

## Dynamic Testing & Evaluation (T&E) formalism

**Johnny Garcia**  
**SimIS, Inc.**  
**200 high Street suite 402**  
**Portsmouth, VA 23704**  
**Johnny@simistech.com**

**Chris Blancett**  
**SimIS Inc.**  
**chris.blancett@simistech.com**

### **Keywords:**

**DODAF, Executable Architectures, Discrete Event System Specification (DEVS), Core Architecture Data Model (CADM), Open Standards**

***Abstract:** Testing and evaluation of architecture remains an issue in the DoDAF community. This paper describes a process for ingesting and storing existing architectures in a standardized format, allowing all or specific artifacts of interest to be reconstructed into an executable format where testing and evaluation could be performed within a simulation environment. The process uses existing DOD standards, specifically the Core Architecture Data Model (CADM). The CADM serves as common approach for organizing, portraying, and relating architecture elements. It is used as a logical and physical data model to construct databases capable of storing architectures. The process assumes the creation of CADM-compliant DoDAF architecture, possibly exported in a format such as CADM XML from a COTS DoDAF modeling tool. Once architectures are ingested and transferred to a data repository, information about DoDAF views, their interrelated elements, and time sequencing information can be utilized to generate corresponding executable models for testing and evaluation. The simulation models are derived from information contained in the common data repository. If a mapping between DoDAF and formalism such as DEVS is established, analysts familiar with DoDAF could generate corresponding simulation models dynamically.*

### **Background**

The development of software and knowledge-based systems has simply not matured to the point where it can be characterized as following a discipline yet alone be considered equivalent to engineering methodologies in other technical domains. Programming could be viewed as the moral equivalent of the construction of an artifact (e.g., building, vehicle, etc) for disciplines like civil or aerospace engineering that occurs once design and engineering analysis has been completed. What software “engineering” lacks is a specification methodology that can guide “construction” or programming per se. Most attempts by Computer Scientists have evolved from knowing how to program or code: pseudo code, specification languages, prototyping, etc.

The entire discipline of architecture development needs a common basis for specifying the design of the target system, one that can be shown to the stakeholders of that architecture (to include purchasing agency, users, testers, and etc.) and which they can understand without significant training. To draw an analogy, a building designer (AKA architect) creates an architectural drawing which becomes an artifact for discussing that design with everyone involved in the project and evolving it over time. Once that architectural drawing is finalized, required engineering details and specifications are worked out and eventually provided to construction specialists who actually build the desired structure. Other development efforts follow a similar approach, e.g., aircraft; ship building, integrated circuits, civil engineering projects, etc. In software we have not achieved a clean division of responsibilities between design and construction and more importantly fail to achieve stakeholder buy-in, not because the system concept is too

complicated, but because current software design artifacts just do not lend themselves to obtaining feedback and discussion of the system to be built until it is actually built.

System software needs architectural frameworks and standards for specifying them which can meet these needs. The Department of Defense has tackled this challenge through its Department of Defense Architecture Framework (DoDAF). While DoDAF may not offer the ultimate common solution for Computer Science or Software Engineering, it serves as a useful artifact for examining how such a framework could be used to support the development of better software that implements the systems needed to support a knowledge-rich business or decision making environment. One specific need is to insure any architectural specification can provide more than a static representation of the system, that it can somehow be simulated or executed as a way to evaluate system concepts and dialog amongst designers as well as with the other stakeholders. Today this is what other designers or “architects” do, they create design artifacts which can be shown to stakeholders using 3-D computer visualizations or dynamic simulations that demonstrate how the end product will look and work.

Research on architectural specification methodologies for software objects, particularly large ones intended to implement complex knowledge-based systems, needs to focus on approaches that will allow dialog during design and support evaluation of that design before it is actually implemented in code through simulations. Executable architectures are one approach that shows promise in meeting this goal but require additional exploration and evaluation.

## **Department of Defense Architecture Framework (DODAF)**

DoDAF defines standard products to capture specific architectural views. DoDAF products are those graphical, textual, and tabular items that are developed in the course of building a given architecture description that describes characteristics pertinent to the architecture's purpose. The current DoDAF prescribes products that provide "static" representations of information for various views. These static products, while capturing enormous amounts of information about the operational, system, and technical architectures, fail to provide a good vehicle for conducting detailed dynamic "behavioral" analysis of how the systems are supposed to interact with each other.

Before you can use executable architecture descriptions for any analysis purposes you must start with an architecture that is integrated, unambiguous, and consistent, then define what an integrated architecture is. A single architecture description is defined to be an integrated architecture when products and their constituent architecture data elements are developed such that architecture data elements defined in one view are aligned with architecture data elements referenced in another view [DoDAF WG Vol.I, 2003]. By alignment we mean that there is a set of symmetric operational and systems architecture elements that have similar meanings, associations/ relationships, properties, and characteristics. An integrated architecture can be defined among multiple architectures when similar or related single architectures, each based on the same set of DoDAF integrated products and constituent aligned architecture elements can be combined for further development and analysis purposes. A final and yet more relevant definition to the research from a memo published by Deputy Secretary of Defense, “an integrated architecture has been defined as an architecture consisting of different views or perspectives (operational view, system view, and technical view) that facilitates integration and promotes interoperability across Family-of-Systems/System-of-Systems and compatibility among related mission area architectures”. Integrated architectures usually have associated with them a time frame, whether by specific years (e.g., 2005-2010) or by designations such as “as-is”, “to-be”, “transitional”, “objective”, “epoch”, etc. In all cases, this reduces to either inventories of current capability (“as-is”) or blue-prints of future capability (“to-be”) based on some future need or objective.

Domain experts, program managers, and decision-makers need to be able to analyze these architectures to locate, identify, and resolve definitions, properties, facts, constraints, inferences, and issues both within and across architectural boundaries that are redundant, conflicting, missing, and/or obsolete. The analysis must also be able to determine the effect and impact of change (“what if”) when something is redefined. In most “as-is” architectures, details about activities, nodes, roles, systems, etc. are fully known and architectures analysis can be readily accomplished.

Integrated architecture elements are keys to the creation of executable architectures. “Executable architecture” refers to the use of dynamic simulation software to evaluate architecture models. These executable architectures differ from the typical simulations because they are often generated directly from the architecture models via a semi-automated or automated process. Several purposes can be achieved with these specialized tools.

- The architecture model itself can be verified for internal self-consistency.

- Operational concepts can be simulated, observed dynamically, verified and refined.
- Operational plans can be examined and assessed.
- Tradeoffs between systems can be assessed.
- Architecture measures can be evaluated, which can support cost-benefit analyses and quantitative acquisition decisions.

There are some key factors in the process of constructing and using executable architectures. First, the aspect of automated or semi-automated generation directly from the architecture models is not simply for convenience. Rather, the driving factor is the accuracy of the executable model, in terms of consistency with the existing architecture models. Many typical simulation efforts diverge from the actual architecture models over time, leading to either the architecture being ignored in favor of the implemented design within the simulation, or, the simulation falls into disuse, as it is not able to keep up with the pace of the architecture modifications. There are currently no standards for the format or process for constructing executable architectures. Research has been done, but additional research is still required.

Most executable models assume a distributed, message-passing paradigm for the architecture operations, which is very applicable in most of the situations encountered in current practice. However, the architecture data elements and the attributes required to construct executable models are specified in the DoD Architecture Framework (DoDAF) products [DoDAF WG Vol.II, 2003] [1], but still no standard process exist. It is also important to make the most of the executable architecture concept. The process by which these types of tools are applied must be integral to the overall systems engineering process. In other words, the development process must be configured to rely upon the results of the executable efforts for validation and refinement. Efforts to construct executable architectures for their own sake have generally not been beneficial to their programs. Executable architectures have immediate implications for process improvement, but also directly support the investment decision process by providing realistic and repeatable cost-benefit analyses. Executable architectures must address not only the performance and effectiveness of the capability represented in the architecture; they must also address the dynamic cost of that capability as it operates to accomplish its mission.

## **Executable Architecture Analysis Modeling Method Approach**

Key elements of the EAAM approach are (1) analyze the use of executable architecture (2) integrate a combat model, a process model, and a communications model to represent the primary components of architectures and (3) implement these models in a simulation environment. The combat model provides the operational context for a system. The process model represents the mission threads that must be executed using the system to accomplish the mission. The communications model represents the network in which the data sent or received by the system must travel. The proposed method applies executable architectures to support verification analysis of a system or capability within one common operational perspective. A major issue is not only how to accurately document capabilities using architectures, but also how to conduct a useful analysis of these capabilities to determine performance and effectiveness of the systems providing the capability in question. This analysis is currently limited to a static analysis where one can examine the static architecture framework products to examine connectivity and other routine requirements. The current architecture frameworks provide no clear means to examine these systems in a dynamic fashion as they function in their operational environment. A successful solution must start with current situational awareness data provided by a Common Operational Picture (COP). These capabilities can support this research with measurable and repeatable environments to virtually reproduce the environment and play back segments of events to support verification analysis of conditions with replaceable components to simulate the system or system of systems.

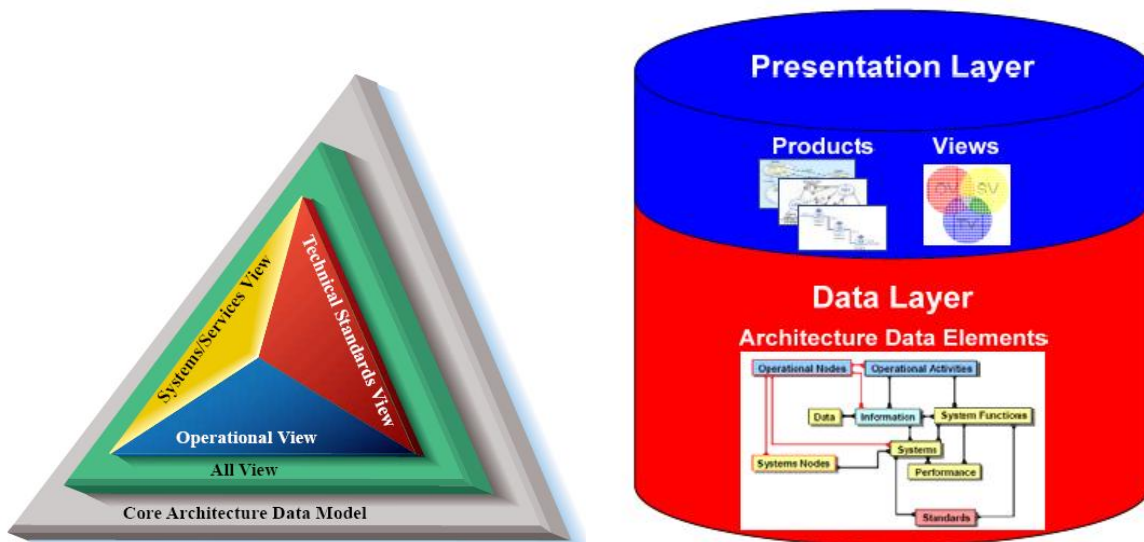
Broader impacts of the research will provide contributions that allow the advancement of executable architecture techniques for testing and experimenting with proposed technical and operational architectures. These experiments will help uncover the relationships and processes in the operational, systems and network architectures in simulated environments and provide a method for optimizing the true cost of the architecture. Similar related contributions may also be applied to training and creation of model simulations and architecture models. These contributions may enhance our scientific and technological understanding of architecture model validation techniques for interoperable solutions. The architecture verification modeling approach for executable architecture consists of the following steps:

1. Starting with the operation architecture descriptions, we formulate the activity models for the operational scenarios within the simulation.

2. After validating the activity models with subject matter experts, the activities are mapped to existing systems.
3. Next is the development of the logical deployment architecture. The logical architecture may be presented at several levels of detail. Physical or network architecture may also be identified as appropriate. Sequence diagrams may be used to describe interaction between services or systems in the architecture.
4. The logical data model provides useful information on data flow. The system data exchange matrix and data schema will deliver further estimates on the data sizes and data transmission latency.
5. Finally, all of the above architecture information is combined to construct the executable architecture model. The executable model is constructed based on the activity model, with additional details for data flow, sequencing and timing from the process model. Discrete event simulation techniques will be used to implement these methods. The method will validate the architecture logic and will provide quantitative decision making information for the end-to-end process. The results of the executable architecture analysis will produce performance parameters.

### T&E formalism using Core Architecture Data Model (CADM)

Most architecture is created using graphical languages like UML. In order to make graphical objects executable, they need to be represented in a machine-readable format that describes the architecture just as like the graphical diagrams. Luckily, the DoDAF is accompanied by a data model capable known as the Core Architecture Data Model (CADM) that is capable of describing the architectures and the relationships between the components within the architecture. The CADM is a DoDAF standard and the entities and relationships defined in the CADM should be constant and followed by all who create CADM-compliant architectures. The CADM is a logical data model and can serve as the design for a database. CADM is supposed to capture the views, their relationships, and timing information.



[ DoDAF 1.5 Volume II ]

If architecture was stored in a database that followed the CADM, not only would the architecture be stored in a machine-readable format, a user should know the tables that describe each of the components in the architecture as well as their relationships. Many of the relationships defined between the views are established with primary key/foreign key relationships in the CADM database. Again, being able to derived models detailed enough to be executable depends on the details of the initial architecture. The CADM is very large and seems detailed enough to handle the information needed for simulation. COTS tools are capable of graphically creating architecture views via drag and drop methods and later exporting the architecture into standard formats such as CADM XML.

```

<ORG_CLASS_CD>U</ORG_CLASS_CD>
<ORG_CAT_CD>B</ORG_CAT_CD>
<ORG_ENTRPRS_TY_CD>1</ORG_ENTRPRS_TY_CD>
</ORG>
</ORG_TBL>
- <SYS_ASSOC_TBL>
- <SYS_ASSOC>
  <SYS_ID>22411522868810677</SYS_ID>
  <ORD_SYS_ID>82891190613854700</ORD_SYS_ID>
  <SUB_SYS_ID>18071312522628445</SUB_SYS_ID>
  <SYS_TY_CD>6</SYS_TY_CD>
  <SYS_NM>Strategic Planning (carrying Interface Contract-St</SYS_NM>
</SYS_ASSOC>
- <SYS_ASSOC>
  <SYS_ID>78665304186038675</SYS_ID>
  <ORD_SYS_ID>88579720663958625</ORD_SYS_ID>
  <SUB_SYS_ID>89644081396713041</SUB_SYS_ID>
  <SYS_TY_CD>6</SYS_TY_CD>
  <SYS_NM>Commodity Order Management (carrying Interface Cus</SYS_NM>
</SYS_ASSOC>
- <SYS_ASSOC>
  <SYS_ID>50636831491898125</SYS_ID>
  <ORD_SYS_ID>81655242442663275</ORD_SYS_ID>
  <SUB_SYS_ID>58222141465223280</SUB_SYS_ID>
  <SYS_TY_CD>6</SYS_TY_CD>
  <SYS_NM>Payment Management (carrying Interface Payment-Sup</SYS_NM>
</SYS_ASSOC>
- <SYS_ASSOC>
  <SYS_ID>46066243280829170</SYS_ID>
  <ORD_SYS_ID>18071312522628445</ORD_SYS_ID>
  <SUB_SYS_ID>88579720663958625</SUB_SYS_ID>
  <SYS_TY_CD>6</SYS_TY_CD>
  <SYS_NM>Contract Management (carrying Interface Contract-C</SYS_NM>
</SYS_ASSOC>
- <SYS_ASSOC>
  <SYS_ID>87135929743078489</SYS_ID>
  <ORD_SYS_ID>18071312522628445</ORD_SYS_ID>
  <SUB_SYS_ID>89644081396713041</SUB_SYS_ID>
  <SYS_TY_CD>6</SYS_TY_CD>
  <SYS_NM>Contract Management (carrying Interface Contract-C</SYS_NM>
</SYS_ASSOC>
- <SYS_ASSOC>
  <SYS_ID>12317622474952317</SYS_ID>

```

### [ ProVition CADM XML export ]

The CADM XML could be parsed directly and the corresponding lower-level DEVS models could be generated, but parsing the XML would be more difficult than querying a database using SQL. A parser would need to be created to read the XML file containing the architecture and place each XML tags into their corresponding database tables. One simple way of doing this is to name the database tables the same as their respective XML tags, but one could always use a lookup list that maps tags to tables.

## Amalgamation of EAAM research and Discrete Event System Specification (DEVS)

Now that architecture can be stored and queried, simulation models need to be built and executed. Models capable of being simulated need to be derived from the architecture stored in the CADM database. Existing simulation formalism capable of hosting this embedding is the Discrete Event System Specification (DEVS). The DEVS formalism is capable of handling hierarchically structured systems and supports coupling and composability of atomic models. DEVS is well adapted to timed simulation and is capable of generating abundant quantitative data via simulation. The structure of DEVS is defined below:

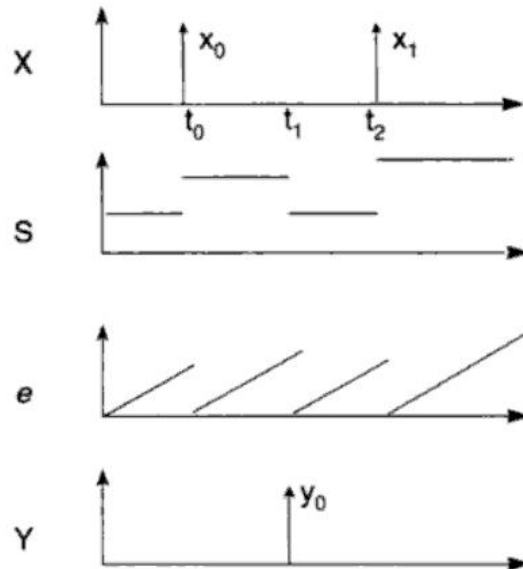
$$M = \{ X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \}$$

- $X$ : set of inputs values
- $S$ : set of states
- $Y$ : set of output values
- $\delta_{int}$ :  $S \rightarrow S$  internal transition function
- $\delta_{ext}$ :  $Q \times X \rightarrow S$  external transition function
- $\lambda$ :  $S \rightarrow Y$  output function

- $ta: S \rightarrow R$  time advance function

Simulating a DEVS model with simulators will produce the following four major trajectories. The figure shown below shows the following information displayed over time:

- Inputs
- States assumed and duration
- Time elapsed since last transition
- Outputs



[DEVS trajectories “Theory of Modeling and Simulation” Zeigler, Praehofer, Kim]

The information contained in these trajectories could play a critical role in determining whether or not a model is behaving as expected. This quantitative data could also play help compare architectures to test equivalence or test their performance after they are integrated.

If enough information is defined in the architecture, such as timing and sequencing information, DEVS models could be derived from the architecture and simulated to produce outputs that analysts can study. The derived models are dependent on the detail on the architecture. The information below describes how elements in DoDAF map to elements specified in DEVS. This DoDAF front-end/DEVS back-end style allows designers to remain familiar with their DoDAF abstractions while also being able to evaluate the architectures using simulation-based execution [2].

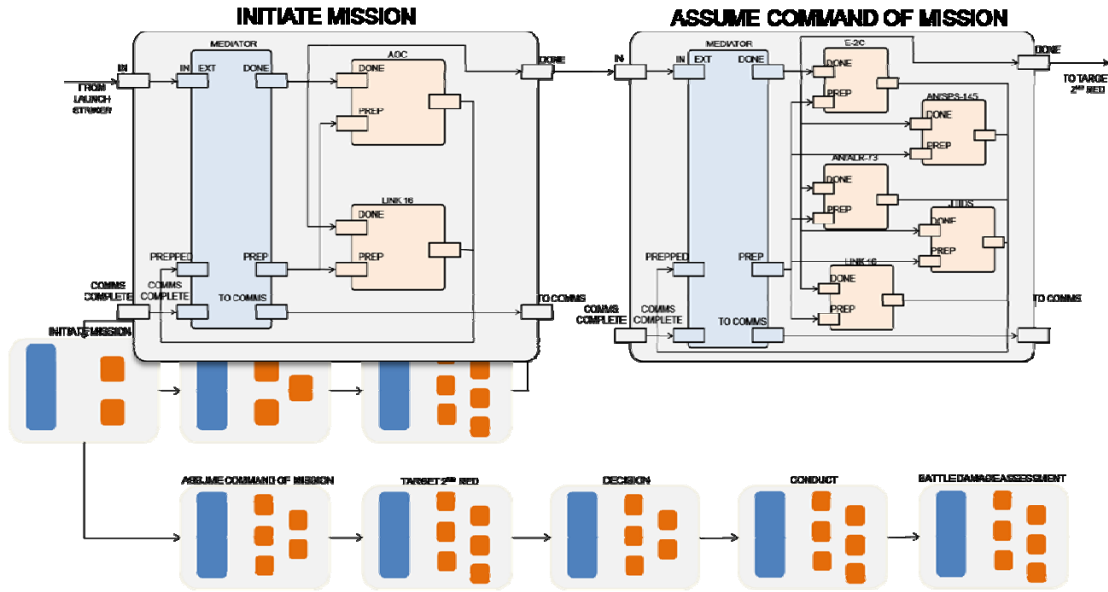
The following is the DoDAF-to-DEVS mapping

- OV-1 High level structure (both atomic and composite models)
- OV-5 I/O pairs, ports and port identification
- OV-6 time-advance function, internal transition function
- OV-2 Coupling information, composite digraphs, Hierarchical component organization
- OV-3 I/O transaction pairs, messages
- OV-4 Hierarchical structure
  
- SV-5 Coupling information
- SV-10 transition functions
- SV-6 I/O pairs, messages
- SV-1 Ports
- SV-2 Coupling

All of the mappings listed above can be obtained by querying the CADM database and dynamically create the appropriate DEVS elements programmatically while querying. For example, one first may want to query database tables that describe the high level structure of the architecture and create the appropriate atomic or composite DEVS models, then secondly query the database tables describing the relationships between views and create the appropriate model couplings between the DEVS models

There are a number of ways to build the DEVS models based on the information in the CADM and recent research has only experimented with one way of constructing and coupling atomic and composite DEVS models in order to make sure every detail is cover when executing the architecture. For instance, when a particular operational process has been reached in the OV-5, every operational node, and all systems that support each operational node must be executed. One way is to find the most dependent views in the DoDAF and couple all required help nodes into a coupled DEVS model. For example, one way would be to lump all nodes required for a particular OV-5 process into a couple OV-5 DEVS model. When it comes time to execute the process, perhaps determined by a temporal view such as OV-6, all the operational nodes, or underlying systems will be executed as well.

The atomic and composite models in the case of EAAM are the nodes contained in the DoDAF views, specifically the OV-2, SV-1, SV-2, etc... Upon import, the EAAM constructs high-level composite operational processes of the OV-5. For instance, if a particular OV-5 process required certain operational nodes specified in an OV-2 and physical systems specified on an SV-1 in order to function, they required helper nodes would serves as atomic DEVS models lumped together to form a composite OV-5 DEVS model. During simulation, if it is time for an OV-5 process to execute, it in turn executes its associate nodes. All along, the DEVS specification logs trajectories including time-stamped sequences of inputs received and outputs produced by any atomic or composite model. During simulation, these results are actually relayed to the web-clients witnessing the simulation. The figure below shows two coupled OV-5 processes with their required operational nodes and system nodes contained within them. The EAAM environment also includes mediator nodes that controls inputs flowing into the helper nodes, interprets their outputs, and determines the next course of action.

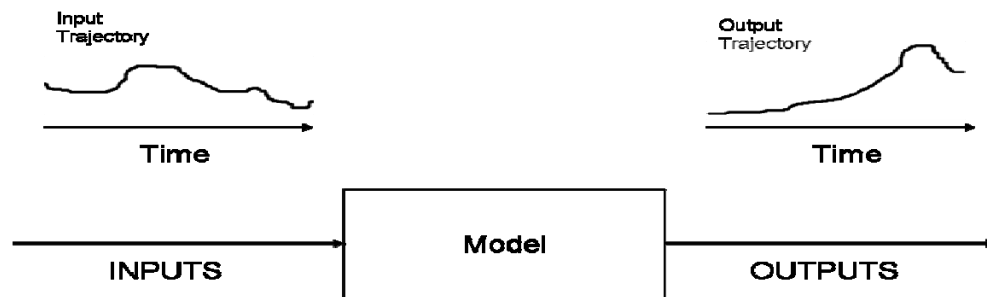


Of course, there will need to be other DEVS models coupled into the composite node to mediate the information flowing into and out of the process. A mediator node is assigned to any composite DEVS model that contains atomic nodes within it. For an OV-5 process, the mediator node serves as the brain of the process. As depicted above, external inputs of the OV-5 process must first flow through mediator nodes before flowing to the helper nodes. For example, a particular incoming request may not require all the helper to be used. Also outputs produced by helper nodes flow into the mediator node for interpretation. For instance, a helper node might output that it is almost done, that it needs more resources (if it is a system requiring operators), that it requires a network transmission and that the EAAM needs to

simulate a transmission on its network model, or that it is finished a particular job. It is the responsibility of the mediator to react accordingly and choose whether or not to proceed to the network model to simulate a transmission or exit the OV-5. Each high-level OV-5 process is coupled to the communications model if it is needed during its execution. The DEVJSJAVA software allows models to send messages to one another via their couplings. Each OV-5 is coupled to the communications model, and messages stating the start node, destination node, type of transmission, etc... are sent to the communications model by OV-5 processes.

These mediator nodes would serve as the brain of the OV-5 process. Of course, this type of DEVS model generation assumes that the OV-5 and OV-6c are the main drivers of the simulation. Again, there could be many ways to generating the simulation models from the CADM database. DEVS is a general and very powerful formalism capable of encompassing most problems arising from making architecture executable.

Since DEVS specifies that each model has an output trajectory over time, these could produced and graphically displayed for analysts to study during or after the simulation.

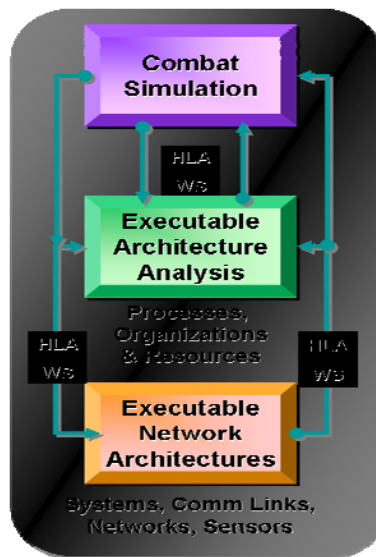


[ Quantitative I/O data via simulation ]

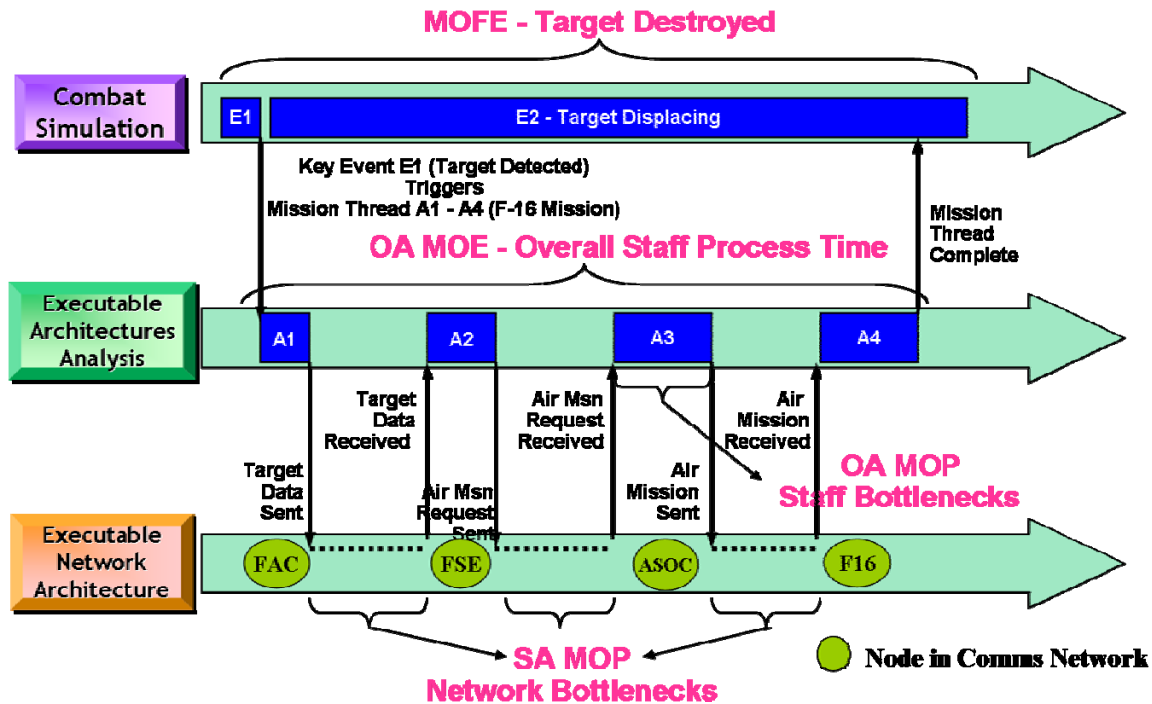
The model above could a particular operational node, a system or a system operator, or the entire operational process lumped together as a whole.

### Model Integration, Testing and Analysis

The DEVS formalism is general enough to integrate with other models to provide more in-depth simulation leading to more in-depth results and analysis. Work has been done in the field of executable architectures that involves model integration between executables architectures, combat models, and network communication models via web service interfaces.



Interfaces between the models allow seamless flow of execution throughout the federated execution.



EAAM Will invoke the appropriate model depending on where execution is occurring in the architecture. For instance, a particular OV-5 process may require the transmission of message over a network. When this step is reached in the architecture, the couple execution environment would invoke the appropriate model and pass the necessary information to it, for example, the start and destination nodes, the capacity of the network, or whether TCP or UDP is used. Each DEVS model requiring the network model could simply create a coupling and some ports to allow integration. The DEVS model representing the network model could contain internal logic that utilizes the network model and passes it the right information. For example, if the actual network model resided on another machine, the DEVS model representing the network model would have to invoke the network model remotely via web services, sockets, RMI, etc... upon receiving a request, or in the case of DEVS, an input that triggers an external transition. The code responsible for making remote connections to other couple models could be hosted in the overridden external transition function in an environment such as DEVSJAVA, created and provided by the Arizona Center for Integrative Modeling & Simulation (ACIMS) at the University of Arizona

### Summary

Research into developing executable capabilities for network centric testing and evaluation has significant potential transform the discipline of architecture development into a state where it more closely resembles the approaches which other technical disciplines follow when designing, specifying, and then constructing the artifacts which are the focus of their work. Architectural specifications which support collaboration amongst designers and programmers and with stakeholders who will procure, test and use such systems are needed. However, such architecture specifications and their consequential products need to be dynamic to support these collaborative dialogs and allow non computer scientists to understand how the system is intended to functions. Executable architectures hold the potential to achieve that goal and are worthy of further research and development at the investigatory level supported by federal research funding agencies.

### 8. References

- [1] DOD Architecture Framework, V1.0, Vol. I and II, 15 August 2003.
- [2] Enhancing DODAF with DEVE-Based System Lifecycle Development process Zeigler, Mittal.

## Author Biography

**Johnny Garcia** is founder and CEO of SimIS, Inc. and a Modeling and Simulation Ph.D. Candidate at Old Dominion University. He has over 19 years of engineering experience that includes systems architecture design, software development, database development, C4I systems development, logistics systems development, and new technology insertion for the Department of Defense. His Dissertation is in the creation of an Executable Architecture Analysis Modeling method discussed in this paper.

**Christopher Blancett** is a senior software Engineer at SimIS and is completing his Masters Degree in Modeling and Simulation from Old Dominion University.

Distribution: Authorized to U.S. Government Agencies and their contractors only; Reason: administrative/operational use. Other requests for this document shall be referred to the Program Manager for Test & Evaluation/Science and Technology (T&E/S&T), Test Resource Management Center, 1225 South Clark Street, Suite 1200, Arlington, VA 22202