Framework for M&S–Based System Development and Testing In a Net-Centric Environment

Bernard P. Zeigler, Ph.D., Dale Fulton, , Phil Hammonds, Ph.D, Jim Nutaro, Ph.D.
Northrop Grumman Information Technology Team
Joint Interoperability Test Command
Fort Huachuca, AZ 85613-7051
and
Arizona Center for Integrative Modeling and Simulation
zeigler@ece.arizona.edu

ABSTRACT:
Department of Defense (DoD) acquisition policy requires testing throughout the systems development process to ensure not only technical certification but also mission effectiveness. Complexity within each new system, as well as composition into families of systems and systems of systems, combines with the extensive use of simulation in the design phase to multiply the challenges over traditional interoperability testing methodologies and processes. In this paper, we present a new methodology and process that integrates system development and testing intended to address the new challenges. The approach goes beyond current software development paradigms in that it rests upon and exploits dynamic systems theory, a modeling and simulation (M&S) framework, and model-continuity development concepts. The process can be applied to development of systems from scratch or in a form of reverse engineering in which requirements have already been developed in an informal manner. We illustrate the latter possibility with an example drawn from a current effort to support automated test generation for an important tactical command and control standard. We also discuss how the new process represents an important addition to current methodologies in that enables simulation-based model continuity in developing and testing specifications that stem from the DoD Architectural Framework (DoDAF). Finally, we address the development of M&S and allied software development infrastructure that can offer web-accessible services for DoD's emerging Net-Centric Enterprise Services on the Global Information Grid.

1. Introduction

Interoperability testing, such as that conducted by the Joint Interoperability Test Command (JITC), employs a structured process in two testing categories (standards conformance and interoperability) across two different environments (synthetic and live). The process examines the entire range of systems, from single interface systems to highly complex multiple systems of systems with multiple interfaces. However, this process is not comprehensive enough to enable an overall mission effectiveness assessment that reaches beyond the individual performance of a proposed system to include its interoperability within joint systems of systems configurations. In this paper, we discuss a framework, based on modeling and simulation formalism, being introduced in JITC's context to evolve toward greater capability to test for mission effectiveness. Other test organizations might adopt this formal approach in a similar way, tailoring it to their mission and long-range goals.

Cost-effective development of today's complex systems must largely, if not exclusively, rely upon modeling and simulation (M&S) principles, methodologies, and technologies. As detailed in recent Department of Defense (DOD)-sponsored reports [1,2], M&S, when properly performed through various stages of the design lifecycle, can provide effective assistance in formulating a system's capabilities, predicting and comparing the cost/benefit ratios of its various alternative architectures and evaluating its projected mission effectiveness. However, currently, there is no comprehensive approach for developing and executing simulation technology for the downstream stages of the lifecycle, including interoperability and overall mission effectiveness testing.

The following section provides a brief background to introduce concepts and an approach that will explore the current problem areas more closely and propose system-level comprehensive solutions.

2. Background: Review of System Theory and Framework for Modeling and Simulation

Hierarchy of System Specifications

Systems theory deals with a hierarchy of system specifications which defines levels at which a system may be known or specified. Table 1 shows this Hierarchy of System Specifications (in simplified form, see [*Theory of Modeling and Simulation* [3] for full exposition).

Level	**Name**	**What we specify at this level**
4	Coupled Systems	System built up by several component systems which are coupled together
3	I/O System	System with state and state transitions to generate the behavior
2	I/O Function	Collection of input/output pairs constituting the allowed behavior partitioned according to the initial state the system is in when the input is applied
1	I/O Behavior	Collection of input/output pairs constituting the allowed behavior of the system from an external Black Box view
0	I/O Frame	Input and output variables and ports together with allowed values

Table 1: Hierarchy of System Specifications

At level 0 we deal with the input and output interface of a system.

At level 1 we deal with purely observational recordings of the behavior of a system. This is an I/O relation which consists of a set of pairs of input behaviors and associated output behaviors.

At level 2 we have knowledge of the initial state when the input is applied. This allows partitioning the input/output pairs of level 1 into non-overlapping subsets, each subset associated with a different starting state.

At level 3 the system is described by state space and state transition functions. The transition function describes the state-to-state transitions caused by the inputs and the outputs generated thereupon.

At level 4 a system is specified by a set of components and a coupling structure. The components are systems on their own with their own state set and state transition functions. A coupling structure defines how those interact. A property of coupled system which is called “closure under coupling” guarantees that a coupled system at level 3 itself specifies a system. This property allows hierarchical construction of systems, i.e., that coupled systems can be used as components in larger coupled systems.

As we shall see in a moment, the system specification hierarchy provides a mathematical underpinning to define a framework for modeling and simulation. Each of the entities (e.g., real world, model, simulation, and experimental frame) will be described as a system known or specified at some level of specification. The essence of modeling and simulation lies in establishing relations between pairs of system descriptions. These relations pertain to the the validity of a system description at one level of specification relative to another system description at a different (higher, lower, or equal) level of specification.

Based on the arrangement of system levels as shown in Table 1, we distinguish between vertical and horizontal relations. A vertical relation is called an association mapping and takes a system at one level of specification and generates its counterpart at another level of specification. The downward motion in the structure-to-behavior direction, formally represents the process by which the behavior of a model is generated. This is relevant in simulation and testing when the model generates the behavior which then can be compared with the desired behavior.

The opposite upward mapping relates a system description at a lower level with one at a higher level of specification. While the downward association of specifications is straightforward, the upward association is much less so. This is because in the upward direction information is introduced while in the downward direction information is reduced. Many structures exhibit the same behavior and recovering a unique structure from a given behavior is not possible. The upward direction, however, is fundamental in the design process where a structure (system at level 3) has to be found which is capable to generate the desired behavior (system at Level 1).

Framework for Modeling and Simulation

The *Framework for M&S* as described in [3], establishes *entities* and their *relationships* that are central to the M&S enterprise (see Figure 1). The entities of the framework are *source system, experimental frame, model,* and *simulator;* they are linked by the *modeling* and the *simulation* relationships. Each entity is formally characterized as a system at an appropriate level of specification within a generic dynamic system.

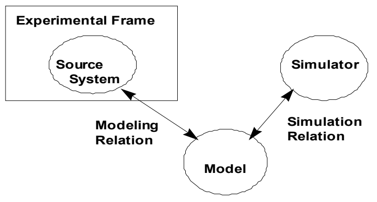

Figure 1. Framework Entities and Relationships

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.

Experimental Frame

An experimental frame is a specification of the conditions within which the system is observed or experimented; it is also the operational formulation of the objectives that motivate an M&S project. A frame is realized as a system that interacts with the source system, or System Under Test (SUT), to obtain the data of interest under specified conditions. An experimental frame specification consists of four major subsections:

input stimuli: specification of the class of admissible input time-dependent stimuli. This is the class from which individual samples will be drawn and injected into the model or system under test for particular experiments.

control: specification of the conditions under which the model or system will be initialized, continued under examination, and terminated.

metrics: specification of the data summarization functions and the measures to be employed to provide quantitative or qualitative measures of the input/output behavior of the model. Examples of such metrics are performance indices, goodness-of-fit criteria, and error accuracy bound.

analysis: specification of means by which the results of data collection in the frame will be analyzed to arrive at final conclusions. The data collected in a frame consists of pairs of input/output time functions.

When an experimental frame is realized as a system to interact with the SUT (or its model), the four specifications become components of the driving system. For example, a generator of output time functions implements the class of input stimuli. In application to testing of systems, as we discuss below, the experimental frame takes the form of a composition of test models that can interact with the SUT through designated input/output interfaces. Testing can be done in both simulation time or in real-time, the difference being that the latter relates to the system or wall clock time while the former does not. Model continuity discussed below, allows the same model to be employed in the two different contexts.

Model

A model is a system specification, such as a set of instructions, rules, equations, or constraints for generating input/output behavior. Models may be expressed in a variety of formalisms that may be understood as a means for specifying subclasses of dynamic systems. The Discrete Event Systems Specification (DEVS) formalism [3] delineates the subclass of discrete event systems and it can also represent the systems specified within traditional formalisms such as differential (continuous) and difference (discrete time) equations.

Simulator

A simulator is any computation system (such as a single processor, or a processor network, or, more abstractly, an algorithm), which is 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).

The DEVS formalism implements a well-defined concept of a simulation engine to execute models and generate their behavior. A *coupled model* in DEVS consists of component models and a coupling specification that tells how outputs of components are routed as inputs to other components. The simulator for a coupled model is illustrated in Figure 2. It consists of a coordinator that has access to the coupled model specification as well as simulators for each of the model components. The coordinator performs the time management and controls the message exchange among simulators in accordance with the coupled model specification. The simulators respond to commands and queries from the coordinator by referencing the specifications of their assigned models. The simulation protocol works for any model expressed in the DEVS formalism. It is an algorithm that has different realizations that allow models to be executed on a single host and on networked computers where the coordinator and component simulators are distributed among hosts.

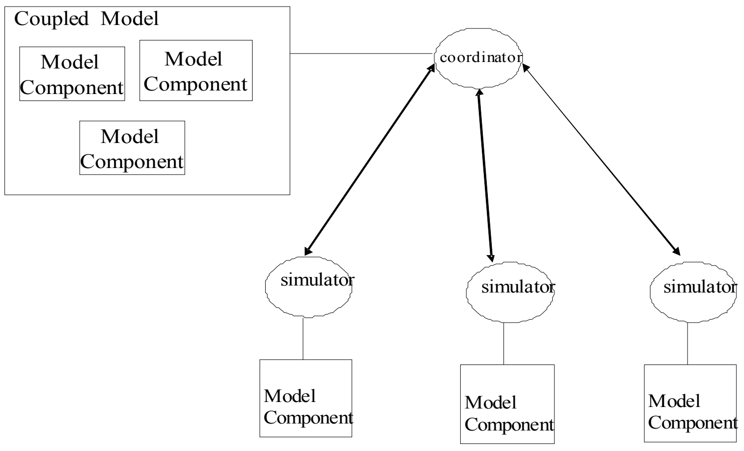

Figure 2. The DEVS Simulation Protocol

Validation and Verification Relationships

The entities *system, experimental frame, model,* and *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*.

The basic modeling relation, *validity,* refers to the relation between a model, a source system, and an *experimental frame*. 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. The term *accuracy* is often used in place of validity. Another 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.

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. In practice, as suggested above, simulators are constructed to execute not just one model but also 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.

Implementation of the above concepts and the M&S framework are key to the correct integration of a systems approach and formalism for DoD testing. By clearly defining and discriminating at fundamental levels the models, simulators, and experimental frames, we can better organize and apply M&S resources to the testing process.

Model Continuity

Model continuity refers to the ability to transition as much as possible of a model specification through the stages of a development process. This is opposite to the discontinuity problem where artifacts of different design stages are disjointed and thus cannot be effectively consumed by each other. This discontinuity between the artifacts of different design stages is a common deficiency of most design methods and results in inherent inconsistency among analysis, design, test, and implementation artifacts [4,5]. Model continuity allows component models of a distributed real-time system to be tested incrementally, and then deployed to a distributed environment for execution. It supports a design and test process having 4 steps (see [6,7]),

1) Conventional simulation to analyze the system under test within a model of the environment linked by abstract sensor/actuator interfaces.
2) Real-time simulation, in which simulators are replaced by a real-time execution engines while leaving the models unchanged.
3) Hardware-in-the-loop (HIL) simulation in which the environment model is simulated by a DEVS real-time simulator on one computer while the model under test is executed by a DEVS real-time execution engine on the real hardware.
4) Real execution, in which DEVS models interact with the real environment through the earlier established sensor/actuator interfaces that have been appropriately instantiated under DEVS real-time execution.

Model continuity reduces the occurrence of design discrepancies along the development process, thus increasing the confidence that the final system realizes the specification as desired. Furthermore, it makes the design process easier to manage since continuity between models of different design stages is retained.

3. Bifurcated Model-Continuity-based Development Process

Combining the systems theory, M&S framework, and model-continuity concepts leads naturally to a formulation of a *Bifurcated Model-Continuity-based Life-cycle Process* for developing and testing military and other software-intensive systems. As illustrated in Figure 3, the process bifurcates into two streams – system development and test suite development – that converge in the system testing phase. The Process has can be applied to development of systems from scratch or in a form of reverse engineering in which requirements have already been developed in informal manner. The Process has the following characteristics:

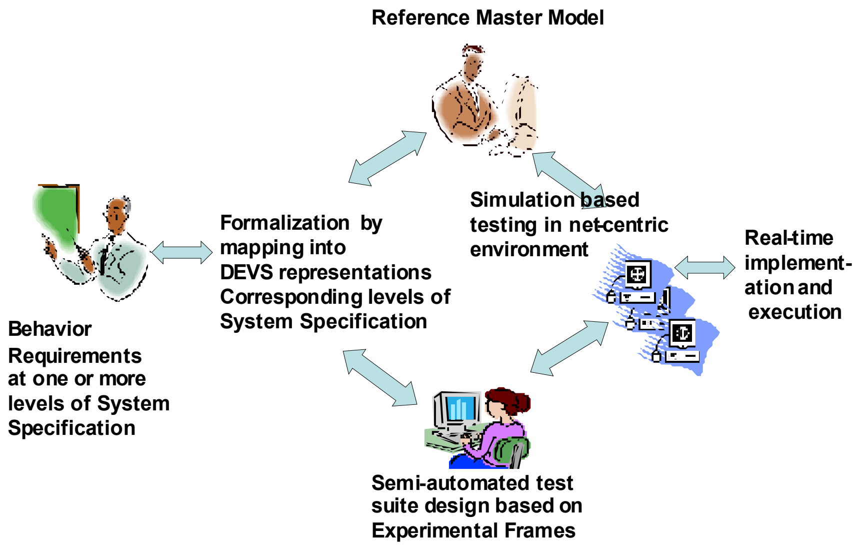
Reference Master Model
Formalization by mapping into DEVS representations Corresponding levels of System Specification
Simulation based testing in net-centric environment
Real-time implement-ation and execution
Behavior Requirements at one or more levels of System Specification
Semi-automated test suite design based on Experimental Frames

Figure 3. The Bifurcated Model-Continuity-based Life-cycle Process

Behavior Requirements at one or more levels of System Specification: The Hierarchy of System Specification offers well-characterized levels at which requirements for system behavior can be stated. Although initially ill-formulated, as the process proceeds, iterative development allows refinement of the requirements and increasingly rigorous formulation resulting from the formalization and subsequent phases.

Formalization by Mapping into DEVS: Concurrently with the formulation or capture of Behavior requirements, they are formalized as DEVS model components that are coupled together to form an overall Reference Master Model.

Reference Master Model: The master DEVS model serves as a reference model for any implementation of the behavior requirements. This model can be analyzed and simulated with the DEVS simulation protocol to study logical and performance attributes Using model continuity, it can be executed with the DEVS real-time execution protocol and provides a proof-of-concept prototype for an operational system.

Semi-automated test suite design: Branching in the lower path from the formalized specification, we can develop a test suite consisting of experimental frames called test models that can interact with a System Under Test (SUT) to test its behavior relative to the specified requirements. .

Simulation based testing: The test suite is implemented in a Net-centric simulation infrastructure and executed against the SUT. The test suite provides explicit pass/fail/unresolved results with leads as to components/ that might be sources of failure.

Optimization and Fielded execution: The reference model provides a basis for correct implementation of the requirements in a wide variety of technologies. The test suite provides a basis for testing such implementations iin a suitable test infrastructure. Test tools should carry into the fielding and operational tests of the system as we have discussed, and provide operationally realistic test cases and scenarios.

Iterative nature of development: The process is iterative allowing return to modify the master DEVS-model and its informal precursor requirements specification. Model continuity minimizes the artifacts that have to be modified as the process proceeds.

Relation to other software development concepts.

We note that the roles of dynamic systems theory and of the M&S framework are critical in supplying the concepts and levels for the statement of behavioral requirements. This is evidenced by the use of DEVS as the basic medium for formalization, analysis, simulation, and execution. The fundamental inclusion of systems and M&S concepts distinguishes the Bifurcated Process from the Model Driven Architecture (MDA), an approach to software development based on the Unified Modeling Language (UML) [8]. MDA separates the specification of system functionality (the model) from its implementation on specific technology platforms. While MDA separates model from implementation, it is intended for application to all of software development and therefore lacks the specific constructs needed from application to modeling and simulation software. The dynamic time-dependent behavior characteristic of simulation software calls for the inclusion of dynamic systems theory and the M&S framework, together with an integrated test development approach, as evident in the above Bifurcated Process.

A comparison with other formalisms is in order. Formal model-based approaches such as Petri nets, Z, and SDL provide syntaxes that are backed up by well-defined semantics. Thus they allow precise specification, which opens the possibility of verification and validation. However, there is a gap between these formal specification languages and computational realization [9]. Research works on state-based object Petri nets [10] and Object-Z [11] try to narrow this gap by integrating them with object orientation. Furthermore, although these models have mathematical bases, they are not derived from a systems theory perspective. For example, they usually do not support hierarchical decomposition. The lack of systems theory concepts make it difficult to scale up to large software-intensive systems. Moreover, efficient development of test models is greatly facilitated by the System Specification Hierarchy, as we show in the next section.

A recent DoD mandate requires that the DoD Architectural Framework (DoDAF) be adopted to express high level system requirements and designs[13]. DoDAF provides broad levels of systems specification related to operational, system, and technical views. However it does not provide a formal algorithmically-enabled process that incorporates dynamical systems theory and the M&S framework and that can support model continuity through a simulation-based development and testing life-cycle. Indeed, mapping of high-level DoDAF specifications into lower-level DEVS formalizations would enable such specifications to be tested in virtual simulation environments before further testing and fielding.

4. The Bifurcated Development Process Applied to Conformance Testing

Joint Single Link Implementation Requirements Specification (JSLIRS) is an evolving standard (MIL-STD-6016c) [14] that presents vexing challenges to automated testing of conformance of systems to it. The specification document states requirements in natural language, employing directives such as the "system shall…," that are open to ambiguous interpretations. The document is voluminous, with many hyperlinked chapters and appendixes, thus rendering its interpretation labor intensive and prone to error. Because of its size and complexity, the specification as a whole is potentially incomplete and inconsistent. As a consequence, it is a major challenge to ensure that a certification test procedure developed from the specification document completely covers the requirements and can be consistently replicated across the numerous military service, national, and manufacturer contexts in which JSLIRS standard certification testing will be executed. With this motivation, in the spring of 2003, the JITC undertook to create a formal version of the JSLIRS that could provide a complete and unambiguous specification and at the same time, enable more automated test development. In the following, we illustrate the Bifurcated Model-Continuity-based Life-cycle Process in this context.

The Design Capture Stream of the Bifurcated Process

In the JITC project, our approach began with the recognition that the JSLIRS specification is, in fact, a description of a system that tells how it should respond to stimuli in various situations. In contrast to the interpretation in the context of system design, the objective in the context of conformance testing is to transform this description into a large family of test procedures and to specify the required outcomes of tests and sequence of tests. Further, sequences of tests can be regarded as the injection of stimuli by experimental frames that will realize the test procedures. To execute a test plan requires developing experimental frames that do the generation of input to the SUT and observe the SUT's output. For conformance at the DoDAF technical level, a set of positive and negative test cases must be selected to provide the desired coverage. From a DoDAF operational view, a set of tests can be designed to cover the normally expected interactions between the SUT and the world, as well as interactions that would not normally occur but cannot be excluded from consideration. The inputs come from either an external source, such as human operator or sensors, or as a result of an internal event in the system, such as a timeout or anomaly detection, which results in an issuance of an alarm. *A system that has discrete event input and output characteristics, such as this, can be characterized by a DEVS model* [3]. This illustrates the formalization step in Figure 3, where the initial JSLIRS document is formalized as a DEVS model.

Figure 4 outlines the process we undertook to formalize the Joint Single Link Implementation Specification. A common practice is to examine such requirements documents from the point of view of threads of linked paragraphs corresponding to a sequence of test actions. For example, a thread may start with the attempt of an operator to drop a radar track; if the track has a moderate level of importance, the implemented system is required to generate a high-level alert requiring the operator to reassert the intention to drop the track. This thread may continue to consider issuance of a drop-track confirmation, a test for its subsequent nonexistence. In the systems-based approach, examining such a thread serves to uncover relevant elements of the state vector. In this case, such elements include status and importance of tracks, whether a drop-track request has been received, etc. Indeed, the state vector concept is a digital state representation in which every requirement thread can be described by changes in the state vector and outputs that result from such transition. The state vector summarizes the effect of testing at any time and we need to capture relevant parts of the state vector to determine the required outcome of testing. Although the complete state vector is not known at the start of analysis, we can build up the state vector as each thread is analyzed by determining the information needed to establish the required outcome of a test case and the items that will be changed as a result of the test. Indeed, the JSLIRS document can be decomposed in a manner that relates to the state vector whose elements emerge as more and more threads are analyzed. This relation can be used to identify the set of requirements that apply to a given vector element and therefore that need to be tested to completely cover the requirements.

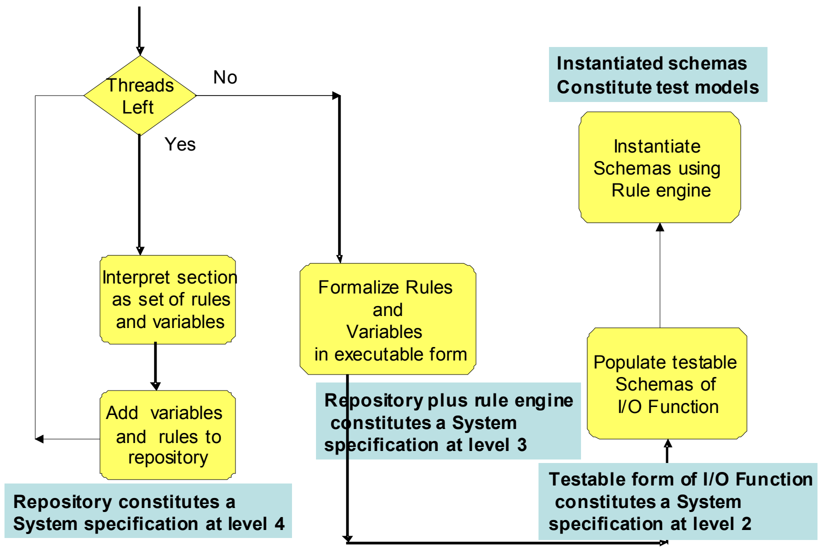
Threads Left
No
Yes
Instantiated schemas Constitute test models
Instantiate Schemas using Rule engine
Interpret section as set of rules and variables
Formalize Rules and Variables in executable form
Populate testable Schemas of I/O Function
Add variables and rules to repository
Repository plus rule engine constitutes a System specification at level 3
Testable form of I/O Function constitutes a System specification at level 2
Repository constitutes a System specification at level 4

Figure 4. Formalization of the Joint Single Link Requirements Specification

Table 2 relates the process of representing the JSLIRS specification as a DEVS (as outlined in Figure 4) to the Hierarchy of System Specification discussed earlier. Each thread is interpreted as a set of condition-action rules with suitable for describing discrete event systems [15,16]. The rules constitute components at Level 4 with the coupling between them determined by the dependencies that arise due the effects of actions of rules on conditions of others. This description becomes a Level 3 specification when the rules are formalized in executable form and augmented with an inference engine capable of executing sequences of rules. This is so since given an initial assignment to the collection of variables forming the state vector, the state transitions and output messages are generated in response to any given time-indexed sequence of input messages. By definition, this constitutes a Level 3 specification. The Level 2 form of this specification can be visualized as the set of all possible executions in which the system state is initialized (by variable assignments), a message input segment is given, and the rule engine is executed to produce the corresponding output segment. By pairing the input/output segments with the initial state that we get in this way, we develop the I/O Function specification at Level 2. This collection is infinite in principle since there are numerous states to start from, and the input segments can be extended indefinitely. Therefore, for practical testing purposes, we need a much more compact representation that captures the essence of this giant set. We'll turn our attention to this testable form of the I/O specification in a moment. Finally at Level 0, the collections of inputs and outputs are derived by collecting together all message types that can be received and transmitted, respectively. We can include operator actions and system displays as well, but we'll restrict our attention to messages, assuming they are the only automatable observables available for testing.

Level	**Name**	**What we know at this level**
4	Coupled Systems	Components – rules; coupling from rule A to rule B is determined by dependency, viz., the condition part of B shares at least one variable with the action part of A.
3	I/O System	Rule engine working in forward direction, takes a given initial assignment to variables, and executes a sequence or rules to yield an output message or operator action.
2	I/O Function	Collection of input/output pairs each with associated initial state and each of the "testable form" discussed in the text.
0	I/O Frame	Inputs: Message types, sensor inputs and operator actions Outputs: Message types, operator displays

Table 2: Levels of System Specification in JSLIRS Formalization.

The Testing Stream of the Bifurcated Process

Although the Hierarchy of System Specifications provides a good framework for structuring the bifurcated design and testing processes, a number of its aspects have to be considered in greater depth to enable it to support actual testing. One of these aspects is the development of a more compact form of the I/O Function Specification that is usable for development of semi-automated test suites. We turn our discussion to this issue by first reviewing the concepts of input/output behavior and their relation to the internal system specification in greater depth. This will put us in position to understand the complexities that arise in specifying test sequences and approaches to their management. We'll use an example drawn from the Correlation/De-correlation portion of the JSLIRS document to illustrate the concepts.

Testable Form of I/O Specification. For a more in-depth consideration of input/output behavior, we start with the top of Figure 5 which illustrates an input/output (I/O) segment pair. The input segment represents message types X with content x and Y with content y arrive at times t_1 and t_2, respectively. Similarly, the output segment represents message type Z occurring twice with content z and z', at times t_3 and t_4, respectively. At the bottom of the figure, shows a more compact representation of this same information, in which arrows about the time line represent inputs received and those below represent outputs sent.

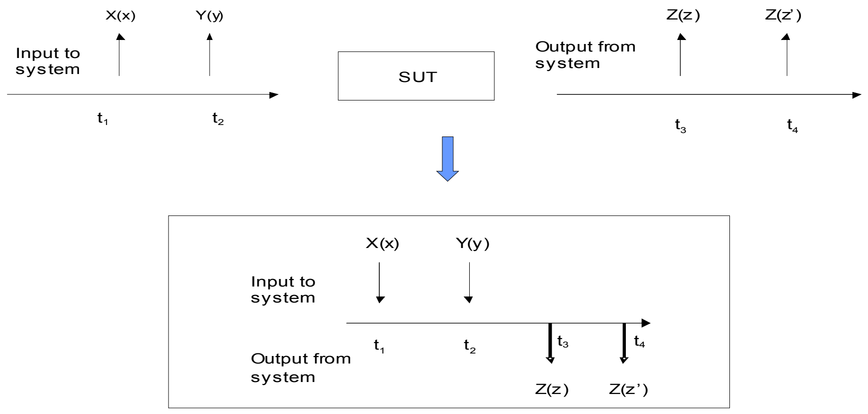

Figure 5. Representing an Input/Output Pair

To illustrate the specification of behavior at the I/O level we consider a simple system – an adder – all it does is add values received on its input ports and transmit their sum as output. However simple this basic operation is, there are still many possibilities to consider to characterize its I/O behavior such as which input values, (arriving at different times) are paired to produce an output value and the order in which the inputs must arrive to be placed in such a pairing.

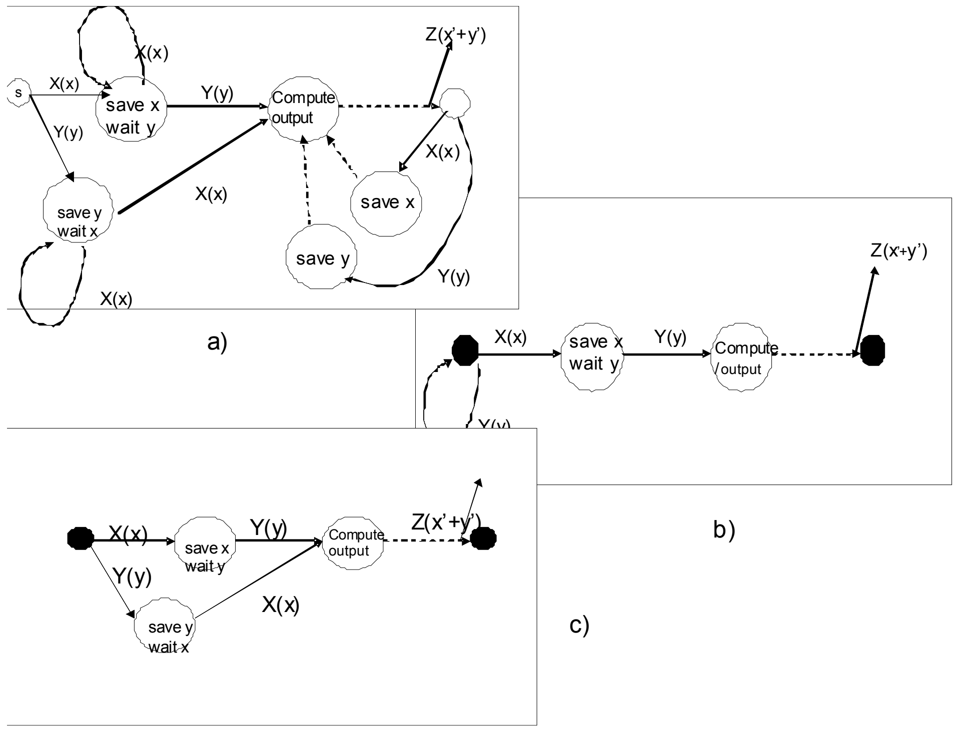

Figure 6. Variants of a DEVS Model for Processing Messages

Figure 6 portrays three possibilities, each described as a DEVS model at Level 3 of the Specification Hierarchy. In a), after the first inputs of types X and Y have arrived, their values are saved and subsequent inputs of the respective types refresh these saved values. The output of message type Z is generated some time after the arrival of an input and its value is the sum of the saved values. In b), starting from the initial state, the output can only be computed when a message of type Y is the next message to arrive after a message of type X. In this case the output is computed from the latest values of X and Y and the system is reset to its initial state. In c) the order of arrival constraint is removed but the reset requirement is retained – from the initial state, both types of messages must arrive before an output is generated (from their most recent values) and the system is reset to its initial state after the output is generated.

This example shows that even for a simple function, such as adding two values, there can be considerable complexity involved in the specification of behavior when the temporal pattern of the messages bearing such values is considered. Two implications are immediate. One is that there may be considerable incompleteness and/or ambiguity in a semi-formal specification such as JSLIRS where explicit temporal considerations are often not made. The second implication follows from the first: an approach is desirable to represent the effects of timing in as unambiguous a manner as possible and in such a way, that a discriminating test suite can be derived from this representation. We outline such an approach in the following.

Figure 7 shows some I/O pairs that serve to distinguish between the triplet of behaviors in Figure 6. In Figure 7a), after initial arrivals of message types X and Y, with subsequent output of type Z, successive arrivals of either message type, X or Y, results in an output of type Z with values shown. In b), however, a second Y message in a row does not result in an output, nor does an X following it. In contrast, for c) while a second Y message still does not result in an output, the latter will be produced when the next arrival is an X message. Note that these patterns are discernable at the Input/Output behavior level and form the basis for testing where access to the SUT is through input/output interfaces. Development of these patterns follows from understanding and manipulation of the Level 3 state-based specification.

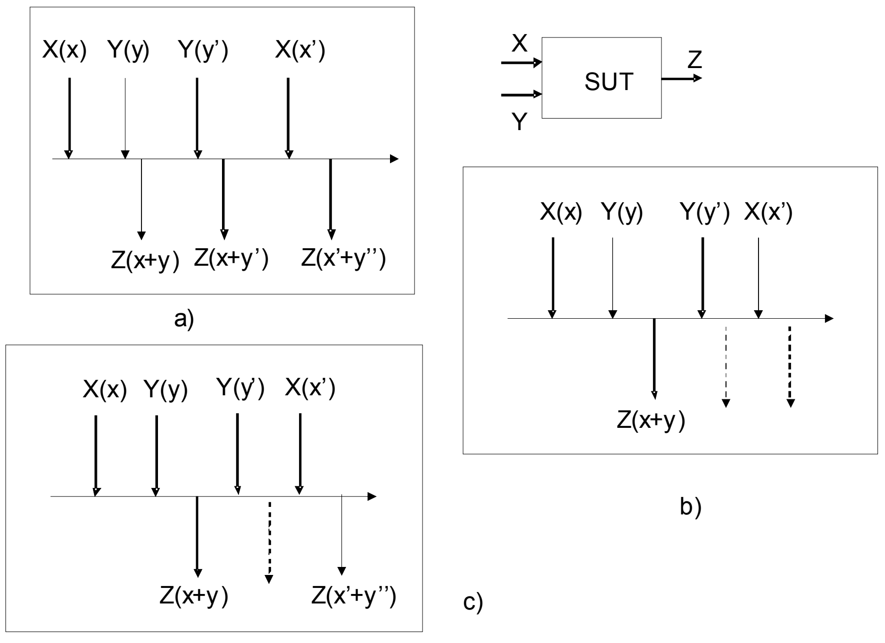

Figure 7. Input/Output Pairs Corresponding to the Variants of Figure 6

Minimal Testable Input/Output Pair. An I/O segment pair may have any finite number of input messages and output messages in its interval of definition. It is much easier to deal with segments of limited complexity and synthesize tests from such segments. Thus we introduce the concept of *minimal testable pair.* This is an I/O pair in which the output segment has at most one event at the end of the segment. Such segments are illustrated in Figure 8. We can easily extract such pairs from an I/O specification at Level 2, by going from a given state to the next one that results in an output event. Figures 8a) and 8b) illustrate this process for the corresponding variants given in Figure 7.

An I/O Specification at Level 2 for which all pairs are minimally testable is said to be a *minimal testable* I/O representation. It is easily seen that the original specification can be reconstructed from a minimal testable I/O that has been the derived from it by concatenating the minimal I/O pairs in the right sequence to replicate an original I/O pair.

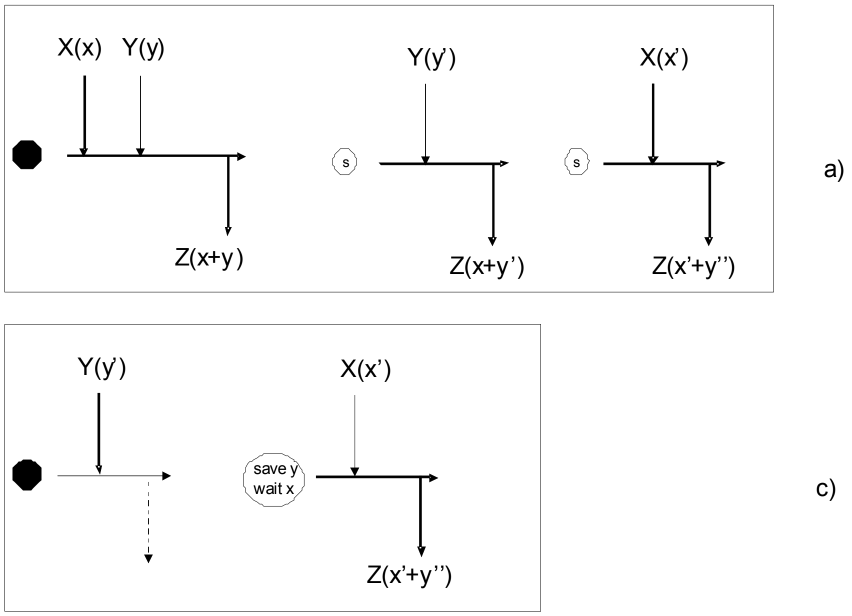

Figure 8 Some Minimal Testable Input/Output Pairs

We can go from an I/O Behavior Specification at Level 2, and in particular from its minimal testable representation, to a test model that interacts with a SUT to assess whether it satisfies the specification. Figure 9 shows an I/O pair as messages incoming and outgoing at the SUT. By reversing the roles of input and output we transform such a pair into one that becomes a required I/O pair for the Test Model. In other words, we require that the Test Model generate outputs that become inputs to the SUT, and receive inputs that are outputs of the SUT in the same time relationship as found in the original I/O pair.

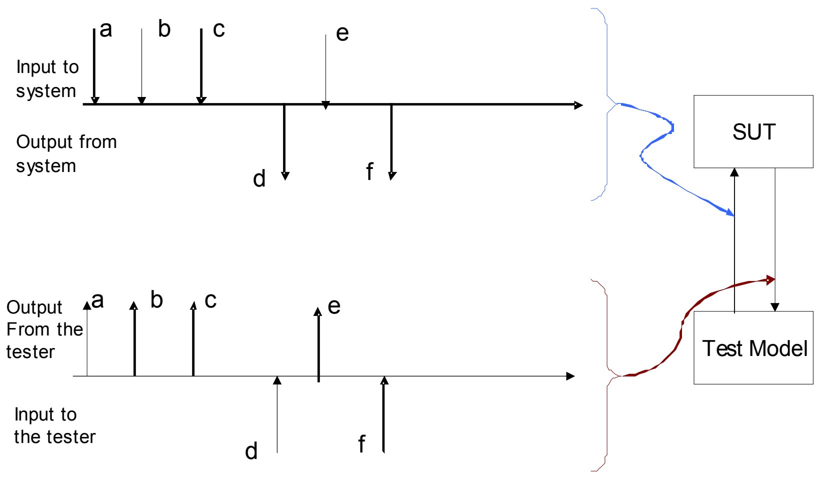

Figure 9 Mirror Imaging Input/Output Pairs

The mirror image concept just stated allows us to create a test model for any minimal testable pair in the SUT's I/O Specification. The test model is constructed as a coupled model whose components are chosen from DEVS atomic model classes called holdSend, waitReceive, and waitNotReceive. The composition and coupling are such that there are one or more holdSend instances followed by either a waitReceive or a waitNotReceive. Figure 10 illustrates how a test model sends a J3.2 message, with content data1, at time t1 – corresponding to a holdSend model with parameters J3.2, data1 and t1. This is followed by another J3.2 sent to the SUT after which the test model waits to receive a J7.2 message by time t4 via its waitReceive component. The messages, data and timing values for these models are derived from the SUTS's I/O specification using the mirror image concept. In other words, a minimal testable pair of the SUT is given in the form of a sequence of receiveAndProcess primitives alone or followed by a transmit primitive. The messages, data, and timing in these primitives determine those in corresponding holdSend, waitReceive, and waitNotReceive atomic models in the test model. Note that the test model takes the form of an experimental frame in which the holdSend models together form the generator and the waitReceive or waitNotReceive primitive forms the acceptor. Due to its stereotypical nature, this test model can be automatically synthesized from the information in the SUT's minimal testable pair.

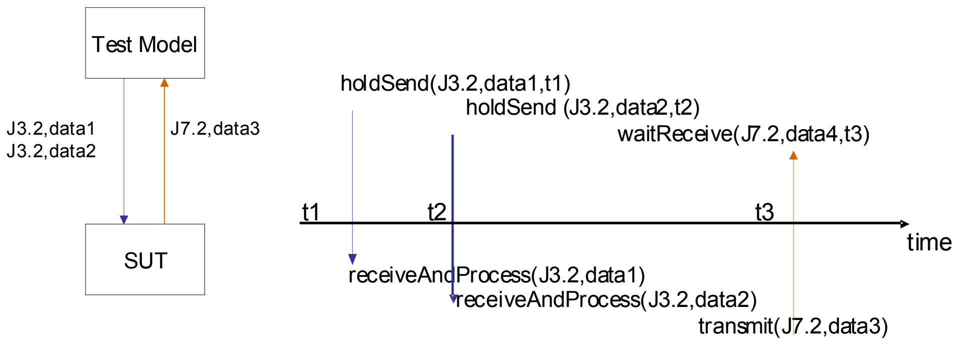

Figure 10. The Primitives Used to Construct a BasicTest Model

Test models derived from minimal testable I/O pairs can be coupled together in a higher level model as illustrated in Figure 11. Here we cascade together models for each of two minimal testable pairs, labeled "Initial correlation" and "Confirmation" respectively. The result forms a test model for the I/O pair formed by concatenating the individual minimal testable pairs. In such a cascade composition, each model (other than the last one) sends its assigned messages and waits for the given response (or non-response). If the response is correct, it starts up the next model in line; if not it stops the test with a report of the point of failure. The last model in line issues a complete pass for the I/O pair tested if the response it receives from the SUT is correct.

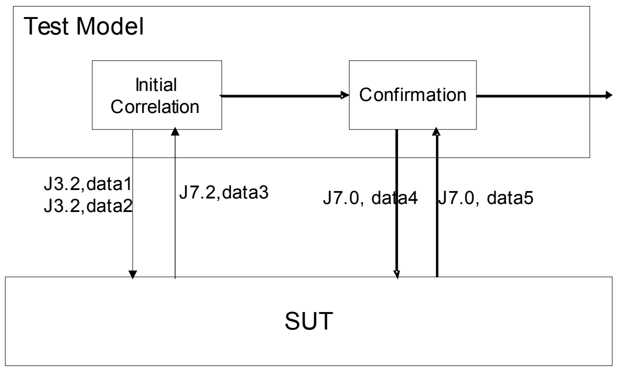

Figure 11. Composite Test Models Formed by Cascading Basic Test Models

JSLIRS Example. To illustrate the theory, consider the thread drawn from the correlation section of the JSLIRS specification depicted in Figure 12 The goal of correlation is to determine with incoming radar tracks are actually continuations of those already identified or represent new ones not yet seen. The thread starts when the SUT receives a J3.2 message bearing a radar track. The new track is compared with the existing tracks in the data base and proceeds further if the system finds one or more tracks that are close enough to it in space and time. The JSLIRS however, considers such a match-up to be only tentative and requires that a second track contained in a second J3.2 message arrive soon after that satisfies the same correlation criteria. If so, the incoming track is considered a candidate for disposal and an intention to drop this track is issued in the form of a J7.2 message. The thread continues on with a confirmation protocol that needs to be executed before the track is actually dropped. In this protocol, the SUT exchanges J7.0 messages with another radar unit that agrees with its correlation conclusion. As shown in the figure, failure to pass the various tests at any point, aborts the thread an allows the incoming track to be added as a bonafide element to the data base. It should be noted that the JSLIRS document goes into excruciating detail on the required checks and counter-checks on the tracks, the types of tests and their parameters, and some of the times involved. The baseline thread had to be extracted and synthesized from this documentation. It represents a one of several variants that account for numerous special circumstances – each of which should be captured for a complete implementation and tested in a thorough suite.

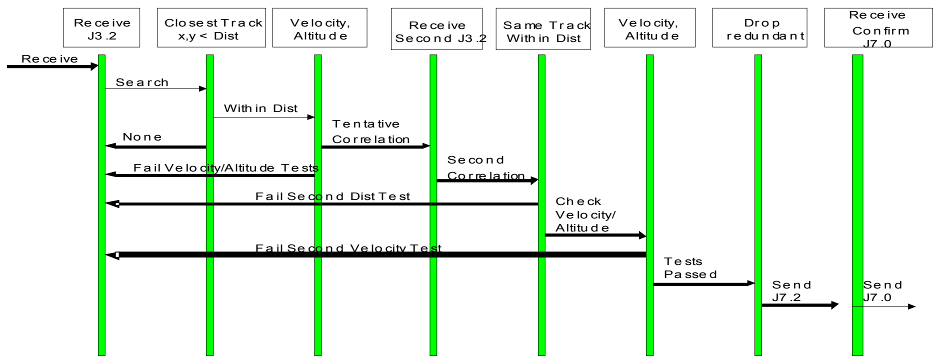

Figure 12. The Baseline Correlation Thread from JSLIRS

The correlation baseline thread just discussed breaks down into two minimal testable I/O pairs – the initial correlation testing and the subsequent confirmation process, respectively. The bottom of Figure 10 presented earlier depicts the required *receive, receive, transmit* pattern for the first I/O pair. At the top, it shows the mirror image pattern *send, send, wait* derived for the test model. In this case, the data to be sent and tested concern track numbers and associated fields. The times involved must be derived from information that is either explicit (such as in a time out) or implicit (such as in an estimate of processing complexity) in the JSLIRS document. Inability to derive these times signals temporal incompleteness in the original specification. A similar analysis produces a minimal test pair for the confirmation protocol. The complete I/O pair that represents the baseline thread is synthesized by cascading the two components as illustrated in Figure 11.

Automating the Mapping from IO Specification to Test Platforms

An important component of the Testing Steam in Figure 3 is that in which test models are derived from minimal testable I/O pairs and then coupled together in a higher level models. Figure 13 fills in some details to describe how a minimal testable IO pair is automatically converted it into a test model that can be executed in a distributed simulation environment. The IO Pairs are expressed as instances of an XML (eXtensible Markup Language,[17]) schema and stored in a repository.

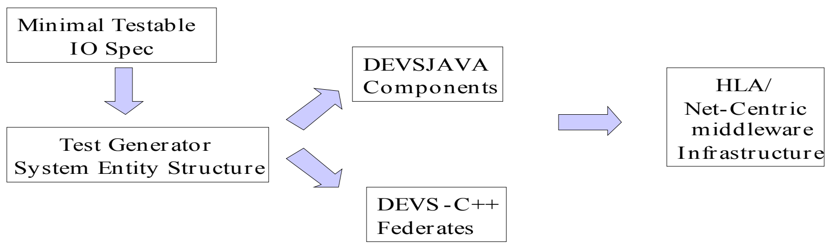
Minimal Testable IO Spec
Test Generator System Entity Structure
DEVSJAVA Components
DEVS-C++ Federates
HLA/ Net-Centric middleware Infrastructure

Figure 13. Automated Generation of Test Models

An XML-based transformation, illustrated in Figure 14, converts these instances into Basic Test Model class files and then into sequences expressed as Composite Test Model class files. The class files can be written into Java to execute in the DEVSJAVA [18,19] environment where they be tested against a stub class representing the SUT. Having been validated in this environment, the class files can be expressed as DEVS federates in C++ to execute in a distributed simulation infrastructure based on HLA [20,21]. As the DoD's Net-Centric Enterprise Services project based on the Global Information Grid [22] evolves, the HLA can be replaced by the SOAP [23] to provide the connectivity middleware required for exchange of data between the DEVS federates and the SUT.

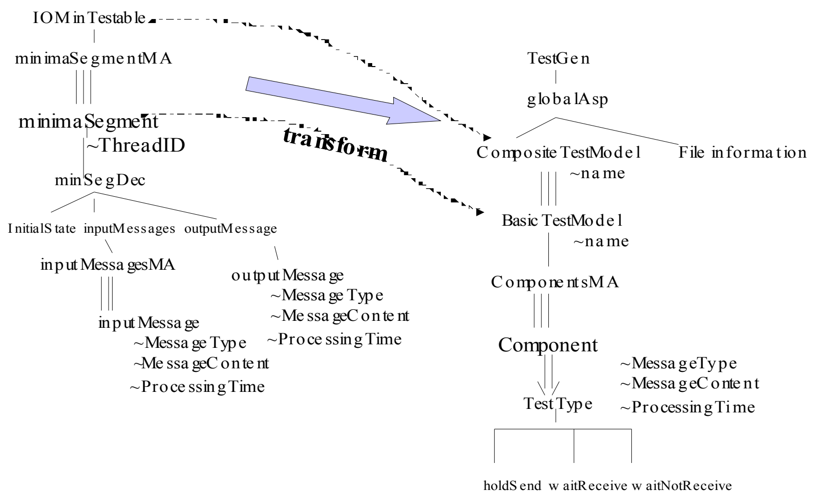

Figure 14. XML-based Transformation of Minimal Testable I/O Pairs into Test Models

Discussion and Conclusions

In this paper, we have presented a Bifurcated Model-Continuity-based Life-cycle process that combines systems theory, M&S framework, and model-continuity concepts to support developing and testing DoD systems. The process can be applied to development of systems from scratch or in a form of reverse engineering in which requirements have already been developed in an informal manner. We have illustrated the latter possibility with a detailed application to support automated test generation for an important tactical command and control standard. We have mentioned the possibility of supporting the complete life-cycle of systems development that might start with high-level DoDAF specifications. We note that, although a DoDAF specification would present a more structured starting point than the JSLIRS document, its views are still quite informal and require the same kind of additional formalization in order to be able to serve as a rigorous common specification for both development and testing streams. Thus, the Bifurcated Process represents an important addition that enables simulation-based model continuity for developing and testing DoDAF-expressed specifications.

Many questions remain for further development concerning the Bifurcated Model-Continuity-based Life-cycle process. A military system may operate in multiple environments under a wide variety of conditions. Therefore the testing stream of the Bifurcated Process needs to be elaborated to guide the development of families of experimental frames that can cover the vast space of potential test cases. The elaboration might start from the DoDAF's operational, system and technical views and develop a taxonomy of frames appropriate to different development and testing objectives associated with these views (for further discussion, see[24]).

Other questions concern practice and pragmatic aspects such as: How much interaction should there be from the testing stream to the development stream (continuous throughout the stages of development, only at designated milestones, etc.)? What happens if the formalisms and tools in the two streams are incompatible with each other? These and other questions of this nature concern the practical applicability of the approach and need to be examined in future research and with experience derived from early applications.

Military systems can be developed using a formal systems methodology that will account for functional requirements and extend to the lifecycle, including developmental and operational testing. Reliable and trusted system development requires integration of the system specification with contextual experimental frames that capture the objectives and experimentation constraints of the varied applications and uses expected of the system. Generic approaches that do not employ a formal methodology for distinguishing and evolving frames, models, and simulators cannot be trusted for robust, cost-effective system development.

The combination of traditional test methodology with rigorous M&S formal methodology provides a much richer and wider range of capabilities and options for the DoD test community. Test organizations may wish to consider a business case paradigm in which they invest in building reusable and shareable infrastructure to support the emerging trend toward M&S-based system development. This trend, together with mandates for prescribed architectural specification (DoDAF) and Net-Centricity, provide a compelling case for such a paradigm In so doing, they will enhance their mission by introducing and integrating M&S-based testing at selected points throughout a system-of-systems lifecycle. In particular, the ultimate objective of the JITC is to improve and enrich mission effectiveness testing of operational systems-of-systems through a rigorous extension of interoperability testing that the JITC and others perform now. Instituting a formal methodology to achieve the versatility demanded by future DoD system testing requires a clear plan for evolving from the current mission and capabilities to one with increased scope. The plan must then be implemented incrementally to incorporate both process and infrastructure, allowing the test organization to increase the certainty that the interests of all stakeholders, (i.e., developers, testers, and warfighters) are consistently served.

Other test organizations may wish to follow the JITC lead in considering the creation of a new Branch dedicated to development of M&S and allied software development infrastructure that can offer web-accessible services that are usable by both internal and external customers. An example of such an approach is meta-data advertisement of the JSLIRS formalized specification and the packaging of the associated test generation methodology as web-based services for DoD's emerging Net-Centric Enterprise Services on the Global Information Grid. Beyond developmental testing, one can imagine the emergence of web based accessible test model/experimental frames that could be used in a fielded system to test its readiness or diagnose problems in its operation.

References

[1] Technology for the United States Navy and Marine Corps, 2000-2035 Becoming a 21st-Century Force: Volume 9: Modeling and Simulation (1997), National Academy Press.

[2] Modeling and Simulation in Manufacturing and Defense Acquisition: Pathways to Success (2002), National Academy Press.

[3] Theory of Modeling and Simulation, Academic Press, 2000.

[4] Gery, E., D. Harel, and E. Palachi, *Rhapsody: A Complete Life-Cycle Model-Based Development System*, in *IFM, Third International Conference on Integrated Formal Methods*, 2002.

[5] W. Schulte, "Why Doesn't Anyone Use Formal Methods? " *Integrated Formal Methods, Second International Conference*, IFM 2000

[6] X. Hu, and B.P. Zeigler, " Model Continuity in the Design of Dynamic Distributed Real-Time Systems", accepted by *IEEE Transactions On Systems, Man And Cybernetics— Part A: Systems And Humans*

[7] Hu, X., *A Simulation-based Software Development Methodology for Distributed Real-time Systems*, Dissertation, University of Arizona, 2003

[8] Model Driven Architecture (MDA), OMG Document number ormsc/2001-07-01, 2001

[9] Bowman, Howard, Derrick, John, *Formal methods for distributed processing: a survey of object-oriented approaches*, Cambridge University Press, 2001

[10] A. Newman, S. M. Shatz, and X. Xie, "An Approach to Object System Modeling by State-Based Object Petri Nets," *Int. Journal of Circuits, Systems, and Computers*, Feb. 1998, Vol. 8, No. 1, pp. 1-20

[12] Graeme Smith. *The Object-Z Specification Language. Advances in Formal Methods*. Kluwer Academic Publishers, 2000

[13] DoD Architecture Framework, Software Productivity Consortium, http://www.software.org/pub/architecture/dodaf.asp, last accessed Jan 9, 2005.

[14] Single Link Interface Reference Specification for Link-16 (JSLIRS -16), 20003.

[15] Bernard P. Zeigler, *Object Oriented Simulation with Hierarchical, Modular* Models, Academic Press, 1990

[16] J. Nutaro, A DEVS Based Inference Engine for the Event Calculus, JITC Report.

[17] eXtensible Markup Language, http://www.w3.org/XML, last accessed Jan 9, 2005.

[18[H. Sarjoughian and R.K. Singh, Building Simulation Modeling Environments Using Systems Theory and Software Architecture Principles, http://www.scs.org/scsarchive/getDoc.cfm?id=1622, last accessed Jan 9, 2005.

[19] DEVSJAVA, http://www.acims.arizona.edu/SOFTWARE/software.shtml, last accessed Jan 9, 2005.

[20] James Nutaro, Phil Hammonds. "Combining the Model/View/Control Design Pattern with the DEVS Formalism to Achieve Rigor and Reusability in Distributed Simulation".*Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*,pp. 19-28, Vol. 1, No. 1, 2004.

[21]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.

[22] DoD Metadata Registry and Clearinghouse, http://diides.ncr.disa.mil/mdregHomePage/mdregHome.portal/, last accessed Jan 9, 2005

[23] SOAP 1.2 Implementation Summary, http://www.w3.org/TR/2002/WD-soap12-part0-20020626/, last accessed Jan 9, 2005

[24] Zeigler, B. P., Fulton, D., Nutaro, J., Hammonds, P., "M&S Enalbed Testing of Distributed Systems: Beyond Interoperability to Combat Effectiveness Assessment":, 9th Annual Modeling and Simulation Workshop, Dec. 8-11, 2003, ITEA White Sands Chapter (available from http://www.acims.arizona.edu/PUBLICATIONS/publications.shtml, last accessed Jan 9, 2005)