2.1 Executable Architecture Research at Old Dominion University

Executable Architecture Research at Old Dominion University

Andreas Tolk
Old Dominion University
atolk@odu.edu
Johnny J. Garcia
Old Dominion University (SimIS Inc.)
johnny.garcia@simisinc.com
Edwin A. Shuman
Old Dominion University (MITRE)
ashuman@mitre.org

Abstract. Executable Architectures allow the evaluation of system architectures not only regarding their static, but also their dynamic behavior. However, the systems engineering community do not agree on a common formal specification of executable architectures. To close this gap and identify necessary elements of an executable architecture, a modeling language, and a modeling formalism is topic of ongoing PhD research. In addition, systems are generally defined and applied in an operational context to provide capabilities and enable missions. To maximize the benefits of executable architectures, a second PhD effort introduces the idea of creating an executable context in addition to the executable architecture. The results move the validation of architectures from the current information domain into the knowledge domain and improve the reliability of such validation efforts. The paper presents research and results of both doctoral research efforts and puts them into a common context of state-of-the-art of systems engineering methods supporting more agility.

1.0 INTRODUCTION

This paper introduces two ongoing related PhD efforts at Old Dominion University. Both efforts contribute to the topic of Executable Architecture research. Being members of the active M&S work force in Hampton Roads, both PhD candidates collected valuable experiences in projects and research. Embedding these experiences into the scholastic education within the Modeling and Simulation program of Old Dominion University ensures academically valuable results that promise to be practically useful as well.

The mentor for this work has experiences in the academic and practical realm as well. In a study on Active Layered Theatre Ballistic Missile Defense (ALTBMD) for NATO, he was member of an international team that used the "Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR) Architecture Framework," which evolved later into the "Department of Defense (DoD) Architecture Framework (DoDAF)" and the "NATO Architecture Framework (NAF)," to define ALTBMD architecture and execute them using simulation systems like the German "Tactical Missile Defense Simulator (TMDSIM)," the US "Extended Air Defense Simulator (EADSIM)," and the US "Extended Air Defense Test Bed (EADTB)" to evaluate and compare the different architecture proposals [1].

Each PhD effort is an individual contribution, but presenting them together in this paper allows focusing on the synergy between the research results. Both contributions address important gaps in the body of knowledge for executable architecture research. To do this, the paper is structured as follows. Section two will deal with a state of the art overview and present a summary of related research. The third section will focus on the necessity for a more formal approach to executable architecture, comprising the definition of elements that are pivotal for such an architecture, evaluating alternative modeling languages to model what needs to be executed, and a formalism allowing to introduce the necessary rigor. The fourth section introduces the idea of executable contexts. While the executable architecture represents the system under development, the executable context represents the oper-

ational environment the system will be applied in. Finally, the concluding section will synthesize both efforts and place them into a broader research agenda.

2.0 STATE OF THE ART

The overview in this section is neither complete nor exclusive. However, it shows the trend of recent developments, in particular for defense related architecture evaluations. The general underlying idea motivating the use of executable architectures is to enable the evaluation of dynamic aspects. A system's architecture is the static blueprint of a system that identifies who (function) is doing what (capability) where (component). Executable architectures allow furthermore to evaluate when (time) something is done. Dead locks, internal loops, and other related problems can be detected in the definition phase of the system.

Zinn [2] investigated the utility of using DoDAF architecture products to provide needed data for agent based simulations. This was accomplished by means of a case study where architecture data from a proposed Air Operations Center architecture was used in the combat model System Effectiveness Analysis Simulation (SEAS). The research concluded that DoDAF, if implemented properly, does provide the needed information for developing agent-based simulations. Zinn proposed a process of taking information from DoDAF architectures and importing it into an agent-based simulation. To model process information, Zinn used information contained in the OV-5 and OV-6a (IDEF3) to feed the agent-based simulation. The OV-5 provides the process and information flow, while the OV-6a provides the decision logic associated with the process.

Wagenhals et al. [3] provide a description of an architecting process based on the object-oriented Unified Modeling Language (UML). They describe a mapping between the UML implementations and an executable model based on Colored Petri nets. They examine DoDAF product sufficiency in terms of the Colored Petri Nets (CPN) simulations end state objective. Wagenhals et al. focus on the UML Sequence Diagram (OV6c), the UML Collaboration Diagram (OV5b) and the Class Diagram (OV5a – with extensions).

In 2005, Ziegler and Mittal [4] described the translation of DoDAF compliant architectures into DEVS simulations. They provided a set of DoDAF foundational Views and related UML diagrams for construction of DEVS-based simulations

In 2006, Mittal [5] addressed the question of extending DoDAF to support integrated DEVS-based modeling. His work cited DoDAF's shortcomings, to include his assertion of ill-defined information exchanges, the need for a coupling of entities, activities, and nodes, and a need to identify ports associated with activity-to-activity communication (since DEVS is a port-based modeling construct). He defined two new OV products, the OV-8 and the OV-9, as extensions of the DoDAF. The OV-8 addresses activities and their logical interface information. The OV-9 maps nodes, entities, and activities. This is similar conceptually to Activities-based methodology [6]. Mittal asserted the need for the OV-8 and OV-9 as intermediate precursor products in the development of the DEVS simulation. Mittal used the OV-5 activity model, the OV-6c (Sequence Diagram) and the OV-6a (Rules diagram – IDEF3), as a basis for generating a DEVS-based simulation.

In 2006, Mittal [7] described a means for semantically strengthening the critical OV-6a Rules Model, through application of Domain Meaning, Units of Measure (UOM), and formatting to domain specific rules, thereby removing ambiguity and aiding in translation of static to dynamic architectures.

In 2009, Risco-Martin et al. [8] described the essential mappings between UML and DEVS modeling. That work focused on the UML Structure and Behavior models that contribute to the development of a DEVS-based system model. Those UML models

are the Component Diagram, the State Machine, the Sequence Diagram, and the Timing Diagram.

3.0 A FORMAL APPROACH TO EXECUTABLE ARCHITECTURES

When evaluating the current approaches to derive executable architectures from static architectures, such as captured in DoDAF or comparable frameworks, it becomes obvious that the objective of these efforts is the use of dynamic simulation software to evaluate architecture models [5]. However, the current research is more concerned about concrete methods and tools, like the use of DEVS and DoDAF, the use of CPN and DoDAF, and similar projects.

The objective of the first PhD thesis is therefore to contribute to a theory of executable architectures. First results of this research are presented in [9]. The research derived from the observations of current approaches that it can be hypothesized that three categories are needed to define the necessary components for an executable architecture.

3.1 Elements

Elements define the static WHO, WHAT, and WHERE parts of an architecture. The elements provide the conceptual, structural, functional and state descriptions needed to describe and analyze a system. An architecture framework helps us to establish the boundaries for the discussion and to give it context and perspective. Examination of relevant elements of an architecture framework from conceptual, structural, functional and state perspectives helps to scope the topic of discussion.

3.2 Language

A modeling language allows us to instantiate the specifics of our architectural subset by describing both static and dynamic aspects of a system. The modeling language provides graphical, symbolic, standard notations designed to address various kinds of analysis and inquiry. A specific example of this would be a System Modeling Language (SysML) instantiation of the DoDAF OV-5, Operational Activity Diagram. That SysML diagram allows us to describe system behavior, or the functional system perspective.

3.3 Modeling Formalism

A modeling formalism for executable architectures should holistically describe the elements of an executable architecture using a standard mathematical notation. This ties the WHO, WHAT, WHERE, and WHEN together in a consistent and complete way. Traditionally, validation and verification supports this task. The formalism provides the mathematical frame to really prove that all functions are provided, interconnected, etc. The DEVS formalism is a promising first candidate. The elements of an executable architecture should be described using a modeling formalism, and minimally in the context of DEVS.

Figure 1 shows the concept triangle, including some examples for the components.

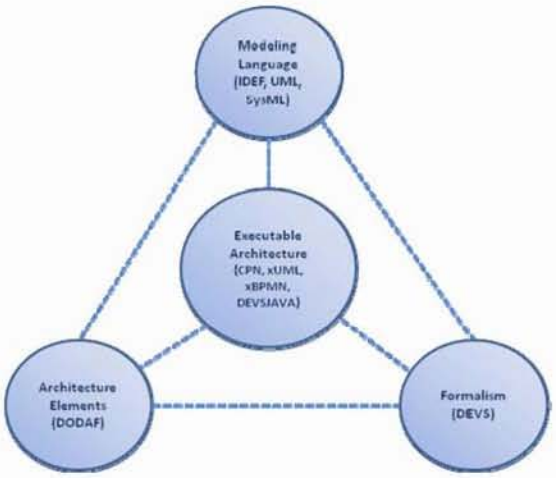

Figure 1. Concept Triangle

A theory of executable architectures must ensure that the architecture can be described completely and consistently through all three components. All elements captured in the Architecture Elements need to be part of the formalism and should be the subject or object of activities modeled with the Modeling Language. The Modeling Language must be the subject of the formalism and should not use elements that are not

captured in the Architecture Elements. The Formalism must bind elements and actions together and provide the mathematics to support validation and validity.

The first results published in [9] already show how to show alignment between Architecture Elements and Modeling Languages and apply metrics to the degree of alignment. Shuman showed, e.g., that for executable architectures the use of the SysML may be preferable to the use of the UML.

He also showed that the Fishwick modeling taxonomy [10], which distinguishes between conceptual, declarative, functional, constraint-oriented and spatial models, has significant value to support the three components of the concept triangle for executable architecture and provides foundational input for the general theory.

Such a theory will help to transfer valuable and practically relevant results between the various contributions so far. It will also support transferability of architecture artifacts, as de facto the components elements, language, and formalism must become a general meta-model of relevant approaches allowing to derive specialized solutions, like used in the examples described earlier.

4.0 ADDING EXECUTABLE CONTEXT

The focus so far has been on the system. Validating system architectures and assessing the contribution and efficiency of the specified systems before a system is built is the objective supported by the research of the second PhD effort. As pointed out before, the current state of the art of validation in practice is limited to static methods answering questions regarding who is doing what where. Executable architectures support system behavior analysis. They support examination of system timing questions (WHEN). They address questions related to the WHY and HOW of system behavior. In addition to this, executable architecture should address system context as well. Garcia presented the theory in [11] and showed an application example in [12].

Buede introduces the system's context as "a set of entities that can impact the system but cannot be impacted by the system. The entities in the systems context are responsible for some of the systems requirements." [13, p. 38] He also introduces external systems that interact with the system under development. Together, they introduce the system environment. Figure 2 captures these ideas.

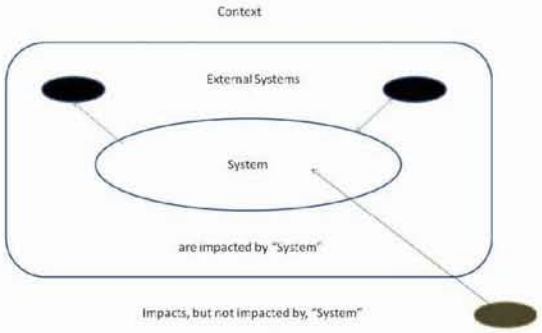

Figure 2. System, External Systems, and Context

While the executable architectures allow evaluation of system behavior (such as deadlocks and infinite loops), Fig. 2 shows that significant effects to system behavior will occur as a result of interaction with other external systems or even as a result of interactions between external systems. It is possible that the same category of dynamic problems that are evaluated in the previous section for the system's internal components by executable architectures – such as deadlocks between components – can occur between the system and external systems in the contexts of operations as well. Without an executable context, such insights are not supported by using an executable architecture alone.

Sage and Rouse aligned the six key interrogatives to information and knowledge categories, distinguishing between those that relate to information and those that relate to

knowledge: *who*, *what*, *where*, *when* refer to information; *how* and *why* deal with knowledge [14, p. 264]. Executable architectures should address both the information and knowledge categories. Adding system context allows us to address why a system acts as it does (and in so doing following the operational requirements of a given scenario) and how it performs its actions (in the collaboration with the other influencing systems (by meeting mission need).

All information needed to provide for the context is normally captured in systems engineering documents. The systems architecture is based on operational requirements (OR) that are derived from mission requirements (MR). These OR are refined into Systems Requirements (SR), Functional Requirements (FR), and Component Requirements (CR), which build the foundation for the systems architecture. MR and OR can be used to identify scenarios and metrics to measure the success of a mission. Figure 3 shows how the executable architecture is derived from SR, FR, and CR to be embedded into an executable context based on MR and OR.

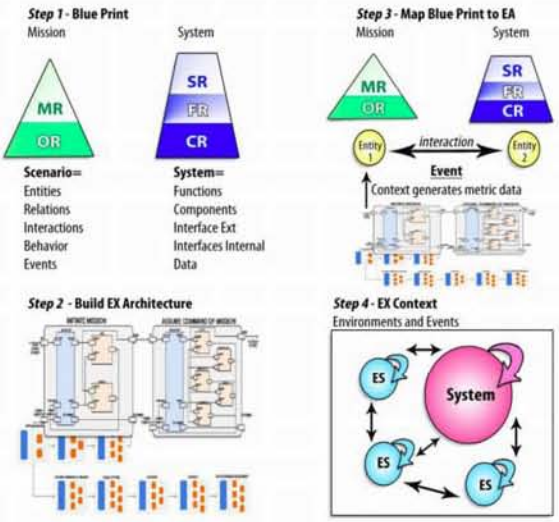

Figure 3. Executable Architecture in the Executable Context

Using the DEVS Unified Process (DUNIP) developed in [15], the system architecture is represented as an executable architecture in JAVA code and can react to inputs as defined in the system architecture and can produce the outputs using the appropriate causal and temporal constraints as defined for the systems.

Using validated simulation systems representing the context and the external systems within critical missions identified in the MR and OR, the validation of the architecture can now be conducted in the context of a valid scenario, using metrics identified by the real user for the critical missions. Garcia applied the NATO Code of Best Practice for C2 Assessment [16] and the Military Missions to Means Framework (MMF) [17].

This approach allows us to identify counter-intuitive effects, such as worst overall results.

5.0 CONCLUSION

The research currently conducted on executable architecture at Old Dominion University will contribute to better processes for validation of system development. Adding the power of M&S solutions to the rigor of systems engineering allows much better decisions on all level, from the stakeholder and future user of the system down to the implementing engineer. Executable architectures following the theory and being embedded into an executable context will allow all partners to display and evaluate operationally relevant data in agile contexts by executing models using operational data exploiting the full potential of M&S and producing numerical insight into the behavior of complex systems.

6.0 REFERENCES

[1] Adshead, S., Kreitmair, T., and Tolk, A. 2001. Definition of ALTBMD Architectures by Applying the C4ISR Architecture Framework. *Proceedings Fall Simulation Interoperability Workshop*, Vol. II, pp. 679 – 689, Orlando, FL, September

[2] Zinn, A. W. 2004. The Use of Integrated Architectures to Support Agent Based Si-

mulation An Initial Investigation. *Master's Thesis, Air Force Institute of Technology Air University*

[3] Wagenhals, L. W, Haider, S., and Levis, A. H. 2002. Synthesizing Executable Models of Object Oriented Architectures. *Proceedings of Workshop on Formal Methods Applied to Defence Systems,* Adelaide, Australia, pp. 85-93

[4] Zeigler, B. P., and Mittal, S. 2005. Enhancing DoDAF with a DEVS-Based System Lifecycle Development Process. *IEEE International Conference on Systems, Man and Cybernetics*, Hawaii, October

[5] Mittal, S. 2006, Extending DoDAF to Allow Integrated DEVS-Based Modeling and Simulation. *JDMS* 3(2):95-123

[6] Ring, S. J., Nicholson, D., and Pallab S. 2007. Activity-Based Methodology for Development and Analysis of Integrated DoD Architectures. *Information Science Reference: Handbook of Enterprise Systems Architecture in Practice*, Chapter 5, pp. 85-113, Systems Modeling Language OMG, 2008

[7] Mittal, S., Mitra, A., Gupta, A, and Zeigler, B.P. 2006. Strengthening 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, Hawaii

[8] Risco-Martin, J.L., de la Cruz, J., Mittal, S., and Zeigler, B.P. 2009. Eudevs: Executable UML with DEVS Theory of Modeling and Simulation, *Simulation* 85(7):419-450

[9] Shuman, E. A. 2010. Understanding Executable Architectures Through An Examination of Language Model Elements. *SCS Proceedings of the Summer Computer Simulation Conference*, Ottawa, Canada, July

[10] Fishwick, P. 1995. *Simulation Model Design and Execution.* Prentice-Hall, Inc.

[11] Garcia, J. J., and Tolk, A. 2010. Adding Executable Context to Executable Architectures: Shifting Towards a Knowledge-Based Validation Paradigm for System-of-Systems Architectures. *SCS Proceedings of the Summer Computer Simulation Conference*, Ottawa, Canada, July

[12] Garcia, J. J. 2010. Methodology Supporting Architecture Validations (MAVS). *SCS Proceedings of the Spring Simulation Multi-Conference*, Symposium Emerging Applications of M&S in Industry and Academia, Orlando, FL, April

[13] Buede, D. 2000. *The Engineering Design of Systems: Models and Methods*, John Wiley & Sons, Inc., New York

[14] Sage, A. P., and Rouse, W. B. (Eds.). 1999. *Handbook of Systems Engineering and Management*, John Wiley and Sons, Inc., New York

[15] Mittal S. 2007. DEVS unified process for integrated development and testing of service oriented architectures. *PhD Thesis. United States -- Arizona: The University of Arizona*

[16] *NATO Code of Best Practice for C2 Assessment*, 2002, CCRP Press, Washington DC

[17] Deitz, P.H., Sheehan, J.H., Harris, B.A., Wong, A.B.H., Bray, B.E., and Purdy, E.M. 2003. The Military Missions and Means Framework (MMF). *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC),* Orlando, FL, December