DEVS Simulation Protocol
HLA/RTI
Networked Parallel/Distributed Computers
DEVS Model in C++
DEVS Simulation Protocol in C++
HLA/C++ RTI
Networked Parallel/Distributed Computers
DEVS Model in Java
DEVS Simulation Protocol in Java
HLA/C++ RTI (Java API)
Networked Parallel/Distributed Computers

Figure 22 Layers Separating Model and Simulator: DEVS/HLA RTI example

The foregoing testing of the first DEVS/HLA C++ implementation in C++ RTI v1.0 revealed an important problem which could have adversely impacted the expressiveness of the DEVS implementation. It was not clear at the outset whether the issue was a flaw in the HLA specification, its implementation in the current RTI, or in the implementation of DEVS itself. However, the separation of layers just described enabled us to take for granted the correct implementation of the DEVS layer – which had been previously independently verified [72]. This enabled us to pin point the problem to the HLA layer and eventually to the HLA specification itself. A resolution was found that exploited formal properties of the DEVS formalism and efficient use of the HLA time management services. This resolution is discussed in [71]. Because it has been formally characterized in terms of the DEVS formalism, its simulation protocol and the HLA specification, the solution is portable to HLA-compliant implementations in other languages such as Java, as suggested in Figure 24.

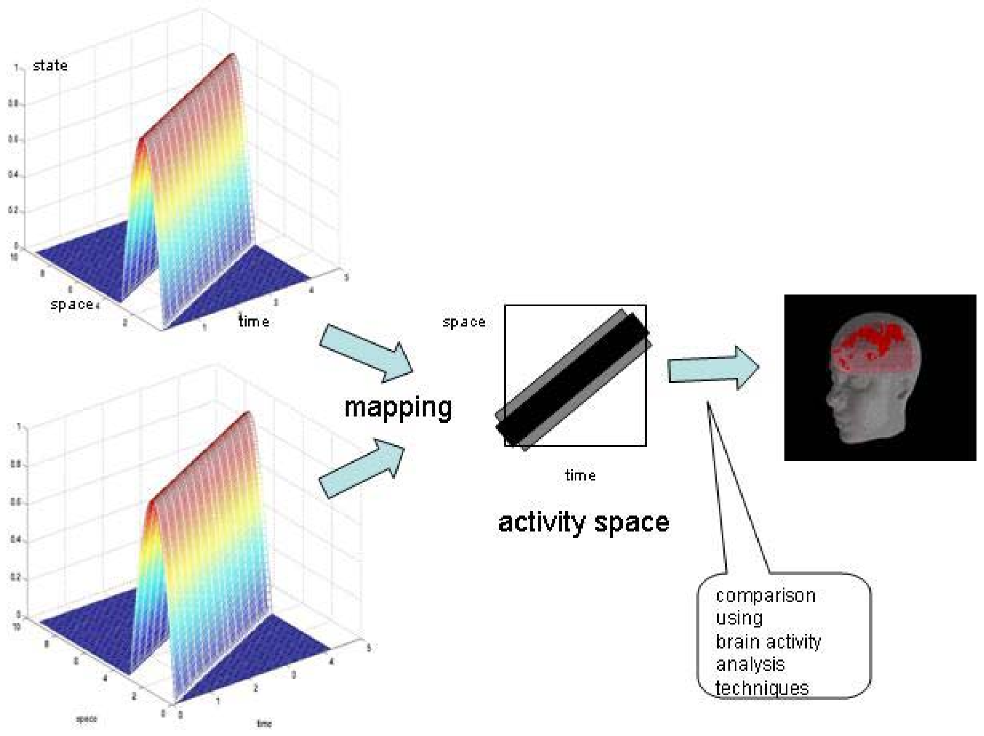
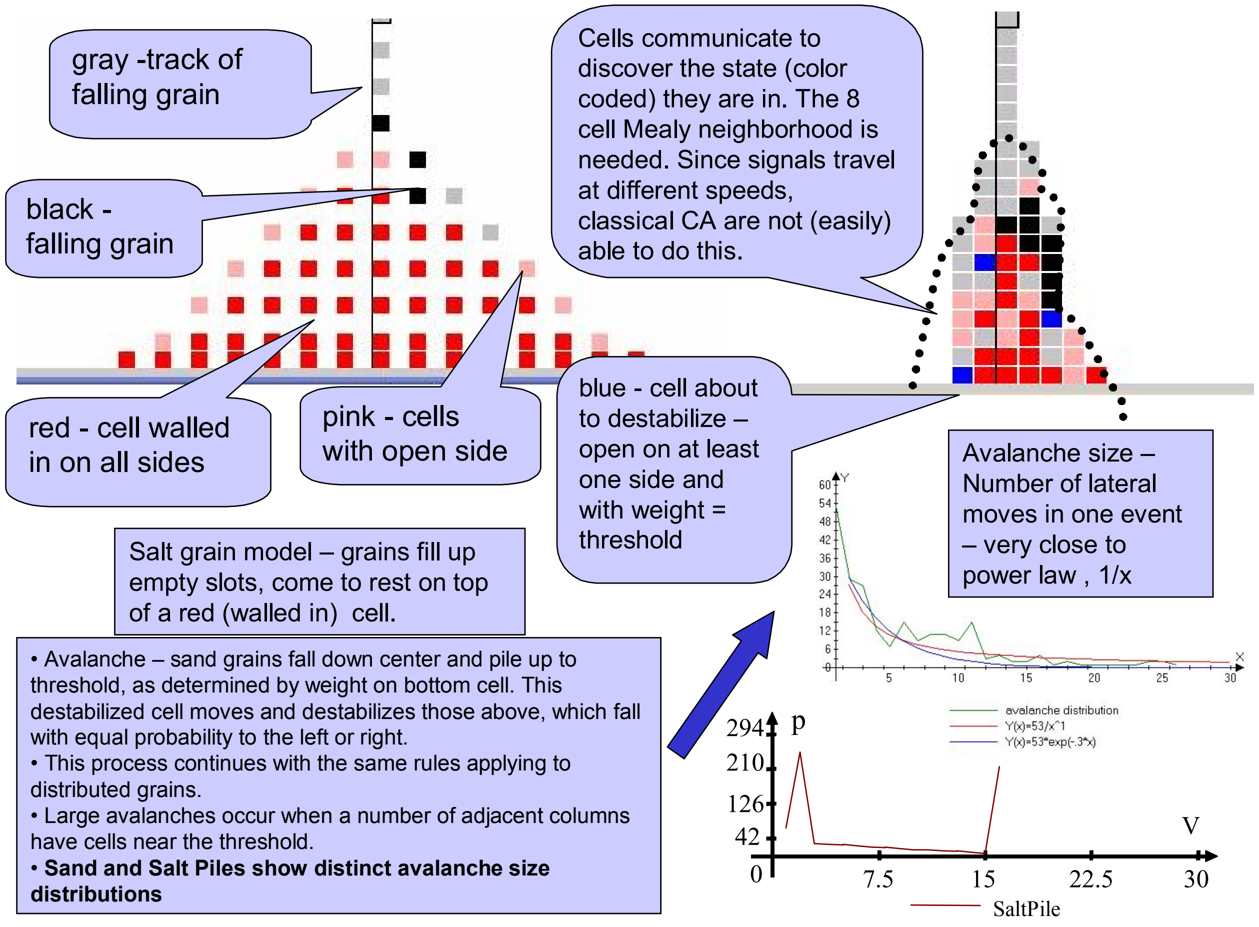


Figure 23 Mapping of large scale system trajectories into activity space

7.2. New General Tools based on the M&S Framework in Support of V&V

For future research, we recommend the search for general spaces that can be interpreted for a variety of specific models and that support the generic approach to V&V introduced earlier in this paper. One possibility for a general mapping space is contained within the concept of a model's activity distribution and evolution while it is being simulated. As background for considering this approach, we consider that within the general systems perspective one looks for isomorphism relations between particular systems and abstractions that characterize the resulting equivalence classes. For example, there are many specific systems having cyclic behavior and the abstract characterization within systems science applies to them all. From this perspective the activity distribution of the sun, and the activity distribution in a brain, are candidates for abstraction into a common context-free concept of “activity.” However, our interest is not in looking for an isomorphism between the sun and the brain – the meanings of their activities are so different that we wouldn’t expect a literal correspondence. Instead, we are interested in employing the technology for observing activity in specific contexts such the brain as a metaphor, or guide, for observing activity in simulation models in general. Metaphorically speaking, one can imagine mapping a simulation model, such as that for the sun’s activity, onto a brain, and employing the methods already developed to track this activity. This metaphor can be a useful one if the mapping and the comparison within the activity space are computationally feasible.


gray -track of falling grain
black - falling grain
red - cell walled in on all sides
pink - cells with open side
blue - cell about to destabilize – open on at least one side and with weight = threshold
Y
X
avalanche distribution
Y(x)=53/x^1
Y(x)=53*exp(-.3*x)
p
V
SaltPile

Cells communicate to discover the state (color coded) they are in. The 8 cell Mealy neighborhood is needed. Since signals travel at different speeds, classical CA are not (easily) able to do this.

Avalanche size – Number of lateral moves in one event – very close to power law , 1/x

Salt grain model – grains fill up empty slots, come to rest on top of a red (walled in) cell.

• Avalanche – sand grains fall down center and pile up to threshold, as determined by weight on bottom cell. This destabilized cell moves and destabilizes those above, which fall with equal probability to the left or right.
• This process continues with the same rules applying to distributed grains.
• Large avalanches occur when a number of adjacent columns have cells near the threshold.
• **Sand and Salt Piles show distinct avalanche size distributions**

Figure 24 Two avalanche models with distinct size distribution characteristics

We might think that a necessary condition for success of this method is that properties of the mapped activity as observed in the “brain” must relate back in useful ways to the properties in the original space. But this is only the case if the model’s mapped behavior is observed in isolation. For example, if, while we are simulating a model we observe that its activity in the activity space diverges from pattern that is known to us a priori, we can infer that the model does not satisfy the corresponding required behavior. This condition would require domain-specific and model-specific knowledge of the expected behavior. So the activity space concept would not be applicable without this apriori information.

However, the situation is different if we map a pair of systems to the same space so that their images can be easily compared. If the mapped activities do not agree, we can infer that the systems are not behaviorally equivalent independently of whether we know what to look for or not. For example, if the pair of systems in question consists of a real system and a model, than this disagreement would invalidate the model. On the other hand, if the mapped activities do agree, then this provides some evidence that they are behaviorally equivalent although this confirmation may be overturned with subsequent comparisons. The point is that this approach is a "weak" method that does not require a priori information about the domain and can therefore be applied to any model provided that the right tools for doing the mapping and comparison are available.

Let's generalize from the brain mapping example. For a V&V approach to be "general" requires that

- we have a universally applicable target space
- a computationally feasible approach to map any model
- comparison of two trajectories in the target space is also computationally feasible

Note that we do not require that the mapping to the target space be a homomorphism – that would take us away from the "general" nature of the method.

Computational feasibility is stressed here since we are interested in scalability to large scale multi-component systems. The computational implementation must be cheap enough to be run along side the main simulation in a high performance environment. If so, it would be possible to keep a set of simulation experiments going in searching for a valid model where a simulation is stopped as soon as a deviation is detected and restarted with another variant. The "weak" nature of the approach means that it would support weeding out of low quality candidates but not necessarily convergence to good candidates.

The example in Figure 24 may help clarify the discussion. The figure presents two contrasting models of avalanche behavior in piles of grains such as salt and sand. These kinds of models have been of interest in the study of self-organized criticality (SOC) or the behavior emerging from slowly-driven, interaction-dominated, threshold systems [73, 74]. Interaction and thresholds are crucial in such systems. This is illustrated by the contrasting behavior between the "salt model" and "sand model" on the left and right hand sides, respectively, of the figure. In the salt model falling grains pass over each other with no interference, while in the sand model, stacked granules in a column cohere together until a critical size threshold is exceeded. This threshold inhibits continuous adjustment of the pile to its relaxed state. Consequently, global states arise in which interactions of almost-critical clumps can propagate destabilization behaviors, similar to rock avalanches in mountainous terrain. SOC is therefore behavior in which there are short periods of high activity punctuating long periods of inactivity. SOC offers new, though still controversial, explanations of species extinction, earthquakes, traffic pile-ups, distributed interactive simulations [75] and other bursty phenomena.

Figure 25 illustrates an experimental frame suited to comparing candidate models for SOC. The models are formulated within cellular spaces and generate streams of active cells identified by their three-dimensional cell space coordinate vectors. Cells are considered active when they undergo certain transitions such as falling from one perch to the next lower one. This activity stream is parsed and analyzed by a series of transducers that identify avalanche events much in the same way that radar tracks are analyzed to detect moving objects. Such avalanches are neighborhood-connected chains of cells, and their sizes, the sequence lengths, are logged into distributions and analyzed for power law and slowly decaying autocorrelation properties, the marks of SOC. The bottom right of Figure 24 shows how the distributions of salt and sand models contrast markedly as seen from this viewpoint.

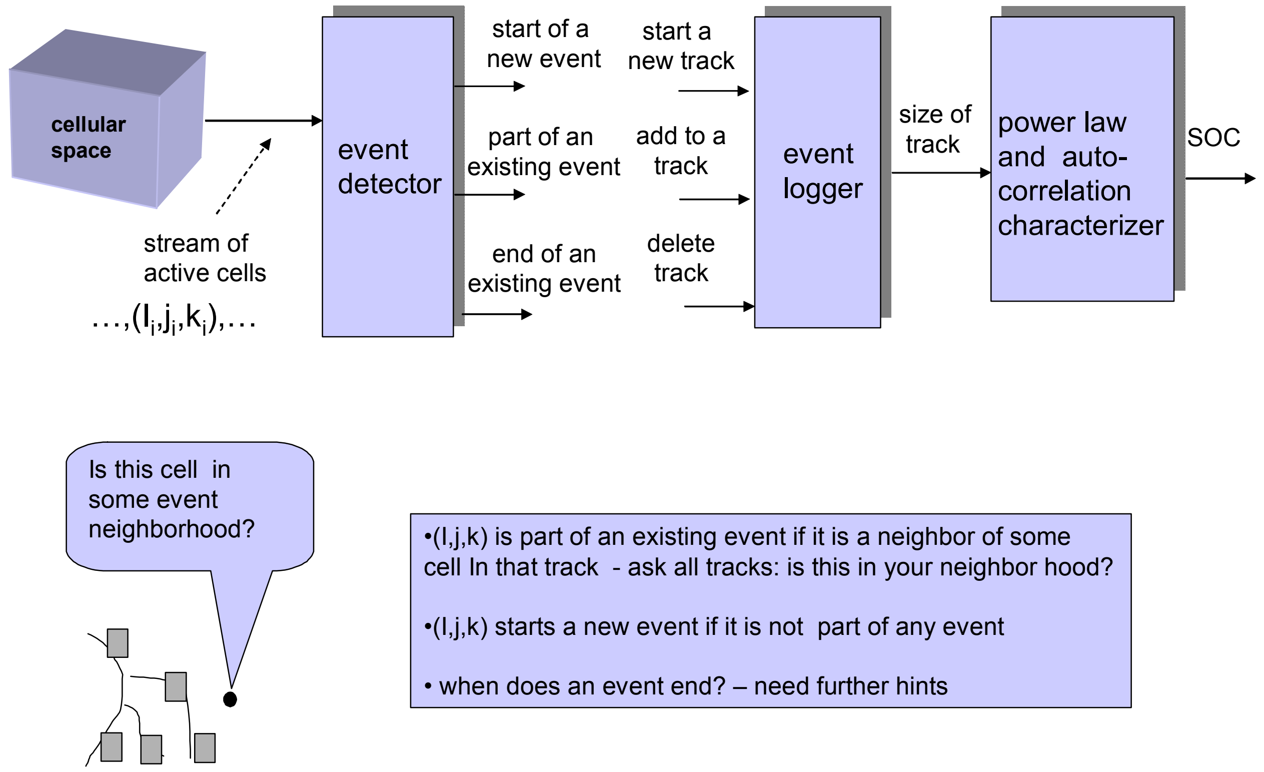

cellular space
stream of active cells
$\ldots,(I_i,j_i,k_i),\ldots$
event detector
start of a new event
part of an existing event
end of an existing event
start a new track
add to a track
delete track
event logger
size of track
power law and auto-correlation characterizer
SOC
Is this cell in some event neighborhood?
•(I,j,k) is part of an existing event if it is a neighbor of some cell In that track - ask all tracks: is this in your neighbor hood?
•(I,j,k) starts a new event if it is not part of any event
• when does an event end? – need further hints

Figure 25 Cellular Space Avalanche Experimental Frame

Now, consider trying to validate, in real- time, models of a real system that is supposed to display SOC behavior. To do this, we feed simulated model activity traces into an online avalanche experimental frame (possibly residing on a dedicated server) which can detect whether the streams evidence power law distributions or not. The salt model, and others like it, that do not exhibit strong power law distributions, would *be invalidated* (i.e., eliminated from further consideration) as soon as statistical confidence exceeds the required threshold. Since activity streams can be generated from any cellular space, this activity processing approach to model invalidation is a general method for such models. And since SOC is a general system property presumed to characterize a wide variety of different phenomena, the method has potential application in many domains. Moreover, the concept of neighborhood-connected activity chains may characterize other spatial phenomena, such as supernova explosion events, whose length distributions or other properties provide activity "signatures" for model invalidation.

8. Conclusions

The systems-theory-based framework for modeling and simulation provides a foundation for addressing problems in the field that benefit from a foundational approach. This paper has suggested several such difficult problems centered on the V&V process for large scale, complex systems. It also has suggested opportunities for research and development that would build on the theory of M&S to provide solutions to these problems.

As one application, we showed how theory-based concepts enable development of a systematic guideline for the role of V&V in the various key steps of the High Level Architecture Federation Development and Execution Process (FEDEP). The approach incrementally introduces appropriate levels/kinds of verification and validation activities throughout the FEDEP, and thus making it possible to handle the complexity inherent in V&V of distributed, legacy-based simulations.

We also discussed the theory's support for V&V within the model composability problem and in component-based model development and reuse. We concluded that V&V must be integrated in an intimate manner within any proposed model composability approach. This is so because successful reuse of existing component models will depend heavily on the confidence levels developed for conformance of the components to their specified behaviors and in the ability to ascertain that their composition displays the behavior desired for the composed model. Research is needed to develop a model-composition framework with a robust range of reusable models, a robust theory to support selection of components and to constrain the potentially exponential number of compositions to those that are plausible given component behaviors and their interdependencies.

The theory and framework for M&S asserts a distinct separation between models and simulators, their relation to each other through the *simulation relation* and to the real world via experimental frames and the *modeling relation*. This separation leads naturally to a distinction between interoperability at the modeling and at the simulation levels. We showed how V&V plays different roles at the two levels of interoperability. Verification of simulator correctness (at the simulation interoperability level) is dependent on ensuring the correct handling of a host of syntactic issues. On the other hand, at the modeling level, V&V is needed to assure that models residing at different abstraction levels in a simulation exercise form a coherent whole and relate to each other in a predictable and verifiable manner. Research is needed to apply the concept of homomorphism, and in particular, of parameter morphism, to provide working tools for verifying the coherency of such families of multi-resolution models. We also showed that in the absence of the theory of M&S, applying the best of formal methods of software verification may miss the most important targets of V&V. This is so, since software verification is most easily applied to the software development processes of simulation engines, as isolated software systems. This may lull developers into the false confidence that there is nothing else requiring quality assurance. However, form the M&S framework perspective, verifying that the simulation engine stands in the correct relation to a target class of models, is an essential V&V concern. On the other hand, we showed that the separation between models and simulators, as embodied in the layered architecture presented earlier, enables a beneficial "separation of concerns" with the consequent advantages for efficient and effective V&V.

Another important consequence of the theory is the possibility of defining and developing universal V&V tools that are entirely domain-independent in the sense that they operate with concepts that are applicable to any model independently of its domain semantics. We discussed recommendations for further research in the development of universal tools for V&V for very large, complex simulation models. In particular, we advocate the search for general "mapping" spaces that can be applied to a wide variety of specific model classes to provide on-line model invalidation and elimination. We provided, in some detail, an example of an activity space mapping that supports such invalidation by employing the power law distributions of neighborhood-connected chains of events as signatures for comparing candidate models of self-ordered criticality of complex systems. The generalization of neighborhood-connection to other causal connection linkages is tempting but might well require some retreat from the universality requirement in that such causality relations might be intimately linked with particular model semantics. This is clearly an area for further research.

Finally, we noted that there is an abundance of concepts, methods, techniques, and tools to relating verification, validation, and accreditation in the various literatures of engineering, science and defense. Unfortunately, these various techniques are only loosely related to one another and fail to display any obvious commonality. Research is needed to integrate such techniques into the theory and framework of M&S so that a coherent approach emerges to aid in selecting the most appropriate tools for the various phases of the V&V process.

References/Bibliography

1. Klir, G., *Architecture of Systems Problem Solving*. 1985: Plenum Press.
2. Booch, G., *Object Solutions: Managing the Object-Oriented Project*. Object-Oriented Software Engineering, ed. G. Booch. 1996, Menlo Park, CA: Addison-Wesley. 323.
3. Pressman, R., *Software Engineering: A Practitioner's Approach*. fourth Edition ed. 1997: McGraw Hill.
4. UML, *Unified Modeling Language*. 2000: http://www.rational.com/uml/index.jtmpl.
5. OMG, *Unified Modeling Language Specification V1.4*. 2002: http://www.omg.org/ technology/documents/formal/uml.htm.
6. SBA, *Simulation Based Acquisition: A New Approach*. 1998, Defense Systems Management College, Report of the Military Research Fellows DSMC.
7. Zadeh, L.A. and C.A. Desoer, *Linear System Theory, The State Space Approach,*. 1963, New York:: McGraw Hill.
8. Wymore, A.W., *A Mathematical Theory of System Engineering: The Elements*. 1967, NY: Wiley.
9. Padulo, L. and M.A. Arbib, *System Theory*. 1974, Philadelphia: Saunders.
10. Mesarovíc, M.D., ed. *Views on general systems theory*. 1964, John Wiley. 178.
11. Mesarovic, M.D. and Y. Takahara, *General Systems Theory: Mathematical Foundation*. 1975, New York: Academic Press.
12. Zeigler, B.P., *Theory of Modeling and Simulation*. 1976, New York: John Wiley and Sons.
13. Zeigler, B.P., *Multi-Facetted Modeling and Discrete Event Simulation*. 1984, New York: Academic Press.
14. Zeigler, B.P., H. Praehofer, and T.G. Kim, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Second Edition ed. 2000: Academic Press.
15. Ho, Y.C., *Special Issue on Discrete Event Dynamic Systems.* IEEE Proceedings, 1989.
16. Meystel, A.M. and J.S. Albus, *Intelligent Systems Architecture, Design, and Control*. 2001: John Wiley & Sons, Inc.
17. Sarjoughian, H.S. and F.E. Cellier, eds. *Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies*. 2001, Springer Verlag. xxxii+397.
18. DMSO, *DoD M&S Verification, Validation, and Accreditation*, S. Youngblood, Editor. 2001, DoD.
19. Wymore, A.W., *Model-based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design*. 1993, Boca Raton: CRC.
20. Mesarovic, M.D., D. Macko, and Y. Takahara, *Theory of hierarchical, multilevel, systems*. 1970, New York,: Academic Press.
21. Knepell, P.L. and D.C. Aragno, *Simulation Validation: A Confidence Assessment Methodology.* 1993, IEEE Computer Society Press: Los Alamitos.
22. Law, A.M. and W.D. Kelton, *Simulation Modeling and Analysis, 3rd Edition*. 1999: McGraw-Hill.
23. Sargent, R.G. *Verification and Validation of Simulation Models*. in *Winter Simulation Conference*. 1994.

24. Balci, O. *Verification, Validation, and Testing*. in *Winter Simulation Conference*. 1998.
25. Nance, R.E. and J.D. Arthur, *The methodology roles in the realization of a model development environment*. 1988, Virginia Tech: Blacksburg.
26. Balci, O. *Verification, Validation, and Accreditation of Simulation Models*. in *Winter Simulation Conference*. 1997.
27. Nance, R.E. *Model Development Revisited*. in *Winter Simulation Conference*. 1984.
28. Johnson, M.V.R., M.F. McKeon, and T.R. Szanto, *Simulation Based Acquisition: A New Approach*. 1998, Defense Systems Management College Press: Fort Belvior.
29. Ewen, D., et al. *Computer Generated Forces Applications to a Simulation Based Acquisition Smart Product Model for SC-21*. in *9th conference on Computer Generated Forces and Behavioral Representation*. 2000. Orlando, FL.
30. Hollenbach, J.W. *Department of Navy (DON) Corporate Approach to Simulation Based Acquisition*. in *Simulation Interoperability Workshop*. 2000. Orlando, FL.
31. SBATF, *A Road Map for Simulation Based Acquisition-Report of the Joint Simulation Based Acquisition Task Force*. 1998.
32. NRC, *Modeling and Simulation in Manufacturing and Defense Acquisition: Pathways to Success*. 2002, Washington DC: National Research Council, National Academy Press.
33. DMSO, *VV&A Recommended Practices Guide*. 2002: http://www.msiac.dmso.mil/vva/.
34. DMSO, *DoD M&S Verification, Validation, and Accreditation*, https://www.dmso.mil/public/transition/vva/, Editor. 2002, DoD.
35. DoD, U.S., *High-Level Architecture Interface Specification (Version 1.3)*. 1998.
36. DoD, U.S., *High-Level Architecture Object Model Specification (Version 1.4)*. 1999.
37. Sarjoughian, H.S., B.P. Zeigler, and S.B. Hall, *A Layered Architecture for Agent-based System Development.* IEEE Proceeding, 2001. 89(2): p. 201-213.
38. Page, E.P., B.S. Canova, and J.A. Tufarolo, *A Case Study of Verification, Validation and Accreditation for Advanced Distributed Simulation.* ACM Transactions on Modeling and Computer Simulation, 1997. 7(7): p. 393-424.
39. Sarjoughian, H.S., et al., *Simulation-based HW/SW Architectural Design Configurations for Distributed Mission Training Systems.* Simulation, 2001. 77(1-2): p. 23-38.
40. Pace, D.K. *Naval Modeling and Simulation Verification, Validation, and Accreditation*. in *Winter Simulation Conference*. 1993. Los Angeles.
41. Kasputis, S. and H.C. Ng. *Composable Simulations*. in *Proceedings of the 2000 Winter Simulation Conference*. 2000.
42. Davis, P.K., *Aggregation, Disagreegation, and the 3:1 Rule in Ground Combat*, MR-638-AF/A/OSD, RAND. 1995.
43. Davis, P.K. and J. Bigelow. *Introduction to Multi-Resolution Modeling (MRM) with an Example Involving Precision Fires*. in *Enabling Simulation Technology for Simulation Science, 13th SPIE*. 1998. Orlando, Florida.
44. Davis, P.K., *Adaptive Designs for Multiresolution Multiperspective Modeling (MRMPM)*, in *A Tapestry of Systems and AI-based Modeling and Simulation Theories and Techniques*, H.S. Sarjoughian, Cellier, F.E., Editor. 2000, Springer Verlag.
45. Davis, P.K., et al. *Model Composability as a Research Investment: Responses to the Featured Paper*. in *Proceedings of the 2000 Winter Simulation Conference*. 2000.
46. Hall, S.B., B.P. Zeigler, and H.S. Sarjoughian. *Joint Measure: Distributed Simulation Issues In a Mission Effectiveness Analytic Simulator*. in *Simulation Interoperability Workshop*. 1999. Orlando, FL.
47. Aronson, J. and D. Wade. *Benefits and pitfalls of composable simulation*. in *Simulation Interoperability Workshop*. 2000. Orlando, FL.
48. Page, E.H. *Composable Simulation*. in *DARPA Advanced Simulation Technology Thrust (ASTT) Workshop*. 1999. Orlando, FL.

49. Dahmann, J., et al. *HLA and Beyond: Interoperability Challenges*. in *Simulation Interoperability Workshop*. 1999. Orlando, FL: IEEE.

50. Sarjoughian, H.S. and B.P. Zeigler, *DEVS and HLA: Complementary Paradigms for Modeling and Simulation?* Transactions of the Society for Modeling and Simulation International, 2000. 17(4): p. 187-197.

51. Zeigler, B.P., S.B. Hall, and H.S. Sarjoughian, *Exploiting HLA and DEVS to Promote Interoperability and Reuse in Lockheed's Corporate Environment.* Simulation, 1999. 73(5): p. 288-295.

52. Reynolds, P.F., S. Srinivasan, and A. Natrajan, *Consistency Maintenance in Multiresolution Simulators.* ACM Trans on Modeling and Simulation, 1997.

53. Zeigler, B.P. and H.S. Sarjoughian. *Object-Oriented DEVS*. in *11th SPIE*. 1997. Orlando, Florida.

54. Zeigler, B.P., *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. 1990, New York: Academic Press.

55. Rozenblit, J.R., J.F. Hu, *Integrated Knowledge Representation and Management in Simulation Based Design Generation.* IMACS Journal of Mathematics and Computers in Simulation, 1992. 34(3-4): p. 262-282.

56. ACIMS, *DEVSJAVA Software*. 1999, Arizona Center for Integrative Modeling and Simulation.

57. DoD, U.S., *High-Level Architecture Rules (Version 1.3)*. 1998.

58. Lutz, R., R. Scrudder, and J. Graffagnini, *High Level Architecture Object Model Development And Supporting Tools.* Simulation, 1998. 71(6): p. 401-409.

59. Dahmann, J.S., F. Kuhl, and R. Weatherly, *Standards for Simulation: As Simple As Possible But Not Simpler The High Level Architecture For Simulation.* Simulation, 1998. 71(6): p. 378-387.

60. Myjak, M.D. *RTI Interoperability Study Group Final Report*. in *Simulation Interoperability Workshop*. 1999. Orlando, FL: IEEE.

61. Zeigler, B.P. *A Theory-based Conceptual Terminology for M&S VV&A*. in *Simulation Interoperability Workshop*. 1999. Orlando, FL.

62. Freigassner, R. and H.S. Sarjoughian. *A Systems Approach to a Verification and Validation Methodology within the FEDEP Six-Step-Process*. in *Europe - Simulation Interoperability Workshop*. 2001. London, UK: IEEE.

63. Sydow, A., et al., *The DYMOS Model System for the Analysis and Simulation of Regional Air Pollution*, in *Modellierung und Simulation im Umweltbereich*, R. Grützner, Editor. 1997, Vieweg-Verlag. p. 209-219.

64. Freigassner, R., *Studie zur Simulation der Generierung und Ausbreitung von bodennahem Ozon am Beispiel Grossraum Linz*, in *Systems Theory and Information Technology*. 1997, University of Linz: Linz.

65. U.S. Army Simulation, T.a.I.C., *OneSAF Operational Requirement Document*. 1998, High-Level Architecture: Orlando, FL.

66. Sarjoughian, H.S. and B. Zeigler. *Models & Representation of their Ownership*. in *Winter Simulation Conference*. 2000. Orlando, FL.

67. Zeigler, B.P., et al. *Predictive Contract Methodology and Federation Performance*. in *Simulation Interoperability Workshop*. 1999. Orlando.

68. Zeigler, B.P., H.S. Sarjoughian, and H. Praehofer, *Theory of Quantized Systems: DEVS Simulation of Perceiving Agents.* Cybernetics and Systems, 2000. 31(6): p. 611-647.

69. Gaudel, M.C. and T. Moineau. *A Theory of Software Reusability*. in *2nd European Symposium on Programming*. 1992: Lecture Notes in Computer Science.

70. Zaremski, A.M. and J.M. Wing. *Specification Matching of Software Components*. in *ACM SIGSOFT Symposium, The Foundation of Software Engineering*. 1995. New York: ACM Press.

71. Zeigler, B.P., G. Ball, and H.S. Sarjoughian. *Implementation of the DEVS Formalism over the HLA/RTI: Problems and Solutions*. in *Simulation Interoperability Workshop*. 1999. Orlando, FL.

72. Zeigler, B.P., et al., *The DEVS/HLA Distributed Simulation Environment and its Support for Predictive Filtering*. 1998, ECE, The University of Arizona.

73. Bak, P., *How nature works: the science of self-organized criticality*. 1996, New York, NY: Copernicus.
74. Jensen, H.J., ed. *Self-organized criticality: emergent complex behavior in physical and biological systems*. Cambridge lecture notes in physics. Vol. 10. 1998, Cambridge University Press: New York.
75. Lin, K.C., A. Sisti, and L. Chow, *Study on Crowded Two-Dimensional Airspace - Self-Organized Criticality.* AIAA Journal of Aircraft, 1998. 35(2): p. 301-306.