# Extending DoDAF to Allow Integrated DEVS-Based Modeling and Simulation

**Saurabh Mittal**
Arizona Center of Integrative Modeling
and Simulation (ACIMS)
ECE Department, University of Arizona
Tucson, AZ 85721
*saurabh@ece.arizona.edu*

A recent DoD mandate requires that the DoD Architecture Framework (DoDAF) be adopted to express high-level system and operational requirements and architectures. DoDAF is the basis for integrated architectures and provides broad levels of specification related to operational, system, and technical views. The combination of DoDAF operational views, which capture the requirements of the architecture, and systems views, which provide its technical attributes, forms the basis for semi-automated construction of the needed simulation models. Unfortunately, DoDAF doesn't mandate any simulation methodology to analyze the system or perform any pre-design feasibility studies. In this paper, we describe an approach to support specification of DoDAF architectures within a development environment based on DEVS (Discrete Event System Specification). The result is an enhanced system life cycle development process that includes both development and testing in an integral manner. We introduce two new operational views (OVs) in the current DoDAF making way for modeling and simulation as a part of the design process. We illustrate the process to build these new OVs from the existing OVs and their impact on the overall DoDAF system development process. We discuss automated model generation using XML through the introduced OVs, which paves the way for OVs to become service-providing components in the web services architecture.

**Keywords:** DoDAF, simulation-based design, DEVS, bifurcated development process, operational view, model continuity, SOA

## 1. Introduction

A recent DoD mandate requires that the DoD Architecture Framework (DoDAF) be adopted to express high-level system and operational requirements and architectures [1]. DoDAF is the basis for the integrated architectures mandated in DoD Instruction 5000.2 [2] and provides broad levels of specification related to operational, system, and technical views. Integrated architectures are the foundation for interoperability in the Joint Capabilities Integration and Development System (JCIDS) prescribed in CJCSI 3170.01D and further described in CJCSI 6212.01D [3, 4]. DoDAF and other DoD mandates pose significant challenges to the DoD system and operational architecture development and testing communities since DoDAF specifications must be evaluated to see if they meet requirements and objectives, yet they are not expressed in a form that is amenable to such evaluation. However, DoDAF-compliant system and operational architectures do have the necessary information to construct high-fidelity simulations. Such simulations become, in effect, the executable architectures referred to in the DoDAF document. DoDAF is mandated for large procurement projects in the Command and Control domain but its use in relation to modeling and Simulation (M&S) is not explicitly mentioned in the documentation [5, 8]. Thus an opportunity has emerged to support the translation of DoDAF-compliant architectures into models that are of sufficient fidelity to support architectural evaluation in capable simulation environments. Operational views capture the requirements of the architecture being evaluated and system views provide its technical attributes. Together these views form the basis for semi-automated construction of the needed simulation models.

DoDAF is a framework prescribing high-level design

*JDMS,* **Volume 3, Issue 2, April 2006 Pages 95–123**

artifacts, but leaves open the form in which the views are expressed. A large number of representational languages are candidates for such expression. For example, the Unified Modeling Language (UML) and colored Petri nets (CPN) are widely employed in software development and in systems engineering. Each popular representation has strengths that support specific kinds of objectives and cater to its user community needs. By going to a higher level of abstraction, DoDAF seeks to overcome the plethora of "stove-piped" design models that have emerged. Integration of such legacy models is necessary for two reasons. Firstly, as systems, families of systems, and systems-of-systems become more broad and heterogeneous in their capabilities, the problems of integrating design models developed in languages with different syntax and semantics has become a serious bottleneck to progress. Secondly, another recent DoD mandate also intended to break down this "stove-piped" culture requires the adoption of the Service Oriented Architecture (SOA) paradigm as supported in the development of Network Centric Enterprise Services (NCES) [6]. However, anecdotal evidence suggests that a major revision of the DoDAF to support net-centricity is widely considered to be needed. Indeed, under DoD direction, several contractors have begun to design and implement the NCES to support this strategy on the Global Information Grid (GIG). The result is that system development and testing must align with this mandate (requiring that all systems interoperate in a net-centric environment), a goal that can best be done by having the design languages be subsumed within a more abstract framework that can offer common concepts to relate to. However, as stated before, DoDAF does not provide a formal algorithmically-enabled process to support such integration at higher resolutions. Lacking such processes, DoDAF is inapplicable to the SOA domain, and GIG in particular. There have been efforts, such as those by Dandashi et al. [7], that have tried to map DoDAF products to SOA, but as it stands there is no clear-cut methodology to develop an SOA directly from DoDAF, let alone their testing and evaluation.

Our earlier work [8] and that of Zeigler et al. [9] described the bifurcated model continuity–based system lifecycle process. In this paper we will explore it in more detail with respect to technologies like XML and UML with their application to DoDAF. Our earlier work [8] also explored the possibility of application of DEVS to the DoDAF design process and developed a mapping between various UML elements and the DEVS formalism. It considered all three elements of the DoDAF, viz., operational view (OV), system view (SV), and technical view (TV), and their mapping with DEVS components. In this paper we will establish that enhancing DoDAF would require extending the DoDAF with more information. We will focus on the proposed extension of DoDAF and examine the information needed to execute the development process. We will demonstrate the procedure and the way in which M&S can be applied with the help of an example. We will also discuss the application of the proposed views. This discussion will show how the enhanced DoDAF can effectively support development of services in SOA environments.

Sections 2 and 3 provide some background on DoDAF views and DEVS specifications, respectively. Section 3 also discusses the key component technologies inherent in DEVS and principles of model continuity. Section 4 describes the idea behind mapping DoDAF into DEVS framework. Section 5 throws light on the prior efforts to employ M&S as an integrated solution to evaluate architectures, and highlights some problem areas encountered. It also discusses the current M&S situation and how DEVS provide solutions to these problems. Section 6 describes the bifurcated model continuity process, the basis for our approach. Section 7 discusses some gaps in DoDAF and proposes solutions on how to fill them. Sections 8 and 9 present the detailed methodology on how to transition from UML description to DEVS specifications. Section 9 also provides the justification of this mapping process. Section 10 provides a full example on how the proposed views are constructed and their significance to M&S areas. Section 11 leads the present discussion toward the benefits of this work in immediate future, followed by conclusions in section 12.

## 2. DoDAF Specifications

The Department of Defense (DoD) Architectural Framework (DoDAF), Version 1.0 (2003), defines a common approach for DoD architecture description development, presentation, and integration. The framework enables architecture descriptions to be compared and related across organizational boundaries, including joint and multinational boundaries.

DoDAF is an architecture description, and it does not define a process to obtain or build the description. The Deskbook [1] provides one method for development of IT architectures that meet DoDAF requirements, focusing on gathering information and building models required to conduct design and evaluation of an architecture. The DoDAF defines three elements for any architecture description:

1) *Operational View* (OV) - The OV is a description of the tasks and activities, operational elements, and information exchanges required

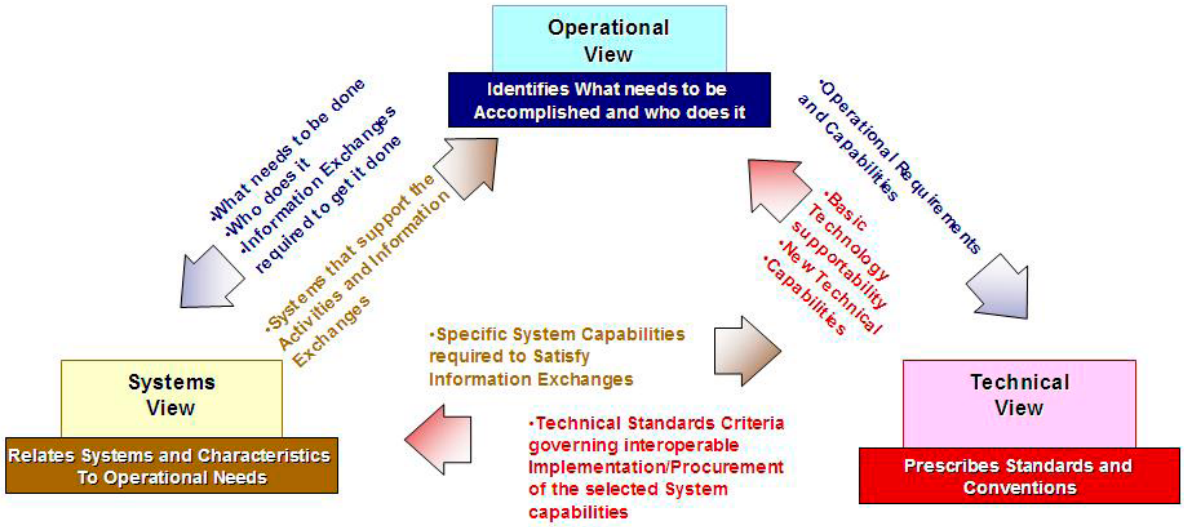


**Figure 1.** Linkages among views

to accomplish DoD missions. DoD missions include both war-fighting missions and business processes. The OV contains graphical and textual products that comprise an identification of the operational nodes[1] and elements, assigned tasks and activities, and information flows required between nodes. It defines the types of information exchanged, the frequency of exchange, which tasks and activities are supported by the information exchanges, and the nature of information exchanges.

2) *System View* (SV) - The SV is a set of graphical and textual products that describes systems and interconnections providing for, or supporting, DoD functions. DoD functions include both war-fighting and business functions. The SV associates systems resources to the OV. These systems resources support the operational activities and facilitate the exchange of information among operational nodes. Within this view, *how* the functionalities specified in OV will be met is elaborated.

3) *Technical View* (TV) - The TV is the minimal set of rules governing the arrangement, interaction, and interdependence of system parts or elements, whose purpose is to ensure that a conformant system satisfies a specified set of requirements. Within this view, the delivery of systems and functionalities is ensured along with their migration strategies toward future standards.

These views provide three different perspectives for looking at an architecture. The emphasis of DoDAF lies in establishing the relationship between these three elements ensuring entity relationships and supporting analysis; see Figure 1. The DoDAF approach is essentially data-centric rather than product-centric. The OV, SV, and TV are further broken down into specialized views whose brief description can be seen in column 3 in Table 3 ahead, as well as in the appendix. A complete description can be see in [1, 14].

1. Operational node: A node specified in OV that performs one or more operations; a functional entity that communicates with other functional entities to implement a collective functionality or a capability.

## 3. DEVS System Specifications

In this section, we review some of the background required for discussion DEVS support of DoDAF.

### *3.1 Hierarchy of System Specifications*

Systems theory deals with a hierarchy of system specifications, which define levels at which a system may be known or specified. Table 1 shows this hierarchy of system specifications (in simplified form, see [10]).

- 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 that 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 I/O 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 a 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.

**Table 1.** Hierarchy of system specifications

| Level | Name | What we specify at this level |
|---|---|---|
| 4 | Coupled Systems | System built up by several component systems that are coupled together |
| 3 | I/O System | System with state and state transitions to generate the behavior |
| 2 | I/O Function | Collection of I/O 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 I/O 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 |

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 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 that is capable of generating the desired behavior (system at level 1).

### 3.2 Framework for Modeling & Simulation

The *framework for M&S* as described by Zeigler et al. [10] establishes *entities* and their *relationships* that are central to the M&S enterprise; see Figure 2. 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. See [10] for detailed discussion.

### 3.3 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 the opposite of 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 [11]. Model

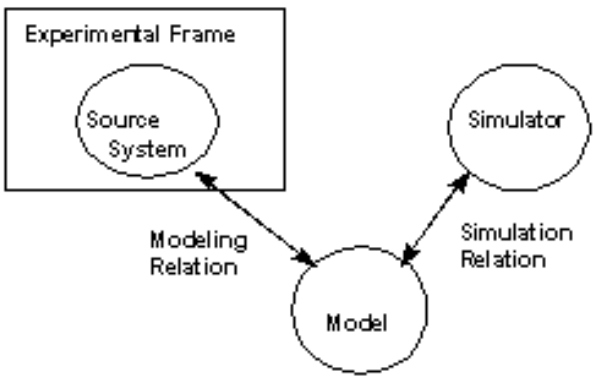


**Figure 2.** Framework entities and relationships

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 four steps; see [11]:

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; and
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 will realize the specification as desired. Furthermore, it makes the design process easier to manage since continuity between models of different design stages is retained.

## 4. Motivation for DoDAF-to-DEVS Mapping

The DoDAF suffers from following shortcomings:

1) Although there is mention of "executable architectures" in DoDAF, there is no methodology recommended by DoDAF that would facilitate the development of executable DoDAF models.
2) It has completely overlooked the model-driven development approach. Consequently, there is no formal M&S theory that DoDAF mandates.
3) DoDAF fails to address performance issues at the OV level.
4) DoDAF fails to include measures of effectiveness (MoEs) that can be evaluated at the OV stage. If any performance measures are considered at all, they are at the SV level. System parameters and performance is at a totally different resolution than MoEs.
5) There is no mechanism to perform verification and validation (V&V) at the OV stage.
6) It fails to address M&S as a potent evaluation and acquisition tool.

We propose a mapping of DoDAF architectures into a computational environment that incorporates dynamical systems theory and an M&S framework. The methodology will support complex information systems specification and evaluation using advanced simulation capabilities. Specifically, the Discrete Event System Specification (DEVS) formalism will provide the basis for the computational environment with the systems theory and M&S attributes necessary for design modeling and evaluation. We will see in the forthcoming sections that the proposed mapping will require augmentation of current DoDAF with more information set that is far from any duplication of the available DoDAF products.

We will demonstrate how this information is added and harnessed from the available DoDAF products toward development of an extended DoDAF integrated architecture that is "executable." This kind augmentation has been attempted earlier by Lee et al. [12] using CORE of the Vitech Corporation as a tool to develop the executable architecture. They developed "architectural templates" that elicit information for both the operational and system views that contained additional information than the usual DoDAF products. In another effort, Rosen et al. [13] proposed a new model called the Rosen-Parenti model that adds another layer of abstraction to the existing DoDAF, augmenting the model with various user-oriented perspectives. Going further, they developed the executable architecture with their proposed model and showed how V&V is applicable in their domain. Their model unearthed a shortcoming of DoDAF: it fails to address the performance issue at the OV level, which the Rosen-Parenti model addressed in one of their perspectives. In our attempt to augment the current DoDAF, our focus shall remain on adding minimal information that would enable DoDAF to become an executable architecture. There are potential advantages to making DoDAF a DEVS-compliant system.

An executable architecture is defined as the use of dynamical simulation software to evaluate architecture models [14]. Such executable architectures provide many benefits [12]:

1) The architecture model itself can be verified for internal self-consistency.
2) Operational concepts can be simulated, observed dynamically, verified, and refined.
3) Operational plans can be examined, assessed with their feasibility reports.
4) Trade-offs between systems can be assessed.
5) Architecture measures can be evaluated (if the metrics have been defined), which can support cost-benefit analysis and quantitative acquisition decisions.

The focus of this effort is to make a DoDAF architecture executable and provide V&V at the operational level, i.e., OV level, as also indicated in [15]. We chose to consider the development of an OV executable model primarily for the reason that the design is abstract at the OV level. There are many tools that can put different system models together and can conduct a simulation exercise. There is not much breadth to explore at the SV level as things are brought down to the implementation level with clearly defined interfaces. We aim to provide the benefits of being an executable architecture at the operational level. DEVS, with its mathematical systems theoretical foundation, serves as the ideal candidate to develop an operational executable model, as the same model can be extended to the systems level using model composition and hierarchical construction leading to multi-resolutional models, as discussed in Table 1.

We seek to employ DoDAF-to-DEVS mapping to unify multiple model representations by expressing their high-level features within DoDAF and their detailed features as subclasses of DEVS specifications. DEVS has been shown to be a universal embedding formalism, in the sense of being able to express any subclass of discrete event systems, such as Petri nets, cellular automata, and generalized Markov chains [10]. DEVS has also been employed to express a wide variety of more restricted formalisms, such as state machines, workflow systems, fuzzy logics, and others [16]. Moreover, DEVS environments have a long history of development and are now seeing ever increasing use in the simulation-based design of commercial and military systems [17]. Providing a DoDAF "front end" to a "back-end" DEVS environment will appeal to military information system designers facing the DoDAF and NCES mandates. Such designers will be able to retain their skills with representations familiar to them, while complying with DoDAF abstractions. At the same time, they can see the results of their specifications evaluated via a simulation-based execution of the model architecture. Moreover, since all mappings are into subclasses of DEVS, the resulting models can be coupled together and, therefore, can interoperate at the systems dynamics level. Thus this approach to the synthesis of system design formalisms leverages design and execution methodologies that are already used, or mandated for use, in commercial and military applications.

DEVS environments, such as DEVSJAVA, DEVS-C++, and others [18], are embedded in object-oriented implementations, thus supporting the goal of representing executable model architectures in an object-oriented representational language. As a mathematical formalism, DEVS is platform independent, and its implementations adhere to the DEVS protocol so that DEVS models easily translate from one form (e.g., C++) to another (e.g., Java) [19]. Moreover, DEVS environments, such as DEVSJAVA, execute on commercial off-the-shelf desktops or workstations and employ state-of-the-art libraries to produce graphical output that complies with industry and international standards. DEVS environments are typically open architectures that have been extended to execute on various middleware such as DoD's HLA standard, CORBA, SOAP, and others [20–23]. Therefore, the proposed design architecture supports interfaces to other engineering and simulation and modeling tools—an example of such networking is provided by Lockheed's satellite cluster mission effectiveness simulator [24]. Furthermore, DEVS operation over a web middleware (SOAP) enables it to fully participate in the net-centric environment of the NCES. As a result of recent advances, DEVS can support model continuity through a simulation-based development and testing life cycle [11]. This means that the mapping of high-level DoDAF specifications into lower-level DEVS formalizations would enable such specifications to be thoroughly tested in virtual simulation environments before being easily and consistently transitioned to operate in a real environment for further testing and fielding.

## 5. Earlier Work and Recommendations for M&S Support in DoDAF

> Reference architecture bridges the gap between processes addressing the development of contingency operations for future systems and the implementation of domain-specific architectures that build on legacy systems while incorporating new technologies and capabilities. Modeling and Simulation is a key tool to support evaluation of the effectiveness of the reference architectures and the resulting domain-specific architectures [25].

The DoDAF is a mission-system reference architecture where the goal is to provide an architecture that can accomplish mission capabilities. It is a domain-specific architecture that is actually an instance of the parent reference architecture with specified applicable domain rules [25]. In one of our earlier works [26], we proposed a rule-based meta-model structure that is applicable to such a reference architecture with M&S as a part of the design cycle. Even if the DoDAF architecture is the same for two missions, it is the domain-specific rules that will eventually decide if the developed architecture is feasible and there are no rule incompatibilities.

Dr. Alexander Levis acknowledges that M&S can provide an integrated solution in evaluation of the

designed architectures [27], but there is no explicit guidance on how to achieve this. Unfortunately, recent attempts to relate M&S to architecture frameworks such as C4ISR (a precursor to DoDAF) and model-driven architecture [5, 28, 29] have established the need for including M&S in it but have not provided a rigorous methodology for doing so. However, there have been efforts, such as NATO Active Layered Theatre Ballistic Missile Defense (ALTBMD) studies, that resulted in alignment of experimentation phases [30]. Despite all the information specified in the constructed OV, SV, and TV, this effort required the generation of a new view, coping with which systems and connections were simulated in which systems, and based on which constraints, etc. The ALTBMD study supports the view that DoDAF in its current state requires the addition of some added/pruned information set that is applicable to the M&S area.

In an attempt to evaluate the completeness of DoDAF document products in providing a sufficient information set for an executable architecture, Zinn [31] integrated the information contained in OV-5 and OV-6a to come up with an intermediate document that fed an agent-based simulation software called SEAS. Agent technology used by Zinn mitigates the solution as the independent agents (i.e., a plane, tank, etc.) can now behave independently and can modify behavior based on their decision rules. This essentially leads to an executable state machine of a component whose behavior is adaptive to the changing environment. However, the problem arises in the absence of interface specifications to port data from these agent architectures into simulation software. In his thesis [31], Zinn concludes that DoDAF contains enough information to build an executable set but it needs synthesis of intermediate documents. He also acknowledges that the development of an executable architecture must address the following:

1) The *third order analysis,* as mentioned in DoDAF, is a critical step in acquisition strategy, but there is no methodology to perform this analysis.
2) Legacy models are too monolithic to be disassembled and recomposed to model individual C4ISR effects [32].
3) Colored Petri nets (CPNs) [33] provide a means to model and simulate the above-mentioned activity. However, they fall short in modeling an adaptive environment (both in the structure and behavior), where the rules of engagement (ROE) are constantly changing as the model learns and evolves. Another area where the CPNs are lacking as a technology itself is the inability to specify timing between states. Consequently, temporal effects cannot be considered in any executable architecture that bases its performance evaluation on CPNs.

In the sequel to our paper, we discuss an approach to enhancing DoDAF with additional views that enable the DEVS formalism and technology to provide a fully capable simulation-based executable model for DoDAF specifications.

### *5.1 Overview of the Role for DEVS-Based Technology*

The Air Force Chief Architect's office (AFCAO) website [34] lists three key impact areas where use of architectures can provide real benefit:

1) Operations enhancement
   - 1.1) Requirement coherence and prioritization
   - 1.2) Better utilization of fewer personnel
   - 1.3) Deliberate exploitation of innovation
2) Programming and planning
   - 2.1) IT investment decisions (support for POM inputs)
   - 2.2) MIL-worth analysis (M&S executable architectures)
   - 2.3) AOA evaluation (trade study)
3) Acquisition support
   - 3.1) Enhanced war-fighter/user capabilities ID
   - 3.2) Execution roadmaps
   - 3.3) Source selection
   - 3.4) Technology application/transition
   - 3.5) Test support (MOE/MOP)
   - 3.6) Interoperability and integration assurance

Even though it has been realized that M&S is necessary in performing evaluation and developing acquisition strategy, there is more opportunity for current simulation technology to help. Table 2, taken from our earlier work [8], summarizes limitations of current M&S methodologies to support DoDAF and shows where DEVS-based technology can contribute.[2]

## 6. DoDAF-to-DEVS and Bifurcated Model Continuity–Based System Life Cycle Development Process

As suggested in Table 2, key to our proposal for extending DoDAF to integrate DEVS-based modeling and simulation into a process model that supports both development and testing of military and other software-intensive systems, the *bifurcated model continuity–based life cycle process* combines the systems theory, M&S framework, and model continuity

2. Although not all of the areas taken from the list above are considered in Table 2, the ones omitted are at a higher level of abstraction and will benefit indirectly from the application of M&S in general.

**Table 2.** Comparison of current technologies in development with DEVS on addressing M&S issues in AFCAO

| AFACO Reference | Desired M&S Capability | Current Working Tools (Agent-Based or CPNs) | Solutions Provided by DEVS Technology |
|---|---|---|---|
| 1.1 | Requirement coherence and prioritization | No formal methodology exists in defining architectures wherein the data model can be put directly to use for simulation modeling. | The present work aims to accomplish this by injecting requirements quite early in the design stage of DoDAF architectures, specifically in the OV phase. |
| 2.2 | MIL-worth analysis (M&S executable architectures) | Work is ongoing in this area. Due to the limitations of the technologies being used, the desired "execution" is not possible. | DEVS provides the capability to:<br>1) Reconfigure simulations on-the-fly [35];<br>2) Control simulations on-the-fly [36];<br>3) Provide dynamic variable-structure component modeling [35, 37];<br>4) Separate the model from the act of simulation itself, which can be executed on single or multiple platforms using DEVS/HLA [10]; and<br>5) Layer the simulation architecture to accomplish the technology migration or run different technological scenarios [19, 38].<br>6) With its bifurcated process, automated test generation is integral to this methodology [9]. |
| 3.1 | Enhanced war-fighter/ user capabilities | 1) Deterministic CPNs<br>2) Stochastic SEAS, but too rigid to reconfigure on the fly<br>3) Agent-based methodology again falls short in variable structure simulation model | |
| 3.2 | Execution roadmaps | There are no capabilities to control the ongoing simulation to steer it in the "right" direction. | |
| 3.3 | Source selection | | |
| 3.4 | Technology application/ transition | No dynamic reconfiguration of model and simulation reported. The simulation architecture itself has to be layered enough to accomplish technology transition. | |
| 3.5 | Test support | Agent-based technology—Zinn's work is essentially in this direction; CPNs are not capable of automated test generation | |
| 3.6 | Interoperability and integration assurance | The methodology itself has limitations; no mechanisms reported so far. | |

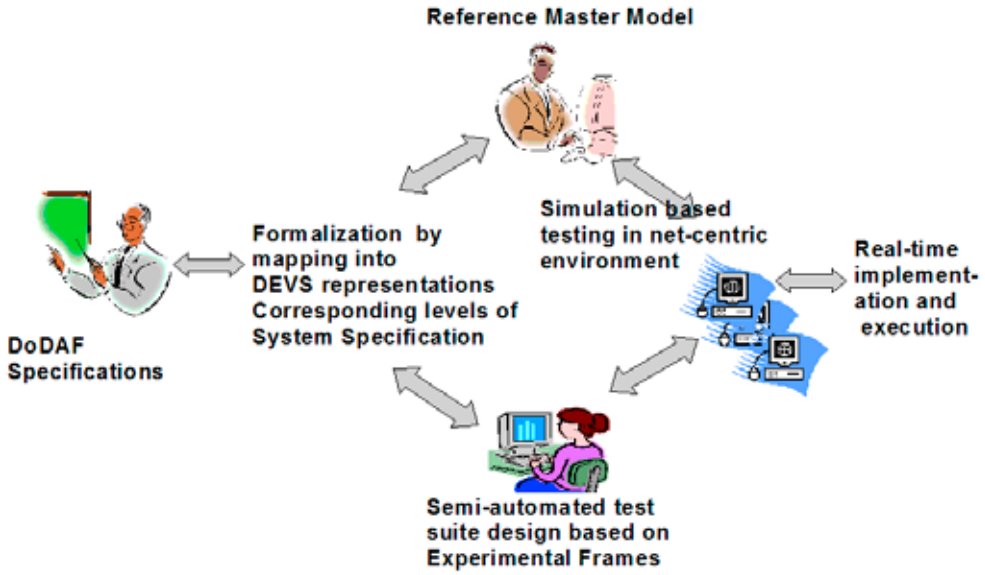


**Figure 3.** The bifurcated model continuity–based life cycle process

concepts reviewed earlier. 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 the following characteristics:

**DoDAF specifications:** As described in greater detail below, DoDAF descriptions in the operational, system, and technical views are created by designers. 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:** Concurrent with the formulation or capture of DoDAF specifications, 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 in a suitable test infrastructure. Test tools should carry into the fielding and operational tests of the system, 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 DoDAF precursor requirements specification. Model continuity minimizes the artifacts that have to be modified as the process proceeds. The design methodology provides a process (see Figure 4) to transform the DoDAF description of an architecture to a DEVS representation supporting evaluation and recommendations for a feasible design. Briefly described, the steps are as follows:

1) The architecture specifications are presented in DoDAF description format as OVs, SVs, and TVs.
2) The system specifications are then mapped to DEVS specifications, according to the translation described in Table 3 (in section 9), which map the DoDAF views to corresponding DEVS elements. The mapping is illustrated with UML elements and is expressed in XML [39]. Table 3 is the updated and extended version of our earlier work [8], focused specifically to OV. The SV is presented in the appendix.
3) Test suites for implementations of the design are developed in the test develop stream.
4) Simulation results and their analyses provide the recommendations for a feasible design.
5) Components are developed from the models using model continuity principles, and the design is verified by the TV specifications developed earlier as a part of the DoDAF process.

Creation of the DEVS model repository and DEVS test suite occur in a concurrent manner. The DEVS repository serves as a collection of models that are used to develop scenarios and experimental frames, and to conduct other simulation-oriented analysis. The DEVS test suite is designed to ensure that the required behavior as expressed in input-output pairs is correctly implemented when integrated in the system with timing constraints. One such semiautomated test suite called Automated Test-Case Generator (ATC-Gen) has been developed at JITC by Zeigler [9], and has been applied for Link-16 testing [40]. Analysis of the experimental frame simulations and the system test results are compared and evaluated to determine departure from required behavior. This error margin is called the *conformance measure.* Ideally, the designed model has a 100% conformance with the test suite. If the departure exceeds a given tolerance, the model is revised to increase the model-test conformance. All this assumes that the initial DoDAF specifications have been cast in stone. Typically, however, the iterative process will also suggest new or modified specifications at the DoDAF level. The iterative loops can be seen in Figure 4. Finally, when the models conform to the system test specifications, the test suite presents the design

and performance recommendations as the outcome of this data-centric process. The model repository serves as the basis of design of components based on model continuity principles, and the test suite serves as the benchmark for performance evaluation and matching the technical specifications as developed in the TV DoDAF descriptions.

In Section 8, we shall demonstrate how the bifurcated model continuity–based life cycle process can provide a framework in which to develop DEVS technology support for DoDAF, leading to model repositories and test suites. However, several gaps in the DoDAF specification have to be bridged to enable the infusion of DEVS to take place. We turn toward these shortfalls and our proposals to address them.

## 7. Filling Gaps in DoDAF

We address three gaps in the DoDAF as currently formulated relating to a) message flow among activities, b) transition from OV-5 to OV-6, c) temporal specifications, and d) accountability for failure of activity execution.

### *7.1 Message Flow Among Activities*

AV-1 deals with the overall functionality of the system. AV-2 deals with the data dictionary and terms used in the DoDAF specification of the system. This leads to the first "functional" document, OV-1, which gives way to OV-5, which describes the intended functionalities in fair amount of detail. It lays out the functionalities in terms of capabilities and activities in hierarchical as well as sequential manner. A *capability* is defined as a functionality comprising of many activities. These activities are linked by "messages"

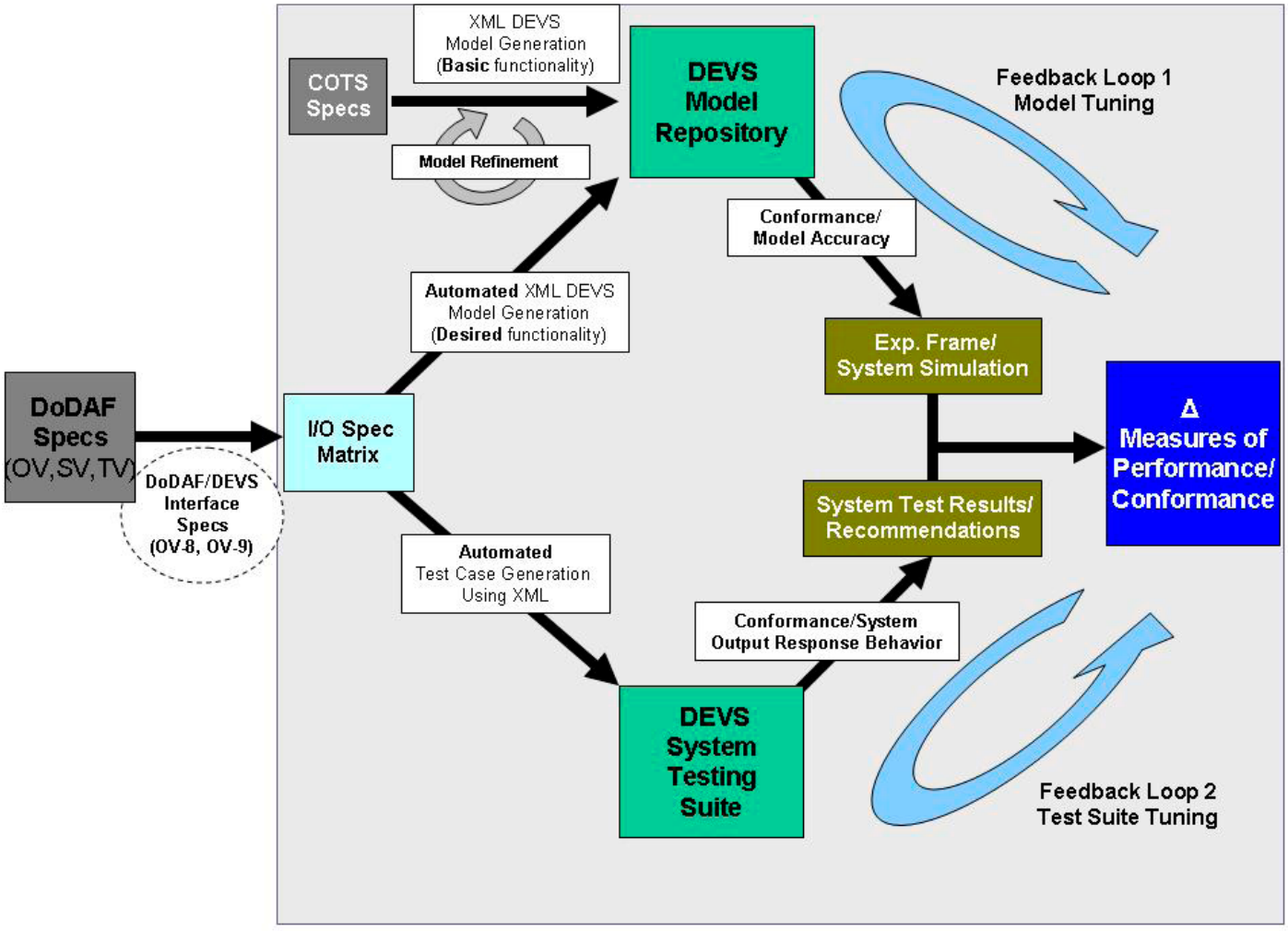


**Figure 4.** Bifurcated DEVS-to-DoDAF system life cycle development process

that are apparently flowing between the defined activities. What flows between these activities is not exactly defined in DoDAF. The abstraction level of "information" flow is not discussed in DoDAF. They can be top-level operational information exchanges (OIEs) or specific data messages. The first gap occurs at this point. DoDAF does not define the interactions between the activities. This is all the more ambiguous since activities are not "entities" that can be physically realized. However, these activities do exchange messages. There is no entity structure developed by this stage of functional development. An *entity* is defined as a component that can be physically realized. There is no mention of any entity taking responsibility for any activity. Were an "activity" to be mapped to the corresponding UML notation, it would map to use-case diagrams. However, unlike the use-case diagram, the OV-5 does not take into account the entities (actors in UML) that take responsibility for this activity. Since there is no apparent entity structure, the interfaces are essentially absent.

Structure in DoDAF does not appear until OV-2 and OV-7. The activities in OV-5 seem to exchange messages or events for that matter but there is no mechanism specified to send or receive events. In other words, there is no specification of activity interface. If activities are considered as potential components of the operational views, there is ample reason to consider them as "components" of a particular *capability* and provide interfaces to enable and define the message communication between activities.

The advantages of considering an activity as a *component entity* are manifold:

1) An activity is grounded in the design at the DoDAF specification level.
2) It can be back-referenced to any particular capability or it may serve to more than one capability.
3) Activities with defined interfaces provide a way to develop a test suite to test the capability definition of the system.
4) An activity can be allotted to the defined entities such that a real-world entity or a group of entities can be held responsible for its execution.
5) It brings specificity to the component design by ensuring the interfaces defined in activities be mapped on a one-to-one basis in the target component entity held responsible for this activity.
6) It provides structure to the functional aspect of DoDAF that can feed the entity structure of the system, which then can be aggregated toward the system views.
7) It paves the way for DoDAF-DEVS mapping and how testing can be applied to the design process (as described in the previous section) at an operational stage.
8) It allows the designer and planner to define the needed entities in the OV phase of DoDAF specification of the system.
9) It provides the framework to incorporate M&S at two different levels of resolution to conduct feasibility studies:
   a) *At the capability level* - At this level the system can be modeled based on the "functionality" of the system. Rules and doctrines can be accounted for at this level of the model. This provides a means to test the compatibility of existing rules and doctrines when testing the feasibility of any activity. There are no means to test and validate the compatibility and inter-operability of various rules and doctrines that constrain any particular capability in DoDAF. The need for such consideration has been recognized by Dickerson and Soules [42] in the proposed CV-6 document (capability evolution document).
   b) *At the entity level* - At this level the system can be modeled based on the entity structure as developed from the capability model. New supporting entities can be introduced at this level that can support the existing capabilities or that are needed by specific capabilities such as fault-tolerance, scalability, etc.

Coming back to the discussion of current OV-5 in DoDAF, the activity diagrams are then detailed further in OV-6. The OV-6 consists of three parts. Our prime interest is in OV-6b (state chart diagrams) and OV-6c (timing sequencing and event trace diagrams), as OV-6a deals with the rules and doctrines, basically a document to describe the constraints on different activities mentioned in OV-5. OV-6b can be mapped to the UML state chart diagrams and OV-6c can be mapped to the UML timing sequence diagrams.

### 7.2 Transition from OV-5 to OV-6

The second gap comes during the transition from OV-5 to OV-6, specifically OV-6c, event trace diagrams. The OV process adds further details to activity diagrams in describing the sub-activities for OV-5 activities. It associates the current activities with the known operational nodes (which can be seen in OV-1). However, the nodes in consideration here may be as big as an organization itself, and abstraction level is

fairly high. Even if the node is the lowest level entity (an unlikely case), the complete behavioral life cycle of this entity is overlooked, and only the activities in consideration are assigned to the operational node with a presumption that the node *will* execute this activity. The life cycle with respect to the attended activities is expressed in OV-6b; however, there is no hierarchy of these nodes present that could account for the hierarchical activities under question. Consider a typical activity diagram in OV-5 and imagine that in order to execute this activity, four different nodes are performing in tandem through a sequence of sub-activities and passing events to one another. The current setup is depicted in the OV-6c timing sequence diagram describing the execution of an OV-5 activity. Similarly, the OV-6b diagram depicts the sequencing of these sub-activities. The problem of developing the life cycle of an operational node is easier if all the sub-activities are happening at one node, but that is usually not the case since multiple nodes are performing and synchronizing to execute the parent activity. The fragmented nature of activities to compose and define a parent activity or a capability makes the construction of an OV-6b diagram for an operational node more difficult. The other drawback of this methodology is the possible occurrence of inconsistency between OV-6b and OV-6c, as the statechart in OV-6b is bounded by the activities called for in OV-5 and explored in fragmented manner in OV-6c in specific event trace diagrams. There is possible loss of information here, and DoDAF provide no means to ensure consistency.

Although the development of OV-5 and OV-6 is an iterative process, it does not ensure a foolproof life cycle of an operational node. This problem does not occur in the UML architecture as the approach there is to start with use-case diagrams, then move to the class diagrams, which leads on to the activity diagrams, timing sequencing diagrams, and state chart diagrams of individual classes, in this specific order. This ensures consistency, as there are defined classes before making the timing sequencing diagrams or state charts. DoDAF has a reverse approach to this design problem, where it groups and aggregates activities and defines classes (in OV-6), then leads on to the OV-2. OV-7 and the data exchanges are simultaneously defined in OV-3. Developing state chart diagrams without developing the class entity structure is error prone. The solution to this problem is to treat an activity as a *component* and develop the activity-entity structure before OV-6b and after OV-5.

(It can be argued that the operational views are concerned with the functional description of the system that does not require any component structure definition. Furthermore, it is in the system views that the structure is made available and interfaces are defined. The counterargument to this approach is that there is no mechanism provided in DoDAF to test the operational views' development and to conduct early feasibility studies[3]).

In order to fill the second gap and maintain the order of OV-5 and OV-6 iterative development process, we suggest the following:

1) Consider the activities as *components* and place them in a structure so that they are better defined and managed.
2) Employ a methodology to transition from OV-6b and OV-6c to the DEVS behavioral model for operational nodes composition and incorporate the activity components as constituent parts with defined interfaces.
3) Incorporate the doctrines and rules of engagement specified in OV-6a to be implemented into the DEVS behavioral model of an activity *component;* see the example in section 10.

The next section describes a methodology to develop the DEVS state systems from OV-6c descriptions, which are essentially time sequencing diagrams. This, incorporated with the OV-6b statechart, will provide much needed consistency between the two documents. Ultimately, it will result in a DEVS model repository of operational nodes for modeling and simulation, testing, and control. The integrated solution to the above two gaps result in the introduction of two new OV documents:

1) OV-8, *activity components document* - This document list the activities as components with defined port interfaces that are essentially logical in structure.
2) OV-9, *activity interface specifications* - This document describes the interface specifications between activities and entities. It holds information about the mapping of a subcomponent inside an operational node that is responsible to execute any particular activity. Again, this mapping requires augmentation with logical port definitions.

Details of the development of these two documents can be seen in the next section. Figure 5 presents the operational view setup in DoDAF and its extended version.

3. When the system views are defined there can be many ways to develop test models since every boundary and interface is almost defined *(though not designed in operational views since there is no document that contains interface specifications except OV-3, which is essentially data-exchange document)*. One can build a simulation model easily as systems are already in place in the system view.

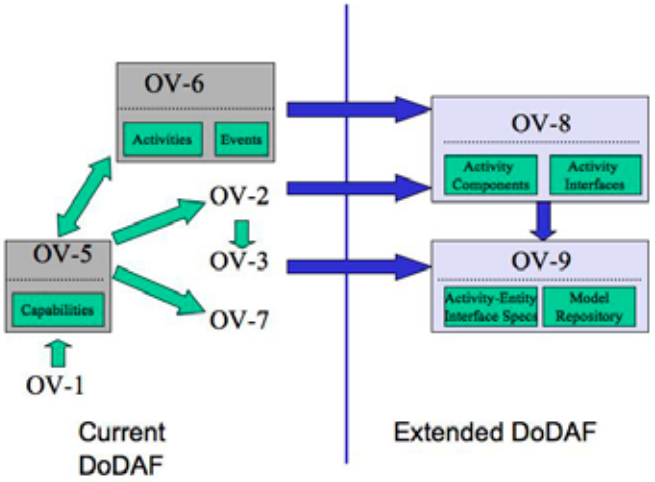


**Figure 5.** State of operational view documents in DoDAF

### *7.3 Temporal Information*

One crucial piece missing in this existing toolset is the timing information that is absent in OV-5. Our proposal attempts to bridge this gap by giving adequate consideration to interface structure and involving other views, namely OV-6b and OV-6c in conjunction with OV-5 and OV-6a. The communication information is not available until SV-2 and SV-6 are constructed, which is derived from OV-3, which is constructed late in the OV phase itself. Introducing timing well before in the design of operational views equips the acquisition strategy with information about those current systems that could meet these "operational" delay requirements.

A similar architecture called Air Operations Center (AOC) Weapon System Block 10.1 Architecture, being developed by MITRE for the Air Force, is under development along the lines of DoDAF specifications. It is considered an "integrated" architecture based on the DoDAF standards. It contains the seven essential products required, and data elements are consistent across the views. Zinn developed a procedure wherein information from different AOC DoDAF views were handled manually and brought to a level where the doctrines can be utilized to impact the simulation model in question. This information is presented in if-else constructs in the form of pseudo-code that can be fed into any XML parser for further processing. More details about the procedure can be seen in his thesis [31].

Our proposal can be seen as extending Zinn's procedures by giving more structure to the compiled information, which includes temporal information in activities as an important part of the information exchange. Zinn used the IDEF process methodology to depict various OVs and noted the inherent inadequacy of the IDEF3 process [43] when being fed to an agent-based simulation software, e.g., SEAS, which has no rules defined in case of OR split that dictate which path to take to resume the activity. We believe that the problem stems from the fact that the underlying architecture does not consider timing an important concept. This problem would not have arisen in the first place, and the situation would have resulted in "timeout" (as can be easily expressed in DEVS) in case of OR split. This is but one example of the limitations of the modeling methodology that does not lead to simulatable models.

### *7.4 Accountability for Failure of Activity Execution*

The third and final gap that we found was the lack of any accountability for failure of activity execution by operational nodes over time. There is an absence of accountability because there are no means provided in DoDAF to test the design principles in operational views. Since M&S is not systematically considered in the current DoDAF specification, there is no means to test the feasibility of the system. Furthermore, there is no support for modeling technologies like model-driven architectures (MDA) and model continuity principles.

Our effort is toward providing accountability to the design process by introducing M&S at the correct development stage, where it is possible to experiment and modify the operational architecture in question. This is very much needed, as it is not reaffirming to presume the capabilities of operational nodes (through COTS specifications) and then move on to the system views without ensuring the functionality of the system in operational views. The reason for choosing the DEVS formalism as a means to M&S is its expressive power and modularity support. The concept of "elapsed time" is one of the key aspects of DEVS formalism, and it provides a means to evaluate the component behavior in a finite time frame. It enables the component to attend, and respond, to any external events in time intervals prescribed in its DEVS specifications. Incorporation of DEVS in DoDAF will make the design process more tractable and controllable.

## 8. From OV-6 UML Diagrams to DEVS Component Behavior Specifications

Figure 6 describes the development of a DEVS description model from a simple time sequencing thread in a time sequencing diagram.

A simple time sequencing diagram is considered to illustrate the DEVS activity component development process and how it fits into the DEVS description of an

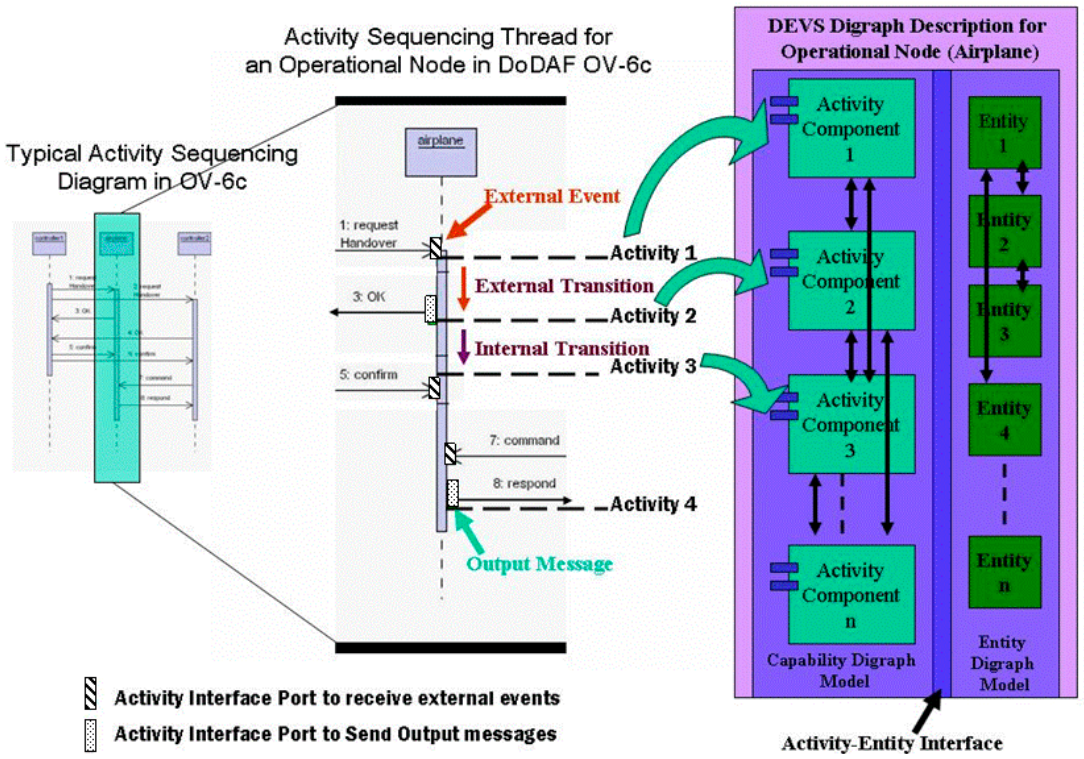


**Figure 6.** Development of DEVS description model from UML time sequencing thread

operational node. Consider that a hierarchical activity is being addressed by three operational nodes, and they are exchanging events between sub-activities in order to perform this activity. In the first diagram in Figure 6 (leftmost), we can see them interacting with each other. The center part of the figure consists of the thread for one operational node and is enlarged for better analysis. The sequencing diagram is represented in UML notation, and this node has a lifeline during the course of which it receives events and sends output messages or events to other nodes. In mapping to a DEVS formalism we need to have information about the *internal* transitions (when no events are received) from one activity to another activity and the *external* transitions (when an event is received at this node sent by another node). The timeline of the node consists of a sequence of activities that the node will undergo in the event of external transition or internal transition. The complete timeline is available in OV-6b, so there is all the more reason to maintain consistency and similar input and output trajectories of sequential activities. Different markings on the thread are self-explanatory. Striped boxes indicate the port interfaces where an external event can be received and dotted boxes indicate the port from which output events can be sent to other nodes. Activity 1 receives an external event and undergoes activity 2 after generating an output message. Activity 2 undergoes internal transition toward activity 3 in absence of any external event. This particular thread displays only a subset of activities performed by this node. Since DEVS employs a port-based component structure system we identify the input and output ports and assign them to specific activity components at this particular developmental stage. This results in the introduction of a new OV document, OV-8, which contains the mapping definitions of logical ports and activity components. Finally, these activities, if not present in OV-6b, are then introduced in OV-6b for a comprehensive set of activities performed by this operational node.

Another byproduct of this stage is the mapping of activity components with entity components that constitute an *operational node.* This is specified in a new OV document called OV-9. This contains information about the activity ports, activity components, entity components, and logical entity ports that are mapped to logical activity ports in OV-8. Introduction of these new OV documents modifies the overall DoDAF OV specification structure illustrated in Figure 6.

## 9. DoDAF-to-DEVS Elements

Table 3 provides the mapping of various DoDAF OV products into DEVS modeling constructs. UML is chosen as the preferred method of DoDAF representation.

**Table 3.** DoDAF-DEVS extended translation table

| DoDAF Elements | | | UML Elements | DEVS Elements (Generated Using XML) | |
|---|---|---|---|---|---|
| | Name | Description | | | |
| Operational View | OV-1 | Top-level operational view | • Use-case diagrams | • Activity component identification<br>• Top-level entity structure | DEVS model repository |
| | OV-5 | Operational activity model | • Use-case<br>• Activity sequencing diagrams<br>• Data flow diagrams | • Activity component updating<br>• Hierarchical organization of activities<br>• Input-output pairs<br>• Port identification | |
| | OV-6 | Operational timing and sequencing diagrams | • Time sequencing diagrams<br>• State machine diagrams | • DEVS atomic model creation (initialize function, internal and external, transition functions, time advance, and output functions) for activity components<br>• Entity identification<br>• Activity-entity component mapping | |
| | OV-2 | Operational node connectivity | • Composite structure diagrams | • Coupling information<br>• Hierarchical component organization | |
| | OV-8 | Activity component description | • Composite structure diagrams<br>• State charts | • Activity component update<br>• Activity port identification and refinement | |
| | OV-3 | Operational information matrix | | • Input-output transaction pairs<br>• Message formats<br>• Activity interface and coupling information | DEVS system test suite |
| | OV-9 | Activity interface specifications | • State charts<br>• Composite structure diagrams | • Acitvity-entity interface<br>• Entity structure refinement<br>• Activity-entity port mapping and refinement | |
| | OV-7 | Logical data model | • Packages (only for xUML)<br>• Class diagrams | • Entity identification<br>• Hierarchical structure | |
| | OV-4 | Organizational relationship chart | • Class diagrams | • Entity identification<br>• Hierarchical entity structure | DEVS model repository |

First the UML element is mapped with the DoDAF product document, and then the same UML element is mapped to the DEVS element(s). Representation of DoDAF into corresponding UML has been presented earlier by Telelogic [41].

Their representation included SV products as well.[4] In Table 3 we have also incorporated the two new OV products, OV-8 and OV-9. Since UML is essentially an object-oriented methodology, work has been attempted in the area of transforming UML models to CPN executable architectures [15]. Our work is similar in nature, where UML elements are transformed to DEVS elements. The last column links the DEVS elements to Figures 3 and 4 by categorizing them into model repository and semi-automated test suite elements.

### *9.1 Justification for the DoDAF-DEVS Mapping Process*

This section discusses the effect of introducing two new operational views in the current DoDAF and their impact on the overall DoDAF design process. Information technology–based systems of the future will be increasingly complex with participants across the globe communicating through disparate channels. Interoperability is very much in question. Scalability and fault-tolerance issues have to be addressed. Capabilities have to be satisfied and

4. For SV mapping refer to the appendix.

reliability has to be ensured. Any large system that DoDAF specification documents intend to build has to realize these important facets of architecture design. Modeling and simulation with its model continuity principles is fast becoming an accepted method of evaluating design principles ensuring accountability to various components within the system. DoDAF has completely overlooked M&S as a possible means to evaluate design, capabilities, and planned expansion of current architectures. There is no provision for testing the constructed system, either in OV or in SV. Ability to configure a system for optimum performance is not allowed in the current DoDAF specification document.

We have introduced two new operational views, OV-8 and OV-9, that add on features to enable M&S of the system under design. We have also demonstrated how these new documents will be created from the existing operational views. A detailed example is presented in the next section. We aim to provide structure to the OV process by shifting the perspective from describing functionality as an activity to an activity *component* with definite interfaces to other activity components as well as identified entities within an operational node. To what extent an operational node is decomposable is a subject requiring further research. We have developed

**Table 4.** Summarizing the contribution of OV-8 and OV-9 to DEVS M&S

| Artifact | SES Elements | Current DoDAF | Enhanced DoDAF | Can DEVS Model Be Created? |
|---|---|---|---|---|
| Tree 1 (Component perspective) | Entities | OV-2 (operational nodes)<br>SV-4 (systems identification) | | Too early! |
| | Hierarchical entity construction | OV-2, OV-7 | OV-9<br>(no mechanism to provide information of hierarchical formation in current OVs) | YES<br>(only the skeleton with well-formed coupled models) |
| | Specified entity-based constraints | SV-7 | OV-9<br>(hierarchical node descriptions help localize contraints at OV design phase) | NO<br>(information missing to develop behavior models) |
| Tree 2 (Capability perspective) | Capabilities | OV1,5, 6b, SV-4 | | NO<br>(no activity components defined) |
| | Hierarchical activities | OV-6, 6b, 6c, SV-5 | | NO<br>(no activity components defined) |
| | Activity-based parameters | *ABSENT* | OV-8<br>(activity as activity components definitions based on OV-5,6b)<br>(documenting procedure has placeholders for parameters and constraints identification)<br>See [44] | YES<br>(DEVS capability skeleton can be created with hierarchical activity composition with defined interfaces) |
| | Activity-based IE | OV-5, 6b | OV-8<br>(may be redundant here) | YES |
| | Activity-based ROE | OV-6a | OV-8 | YES |
| Tree 3 (Rule perspective) | Rule hierarchy | OV-6a | | YES<br>(ATC-Gen project [9, 40]) |
| | Rule-activity mapping | *ABSENT* | OV-8<br>(the whole purpose of OV-8 is realized here) | YES<br>(with full behavior for capability testing) |
| | Rule-entity mapping | *ABSENT* (partially in OV-6a) | OV-9<br>(the whole purpose of OV-9 is realized here) | YES<br>(with full behavior for system testing) |

a testing process for defined capabilities (that were defined during the conceptual design process in OV-5) and the way in which various rules and doctrines (in OV-6a) can be evaluated for interoperability with different capabilities. By purview of the information contained in OV-9 we have introduced the model repository as an important aspect of DoDAF system specification that enhances the DoDAF by making way for the M&S area.

DEVS modeled systems are inherently object oriented, and DoDAF at the OV stage does not have full expressiveness to be transformed to an executable model. In one of our other systems engineering approaches using System Entity Structure (SES) (see the appendix), we developed a hierarchical perspective representation that would enable DEVS to step in at various levels of resolutions. The three main perspectives are 1) component based, 2) Capability based, and 3) rule based. The DEVS bifurcated model continuity–based system requires all three perspectives to be available in order for the system model be deployable. As you can see in Table 4, the current DoDAF, if enhanced with the new OV documents, does make the DoDAF a DEVS-compliant system.

In the next section we shall demonstrate how one can construct a DEVS executable OV model from the enhanced DoDAF. We will also show the sample OV-8 and OV-9 documents and their construction process.

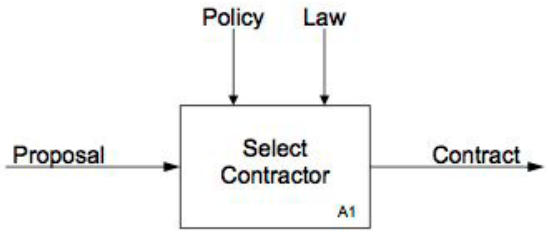


**Figure 7.** OV-5 diagram for "select contractor" in IDEF0 notation (from [31])

## 10. Example: Implementation of an Activity Component

Consider an activity as mentioned in Zinn [31, p. 65] described in IDEF0 format; see Figure 7. This activity is governed by the doctrines specified in OV-6a, IDEF3 format, which are described in [43]. Figure 7 is a sample OV-5 diagram for "select contractor," and Figure 8 is the OV-6a description in IDEF3 format where *X* represents an XOR split and *O* represents an OR split. These are the critical decision-making points that impact the outcome of the activity based on the previous step. It is at this point that timing needs to be specified so that "timeouts" can occur without leading to any ambiguity. Zinn acknowledged this problem in the process.

The information from these two figures is compiled manually to generate the pseudo-code in the following format. This manual process amounts to the integration of OV-5 and OV-6a into a single document. The pseudo-code is provided in Figure 9.

The graphical representation in Figure 7 is represented textually through the Popkin System Architect as shown in Figure 10. Consequently, Figure 9 and Figure 10 gives us the comprehensive information about the activity, its purpose, its input-output information through ICOM[5] lines, and pseudo-code for operational rules (as defined in OV-6a). Figures 7, 8, and 9 describe a general step approach to arrive at this pseudo-code, which is then utilized by an agent-based modeling software (e.g., SEAS) via Tactical Programming Language (TPL). Once pseudo-

5. In IDEF0 diagrams, inputs, controls, outputs, and mechanisms are collectively referred to as ICOM arrows.

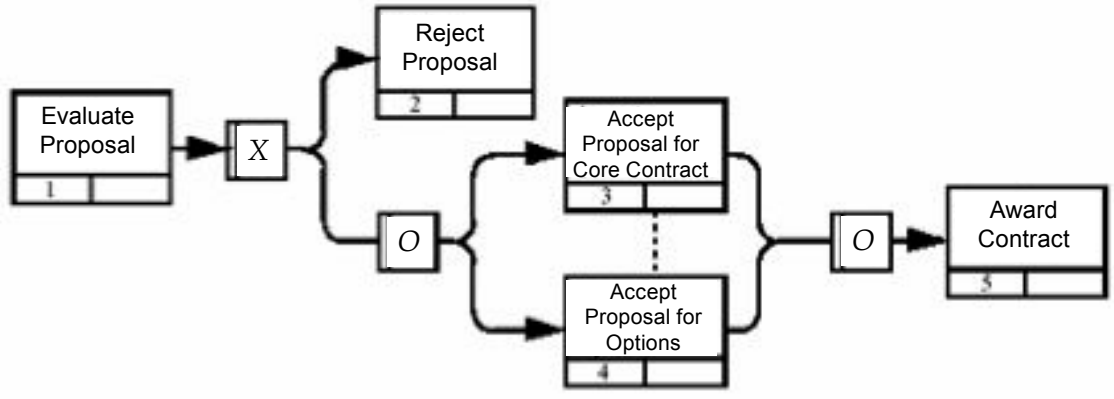


**Figure 8.** OV-6a diagram for "select contractor" in IDEF3 notation, (from [31])

**Activity 1: Select Contractor**

Description: The process used by the company to select the contractor for a new project

**Inputs:**
Proposal: contains the cost, schedule, and technical information as proposed by the contractor

**Outputs:**
Contract: the awarded contract

**Controls:**
Policy: Company contracting policy
Law: Federal, State and Local regulations

**Pseudo-code for Activity 1**

**Evaluate Proposal**

```
IF (cost > budget) THEN
   Reject Proposal
ELSE
   (Accept Proposal for Core Contract) OR
   (Accept Proposal for Options) OR
   ((Accept Proposal for Core Contract) AND (Accept
     Proposal for Options))
```

**Figure 9.** Pseudo-code as per Zinn's interpretation and integration procedure [31]

code has been made available, any software developer who is versed with TPL or any other language can interpret it. This process is then followed for the case study (for all the eleven activities) considered in Zinn [31]. Zinn brought forward the information expressed in graphical format in OV-5 diagrams and OV-6a doctrines in the form of psedo-codes that are realizable into software code. We utilize his efforts and demonstrate how this information can be used to feed the integrated DEVS methodology and development of OV-8 and OV-9.

### *10.1 Activity Taken from Zinn [24] as an Example*

Let us consider the same example that is described in Zinn[31]. Let the activity that is being modeled be defined as Activity 6: TCT - Determine target significance/urgency. There are about eleven activities that are being evaluated and pseudo-code provided in [31]. Figure 10 provides the activity model report as generated by Popkin System Architect.

This activity report is nothing but the interface descriptions for an activity in OV-5 diagram. It tells us from which activities Activity 6 receives input and to which activities it sends output. It also provides us with the information about the "control" interfaces that are needed to execute the doctrines and rules. Figure 11 depicts the IDEF3 model that implements the OV-6a doctrines and rules for Activity 6.

The pseudo-code for Activity 6 is provided in Figure 12, which is compiled manually from the information

**Operational Activity 6:**
**TCT - Determine target significance/ urgency (track)**
**[within OV-5 diagram "TCT - Level 1"]**

**Glossary Text:** Utilizing track data and other target information, C2 Warriors determine if the target/target set is threatening and/ or fleeting, and estimate target availability, i.e., how long the target will remain susceptible to attack.

From 2005 C2 Constellation 3.2.5.2 and CAOC-4.5.2.7

**ICOM line:** Air Track (J3.2)
**Output**: going to TCT-Validate target/target set (Target) as input
**Glossary Text:**

**ICOM line:** Current Intelligence - Dynamic Assessment/ Target Status
**Input**: coming from <offpage>
**Glossary Text:**

**ICOM line:** Current Intelligence - Target Classification
**Input**: coming from TCT-Define target/target set (Fix) as output
**Glossary Text:**

**ICOM line:** Current Intelligence - Target Identification
**Input**: coming from <offpage>
**Glossary Text:**

**ICOM line:** line: Doctrine, Policy, LOAC, SROE, ROE
**Control**: coming from <offpage>
**Glossary Text:**

**ICOM line:** line: Dynamic Target Nomination
**Output**: going to <offpage>
**Glossary Text:**

**ICOM line:** line: Dynamic Targeting Execution Direction and Guidance
**Control**: coming from <offpage>
**Glossary Text:**

**ICOM line:** JMSNSTAT
**Input**: coming from <offpage>
**Glossary Text:**

**ICOM line:** Land (Ground) Point/Track (J3.5)
**Output**: going to TCT-Validate target/target set (Target) as input
**Glossary Text:**

**ICOM line:** Reattack Recommendation
**Output**: going to TCT-Nominate engagement option (Target) as input
**Glossary Text:**

**ICOM line:** TRKREP
**Output**: going to TCT-Validate target/target set (Target) as input

**Figure 10.** Activity report model for Activity 6 generated through Popkin System Architect

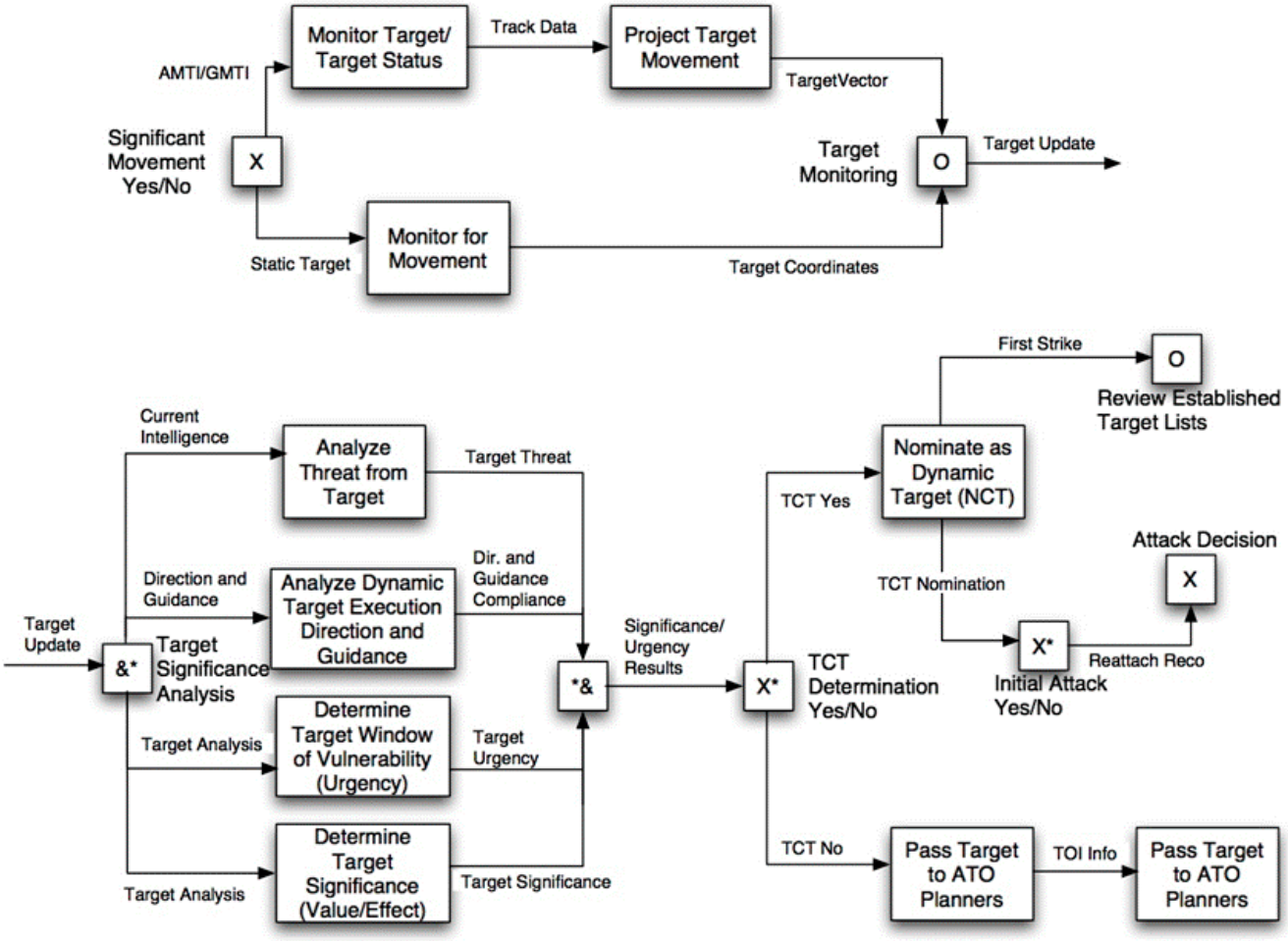


**Figure 11.** IDEF3 representation of Activity 6 ("conduct dynamic assessment of target" TCT 2005 Architecture, 2003: OV-6a) [31]

**IF** Significant Movement of target
**THEN** Monitor Target/Target Status
Project Target Movement
Target Vector = . . . .. ?
**ELSE** Monitor for Movement

Analyze Threat from Target (Is the target closing on Friendlies or Fleeing?)

Analyze Dynamic Targeting Ex Direction and Guidance (Does this agree with the commander's requirements?)

Determine target window of vulnerability (urgency)

Determine target significance – partly based on above findings

**IF** it is determined to be a TCT based on the above info
**THEN IF** this is the first strike attempt on this target
**THEN** Goto Activity 7 (Validate Target/Target set)
**ELSE** Goto Activity 8 (Nominate engagement option)
**ELSE** Pass target to ATO Planners
Monitor Target of Interest for Status Change

**Figure 12.** Pseudo-code for Activity 6 – based on IDEF3 diagram in Figure 11, taken from [31]

contained in OV-6a. For a complete description of Activity 6, refer to [31]. Briefly, the context of Activity 6 in TCT architecture is immediately after a target (or target set) is found and fixed. The upper half of Figure 11 shows an XOR junction that indicates only one path is taken. The resulting "target update" is then put through four simultaneous analyses indicated by the *AND* junction. This results (after integrated processing) in "Is the target time critical?" If it passes this TCT test it is again presented with a decision point: "Is the initial attack on the target?" The answer to this question results in two different modes of action, indicated by the XOR junction. Zinn acknowledges the fact that even though there is certain sequencing present, precise information about the rules defined are left to the imagination [31].

The next section demonstrates how the information in Figure 10 and Figure 12 is transformed into DEVS component modeling framework. It also shows how OV-8 and OV-9 gets populated. However, it must be

realized that an "operational node" hasn't been defined with respect to the current example. Consequently, we will assume an entity structure that will illustrate the concept.

### *10.2 DEVS Interpretation of Activity 6*

Based on the available information let us assume that dynamic target assessment happens at a particular node. Assume that Activity 6 and its sub-activities are all happening at TCT. Let us call this Operational Node 1, (with identification number O1). This will comprise our OV-2 diagram containing only one operational node executing all the eleven activities [31]. Again, a simple example has been considered to demonstrate the construction of the new OV document, namely OV-8 and OV-9.

Table 5 assigns identification numbers to various activities.

Based on the IDEF3 diagram (graphical information for OV-6) in Figure 11, and our constructed OV-2 in previous paragraph, we can construct our OV-8 document that lists activities and their *logical* interface information. We need such port information to be able to create components. Such logical port construction has been attempted in [41] where the focus was to create an SV executable model. Developing and specifying activity port interfaces at this level is a logical step toward SV interface design as tractability is ensured. The OV-8 document below does not address the performance issue at the OV level, and its refined structure is presented in [44]. Table 6 shows a sample OV-8 document.

Based on the information provide in Figure 11, we have constructed and identified the interfaces that are being used by different activities to communicate. However, we have not considered the information contained in Figure 10 that describes how Activity 6 communicates with the other ten activities. We did not explore the connectivity between other destination activities just to keep the example in the needed perspective. However, the procedure is essentially the same with more rows being added to Table 6. To give a glimpse of how these interconnected activities

**Table 5.** Activity-ID mapping for OV-8 and OV-9

| S. No. | Activity | Sub-activity | Internal Activity | ID |
|---|---|---|---|---|
| 1 | Activity 6 | Dynamic target assessment | | A6 |
| 2 | | Monitor target/ target status | | A6.1 |
| 3 | | Monitor for movement | | A6.2 |
| 4 | | Project target movement | | A6.3 |
| 5 | | Analyze threat from target | | A6.4 |
| 6 | | Analyze dynamic target execution/direction and guidance | | A6.5 |
| 7 | | Determine target window of vulnerability (urgency) | | A6.6 |
| 8 | | Determine target significance (value/effect) | | A6.7 |
| 9 | | Nominate as dynamic target (NCT) | | A6.8 |
| 10 | | Pass target to ATO parameters | | A6.9 |
| 11 | | Pass target to ATO planners | | A6.10 |
| 12 | | | Significant movement Yes/No | A6.11 |
| 13 | | | Target monitoring | A6.12 |
| 14 | | | Target significance analysis | A6.13 |
| 15 | | | Synthesize results | A6.14 |
| 16 | | | TCT determination Yes/No | A6.15 |
| 17 | | | Initial attack Yes/No | A6.16 |
| 18 | | | Review established target lists | A6.17 |
| 19 | | | Attack decision | A6.18 |

**Table 6.** Sample OV-8 document

| S. No. | Activity ID Component | Connection ID | Source Activity | Input Interface Name (Logical Port) | Message Description/OIEs | Container Operational Node | Source Document/ Diagram |
|---|---|---|---|---|---|---|---|
| 1 | A6 | | | | | O1 | |
| 2 | A6.1 | CA6.1 | A6.11 | inSigMovY | AMT/GMTI | O1 | Figure 12/OV-6b,c |
| 3 | A6.2 | CA6.2 | A6.11 | inSigMovN | StaticTarget | O1 | Figure 12/OV-6b,c |
| 4 | A6.3 | CA6.3 | A6.1 | inTrkData | TrackData | O1 | Figure 12/OV-6b,c |
| 5 | A6.4 | CA6.4 | A6.13 | inCurrInte | Current intelligence | O1 | Figure 12/OV-6b,c |
| 6 | A6.5 | CA6.5 | A6.13 | inDirGuid | Direction and guidance | O1 | Figure 12/OV-6b,c |
| 7 | A6.6 | CA6.6 | A6.13 | inTarAnaly | Target analysis | O1 | Figure 12/OV-6b,c |
| 8 | A6.7 | CA6.7 | A6.13 | inTarAnaly | Target analysis | O1 | Figure 12/OV-6b,c |
| 9 | A6.8 | CA6.8 | A6.14 | inTctYes | TCT Yes | O1 | Figure 12/OV-6b,c |
| 10 | A6.9 | CA6.9 | A6.14 | inTctNo | TCT No | O1 | Figure 12/OV-6b,c |
| 11 | A6.10 | CA6.10 | A6.9 | inToiInfo | TOI Info | O1 | Figure 12/OV-6b,c |
| 12 | A6.11 | CA6.11 | | inIsSigMov | Significant movement | O1 | Figure 12/OV-6b,c |
| 13 | A6.12 | CA6.121 | A6.2, | inTargCoord | Target coordinates | O1 | Figure 12/OV-6b,c |
| | | CA6.122 | A6.3 | inTargVec | Target vector | O1 | Figure 12/OV-6b,c |
| 14 | A6.13 | CA6.13 | A6.12 | inTarUpdate | Target update | O1 | Figure 12/OV-6b,c |
| 15 | A6.14 | CA6.141 | A6.4 | inTarThreat | Target threat | O1 | Figure 12/OV-6b,c |
| | | CA6.142 | A6.5 | inDGCompl | Direction guidance compliance | O1 | Figure 12/OV-6b,c |
| | | CA6.143 | A6.6 | inTarUrg | Target urgency | O1 | Figure 12/OV-6b,c |
| | | CA6.144 | A6.7 | inTarSig | Target significance | O1 | Figure 12/OV-6b,c |
| 16 | A6.15 | CA6.15 | A6.14 | inSigUrgRes | Significance/ urgency results | O1 | Figure 12/OV-6b,c |
| 17 | A6.16 | CA6.16 | A6.8 | inTctNom | TCT nomination | O1 | Figure 12/OV-6b,c |
| 18 | A6.17 | CA6.17 | A6.16 | inFirstStr | First strike | O1 | Figure 12/OV-6b,c |
| 19 | A6.18 | CA6.18 | A6.16 | inReAtkRec | Reattack recommendation | O1 | Figure 12/OV-6b,c |

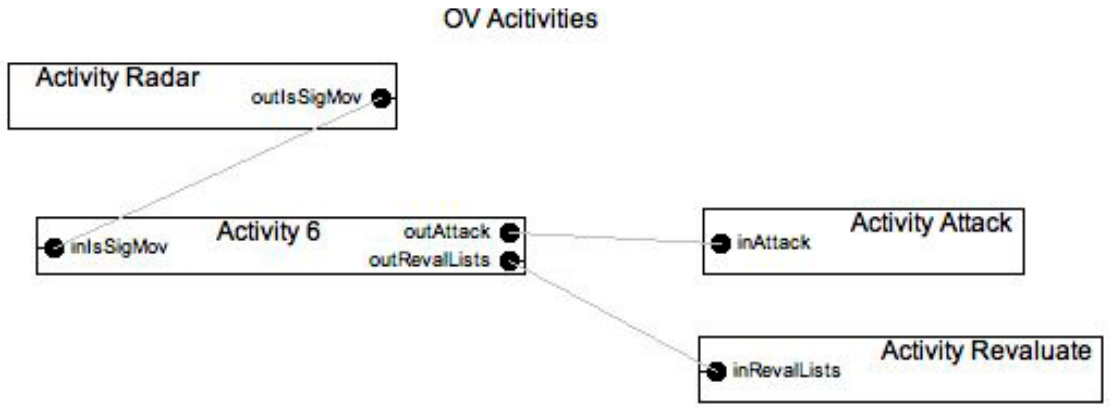


**Figure 13.** DEVS interrelationships of Activity 6 with other activities

(as components) will perform in tandem, notice the inports and outports of Activity 6 in Figure 13. The other activities shown in this figure do not have any resemblance to the actual example in [31]. They are just meant for understanding. To understand how Activity 6 works internally based on the different activities in Table 6, please see Figure 14.

The coupling relations shown in Figure 14 are generated in an automated manner from the data presented in Table 6. Columns 2, 3, 4, and 5 provide sufficient information to generate the following lines of code with simple string manipulations. Consequently, an automated generation of DEVS model is realizable. Hence the OV-8 document provides sufficient information to develop a skeleton DEVS model that can make its entry into the model repository. Let us name the model for Acitivity 6 MA6. The inner models are identified in the same predictable manner as MA6.1, MA6.2,…MA6.18.

```
ViewableAtomic a61 = new ViewableAtomic("A6.1");
add(a61);
ViewableAtomic a62 = new ViewableAtomic("A6.2");
add(a62);
...
ViewableAtomic a611 = new ViewableAtomic("A6.11");
add(a611);
...
a611.addOutport("outSigMovY");
a61.addInport("inSigMovY");
addCoupling(a611,"outSigMovY",a61,"inSigMovY");
a611.addOutport("outSigMovN");
a62.addInport("inSigMovN");
addCoupling(a611,"outSigMovN",a62, "inSigMovN");
...
```

The next task in line is the inclusion of pseudo-code that contains the doctrines and rules form OV-6a, described in Figure 12. Consider these four initial lines from Figure 12.

```
IF Significant Movement of target
THEN Monitor Target/Target Status
    Project Target Movement
    Target Vector = . . . .. ?
ELSE Monitor for Movement
```

This particular doctrine is to be implemented at A6.11; refer to Table 6. This has far-reaching advantages. By assigning doctrines and rules to specific activity components, we are ensuring that each rule is formally implemented and is synchronized with other rules that are "in operation" at that instant of time. In a sense, which rules are compatible and which can cause "deadlocks" can be determined by execution of the above Activity 6 DEVS model. The sample lines above are implemented in the *deltext*( ) function of component A6.11. The *deltint*( ) function defines the natural course of the activity.

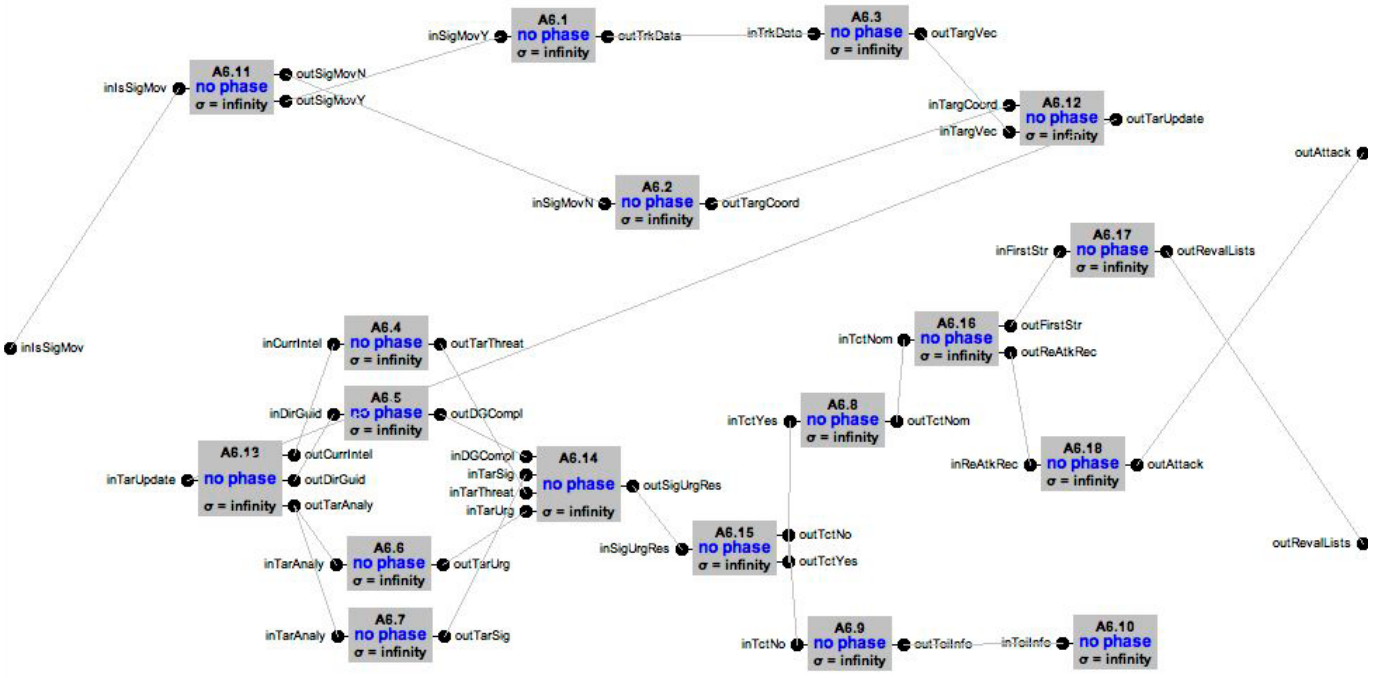


**Figure 14.** DEVS description of Activity 6 in relation to Table 6 activity components

```
public void deltext(double e, messagex){
...
   for(int i=0; i<x.length; i++){
      if(messageOnPort("inIsSigMov"){
         MessageTypeA msg
          = (MessageTypeA)x.getValOnPort
          (i, "inIsSigMov");
         If(msg.equals("yes"))
            holdIn(0, "yesSigMov");
         else
         if(msg.equals("no"))
            holdIn(0, "noSigMov"));
   }
...
}
public message out(){
...
   if(phaseIs("yesSigMov")){
      m.add(makeContent("outSigMovY",
       new entity("start")));
   if(phaseIs("noSigMov"))
      m.add(makeContent("outSigMovN",
       new entity("start")));
...
}
```

Similarly, all other activities will receive inputs from other source activities in their *deltext*( ) functions that will contain the logic for implementation of doctrines. For convenience purposes, the execution time of these doctrines is considered zero. Notice the *holdIn*( ) function in the code above. This is an important place where we can tune and implement the realistic time in issuing commands by human commanders; for example, in a situation where the system is "waiting" for a command from an authority figure and decision has to arrive until a "timeout" occurs. In addition, consider that the activity component is executing certain process with respect to its *deltint*( ) function and is in a certain "phase" waiting for any external input from other activities. In the situation of not receiving this input within allowable time window, timeouts can very effectively guide the simulation to its completion and prevent the wait-to-infinity problem.

The OR split problem pointed out by Zinn [31] in IDEF3 methodology has no effect in DEVS methodology. This problem is resolved by making the *&*, *X*, and *O* constructs in IDEF3 methodology "internal activity" components; see Table 6. Once they are componentized, timeouts can be implemented very easily that will completely eradicate this problem. These components are very well documented in DEVS *SimpArc* package version 3.0. This solution also puts the focus back on the system-logic implementation and test if the communication delays are significant enough that timeouts are occurring frequently.

Finally, the last task is the description of the OV-9 document. This document contains information about the activities happening inside an operational node and the way in which the sub-activities are mapped onto the components inside the operational node. For simplicity, we are working on the assumption that there is only one operational node, O1, in the example. As there is no information present on what its inner components are in [31], we will assume that there are, say, seven inner components that make up this node. Four of these seven components are associated with Activity 6 and the other three components are associated with some other activities, not considered for illustration purposes.

The defined components are essentially COTS components with defined behavior. They can even come from system view document SV-4. Consequently, each of them has its "models" for simulation purposes specified in DEVS formalism. These models are essentially open-source models available to the public through a common repository and are "standardized." Table 8 depicts the

**Table 7.** Inner components within operational nodes and their mapping with "standardized" DEVS models

| S. No. | Operational Node | Inner Component Entities | Component Name | Associated Models Added to Repository | Hierarchical Parent/Container | DEVS Model Type |
|---|---|---|---|---|---|---|
| 1 | O1 | OCE1 | TCT | ME1 | - | Digraph |
| 2 | | OCE1.1 | Radar tracking system | ME1.1 | ME1 | Atomic |
| 3 | | OCE1.2 | Significance analyzer | ME1.2 | ME1 | Atomic |
| 4 | | OCE1.3 | Urgency analyzer | ME1.3 | ME1 | Atomic |
| 5 | | OCE1.4 | Vigilance controller | ME1.4 | ME1 | Atomic |
| 6 | | OCE1.5 | Attack evaluator | ME1.5 | ME1 | Digraph |
| 7 | | OCE1.6 | Attack initiator | ME1.6 | ME1.5 | Atomic |
| 8 | | OCE1.7 | Attack terminator | ME1.6 | ME1.5 | Atomic |

**Table 8.** OV-9 description document mapping the entity component inside operational node O1 with the activity components defined in OV-8 with port interfaces

| S. No. | Operational Node | Inner Component Entities | Component Name | Activity Component | Activity Component Name | Interface Description |
|---|---|---|---|---|---|---|
| 1 | O1 | OCE1 | TCT | | | |
| | | OCE1.1 | Radar tracking system | A6.1 | Monitor target/target status | monTarE |
| | | | | A6.2 | Monitor for movement | monTarMovE |
| | | | | A6.3 | Project target movement | proTarMovE |
| | | | | A6.11 | Significant movement Yes/No | sigMovYesNoE |
| | | | | A6.12 | Target monitoring | tarMonE |
| | | | | A6.10 | Monitor target of interest for status change | monTarInterE |
| | | OCE1.2 | Significance analyzer | A6.13 | Target significance analysis | tarSigAnalyE |
| | | | | A6.4 | Analyze threat from target | analyThrTarE |
| | | | | A6.5 | Analyze dynamic target execution direction and guidance | analyEDGE |
| | | | | A6.7 | Determine target significance | detTarSigE |
| | | | | A6.14 | Synthesize results | syncE |
| | | OCE1.3 | Urgency analyzer | A6.6 | Determine target window of vulnerability | detWinVulE |
| | | OCE1.4 | Vigilance controller | A6.15 | TCT determination Yes/No | tctDetYesNoE |
| | | | | A6.8 | Nominate as dynamic target | nomDynTarE |
| | | | | A6.9 | Pass target to ATO planners | passTarAtoE |
| | | | | A6.16 | Initial attack Yes/No | initAtckYesNoE |
| | | | | A6.18 | Attack decision | atckDecE |
| | | | | A6.17 | Review established target lists | revEstTarListsE |
| | | OCE1.5 | Attack evaluator | A6.16 | Initial attack Yes/No | initAtckYesNoE |
| | | OCE1.6 | Attack initiator | | | |
| | | OCE1.7 | Attack terminator | | | |

information assumed for construction of OV-9. The inner components depicted in this table are only for illustration purposes.

Having Table 7 as available resources for OV-9, we have enough information to construct the activity-entity mapping. We identify and define port interfaces that need to be added to the entity component models so that they can be coupled to the activity components. Once the OV-9 document is in place, the added interface information is used to update the models defined during the construction of these two documents. We saw in construction of the OV-8 document that the resulting model is a stand-alone model that is capable of executing the simulation in "capability" mode, testing the OV-5 and OV-6 description of the system. A sample OV-9 document is shown in Table 8

The OV-9 document aids in bringing the systems perspective to the design and how the system's components initiate the designated activities. Assignment of an activity to appropriate component entity is a job of an experienced "designer," as per the definition of designer in the DoDAF document. This document ensures accountability: there is at least one component entity that is responsible for the execution of that particular activity. Notice that all

the activity components addressed in the example have been assigned at least one operational node inner component entity. After the creation of the OV-9 document, the interface information, in the last column, is used to update the corresponding activity and the entity models in the model repository that were created during the construction of OV-8. This is again an automated task with simple string manipulation as described earlier, during the construction of OV-8 models.

Hence, during the creation of OV-8 and OV-9 we have populated the model repository with activity models (MA6.1–MA6.18) and operational nodes' inner component models (ME1, ME1.1–ME1.6), and have created an interface between these two aspects of DoDAF design.

### *10.3 Synopsis*

Looking at Figure 6 in an activity component perspective, we have our defined inputs and outputs, and eventually the logical activity ports. In the example above, we have defined the interfaces of an activity that could be subjected to component coupling and testing. The coupling information can be integrated using the OV-3 document, as described in Table 3 and in more detail in Table 6. The timing information is added using the OV-6b and OV-6c diagrams as we have defined "components," the effects of which have been highlighted earlier in Sections 7 and 8. This information, along with the pseudo-code provided by Zinn, is integrated to develop the DEVS model of the activity in question. The pseudo-code is very well directed to the activity that is best responsible to execute those "rules." At this point the whole purpose of creating OV-8, the rule-activity mapping, is realized.

The OV-9 document deals with the mapping of the activity components with the entity components. Since Zinn [31] did not define internal components for any operational node, we assumed certain inner components and mapped the activities to these components. Having ensured accountability for each of the activities, another area that OV-9 contributes to is system design, reuse, and composability. We have available to us a document that contains information of the functionalities any particular component can perform or participate in collective functionality. Consider the situation when two or more inner components from systems perspective are thrown together to observe if the system is capable of performing "something." This allows us to experiment with different systems that claim to exhibit certain functionality. It allows us to test interoperability.

Hence, the resulting integrated information from OV-3, OV-2, and OV-6 is converted to the information in documents OV-8 and OV-9, with the addition of logical ports, dedicated to the M&S area that are focused toward operational views. Referring to Figure 4 again, the complete DEVS process life cycle, the area in the grey box is initiated at this stage, wherein we have XML (OV-8, OV-9) or any form of pseudo-code to define the operational view descriptions. Manual/automated design of DEVS models, based on Table 3 interpretation, and semiautomated model–test suite development based on Zeigler et al. [9] and Mak et al. [40] stems from the DEVS activity model description documents, viz., OV-8 and OV-9.

## 11. Future Work

The present work has made two contributions:

1) It provides a methodology to incorporate automated testing during the early stages of DoDAF design process.
2) Operational view activities can be considered as activity components with defined interfaces and port definitions.

The work related to the first contribution is already proved in projects that are being done currently at JITC [9, 40]. How it relates to the DoDAF design process is left for future considerations. The second contribution presents us with a perspective of considering functionality as "components" and the way in which different functionalities are dependent upon each other in order to implement any capability. This perspective enables us to think of an abstract component that is capable of information exchange and can provide services to other functional components. The abstract functional component can very well be a part of a service-oriented architecture.

Consider the situation when there is a capability (composed of various functionalities) that is a part of a certain DoDAF architecture and a model needs to be simulated in order to analyze the compatibility between these functionalities (through OV-6a). Also consider that the number of functionalities is quite large, and that these are complex enough to be at one simulation station. Further, there has already been a subset of functionalities (within this capability) that has been simulated by some other organization. The cost effective solution is to utilize the work already done by this organization and build the remaining functional components (through OV-8 and OV-9). The means by which an online simulation model can be realized is through a web services architecture implemented over the Internet. The common mode of exchange is through XML. Now, if these components are described in XML, and there is a definite interface

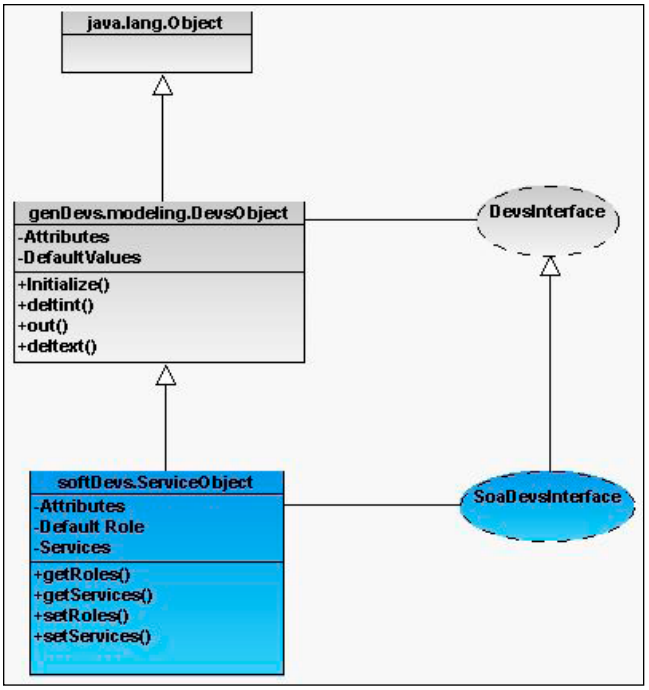


**Figure 15.** An SOA object capable of DEVS modeling

port for a particular service, creating a model for the capability can be effectively realized. This work is currently in progress at the ACIMS lab and will be reported in future publications.

The abstract component that is capable of being a part of DEVS modeling framework is shown in Figure 15. Its XML representation is shown in Figure 16. The point worth stressing here is that the XML code is generated automatically with the information coming from DEVS DTD and OV-8. What is required here is the addition of code for a "services" tag. Once implemented on SOA, the code with respect to the "services" tag can be exchanged through a SOAP message, and a DEVS model is made ready for simulation. In ongoing research [45], web services–based DEVS model interoperability has been achieved using XML as the communication medium with SOAP middleware. This technology is made possible by DEVS Modeling Language (DEVSML), which allows distribution of DEVS models in the form of XML documents to remote nodes where they can be coupled with local service components to compose a federation. The layered middleware architecture capability is shown in Figure 17.

At the top is the application layer that contains a model in DEVSJAVA or DEVSML. The second layer is the DEVSML layer itself, which provides seamless integration, composition, and dynamic scenario construction resulting in portable models in DEVSML

```
<?xml version="1.0" encoding="UTF-8"?>
<xml-body>
<model>
  <atomic>
    <name>Hello</name>
    <params>  </params>

  <construct>
      <args> </args>
      <ports>
        <inports>
          <inport>in</inport>
        </inports>
        <outports>
          <outport>out</outport>
        </outports>
      </ports>
    </construct>

    <initialize>
    </initialize>
       . .

    <services>

      <function>
        <access> public </access>
        <return> int </return>
       <inport> in </inport>
       <outport> out </outport>
        <fname> decrement() </fname>
        <logic>    </logic>
      </function>

    </services>
  </atomic>
</model>
</xml-body>
```

**Figure 16.** Automated XML snippet for an activity component created with OV-8 (port definitions); logic is added later or exchanged through SOA implementation

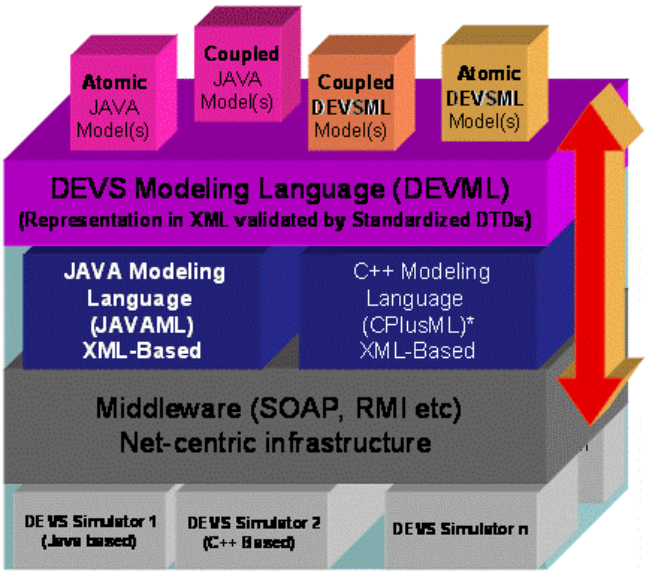


**Figure 17.** DEVSML layered architecture providing simulator transparency

that are complete in every respect. These DEVSML models can be ported to any remote location using the net-centric infrastructure and can be executed at any remote location.

The simulation engine is totally transparent to model execution over the net-centric infrastructure. The DEVSML model description files in XML contain metadata information about its compliance with various simulation "builds" or versions to provide true interoperability between various simulator engine implementations. Such run-time interoperability provides a great advantage when models from different repositories are used to compose models using DEVSML seamless integration capabilities. Making DoDAF a DEVS-compliant architecture will enable cross-platform integration and simulation opportunities.

## 12. Conclusions

Although the current DoDAF specification provides an extensive methodology for system architectural development, it is deficient in several related dimensions: its absence of integrated modeling and simulation support, especially for model continuity throughout the development process, and its lack of associated testing support. To overcome these deficiencies, we described an approach to support specification of DoDAF architectures within a development environment based on DEVS-based modeling and simulation. The result is an enhanced system life cycle development process that includes model continuity–based development and testing in an integral manner.

We have also introduced two new operational views, OV-8 and OV-9, to address the additional information that is needed to make the DoDAF M&S compatible. We have also demonstrated the process to create OV-8 and OV-9 from the existing operational views. OV-8 contains the information about the activity component structure and how different activities interface with each other using the specified logical interfaces. OV-9 contains information about the constituent components inside an operational node and its corresponding DEVS model structure along with their mapping to the activity components in OV-8. Together OV-8 and OV-9 provide a means to correlate activity components with accountable entities in an operational node using logical interfaces. It is after the transformation of OV-8 and OV-9 into DEVS models that rules assigned to specific activity or entity components make OV-8 and OV-9 serve their complete purpose. Automation using XML and simulation tuning are important concepts that can be well executed and performed under current DEVS technology. Composing simulations that are hierarchically stable and realizable is a step forward in evaluation of multi-resolutional architectures. Issues such as personnel management and task assignment at proper resolution of architectural execution are worth exploring further in future work. Capability to objectify parameters and visualize them with respect to the end goal in mind is critical for success. Current DEVS technology is well equipped to accomplish such a capability.

## 13. References


[1] DoDAF Working Group. DoD Architecture Framework Version 1.0 Vol. III: Deskbook, DoD, Aug 2003.

[2] DoD Instruction 5000.2. Operation of the Defense Acquisition System; 2003 May 12.

[3] Chairman, JCS Instruction 3170.01D. Joint Capabilities Integration and Development System; 2004 Mar 12.

[4] Chairman, JCS Instruction 6212.01D. Interoperability and Supportability of Information Technology and National Security Systems; 2006 Mar 8.

[5] Atkinson K. Modeling and Simulation Foundation for Capabilities Based Planning. Simulation Interoperability Workshop; Spring 2004.

[6] DoD Metadata Registry and Clearinghouse. [cited 2005 Jan 9]. Available from: http://diides.ncr.disa.mil/mdregHomePage/mdregHome.portal

[7] Dandashi F, Ang H, Bashioum C. Tailoring DoDAF to Support a Service Oriented Architecture. White paper; Mitre Corp.; Dec 2004.

[8] Zeigler BP, Mittal S. Enhancing DoDAF with DEVS-Based System Life-Cycle Process. IEEE International Conference on Systems, Man and Cybernetics; Oct 2005; Hawaii; 2005.

[9] Zeigler BP, Fulton D, Hammonds P, Nutaro J. Framework for M&S-Based System Development and Testing in Net-centric Environment. ITEA Journal; Nov 2005.

[10] Zeigler BP, Praehofer H, Kim TG. Theory of Modeling and Simulation. Academic Press; 2000.

[11] Hu X, Zeigler BP. 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; 2006.

[12] Lee J, Choi M, Jang J, Park Y, Jang J, Ko B. The Integrated Executable Architecture Model Development by Congruent Process, Methods and Tools. The Future of C2, 10th International Command and Control Research and Technology Sysmposium.

[13] Rosen JD, Parenti JL, Hamilton J. Cutting the Guardian Knot. In: The Domain of Interoperability in the US Department of Defense. Joint Command and Control Interoperability. Available from: http://www.eng.auburn.edu/users/hamilton/security/spawar

[14] DoD Architecture Framework Working Group 2004. DoD Architecture Framework Version 1.0, Volume 1. Definitions and Guidelines; 2004 Feb 9; Washington, D.C.

[15] Wagenhals LW, Haider S, Levis AH. Synthesizing Executable Models of Object Oriented Architectures. Workshop on Formal Methods Applied to Defense Systems; Jun 2002; Adelaide, Australia.

[16] Sarjoughian HS, Cellier FE, editors. Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies. New York: Spring-Verlag; 2001.

[17] Zeigler BP. DEVS Today: Recent Advances in Discrete Event-Based Information Technology. MASCOTS Conference 2003.
[18] ACIMS Software Development website. [cited Jan 2005]. Available from: http://www.acims.arizona.edu/SOFWARE/software.shtml
[19] Sarjoughian H, Zeigler B, Hall S. A Layered Modeling and Simulation Architecture for Agent-Based System Development. In: Proceedings of the IEEE. 2001; 89(2): 201–213.
[20] Cho Y, Zeigler BP, Sarjoughian HS. Design and Implementation of Distributed Real-Time DEVS/CORBA. IEEE Sys. Man. Cyber. Conference; Oct 2001; Tucson, AZ.
[21] Kim KH, Kang WS. A Web Services–based Distributed Simulation Architecture for Hierarchical DEVS Models. AIS Conference; Oct 2004; Jeju, Korea.
[22] Kim K, Seong Y, Kim T, Park K. Distributed Simulation of Hierarchical DEVS Models: Hierarchical Scheduling Locally and Time Warp Globally. Transactions of the Society for Computer Simulation International. 1996; 13(3): 135–154.
[23] Kim K, Kang W. CORBA-based, Multi-threaded Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One. LNCS. 2004; 3046: 167–176.
[24] Davis KP, Anderson AR. Improving the Composability of Department of Defense Models and Simulations; RAND Technical Report (Appendix D); 2003.
[25] Shelton GS, Case R, DiPalma LP, Nash D. Advanced Software Technologies for Protecting America. CrossTalk: Journal of Defense Software Engineering; May 2004. Available from: http://www.stsc.hill.af.mil/crosstalk/2004/05/0405shelton.html
[26] Mittal S, Gupta A, Mitra A, Zeigler B. Strengtheing OV-6a Semantics with Rule-Based Meta-models in DEVS/DoDAF-based Life-Cycle Architectures Development. IEEE-Information Reuse and Integration, Special Section on DoDAF; Sep 2006; Hawaii; 2006.
[27] Levis A, Wagenhals L. C4ISR Architectures I. Developing a Process for C4ISR Architecture Design. System Architectures Lab, C3I Center, MSN 4D2; Geroge Mason University; Jul 2000.
[28] Tolk A, Muguira JA. M&S with the Model Driven Architecture. I/ITSEC: Interservice/Industry Training, Simulation, and Education Conference; 2004.
[29] Tolk A, Solick S. Using the C4ISR Architecture Framework as a Tool to Facilitate V&V for Simulation Systems Within the Military Application Domain. Simulation Interoperability Workshop; Spring 2003.
[30] Adshead S, Kreitmair T, Tolk A. Definition of ALTBMD Architectures by Applying the C4ISR Architecture Framework. Fall Simulation Interoperability Workshop; Orlando, FL; 2001.
[31] Zinn AW. The Use of Integrated Architectures to Support Agent Based Simulation: An Initial Investigation.MS Thesis; AFIT/GSE/ENY/04-M01; Air Force Institute of Technology (AU), Wright-Patterson AFB, OH; Dec 2004
[32] Gonzales D, Moore L, Pernin C, Matonick D, Dreyer P. Assessing the Value of Information Superiority for Ground Forces – Proof of concept. RAND; 2001.
[33] Wagenhals L, Shin I, Kim D, Levis A. C4ISR Architectures: II. A Structured Analysis Approach for Architecture Design. Journal of Systems Engineering. 2000; 3(4).
[34] https://cao.hanscom.af.mil/af-cio.htm
[35] Mittal S, Zeigler BP, Hammonds P, Veena M. Network Simulation Environment for Evaluating and Benchmarking HLA/RTI Experiments. JITC Report; Fort Huachuca; Dec 2004.
[36] Mittal S, Zeigler BP. Dynamic Simulation Control with Queue Visualization. SCSC'05: Summer Computer Simulation Conference; Jul 2005; Philadelphia, PA.
[37] Hu X, Zeigler BP, Mittal S. Dynamic Configuration in DEVS Component-Based Modeling and Simulation. SIMULATION: Transactions of the Society of Modeling and Simulation International. Nov 2003.
[38] Mittal S, Zeigler BP. Modeling/Simulation Architectures for Autonomous Computing. Autonomic Computing Workshop: The Next Era of Computing; Jan 2003.
[39] Curbera F, Duftler M, Khalaf R, Nagy W, Mukhi N, Weerawarana S. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing. 2002; 6(2 Mar-Apr): 86–93.
[40] Mak E. Automating Testing Using XML and DEVS. University of Arizona; Spring 2006. Available from: http://www.acims.arizona.edu/PUBLICATIONS/publications.shtml#MSPHDReports
[41] Kobryn C, Sibbald C. Modeling DoDAF Compliant Architectures: The Telelogic Approach for complying with DoD Architectural Framework. Telelogic White Paper; Oct 2004.
[42] Dickerson C, Soules S. Using Architecture Analysis for Mission Capability Acquisition. 2002. Available from: http://www.dodccrp.org/events/2002/CCRTS_Monterey/Tracks/pdf/123.pdf#search=%22Dickerson%20Soules%20Mission%22
[43] Integration Definition Methods. IDEF3 Process Description Capture Method. Available from: http://www.idef.com/IDEF3.html
[44] Mittal S, Mak E, Nutaro J. DEVS-Based Dynamic Simulation Control and Reconfiguration in Enhanced DoDAF Design Process. JDMS To be published 2006; 3(4).
[45] Mittal S, Martin JLR. DEVS Modeling Language (DEVSML). Available from: http://devsml.sourceforge.net

## Appendix

### *System Entity Structure (SES)*

The SES formalism 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 any entity concerns how an entity may be broken down into subentities. Coupling specifications tell us how different subentities can be coupled together to reconstitute an entity. The taxonomic relationship concerns admissible variants of an entity. It also provides a formal framework for representing the family of possible structures. From a design point of view, SES represents the design space with various possible design configurations. Thus, the process of design/analysis is to prune SES—in other words, to search the best design configuration. For complex systems, the number of the combination of different configurations is very large. Thus, it is desirable to be able to emulate SES and automatically search the best design configuration. For a detailed description on SES see [10].

**Table 9.** DoDAF-DEVS extended translation table focusing on SV, TV

| DoDAF Elements | | | UML Elements | DEVS Elements (Generated Using XML) | |
|---|---|---|---|---|---|
| | Name | Description | | | |
| Systems View | SV-4 | System functional description | • Use-case description<br>• Activity sequencing diagrams | • Hierarchical functional components organization | DEVS model repository |
| | SV-5 | System functional traceability matrix (based on OV-5) | | • Coupling info refinement | |
| | SV-10 | System state description and event trace (based on OV-6) | • Sequence diagrams<br>• State charts | • DEVS atomic model transition functions refinement | |
| | SV-6 | System data exchange matrix | | • Input-output pair refinement | |
| | SV-1 | System interface description (based on OV-2) | • Composite structure diagram | • Port assignment refinement<br>• Entity refinement | |
| | SV-2 | System communication description | • Deployment diagrams | • Coupling info refinement (hierarchical management) | |
| | SV-7 | System performance parameters matrix | | • Experimental frame | DEVS system test suite |
| | SV-3 | System-systems matrix | | • Hierarchical model organization<br>• Entity refinement | DEVS model repository |
| | SV-11 | Physical schema | • Class diagrams | • Hierarchical model organization | |
| Technical View | TV-1 | Current standards | • Timing response | • Basic DEVS model for COTS component | |
| | TV-2 | Future standards | | • Improved DEVS model for desired functionality | |

## Acknowledgments

I would like to thank Prof. Bernard P. Zeigler for his expert guidance, resources, and concept validation. I would also like to thank Jerry M. Couretas of Lockheed Martin Corp. for invaluable discussions and reference material to continue research. Further, I would like to thank the *JDMS* reviewers who provided valuable suggestions to enhance the content of the paper.

## Author Biography

**Saurabh Mittal** *is a research engineer at Arizona Center of Modeling & Simulation (ACIMS) at University of Arizona. He is also a Ph.D. candidate in Electrical & Computer Engineering (ECE) at the University of Arizona. He is a recipient of JITC's highest civilian contractor award 'Golden Eagle.' He holds an M.S in ECE from the University of Arizona. His research interests include DEVS based integrated design methodologies, DoDAF, modeling languages, automated testing and design of software systems.*