"Model Based Systems Engineering for Acquiring Vehicle Training Simulations"

R. Bradley Cope, Devarshi Desai, Cattien Nguyen, Naomi Acosta
Naval Air Warfare Center Training Systems Division
Orlando Florida
bradley.cope@navy.mil, devarshi.desai@navy.mil, cattien.nguyen@navy.mil, naomi.acosta@navy.mil

ABSTRACT

The constraints of a documentation-based engineering approach limit designers to chronicling the development of systems with written descriptions and illustrations that fail to keep pace as the design progresses. A transformational shift from the documentation-based approach is Model Based Systems Engineering (MBSE), an emerging disciplined innovation for blueprinting architectures of complex systems. With MBSE, the system model is independent of the views a system designer creates for acquisition stakeholders and when the model changes, the modeling tool automatically updates views to reflect the changes. Furthermore, the effectiveness of MBSE is exhibited when there is a common architecture for establishing the reuse of components across many similar systems. For these reasons, vehicle training simulations seemingly are an ideal application for MBSE. MBSE unleashes the potential to unify a broad range of disciplines involved with the design of vehicle training simulations where participants communicate in a standardized manner through a graphical Systems Modeling Language (SysML). The approach taken in this paper is to illustrate a core set of MBSE capabilities in a simplified manner enabling the reader to see how it might be used for managing requirements. To demonstrate a possible approach for modeling vehicle training simulations, several sample stakeholder views are presented. The structure diagram depicts architecture, the requirements diagram reveals traceability, the use case diagram conveys stakeholder expectations, and the activity diagram specifies the use cases. An interpretation of how an MBSE approach could replace the traditional documentation approach is shown with a categorical mapping of system design attributes across the two approaches. Considerations are provided for using an MBSE approach to manage requirements vice the documentation-based approach for acquiring vehicle training simulations.

ABOUT THE AUTHORS

Brad Cope earned a Master's degree in Engineering Science from Penn State University and was trained by MIT and Georgia Institute of Technology in System Architecture and Model Based Systems Engineering. He specializes in the acquisition of software for training systems.

Devarshi Desai has a Master's degree in Software Engineering from Mercer University and 13 years of experience working for the Air Force and Navy. He has several years of experience working an MBSE approach using SysML to define and verify requirements used in the design of vehicle training systems.

Cattien Nguyen graduated from the University of Central Florida with a Bachelor of Science degree in Computer Science and holds a Master's of Science degree in Systems Engineering from the Naval Post Graduate School. Cattien's areas of expertise are in requirements analysis, system modeling, test, and systems engineering.

Naomi Acosta has Master's degree in Software Engineering from the University of Puerto Rico and 10 years of experience in defense software acquisition. Naomi's areas of expertise are in software development and human/machine interaction. She is a trained SysML modeler experienced in modeling components for training systems.

"Model Based Systems Engineering for Acquiring Vehicle Training Simulations"

R. Bradley Cope, Devarshi Desai, Cattien Nguyen, Naomi Acosta
Naval Air Warfare Center Training Systems Division
Orlando Florida
bradley.cope@navy.mil, devarshi.desai@navy.mil, cattien.nguyen@navy.mil, naomi.acosta@navy.mil

RATIONALE

The logical basis of this paper is to highlight how Model Based Systems Engineering (MBSE) can be applied to improve the acquisition of complex vehicle training simulations. A highly functional modeling language for MBSE called Systems Modeling Language (SysML) is reviewed for its features with managing requirements. SysML is an extension of an enduring modeling language that engineers apply for designing software called Unified Modeling Language (UML). The benefits of UML are proven and well documented. Several disciplines are involved with the design of vehicle training simulation systems – engineering, physics, image generation, computational systems, instructional design, and test. With SysML, each of these disciplines can have customizable stakeholder views for illustrating a complex system architecture while reducing ambiguity. SysML can unify these disciplines by having a common model that is communicated with standardized semantics, a step in the direction of moving away from an antiquated error prone documentation-based process [1]. SysML eloquently describes system design for implementation, and in some instances designed systems can even be simulated within the SysML environment. Having a SysML model provides the ground truth for not only the design of systems, but also for conducting analyses, verification, and validation. A premise for using MBSE in the acquisition of vehicle simulators is to characterize the acquiring activity as the initial architect of the system. Another premise is there is a product line where a foundational common architecture can be established allowing the reuse of components across many similar vehicle training simulation systems. To evaluate the notion of using MBSE for the acquisition of vehicle training simulations, it's necessary to explore how requirements are managed, the business case, skillsets needed, change management and the reuse of models.

MODEL

A model is an abstraction of a system. Documentation or SysML, or a combination of both, can capture the abstraction of a system with the perspectives stakeholders need. Requirements are a reflection of the system model where component dependencies are mapped for traceability. At the core of a system architecture defined with SysML is one model that connects everything together. Similar to the systems engineer being the connective tissue that unifies subject matter experts in solving system problems, a model unifies the system [2]. When implemented correctly, a model reflects the interconnected nature of the system whereby views spanning the disciplines are based on a unified set of requirements. As the design progresses, the model joins designers and stakeholders to improve analysis and communication.

DIAGRAMS

Diagrams make particular aspects of a system specific, where it is easier to emphasize information in a diagram than to describe something in words. Even the act of diagramming forces specificity [3]. The diagrams used in SysML fall into two major categories, structure and behavior. Structure diagrams represent the decomposable architecture of the system. Behavior diagrams express the operation of the system i.e. the flows of execution or use cases. SysML makes it easier to consider both structure and behavior as interconnected yet distinct diagrams [4]. SysML diagrams contain the same information conveyed in conventional documentation-based engineering, the difference is SysML interconnects the underlying model. The third category is the Requirements diagram (req) which is neither structure nor behavior. The Requirements diagram is intertwined with the structure and behavior diagrams providing a more

holistic view of the system model where requirements that relate to a behavior or structure can be shown with relationships directly on the diagrams. This allows depiction of the architectural design the way it is, because there are requirements for it to be that way. The fourth category of diagram is the Parametric diagram (par) which is a special kind of Internal Block diagram (ibd) for conveying mathematical models and numerical quantities related to system performance e.g. the maximum speed, the amount of force needed, the amount of energy used, or the maximum acceleration. The parametric diagram is ideal for the verification of performance requirements.

Structure

The block is the fundamental element for defining structure and it appears on two kinds of structure diagrams, the Block Definition diagram (bdd) and the Internal Block diagram (ibd). The Block Definition diagram defines the major blocks within a system and their relationship to each other, where a block represents a system, a part of a system or some aspect of a system. Blocks can have ports for exchanging representations of data, matter or energy between other blocks. Blocks in SysML define types of modeled elements, similar to how the blueprint of a house is used to model a house. Just as the house blueprint can be reused to build many houses with the same structure, the block can be reused to create many instances of modeled elements for constructing a system design. Similarly, the Block Definition diagram shows types, and the Internal Block diagram shows instances of the types defined in the Block Definition diagram. The Internal Block diagram shows the structure within blocks, effectively decomposing blocks into sub-parts. Shown in Figure 1 is a Block Definition diagram depicting a basic vehicle simulator. While blocks are used to type structure, they also are used to type behavioral features.

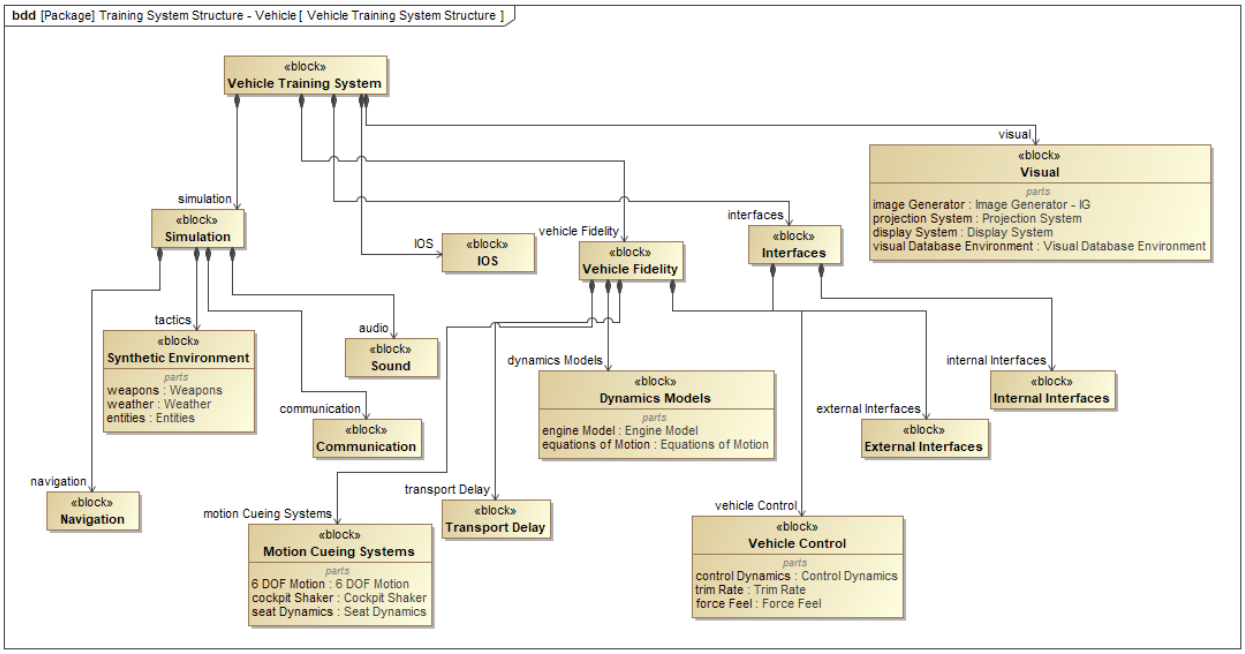

Figure 1. Block Definition Diagram for a Basic Vehicle Simulator

Behavior

There are several types of behavioral diagrams: Use Case diagram (uc), Activity diagram (act), State Machine diagram (stm), and the Sequence diagram (sd). Use cases describe what the system is actually doing and what it's going to do for the users. The use case is a sequence of operations or events that expresses a needed goal of the system. It conveys a particular capability or interaction between the user and the system that produces some result of value. The use case always describes a complete usage of a system. It's a manifestation of a requirement. If the system must accomplish a uses case, this becomes a design requirement. Actors represent external roles performed when interacting with the system and invoke use cases. An actor can be a person, it can be another system, or it can be a component within the system responsible for actions [5]. Systems that interact with other systems are considered actors, they are typically illustrated as a rectangular box rather than a stick figure. Inside the rectangle, use cases are shown as ovals. The oval

contains an action verb phrase representing the name of the use case. Another thing to note about use cases is that all of the use cases taken together, express the totality of the required system behavior. Use cases reveal the behavior that the architecture can produce. Figure 2 shows a general Use Case diagram for a vehicle training simulation system. Each use case requires specificity, a use case specification. The Activity diagram fills this role and tells the story that unfolds when the actor comes along and invokes the use case. The Activity diagram is effectively a descendant of the flowchart. It expresses the flow of action through the use case and can have decisions with branches to show all outcomes of the use case. Swim lanes in Activity diagrams show the responsible actor for actions. Figure 3 is the Activity diagram that specifies the "Provide Simulation" use case shown in Figure 2.

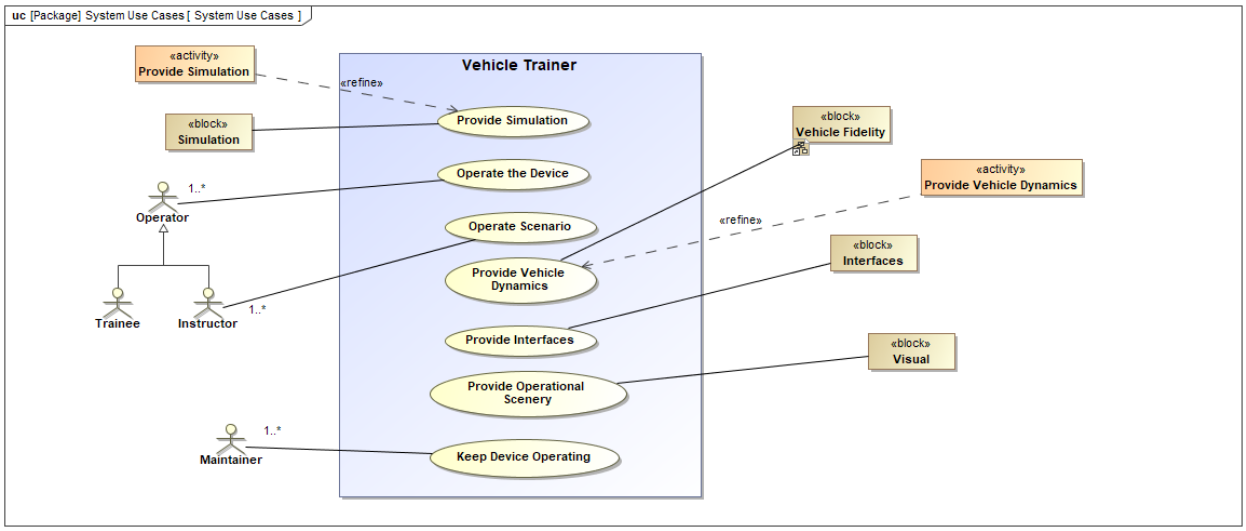

Figure 2. General Use Case Diagram for a Vehicle Training System

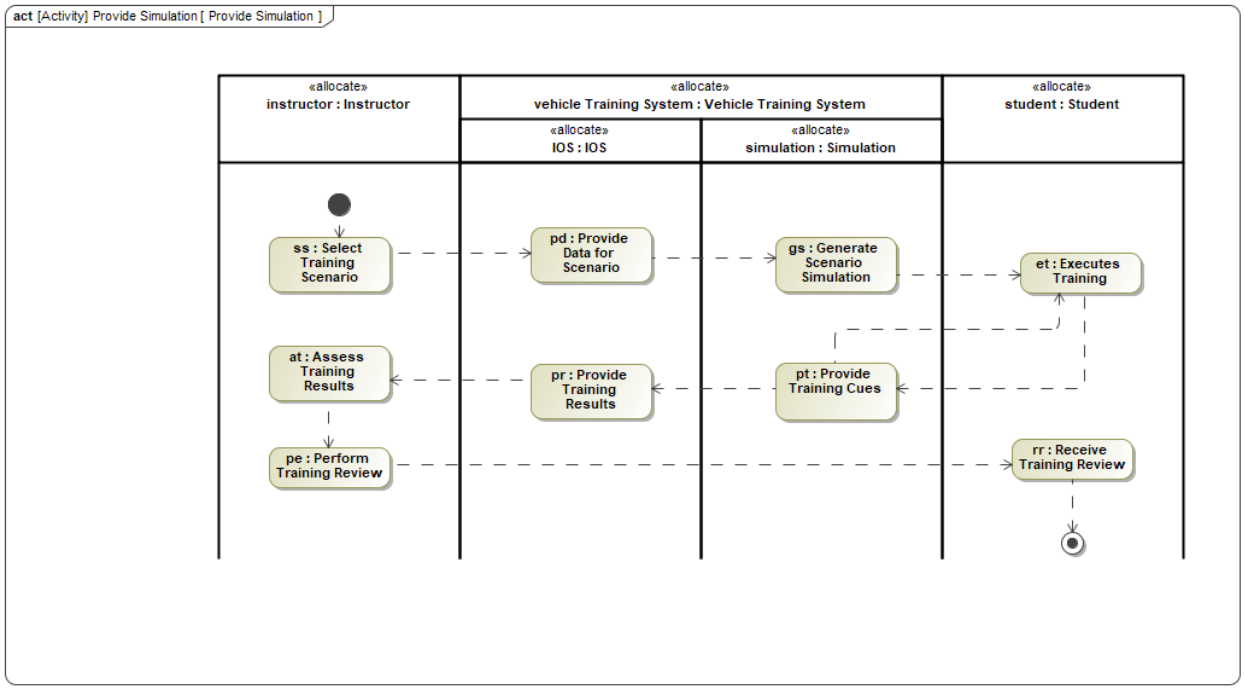

Figure 3. Activity Diagram for Specifying the "Provide Simulation" Use Case

Requirements

While the combination of use cases and Activity diagrams is a widespread technique for establishing requirements, SysML provides the means to diagrammatically represent text-based requirements using the Requirements diagram which can alternatively be used to display the relationships of requirements with model elements [6]. Requirements can also appear on other diagrams to show how they relate to elements shown specifically on that diagram. Since the model is an interconnected view of the underlying requirements and has built-in traceability, the requirements diagram is particularly useful for tracing back to elements added to the model. The advantage of entering a long list of requirements from a sizeable program into a diagram is the automation of requirements traceability. When there are model changes, because there is a single source of information, it's no longer necessary to find and update the associated requirements in each of many documents. Furthermore, patterns are used to constrain designers e.g. ensure a component is only composed of other components, and that a component performs a function.

In SysML, Requirement diagrams can be represented as matrices for providing additional views enabling users to easily create, modify, and analyze requirement relationships. Within SysML, the Satisfy Requirement Matrix provides a compact representation for identifying the components that satisfy each requirement. This is practical when a large number of requirements complicates the Requirements diagram. The Satisfy Requirements Matrix displays traceability and can show the supplier dependency relationships along with the model elements necessary to satisfy requirements. These are desirable features for requirements verification. The dependency relationships in SysML are called Derive, Verify, and Refine [7].

VEHICLE SIMULATOR EXAMPLE

Vehicle simulators used for training typically have an architecture similar to the one shown in Figure 1 where blocks blueprint the components for behaviors. The Simulation blocks consist of navigation, radio communication, vehicle sounds, and a synthetic environment for emulating real world behavior. The Vehicle Fidelity blocks model vehicle dynamics, and provide motion cueing for vehicle control. The Instructor Operator Station (IOS) provides the instructor the means to operate the vehicle simulator while in use by students. Internal and External interfaces are components of the Interfaces block e.g. power and external data. The visual system is composed of image generators, projection, displays and databases for creating the imagery the student would see during the simulation e.g. terrain, ownship, moving models, celestial bodies, weather, lights, and other visual effects. Optimally, this would be a solution neutral vehicle simulator model, meaning that it could be a starting point for more specific solutions.

In the "Provide Simulation" use case, both the human and system actors are represented. Seven use cases are shown, each stated with action verbs. The "<<refine>>" stereotype indicates there is more detail about the use case at the end of the dashed line which is further described in the Activity diagram depicted in Figure 3. This Activity diagram begins with an initial node and ends with an activity final node. The initial node is the solid dot and the activity final node is the solid dot with the circle around it. The rounded edge rectangles in between the initial and the final nodes represent the actions that the system performs to provide the simulation. The dotted lines represent control flow in between the actions. Swim lanes delineate the responsible actors i.e. the Instructor performs "Select Training Scenario", "Assess Training Results", "Perform Training Review"; the IOS performs "Provide Data for Scenario", "Provide Training Results", the Simulation performs "Generate Scenario Simulation", "Provide Training Cues", and the Student performs "Execute Training", and "Receive Training Review". The "<<allocate>>" symbol indicates that properties of the block have been allocated. One of the nice things about Activity diagrams is that they can be nested.

The Figure 4 Requirements diagram displays the organization and relationships of the vehicle simulator requirements. The numbering shown here follows what would normally be seen in traditional specification documentation. Requirements are named and have an identification (ID) e.g. "Required Modes" is the name and it has an ID of "3.1". Requirements diagrams can depict other attributes such as the verification method. At the top of each requirement is a stereotype which is used for categorization and is typically denoted by "<<requirement>>". Specific notation depicting the relationships between requirements can be for the functions of Containment, Derive, Refine, Satisfy, and Verify. In Figure 4, the function of containment shows sub-requirements in a hierarchy using the circled crosshair symbol e.g. Required Modes contains "Initial mode", "Freeze mode", and "Maintenance mode".

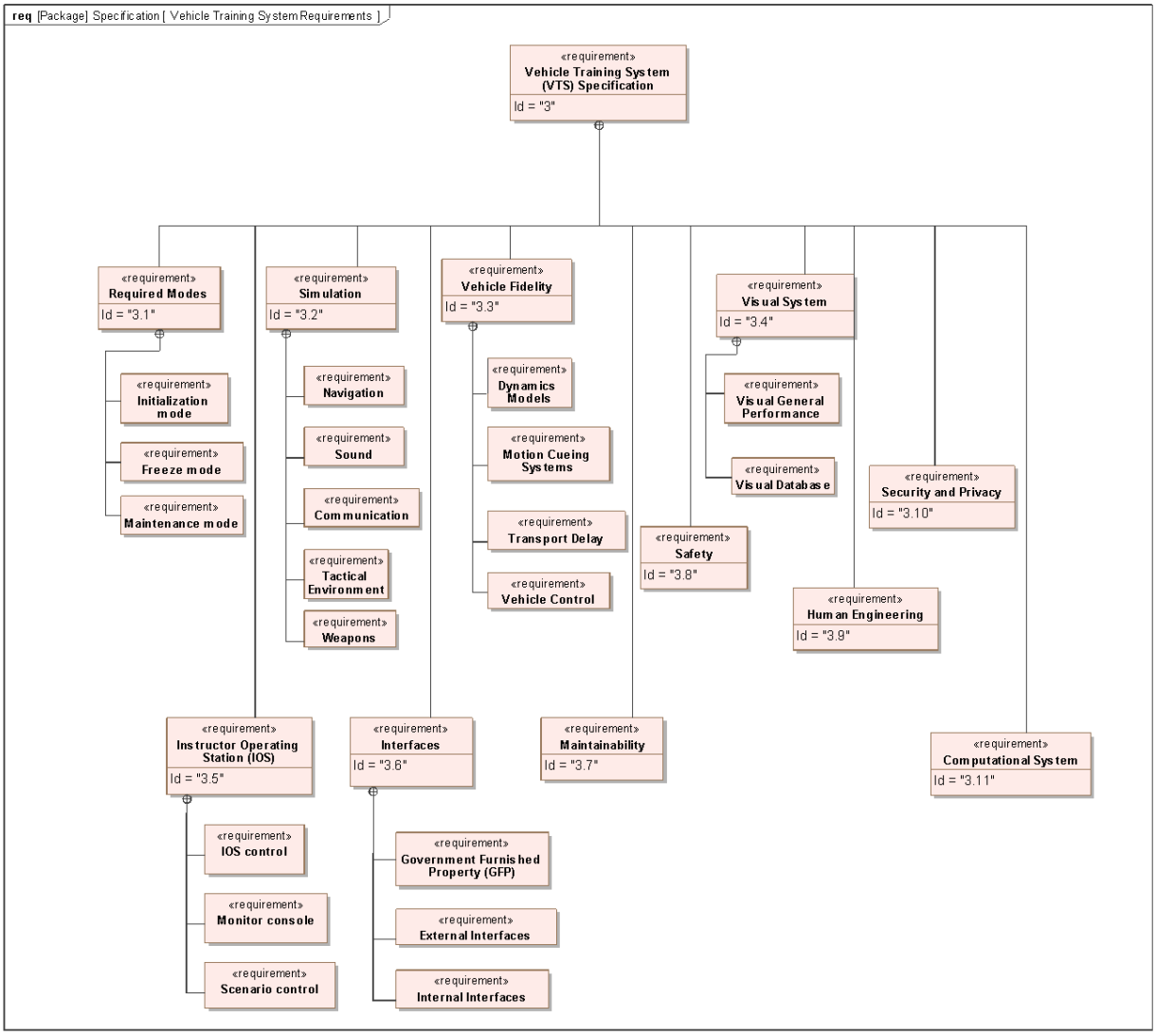

Figure 4. Requirements Diagram for the Basic Vehicle Simulator

DOCUMENTATION

System design information contained within a model that was developed using SysML is the same as that contained within traditional documentation-based methods. Data Item Descriptions (DIDs) define the content and format of design data. DIDs can be prepared in a manner suitable for using SysML diagrams in place of conventional documentation for deliverables such as the System/Subsystem Design Description (SSDD), Requirements Traceability Verification Matrix (RTVM), Software Design Description (SDD), and the Interface Design Description (IDD). Once an acquiring activity establishes a product-line based architecture, designers can then provide the details typically found in these documentation-based data item deliverables directly into a system model using SysML. This method eliminates the practice of manually updating each of many design related documents whenever there are design changes. Figure 5 shows a mapping of the SDD and SSDD to the system design attributes described in those documents and which SysML diagram might be used to convey the same information.

	SDD & SSDD	SDD & SSDD	SDD	SDD	SysML
Function	**System Items**	**Components**	**Units**	**Data Elements**	**Diagrams**
Architecture	re-use components dependencies modeling behavior	considerations units specifications assumptions purpose	conditions elements traceability constraints		**bdd, ibd** **bdd, ibd** **bdd, ibd** **bdd, ibd** **bdd, ibd**
I/O	inputs and outputs user interface	inputs and outputs	interfaces procedures method of control	interfaces	**uc, act, sd, stm** **uc, act, sd, stm** **uc, act, sd, stm**
Control	improper conditions state	status	logic sequence states or modes	 sequencing source destination	**uc, act, sd, stm** **uc, act, sd, stm** **uc, act, sd, stm** **uc, act, sd, stm** **uc, act, sd, stm**
Performance	performance	interrupts resources units measures	response time features	precision type units range accuracy	**par** **par** **par** **par** **par**
Stability	stability	diagnotsic	exception and error handling		**uc, act, req, sd, stm**
Cyber/Security/Safety	 safety security			Cybersecurity safety	**req** **req** **req**
Software System Specific	language algorithms databases memory utilization processor utilization network	 local data, spare spare storage utilization	programming language database memory data frequency	 bandwith	**uc, act, req, par** **uc, act, req, par** **uc, act, req, par** **uc, act, req, par** **uc, act, req, par** **uc, act, req, par** **uc, act, req, par**
Identification		unique identifier	format	project identifier data element name descriptive Name	**req** **req** **req**

Figure 5. Mapping of the SDD and SSDD to SysML

ROLE OF THE ARCHITECT

SysML integrates design information needed for architecting systems and having a centrally located authoritative source of information makes it convenient for an architect to provide a framework for how SysML is used to design systems, analyze architectures, and verify requirements. One of the many roles of the architect is to define what is to be modeled while knowing the boundaries of the desired model. The architect establishes the questions being answered by the model and the risks to be managed. The architect is responsible for defining the patterns the team will use, whether the model will be single model or a federation of models, and the types of views the stakeholders will need to accomplish their roles. Typically, an architect would prepare a model management plan describing how disparate groups access a system model. This ensures each of the disciplines is getting the specific views they need. Another role the architect may take on is leading a governance board. The governance board would be responsible for enforcing the use of patterns, adherence to established rules, setting access permissions to ensure only MBSE certified subject matter experts have read-write access" to for making model changes and that stakeholders have "read access" either directly through the modeling software or by way of a web based application. Analogous to SysML providing the ground truth of information, the architect is like the band conductor who manages to get the designers and stakeholders all playing from the same sheet of music.

CONSIDERATIONS

Four areas of consideration for using MBSE to acquire vehicle training simulations are (1) the business case, (2) the skillsets needed for implementing MBSE, (3) a process change from documentation-based to model-based, and (4) the reuse of models.

Business Case

Transforming an enterprise from traditional documentation-based processes to using MBSE is like trying to reverse the direction of a freight train. Convincing program managers that investing funds in the present may provide avoided cost in the future needs a strong lifecycle business case where their program is likely a new start or a product line that will see a considerable return on investment [8]. There is usually a strong business case for new systems and those that may have many variants, but certainly systems with few projected modifications wouldn't be good candidates for MBSE. Program management buy-in from across the organization is needed to achieve commonality and to yield the cost benefits.

Skillsets

A drawback of MBSE is the potential separation of modeling and engineering whereby the engineer has to rely on a modeler to create a design that's inside the engineer's head. To avoid this, there are several skills to be learned by the engineer each with a considerable learning curve for implementing MBSE. The art of modeling with SysML is more than just learning to use a new tool. Both a systems engineering skillset and the ability to write software are required. In the area of software, engineers need to be knowledgeable with writing scripts and customizing application interfaces. Having both the systems engineering and software skillsets enables a deeper understanding of the underlying mechanics of MBSE which increases the likelihood of making good design decisions that are provisional and for influencing the architecture of a product line in an effective manner.

Process Change

Familiarity with documentation-based methods may be difficult to retrain and the current systems engineering paradigm is documentation-based. The focus of MBSE is having an integrated system model where documents are a natural byproduct rather than a separate task requiring the engineer to write a representation of the design. The documentation-based process increases risk as it typically produces stale information where the most recent version is usually a local copy on someone's computer. With MBSE, the model is accessible to the entire team and requirements can be traced automatically.

Model Reuse

While the vision of automating requirements traceability and having reusable components to reduce rework are desirable attributes of MBSE, there are limitations. Models are essential for engineers to make sound design decisions and for stakeholders to visualize the operation of systems before they are constructed. Models can be mathematical or architectural. Mathematical models have two types, analytical i.e. with simulation, or numerical i.e. without simulation. Mathematical models are typically created in the domain-specific languages used by subject matter experts. Architectural models integrate disparate elements and span requirements definition through verification & validation. The architectural model is the space for inspiring innovation, managing complexity, and influencing system wide characteristics. Moreover, the architectural model is effectively a specification for depicting the system and it integrates mathematical models into a structure. Reuse thrives when architectural models are applied to product lines. With the exception of where there is a product line, the drawback with architectural models is that they are much less conducive to reuse than mathematical models as they predominantly require a broader scope with an overhead investment to make them extensible. Furthermore, with agile processes, development not directly connected with the project is wasteful thus making extensibility even more difficult to achieve [1]. For MBSE to be successful within an industry, there needs to be a community for sharing experiences to establish common practices and patterns of usage. Patterns and standard interfaces simplify how information is exchanged and are essential for reuse. Patterns defined by reference architectures such as DoD Architecture Framework (DoDAF) and The Open Group Architectural Framework (TOGAF) create inherently reusable models. The acquisition of systems with DoDAF and TOGAF reference frameworks is an emerging strategy where the Request for Proposal (RFP) provides the foundational framework of a model and submitted proposals become the manifestation of performance requirements in the form of a SysML model for consideration.

CONCLUSIONS

Taking a solution neutral MBSE approach, an architecture for vehicle training simulations was introduced. MBSE using SysML makes sense when systems are being designed for the first time, for systems with many potential variants, and when there is the potential for reuse of common components. Given the similarity across vehicle training simulation architectures, the application of MBSE for acquisition of these systems is seemingly ideal and the practice of managing requirements is at the core for ensuring systems are acquired correctly. The method for establishing requirements in SysML, is to specify use cases with Activity diagrams. The intertwining of Requirements diagrams and textual requirements matrices with many of the diagrams stakeholders use and having standardized semantics improves communication across multiple disciplines further improving how systems are acquired. A key aspect of SysML for acquisition is the inherent interconnectivity of information within SysML modeling tools for automating requirements traceability whereby eliminating the practice of manually updating each of many traditional acquisition design related documents such as the SDD and the SSDD whenever there are design changes. Overall, MBSE brings order to systems engineering. The systems engineer is the connective tissue that unifies subject matter experts in solving system problems and the system architect is like the band conductor for getting the team playing from the same sheet of music. While each may need to rely on others with the ability to create models and write software to integrate disparate components, their skillsets are complimentary. Ideally, having both the systems engineering and software skillsets enables a deeper understanding of the underlying mechanics of MBSE which increases the likelihood of making good design decisions. And, good design decisions increase the potential for reuse, a main goal of MBSE. Generally, the reusability of mathematical models is higher than that of structural models, and where there is a product-line, the potential for reuse of architecual models is increased. Lastly, patterning with reference architectures such as DoDAF and TOGAF improves the potential for reusability in the acquisition of systems.

REFERENCES

[1]. Friedenthal S., Wolfrom J., (2010), *Modeling with SysML*, INCOSE 2010.

[2]. Long D, (2016), *One Model to Coordinate them All*, http://community.vitechcorp.com/index.php/One-Model-to-Coordinate-them-All.aspx,

[3]. Nolan B., Brown B., Balmelli L., Bohn T., Wahli U., (2008), *Model Driven Systems Development with Rational Products*, http://ibm.com/redbooks

[4]. Friedenthal S., Moore A., Steiner R., (2009), *OMG Systems Modeling Language (OMG SysML™) Tutorial*, Object Management Group, http://www.omgsysml.org/INCOSE-OMGSysML-Tutorial-Final-090901.pdf,

[5]. Douglas B., (2015), *Harmony MBSE Modeling Standards for use with UML, SysML, and Rhapsody*, https://www.bruce-douglass.com/

[6]. Deligatti L., (2014), *SysML Distilled*, Addison-Wesley, Pearson Education Inc

[7]. Roques P., *Modeling Requirements with SysML*, Requirements Engineering Magazine, https://re-magazine.ireb.org/articles/modeling-requirements-with-sysml

[8]. Mitchell S.W., (2019), *Model-Based System Development for Managing the Evolution of a Common Submarine Combat System*, Lockheed Martin

DISCLAIMER

The views of the author expressed herein do not necessarily represent those of the U.S. Navy or Department of Defense (DoD). Presentation of this material does not constitute or imply its endorsement, recommendation, or favoring by the DoD.