2022 NDIA MICHIGAN CHAPTER
GROUND VEHICLE SYSTEMS ENGINEERING
AND TECHNOLOGY SYMPOSIUM
DIGITAL ENGINEERING / SYSTEMS ENGINEERING TECHNICAL SESSION
AUGUST 16-18, 2022 - NOVI, MICHIGAN

MODEL BASED APPROACHES FOR SYSTEMS IMPLEMENTING MODULAR OPEN SYSTEM APPROACH (MOSA)

Robert Peters[1], Brooke McDonald[1], Robin Mikola[1], Macam Dattathreya, PhD[2]

[1]System Strategy, Inc., Troy, MI
[2]U.S. Army DEVCOM, MI

ABSTRACT

In the continued endeavor to abstract higher levels of implementation and generalize core features, the government is requiring the use of a Modular Open System Approach (MOSA) [1] to architectures that have a common set of services while conforming to portable interfaces. This paper discusses how to model such restrictions in SysML, including the why, how, and downstream effects.

Citation: R. Peters, B. McDonald, R. Mikola, M. Dattathreya, "Model Based Approaches to Systems Implementing Modular Open System Approach (MOSA)," In Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium (GVSETS), NDIA, Novi, MI, Aug. 16-18, 2022.

1. BACKGROUND

Modular Open System Approach (MOSA) is "a technical and business strategy for designing an affordable and adaptable system" [1]. Because of its variety of solutions to acquisition issues, MOSA is now required for new platforms [1]. "The objective in implementing this approach is to ensure systems are designed, where possible, with highly cohesive, loosely coupled, and severable modules that can be competed separately and acquired from independent vendors" [2]. MOSA effectively enables a system to integrate severable and modular components that can be replaced or upgraded without modifying the whole system architecture. In systems implementing MOSA the modules must have certain boundaries with open standards-based interfaces appropriate for separating out the important aspects of each feature, so that the vendors create whole modules that serve their functions in their entirety.

Using Model-Based Systems Engineering (MBSE) for acquisition is the current trend, and its top challenge is to model systems to demonstrate MOSA principles. This paper describes at a high level the considerations and approaches that can be followed to effectively model a system applying MOSA, using the System Modeling Language (SysML). Instead of developing a system architecture, the approach described in this paper uses a concept of "MOSA Objective Architecture" to model the rules and constraints to be followed by the system

implementers to achieve MOSA principles in their designs. The Objective Architecture includes the minimum required attributes and capabilities of a family of systems at a high level. The goal of the Objective Architecture is to define the minimum of implementation specific constraints on the vendor system. The specifics of this system will eventually be developed and modeled in a system architecture that implements the high-level specifications from the Objective Architecture.

The MOSA Objective Architecture, based on open standards, is the first step in building portability and modularity while promoting interoperability. In an open system, portability means that a component can run on multiple different platforms. The aim is to minimize the human efforts in redesign and redeployment of each new component to a platform. Interoperability means the components can regularly interact and exchange information with one another through a standardized and well-documented interface. Component-specific interfaces do not need to be adapted for each system.

The Objective Architecture also defines common capabilities and services that must be part of any compatible platform. The intent is to allow the architects to make some capabilities available on all platforms and encourage or enforce usage from the platform specific modular components. It should be noted that the Objective Architecture may vary how much of any interface is pre-defined vs how much of the interface would need to be defined by the implementor.

The Objective Architecture defined herein ensures that any vendor-derived system architectures are portable, modular, and interoperable with all other systems by requiring the compliance to the following definitions. Common services that are defined without specific definition of their implementation allow the vendor to control the specificity within their component. Common services and messages are defined allowing for the reuse and interoperability of the components. Finally, the physical interfaces and software interfaces are constrained to allow for portability of components. These definitions within the Objective Architecture ensure that complete, portable, modular, and interoperable components can be effectively designed by a vendor.

2. MODELING CONSIDERATIONS

One aspect of modeling a MOSA Objective Architecture that requires extra consideration is the nature of the architecture being a set of rules for a vendor system architecture more than a system architecture itself. Since the driving factor was a MOSA, it would be perceived as "too prescriptive" for the Objective Architecture to fully define how the vendors need to design their system. For this reason, the Objective Architecture is extendable for implementation and integration into the target platform.

2.1. DIFFICULTIES MODELING AN OBJECTIVE ARCHITECTURE

MOSA Objective Architectures, by definition, are under-prescribed. Modeling an under-prescribed system has its challenges. The lack of description of functionality and attributes causes models to be incomplete. Activity diagrams need to have a significant amount of information for them to be understandable, and Objective Architectures cannot provide the usual amount, or any context to derive it. Activities cannot connect between the defined common services and platform specific components because the platform requirements are not described. The interfaces fully describe the inputs and outputs abstractly per Entity. However, modeling connections and interactions from an example client would mean having to assume why and what one entity may need to communicate with another. The fact that this

modeling difficulty exists means that there is low coupling between entities in the model.

Getting cohesiveness and coherence in a model of something that is not meant to be complete is done using weak relationships between elements to show connections that are implicitly defined. Specifically, allocate relationships are used extensively to show how specifications relate to other model elements and how model elements relate to one another between layers of the Objective Architecture model.

A final challenge with modeling is that the specifications may be organized by subject areas as opposed to logical model elements. More on this challenging aspect and a solution to it is described in section 3.1.

3. MODEL STRUCTURE AND RELATIONS

For the approach described in this paper, an Objective Architecture model is organized into three main categories: Specification Structure, Functional Architecture, and Logical Architecture. This allows for splitting up the model into the requirements views, functional views and logical views. This supports minimal descriptions and modularity as shown in the following sections. Figure 1 shows the three packages in an example structure. Figure 2 shows how the three levels relate to each other.

Figure 1: Containment Structure Example

Figure 2 shows how the three levels relate to each other.

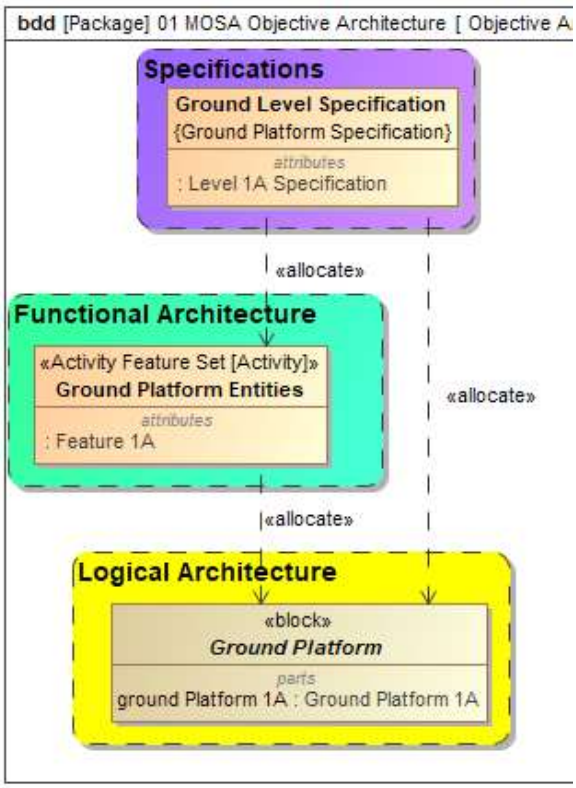

Figure 2: Relationship Diagram Between Specification, Logical and Functional Example

3.1. SPECIFICATIONS

The specifications for an Objective Architecture may be organized by subject area/topic (e.g., specific services, general services, general hardware, cybersecurity, etc.) as opposed to element (a.k.a. Component or Entity). There can be a main specification that can reference other specifications and in turn those may reference other specifications. In order to model a MOSA Objective Architecture, each specification has its own Package with sub-Packages for referenced specifications. Classes are added within the package holding each specification table. The specification classes are loosely mapped (allocate relationship) to the elements in the model that are necessary to satisfy the requirements in each specification within the Functional Architecture and the Logical Architecture

sections of the model. The intent is for the Objective Architecture to only contain the minimum required elements. As such, the Functional Architecture and the Logical Architecture are only made up of the model elements that the specification classes are mapped to. This mapping connects the structure of the specifications to the rest of the model. If this isn't done, the relationship between the two would not be clear.

Note: An alternative that was considered and rejected was having the rest of the model structure match that of the specification structure, but it made the relationship of elements within the Logical and Functional Architectures nearly impossible to describe effectively.

For greater requirements tracing, there are two ways that specifications are directly tied into the Functional and Logical Architectures:

1) The requirements are used as constraints on model elements where other modeling is unclear or not possible.
2) The requirements link to model elements with satisfy relationships which also show where the requirements have been satisfied – a useful tool for model viewers.

The general structure of the Specification Structure start with the Top-level MOSA specification, then continues with the references Specifications. Hierarchy outline of Specification Structure for 'Current' Specification:

1) {Parent} : Package [1]
    a) {Current} : Package [1]
        i) {Child} : Package [0..*]
        ii) 00 Introduction - {Current} : content [1]
        iii) {Current} - Allocations : BDD [1]
        iv) {Current} : Requirement Table [1]
        v) ??? : Requirement [1..*]
        vi) {Current} : Class [1]
            (1) {Current} : Constraint [1]

Figure 3 shows a sample of what a Specification Structure could look like if it has 3 levels of description.

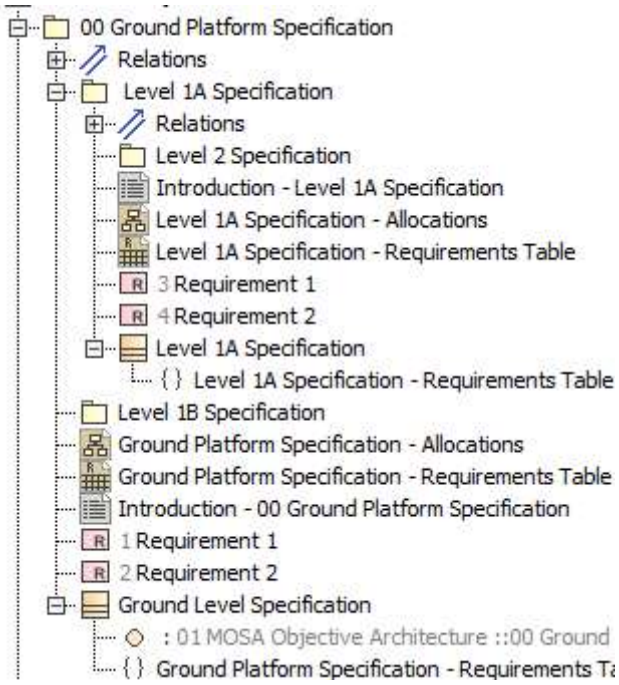

Figure 3: Specification Structure Example

Figure 4 shows a sample of a Specification allocations and associations.

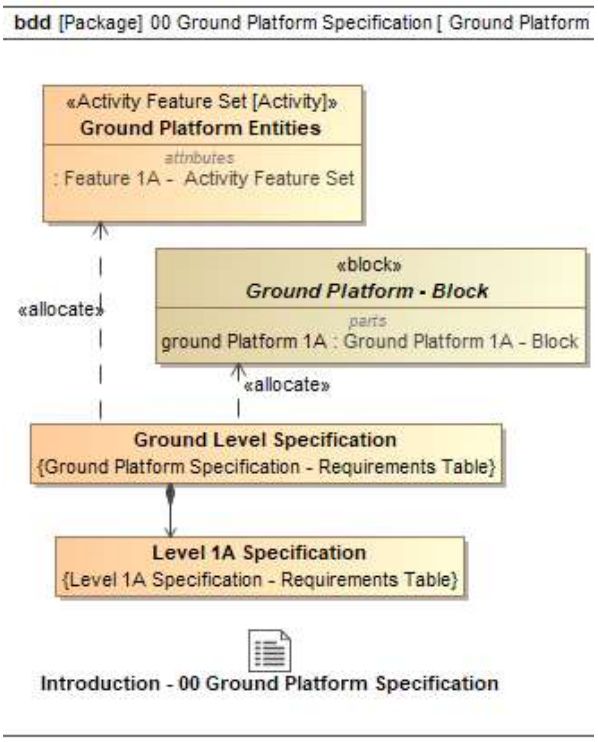

Figure 4: Specification Allocations Example

3.2. FUNCTIONAL ARCHITECTURE

The Functional Architecture is a set of features, capabilities and behaviors contained within Entities using a special stereotype called "Activity Feature Set." An Entity is a functional Element that contains a set of behaviors described in the requirements and represents an endpoint of communication between data, messages and/or information. The "Activity Feature Set" stereotype, which extends the use of an activity, differentiates the blocks in the Logical Architecture from those in the Functional Architecture. This approach allows the difference between logical ports and purely functional ports to be more obvious, but still have roughly the same model representation. Logical ports show a more physical type of interface, as where functional ports are used to show the interfaces for the information sent between Entities. The use of a Block is specifically avoided to clarify to a viewer of the model that these are not logical structural elements of the system. An Activity Feature Set stereotype is composed of behavioral elements not directly allocated to any structural logical element.

The important part of the Functional Architecture is to show the how the Signals are used between defined Entities of the system. Behavior diagrams are used to capture how Signals move and are handled within each Entity. Using Send and Accept Event Actions circumvents the need to know who sends data and who receives it since all they say are what port the data is received or sent on. It can be assumed, using Send and Accept Event Actions, that any other activity could be sending or receiving the data, which exemplifies the openness of a MOSA Objective Architecture. When the interface is an open standard anything in the system can use it.

The Entities match the description of a component, service, etc. from within the requirements that perform an action based on passing information between Entities. The Entities are modeled as Activity Feature Sets and contain the owned interface blocks and behaviors. The Signals described here are not meant to represent a message; they represent a set of information being passed between Entities. These Signals are mapped to one or more messages within the Logical Architecture either within the Objective Architecture if required or left open to be mapped by the vendor implementation.

The structure of a Functional Architecture generally follows the hierarchy of the Entity generalizations with some additional grouping as makes sense by the modelers. It starts with the Top-level Activity Feature Set, then continues with the owned Entities (Activity Feature Sets). Hierarchy outline of Functional Architecture for {Current} Activity Feature Set:

1) {Parent} : Package [1]
    a) {Current} : Package [1]

i) {Child} : Package [*]
ii) 00 Introduction - {Current} [1]
iii) {Current} - Parts : BDD [0..1]
iv) {Current} - Generalizations : BDD [0..1]
v) {Current} - Signals : BDD [0..1]
vi) {Current} IF : Interface Block [0..*]
  (1) {Current} Message : Signal [*]
  (2) {Current} Parameter : Data Type [*]
vii) {Current} : Activity Feature Set [1]
  (1) Do {Current} : Behavior [*]
  (2) +port : Interface Block [0..*]

Figure 5 shows a sample layout of 3 levels of features. This includes the chosen containment structure of related Elements for each Activity Feature Set.

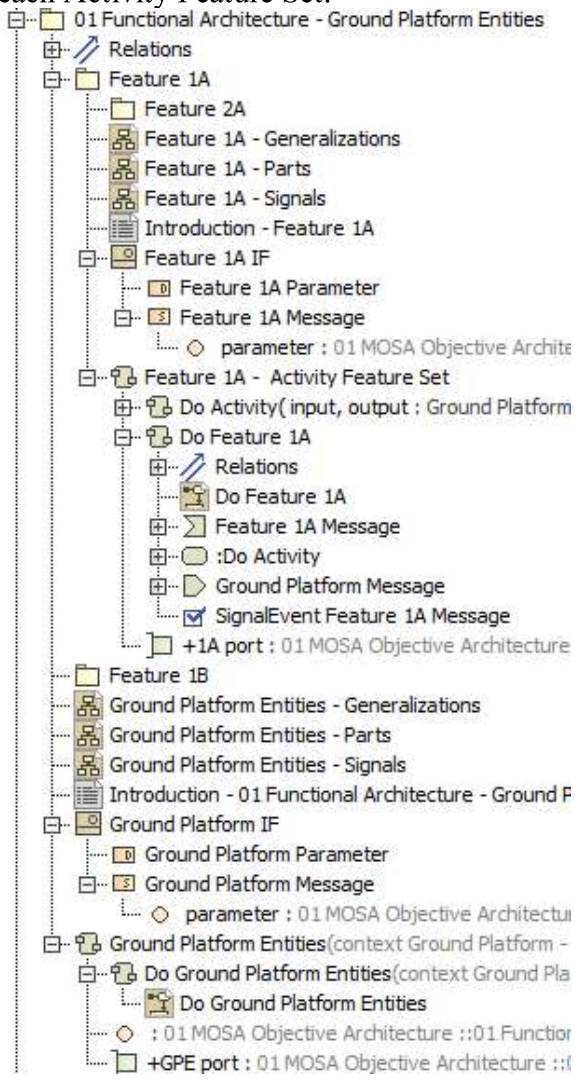

Figure 5: Functional Structure Example

Figure 6 shows a sample Activity Diagram using Accept/Send with Signals that are part of the Interface Blocks on a specific Port. It also shows the parameters moving through the Activity Diagram.

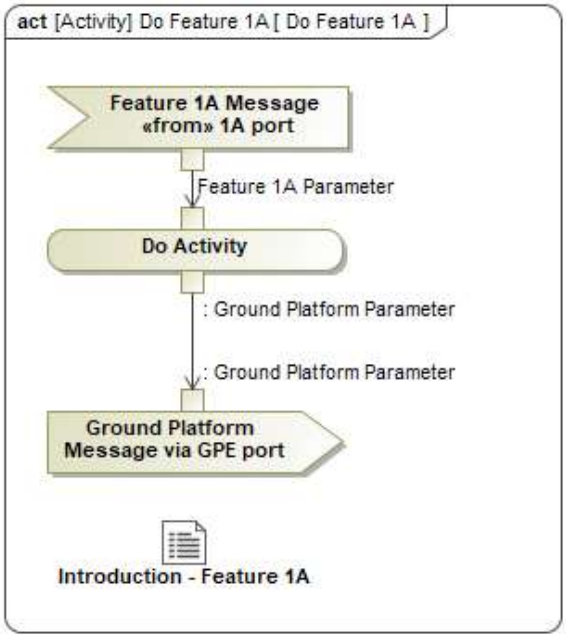

Figure 6: Activity Diagram Example

3.3. LOGICAL ARCHITECTURE

The definition of "Logical Architecture" in this context is a level of abstraction that defines components of the system that perform the functionality in the Functional Architecture. It describes the logical properties of each component characterized by the Objective Architecture specifications. There is a direct allocation from the elements within the Specifications and Functional Architecture to those in the Logical Architecture which allows viewers to understand the connection of the two architectures. All Logical Components have an allocate relation from the Functional Architecture and may have an allocate directly from the Specifications. But, the set of Components may be small, if there are only a few required logical elements.

The connections between logical elements are modeled on interfaces that are logical or physical, and this is the main difference between the Functional Architecture and the Logical Architecture. The Logical Architecture is a way of showing vendors the minimum components and interfaces needed within their system whereas written specifications may veil this information.

Hierarchy outline of Logical Architecture for {Current} Block

1) {Parent} : Package [1]
   a) {Current} : Package [1]
      i) {Child} : Package [*]
      ii) 00 Introduction - {Current} [1]
      iii) {Current} - Parts : BDD [0..1]
      iv) {Current} - Generalizations : BDD [0..1]
      v) {Current} - Signals : BDD [0..1]
      vi) {Current}: Block [1]
         (1) Message : Signal : Parameter Set [*]
         (2) Parameter: Data Type [*]
         (3) Defined Block : Abstract Block [*]
         (4) +part : Typed by Block [*]
         (5) +port : Interface Block [*]

Figure 7 shows a sample layout of 3 levels of logical Blocks. This includes the chosen containment structure of related Elements for each Block.

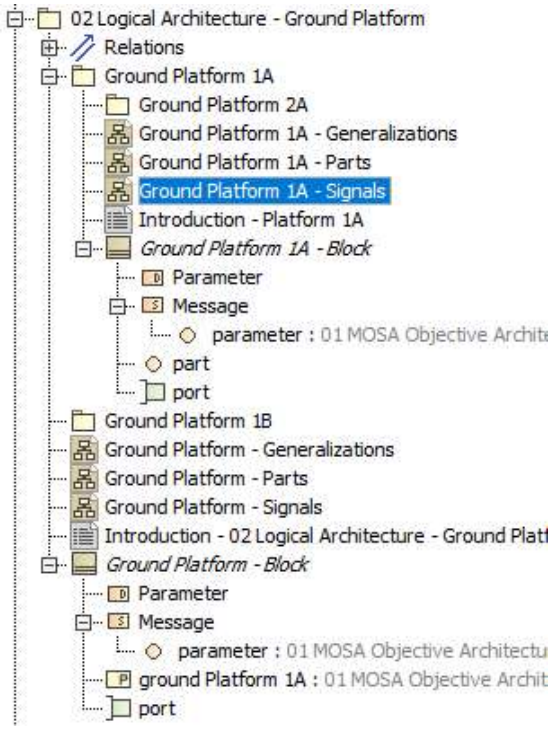

Figure 7: Logical Structure Example

3.4. DOCUMENTATION AND NAVIGATION

To support understanding the model structure, it is important to create documentation that supports navigating through the related elements of the model. For each major element of the model (Class from Specifications, Activity Feature Set and Blocks) there is a content diagram with a relatively common layout that puts links to related element Diagrams, such as:

1) Parent content diagram
2) Related requirements
3) Owned Elements' content diagrams
4) Element relation diagrams (IBD, BDD)
5) Related Elements (as determined by the modeler)
6) Behavior diagrams
7) Allocations between levels

Figure 8 shows an example Content diagram with related documentation and links. (Modified to fit the paper format. It

should be better spaced for visibility.)

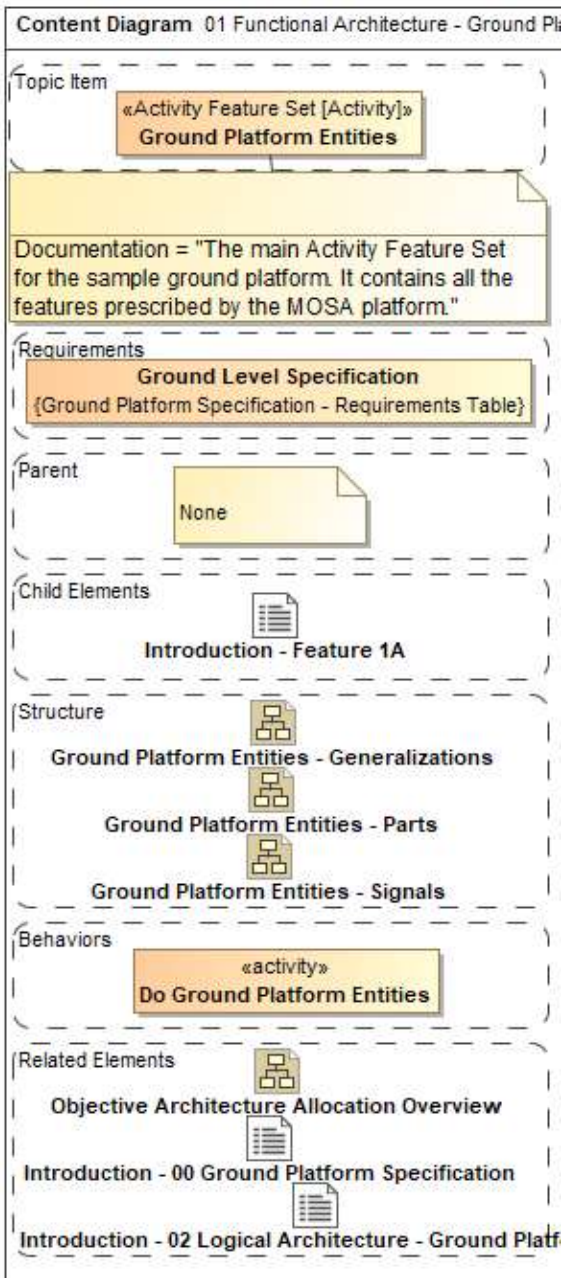

Figure 8: Content Diagram Example – Activity Feature Set

Each diagram should also contain a link to its related content diagram to allow full linkage throughout most of the model's diagrams. (Tables, Decomposition maps, etc. will not support linking back to the content diagram.)

4. MODEL COHESIVENESS CHECKS

Many measures need to be taken to ensure cohesiveness within the model. Beyond the normal model check, there are at least two specific checks for this Objective Architecture. First, check that every Activity Feature Set has activities or state machines located within them. Second, make sure that all requirement-specified functionality and communication on interfaces is in the model activity diagrams and, when both Entities are described, shown on internal block diagrams for the Functional Architecture. To determine whether the interfaces are all accounted for and if all communication is displayed, modelers double check the requirements and their satisfy relationships. For this reason, it is imperative to have as many satisfy relationships to each requirement as is justifiable. Doing this allows the modeler to more quickly view how the requirement is satisfied without the need to always look in other parts of the model. Another aspect to check is to make sure that Send and Accept Event Actions in activity diagrams have an opposite of the same or generalized type in another activity diagram when both are specified. Checking for matching Accept Event Actions to Send Event Actions is one way of ensuring cohesiveness and continuity in the model.

An important model aspect to check is that all model elements are appropriately allocated from Functional Architecture to the Logical Architecture and that all model elements generalized all necessary parents. The generalizations would show the inherited properties the element needs according to the specification structure. Along with allocations, this allows vendors to know which elements they need to implement to create each part of and be compliant with the Objective Architecture.

5. INTENDED USAGE

The behavior modeled within the Functional Architecture will be used by the vendors to show compliance to the MOSA Objective Architecture. Vendors will take the current behaviors (activity diagrams and state machines), Activity Feature Sets and Blocks, and add the specifics/complexity of their system while still using the ports and signals in the diagrams. Any extra functionality that the vendor adds outside of what was already in the Objective Architecture would have to use a similar approach to cohesively show communication between functional elements. This allows the Program Manager to be able to easily identify whether the model component(s) are still ultimately doing what they are specified to do. This also provides the capability of tailoring components specific to the program or platform specific objectives consistently.

How to do this is outside the scope of this document but is discussed in the Future Investigation section.

A desired usage for this MOSA Objective Architecture is to give industry the minimum number of rules to create a design without describing the exact details, thus leaving them open to creative solutions.

Having the model and requirements setup in a solution independent manner makes it easier for vendors to look at an element and all its related parts in the model. And if a vendor is unable or unwilling to implement the entire Objective Architecture, they can tailor it to what suits their needs. The chosen functionality and the allocations from the Functional Architecture to the Logical Architecture gives them a complete picture of what they are expected to implement for their tailored design to still be compliant and cohesive.

6. INTENDED APPLICATION

The Objective Architecture at the highest level is structured to cover all specifications and standards applicable to the compliance of a MOSA. With the decomposition of the specification to the functional and logical architecture definition verification steps can be created to ensure compliance to the common definition of these parts.

Implementation of the Objective Architecture by a vendor extends the definition from the Logical and Functional Architectures with complete traceability to the specifications and standards driving the MOSA system. This ensures that each component of the platform system complies with the Objective Architecture model and by extension will be portable and interoperable within all MOSA enabled platforms derived from this Objective Architecture.

7. FUTURE INVESTIGATION

Future investigations include integration with other models within the acquisition, design and implementation process. This includes considering the impacts of using a ‘Conceptual Architecture’ instead of the Functional Architecture and mapping to a ‘Logical Architecture’. This would be helpful throughout the platform lifecycle including the definition, acquisition and verification phases.

This proposal would more broadly define an Objective Architecture to be applied to any incomplete set of requirements and allow maximum flexibility and links to a completed system. It can be used as the basis of a common interoperable platform (as in this case). It can also be used to describe the customer’s requirements of any component that is part of a single system design. This flexibility and linkage can be useful throughout MBSE acquisition and integration lifecycle.

Other improvements in methodology include automated of checking of structure, linkage and rules, and changing how the Specifications are created and related to the rest of the model. For example, the trace

relation may be more appropriate for relating Specifications to Functional and Logical Architectures.

8. SUMMARY

Future system acquisition within the government will increase its MOSA usage to obtain modular and open systems. Using MBSE for request for proposals and vendor responses will make this process easier and more standardized, but it will be an ongoing effort to model Objective Architectures effectively. This paper introduced some solutions to model MOSA related problems and offered suggestions for best practices for modeling approaches to the objective architecture structure.

9. REFERENCES

[1] "DSP :: MOSA," *www.dsp.dla.mil*. https://www.dsp.dla.mil/Programs/MOSA/

[2] National Defense Industrial Association Systems Engineering Architecture Committee, "Modular Open Systems Approach: Considerations Impacting Both Acquirer and Supplier Adoption," INCOSE, Jul. 2020.

[3] "Model-based systems engineering," *Wikipedia*, May 23, 2022. https://en.wikipedia.org/wiki/Model-based_systems_engineering (accessed Jun. 09, 2022).

[4] Wikipedia Contributors, "Systems Modeling Language," *Wikipedia*, Jan. 15, 2020. https://en.wikipedia.org/wiki/Systems_Modeling_Language (accessed Feb. 02, 2020).

10. GLOSSARY

Name	Definition
BDD	*A Block Definition Diagram is a static structural diagram that shows system components, their contents (Properties, Behaviors, Constraints), Interfaces, and relationships.*
Component	*A logical representation of an Entity. A Component can communicate to other Components via logical Interfaces.*
Conceptual Architecture	*The very high-level structural representation of the system, independent of design choices, allowing for exploration and comparison of multiple Logical architectures.*
Data Architecture	*A Data Architecture defines the rules of construction for required data models and is focused on the representation of data exchanges in software.*
Element	*A single selectable item of the model.*
Entity	*A functional Element that contains a set of behaviors described in the requirements and represents an endpoint of communication between data, messages and/or information. Related to Component.*
Functional Architecture	*The functional representation, independent of design/implementation, of a system, including the exchange of information that happens as part of performing those functions*
IBD	*An Internal Block Diagram is a static structural diagram owned by a particular Block that shows its encapsulated structural contents: Parts, Properties, Connectors, Ports, and Interfaces.*
Interoperability	*Components can regularly interact and exchange information with one another through a standardized and well-documented interface.*
Logical Architecture	*The basic structural representation of a system that includes high level design choices but is independent of specific design choices.*
MBSE	*Model-based systems engineering (MBSE), according*

	to INCOSE, is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases.[3]
MOSA	*A Modular Open Systems Approach (MOSA), formerly known as Open Systems Architecture or Open Systems Approach, can be defined as a technical and business strategy for designing an affordable and adaptable system. [1]*
MOSA Objective Architecture	*An Objective Architecture to meet MOSA's goals of adaptability.*
Objective Architecture	*The minimum MBSE representation of the requirements of a system, as described by customer looking for vendors to implement. It allows for the maximum implementation freedom that meets the needs of the customer.*
Physical Architecture	*The detailed structural and functional representation of the system, includes specific design choices and sufficient detail with which to describe the "design to" condition of the system.*
Platform	*The top-level system that MOSA is applied to. The typical context is a vehicle, but can refer to other types of systems as well.*
Portability	*Components are HW and/or SW Elements that can integrate with multiple different platforms without recompiling or modifying the HW.*
SysML	*Systems Modeling Language. A general-purpose modeling language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. [4]*
System Architecture	*A model that defines the structure, behavior and views of a complete system.*