28th annual INCOSE international symposium
Washington, DC, USA
July 7 - 12, 2018

Evolving Model-Based Systems Engineering Ontologies and Structures

Warren K. Vaneman, Ph.D.
Naval Postgraduate School
Monterey, CA
wvaneman@nps.edu

Abstract. Model-Based Systems Engineering (MBSE) is a mysterious concept that means many different things to different stakeholders. MBSE was envisioned to manage the increasing complexity within systems, by replacing traditional document-based system engineering with a model-based approach. However, more than a decade after MBSE was introduced, many systems engineering efforts still default to a "document-like view" rather than integrated, "virtual," representation of the system. This paper suggests a revised definition for MBSE which supports system design and analysis, throughout all phases of the system lifecycle, and through the collection of modeling languages, model-based processes, structures, and presentation frameworks used to support the discipline of systems engineering in a model-based or model-driven context. To realize this definition, and an environment where the system is virtually represented, the long-sought ontology must be attained for better definition and structure within MBSE. This paper explores how current MBSE methods can be extended to include an ontology.

Introduction

Today's systems are more complex, and change more rapidly than ever before. As sub-systems are added to systems, and systems are added to system of systems, interfaces grow nonlinearly. As a result, interfaces and interactions are often difficult to comprehend, with cascading effects leading to an uncertain and incomplete architecture that fails to account for emerging issues within the system. The challenge is to deal with this increased complexity within the systems environment.

Model-Based Systems Engineering (MBSE) was conceived by the International Council on Systems Engineering (INCOSE) to transform systems engineering by replacing the traditional document-based approach with a model-based approach, where the models can be easily tailored to changing conditions and needs, re-used, and have the ability to observe the system architecture from a static perspective, or explore system behavior from a dynamic perspective.

However, more than a decade after it was defined, MBSE remains a mysterious concept that means many things to many different people and organizations. Some will contend that MBSE is the Unified Modeling Language (UML) or Systems Modeling Language (SysML), while others believe that MBSE is the Department of Defense Architecture Framework (DoDAF) or the Zachman Framework. As a result, MBSE is frequently implemented as a series of "architectural products," instead of a singular virtual representation of the system that can be used for a wide array of systems engineering analysis. This due, in part, to lack of an adopted ontology that defines entities, and provides the structure, used to represent systems.

The purpose of this paper is twofold. First it provides a revised definition that addresses the multiple facets, that some singularly believe defines, MBSE. Second, the next-generation of MBSE is examined with respect to a long-sought ontology. This ontology provides a better definition and

structure for MBSE languages. Regardless of the herculean efforts to date, MBSE must continue to move the sub-disciple forward to better support the engineering and management of system complexity.

MBSE Defined and Re-Defined

Two MBSE definitions have been prominent during the past decade. INCOSE (2007) initially defined MBSE as:

> "the formalized application of modeling to support system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout the later lifecycle phases."

Later, Long and Scott (2011) introduced a new definition that stated:

> "MBSE is fundamentally a thought process. It provides the framework to allow the systems engineering team to be effective and consistent right from the start of any project. At the same time, it is flexible enough to allow the 'thought' process to adapt to special constraints of circumstances present in the problem."

While both definitions advanced the nascent concept, neither captured both the full-scope of MBSE or address the elements required for implementation. The INCOSE definition captured the lifecycle perspective, but does not give any indication how MBSE is different than traditional system engineering given that "models" have always been a cornerstone of the discipline. Long and Scott's definition suggest a "framework" but does not discuss the framework elements.

This paper defines MBSE as the formalized application of modeling (static and dynamic) to support system design and analysis, throughout all phases of the system lifecycle, through the collection of modeling languages, structures, model-based processes, and presentation frameworks used to support the discipline of systems engineering in a model-based or model-driven context (Vaneman, 2016). The four tenets of this definition are:

- Modeling Languages – Serves as the basis of tools, and enable the development of system models. Modeling languages are based on a logical construct (visual representation) and/or an ontology.
- Structure – Uses the ontology, and defined relationships between the systems entities, to establish concordance, thus allowing for the emergence of system behaviors and performance characterizations within the model.
- Model-Based Processes – Provides the analytical framework to conduct the analysis of the system virtually defined in the model. The model-based processes may be traditional systems engineering processes such as requirements management, risk management, or analytical methods such as discrete event simulation, systems dynamics modeling, and dynamic programming.
- Presentation Frameworks - Provides the framework for the logical constructs of the system data in visualization models that are appropriate for the given stakeholders. These visualization models take the form of traditional systems engineering models. These individual models are often grouped into frameworks that provide the standard views and descriptions of the models, and the standard data structure of architecture models.

Maximum effectiveness occurs at the convergence of the four MBSE tenets in Figure 1. This is where most MBSE tool vendors strive to create a proper balance of the tenets to develop their modeling environments.

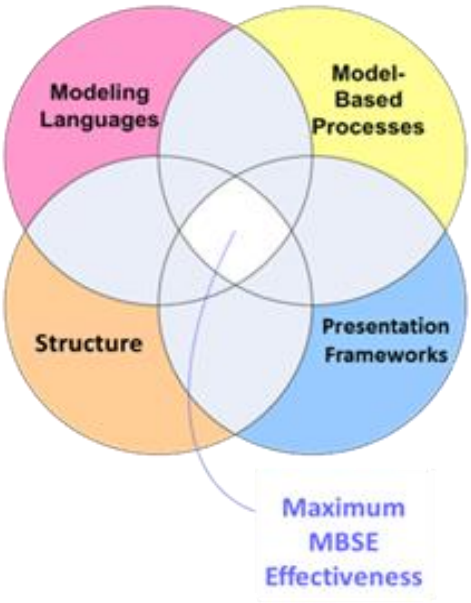

Figure. 1. Four Tenets of MBSE.

The Essence of MBSE: Language and Structure

The fundamental objective of systems engineering is to facilitate a process that consistently leads to the development of successful systems (Long and Scott, 2011). MBSE was envisioned to transform systems engineering's reliance on document-based work products to an engineering environment based on models. This transformation means more than using model-based tools and processes to create hard-copy text-based documents, drawings, and diagrams. Data in a MBSE environment is ideally maintained within a single repository, where each entity has a singular definition, with all necessary attributes and relationships of the entities being portrayed. This allows for the static and dynamic representations of the virtual system from several different engineering and programmatic perspectives, and levels of decomposition.

The foundation of the MBSE environment are the modeling languages that enable the tools. Modeling languages are based on a visual representation (logical construct), an ontology, or both. Ontology is a collection of standardized, defined terms and relationships between the terms to capture the information that describes the physical, functional, performance, and programmatic aspects of a system (LML Steering Committee, 2015). While the languages serve as the foundation for MBSE tool development, tool vendors often interpret the languages to enable the best implementation for the goals of their tools. While a common language is used, the tools can be very different, with limited ability to effectively exchange data. For example, MBSE tools that are based on SysML support a common set of visual models, but typically have unique data schemas that do not easily facilitate data exchange between other tools (Vaneman, 2016).

Systems have structures that consist of "building blocks" and their relationships to each other that allows them to come together in a designed form that satisfies the desired system capabilities and functionality. These structures are governed by the property of concordance. Concordance is the ability to represent a single entity, such that data in one view, or level of abstraction, matches the data in another view, or level of abstraction, when talking about the exact same thing. This allows for complexity to be managed more efficiently because each entity is ideally represented in the model only once, essentially creating a virtual representation of the system in the model. Systems engineering views are generated from the data.

Heretofore, the systems engineering community has made great progress in moving towards a model-based environment. However, current MBSE implementations often do not apply model ontologies and structures efficiently, thereby causing data to be represented in the model more than once. This results in data maintenance errors, leading to different representations of the same system entity in various viewpoints. In a practical sense, this leads to conflicting system requirements.

The leading MBSE language is SysML, a general-purpose graphical modeling language for supporting requirements, design, analysis and validation and verification, for systems that include hardware, software, information, process, and people (Vaneman, 2016). The SysML taxonomy is composed of nine models to represent behaviors, structures, requirements and parametrics (Figure 2). SysML has made tremendous progress towards the MBSE vision, but the lack of ontology hinders the language from representing precise complex system relationships and interactions, and forces tool developers to define data schemas for tool execution (Vaneman, 2016).

As an attempt to advance MBSE, the Lifecycle Modeling Language (LML) was developed as an approach for incorporating visualization models and corresponding ontologies within the same framework. By doing so, it provides the structure that expresses a wide range of entities, relationships, and attributes to capture systems engineering and programmatic information.

LML defines visualization models in four areas: functional models; physical models; documentation entities; and parametric and program entities (Figure 3) (LML Steering Committee, 2015). Many of the LML models are equivalent to the familiar models that have been developed over time by SysML, Business Process Modeling Notation (BPMN), and other engineering disciplines. LML uses a simplified ontology of 12 primary entities and eight secondary entities to capture the system characteristics, relationships, and interactions. LML's primary and secondary entities are (LML Steering Committee, 2015):

- Asset – An Asset entity specifies an object, person, or organization that performs Actions, such as a system, sub-system, component, person, or element. Asset is the parent to the Resource child entity (e.g. component, service, sub-system, system);
- Resource – A Resource is a child entity of Asset that specifies a consumable or producible Asset (e.g. fuel, ammunition, people);
- Characteristic – A Characteristic entity specifies properties of an entity. Characteristic is the parent for the Measure child entity (e.g. category, attribute, power, size, weight);

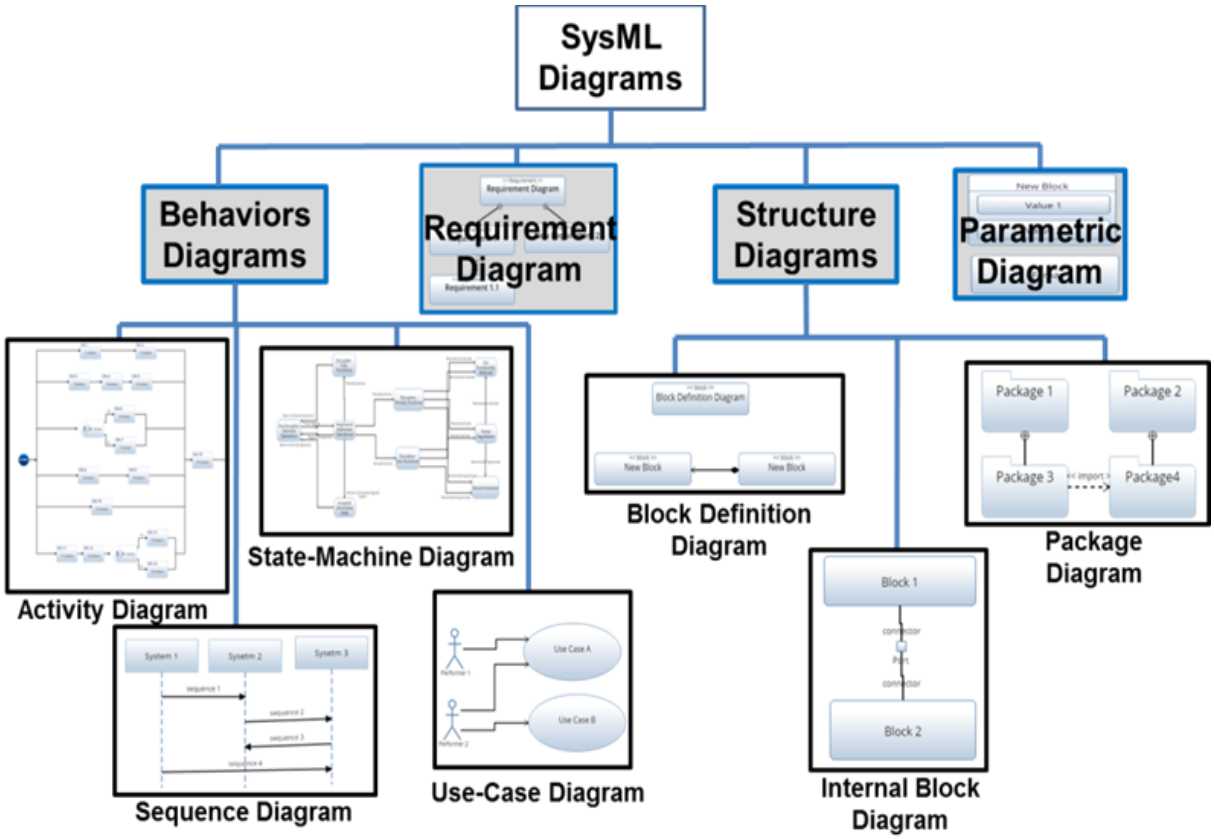

Figure 2. SysML Diagram Taxonomy.

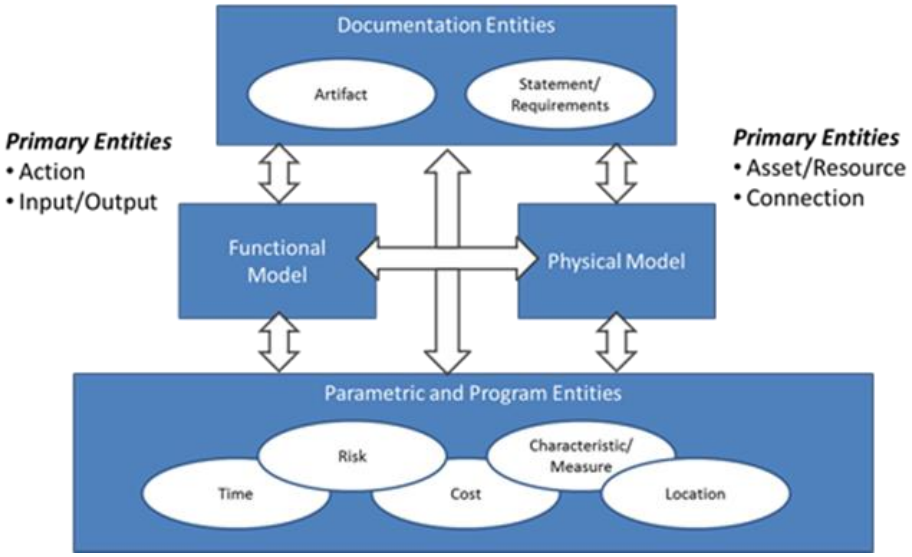

Figure 3. Categories of LML Models.

- Measure – A Measure is a child entity to Characteristic that specifies properties of measurement and measuring methodologies, including metrics (e.g. Measures of Effectiveness (MOE), Key Performance Parameters (KPP), Measures of Performance (MOPS));
- Connection – A Connection entity specifies the means for relating Asset instances to each other. Connection is the parent to the Conduit and Logical child entities (e.g. abstract entity);
- Conduit – A Conduit is a child entity to Connection that provides a means for physically transporting Input/Output entities between Asset entities. Conduits are constrained by the attributes of capability and latency (e.g. data bus, communications interface, and pipe);
- Logical – A Logical entity is a child entity to Connection that represents the abstraction of the relationship between any two entities (e.g. “Has”, “is a”, “relates to”);
- Cost – A Cost entity specifies the outlay of expenditure made to achieve an objective associated with another entity (e.g. Earned Value, Work Breakdown Structure, actual cost, planned cost);
- Decision – A Decision entity specifies a challenge and its resolution (e.g. major decision, challenge, issue, problem);
- Input/Output – An Input/Output entity specifies the information, data, or object input to, trigger, or output from an Action (e.g. item, trigger, information, data, matter, energy);
- Location – A Location entity specifies where an entity resides. Location is the parent entity for Orbital, Physical, and Virtual child entities (e.g. abstract entity);
- Orbital – An Orbital entity is a child entity of Location that specifies a location along an orbit around a celestial body (e.g. orbit, ephemeris);
- Physical – A Physical entity is a child entity of Location that specifies the location on, above, or below the surface (e.g. geospatial coordinates, altitude, depth);
- Virtual - A Virtual entity is a child entity of Location that specifies a location within cyber space or a physical network (e.g. Uniform Resource Locator);
- Risk – A Risk entity specifies the combined probability and consequence in achieving objectives (e.g. cost risk, schedule risk, technical risk);
- Statement – A Statement entity specifies text referenced by the knowledgebase and is usually contained in an Artifact. Statement is the parent entity to the Requirement child entity (e.g. need, goal, objective, assumption);
- Requirement – A Requirement is a child entity to Statement that identifies a capability, characteristic, or quality factor of a system that must exist for the system to have value and

utility to the system or user (e.g. functional requirement, performance requirement, safety requirement);
- Time – A Time entity specifies a point or period when something occurs or during which an action, asset, process, or condition exists or continues (e.g. milestone, phase).

Two additional entities (equation and port) are being considered for inclusion in the LML Specification. These entities, originally defined by SysML provide the LML schema with further opportunities for modeling systems, and collaboration with other modeling languages and tool.

- Equation - An Equation entity specifies a mathematical or logical equation that can be used to describe a portion of the model;
- Port - A Port entity is a child entity of Asset that represents an interaction point of a block, specifying the input and output flow.

A fundamental tenet of LML is that each entity has at least one corresponding visualization. The language contains three mandatory diagrams: Action diagrams for functional modeling; Asset diagrams for physical modeling; and, Spider diagrams for traceability. All other visualizations are optional, but recommended, to represent the full set of LML entities. Additional visualizations are possible due to the flexibility and extensibility of LML. Table 1 provides of a list of possible visualization models that can be used to represent LML entities (Vaneman, 2016).

Table 1. LML Entities and Corresponding Visualization Models.

LML Entity	LML Model
Action	Action Diagram
Artifact	Photo, Diagram, etc.
Asset	Asset Diagram
Resource (Asset)	Asset Diagram
Port (Asset)	Asset Diagram
Characteristic	State Machine, Entity-Relationship, and Class Diagrams
Measure (Characteristic)	Hierarchy, Spider, and Radar Charts
Connection	Asset Diagram
Conduit (Connection)	Asset Diagram
Logical (Connection)	Entity-Relationship Diagram
Cost	Pie/Bar/Line Charts
Decision	
Input/Output	State Machine Diagram
Location	Map
Physical (Location)	Geographic Maps
Orbital (Location)	Orbital Charts
Virtual (Location)	Network Maps
Risk	Risk Matrix
Statement	Hierarchy and Spider Charts
Requirement (Statement)	Hierarchy and Spider Charts
Time	Gantt Chart, Timeline Diagram
Equation	*Equation*

Figure 4 shows LML's principal entities and relationships. LML was designed to support the systems engineering lifecycle (i.e. the traceability of requirements to their implementation to system elements (Assets)) (LML Steering Committee, 2015). The systems engineering process begins with the communication of a need, or capability, typically in the form of an authoritative document (Artifact). Statements of need (Statements or Requirements) are contained in the documents, and serve as the basis for the beginning of the systems architecting and engineering process. Functional models define operational activities (Actions) that the system must perform based on the system's inputs, controls, and outputs (Input/Output). Operational activities are performed by systems, sub-systems, and components (Asset or Resources) and are connected through a series of physical interfaces (Connection or Conduit). Systems have physical properties (e.g. size, weight, power required) that are described by Characteristic or Measure entities (Vaneman, 2016).

The LML ontology is an entity, relationship, and attribute (ERA) data schema. An attribute is an inherent characteristic or quality, that further describes an entity. The ERA approach allows for the efficient use of entities due to the attributes and relationships defined (LML Steering Committee, 2015). As such, LML can represent the complexity of systems with only 12 primary entities. Table 2 (LML Steering Committee, 2015) depicts the relationships between the entities. It is these relationships that allow a modeling language with 12 primary entities to comprehensively represent the complexities in today's systems. Note that the same verb is used for the inverse relationships.

Evolving MBSE with a Common Ontology

The INCOSE Systems Engineering Vision 2025 (INCOSE, 2014, p. 38) describes the future MBSE state as: "Formal systems modeling is a standard practice for specifying, analyzing, designing, and verifying systems, and is fully integrated with other engineering models. System models are adapted to the application domain, and include a broad spectrum of models for representing all aspects of systems. The use of internet-driven knowledge representation and immersive technologies enable highly efficient and shared human understanding of systems in a virtual environment that span the full lifecycle from concept through development, manufacturing, operations, and support."

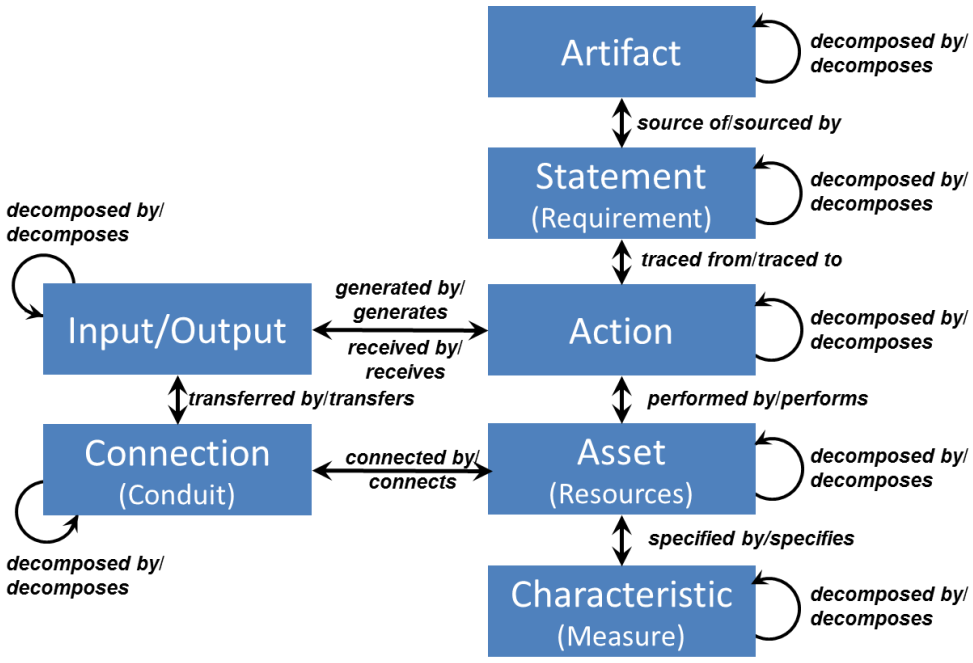

Figure 4. Principal Entities and Relationships for Design LML Traceability.

Table 2. Summary Table of LML Relationships.

	Action	Artifact	Asset (Resource)	Characteristic (Measure)	Connection (Conduit, Logical)	Cost	Decision	Input/Output	Location (Orbital, Physical, Virtual)	Risk	Statement (Requirement)	Time
Action	decomposed by* related to*	references	(consumes) performed by (produces) (seizes)	specified by	-	incurs	enables results in	generates receives	located at	causes mitigates resolves	(satisfies) traced from (verifies)	occurs
Artifact	referenced by	decomposed by* related to*	referenced by	referenced by specified by	defines protocol for referenced by	incurs referenced by	enables referenced by results in	referenced by	located at	causes mitigates referenced by resolves	referenced by (satisfies) source of traced from (verifies)	occurs
Asset (Resource)	(consumed by) performs (produced by) (seized by)	references	decomposed by* orbited by* related to*	specified by	connected by	incurs	enables made responds to results in	-	located at	causes mitigates resolves	(satisfies) traced from (verifies)	occurs
Characteristic (Measure)	specifies	references specifies	specifies	decomposed by* related to* specified by*	specifies	incurs specifies	enables results in specifies	specifies	located at specifies	causes mitigates resolves specifies	(satisfies) spacifies traced from (verifies)	occurs specifies
Connection (Conduit, Logical)	-	defined protocol by references	connects to	specified by	decomposed by* joined by* related to*	incurs	enables results in	transfers	located at	causes mitigates resolves	(satisfies) traced from (verifies)	occurs
Cost	incurred by	incurred by references	incurred by	incurred by specified by	incurred by	decomposed by* related to*	enables incurred by results in	incurred by	located at	causes incurred by mitigates resolves	incurred by (satisfies) traced from (verifies)	occurs
Decision	enabled by result of	enabled by references result of	enabled by made by responded by result of	enabled by result of specified by	enabled by result of	enabled by incurs result of	decomposed by* related to*	enabled by result of	located at	causes enabled by mitigated by result of resolves	alternative enabled by traced from result of	date resolved by decision due occurs
Input/Output	generated by received by	references	-	specified by	transferred by	incurs	enables results in	decomposed by* related to*	located at	causes mitigates resolves	(satisfies) traced from (verifies)	occurs
Location (Orbital, Physical, Logical)	locates	locates	locates	locates specified by	locates	locates	locates	locates	decomposed by* related to*	locates mitigates	locates (satisfies) traced from (verifies)	occurs
Risk	caused by mitigated by resolved by	caused by mitigated by references resolved by	caused by mitigated by resolved by	caused by mitigated by resolved by specified by	caused by mitigated by resolved by	caused by incurs mitigated by resolved by	caused by enables mitigated by results in resolved by	caused by mitigated by resolved by	located at mitigated by	caused by* decomposed by* related to* resolved by*	caused by mitigated by resolved by	occurs mitigated by
Statement (Requirement)	(satisfied by) traced to (verified by)	references (satisified by) sourced by traced to (verified by)	(satisified by) traced to (verified by)	(satisfied by) specified by traced to (verified by)	(satisfied by) traced to (verified by)	incurs (satisified by) traced to (verified by)	alternative of enables traced to results in	(satisfied by) traced to (verified by)	located at (satisfied by) traced to (verified by)	causes mitigates resolves	decomposed by* traced to* related to*	occurs (satisified by) (verified by)
Time	occurred by	occurred by	occurred by	occurred by specified by	occurred by	occurred by	date resolves decided by occurred by	occurred by	occurred by	occurred by mitigates	occurred by (satisfies) (verifies)	decomposed by* related to*

To achieve this vision, the next-generation modeling language must include the following characteristics (Friedenthal and Burkhart, 2015):

- Long-sought ontology, coupled with visual representations, to more fully represent system types, properties, and interrelationships;
- Robust data model that expresses core systems engineering concepts and processes;
- Precise definitions and semantics to provide an unambitious and concise data representation;
- Flexible visualization models to support a wide spectrum of stakeholders;
- Adaptable and customizable to support a wide-range of engineering and programmatic issues.

Coupling SysML with LML could immediately serve as the foundation to achieve the long-sought ontology needed for the next generation modeling language. This coupling begins with exploring commonality, and creating a mapping between the SysML and LML visualization models. Once mapped, the LML visualization models can be associated to the corresponding LML entity, and by extension be associated with the appropriate SysML visualization model, thereby providing an ontology for SysML. This coupling will also allow SysML visualizations to more precisely represent system concepts, especially where complexity is involved, and allow for better presentation of system data. This common ontology will also allow for better interoperability among various MBSE tools that use SysML, by reducing the need for proprietary ontologies. Table 3 depicts the mapping between the SysML diagrams and the LML entities (LML Steering Committee, 2015; Vaneman, 2016).

Heretofore, the MBSE languages have supported primarily the systems engineering community. However, Friedenthal and Burkhart (2015) recognized the need for a broader range of related disciplines. LML has taken initial steps to meeting a broader range of stakeholders throughout the system's lifecycle by defining entities that support systems architecting and program management. Table 4 maps the information needed by key stakeholders to the corresponding LML entities (LML Steering Committee, 2015). These entities have corresponding LML visualization models, as shown in Table 1, which could be used to further expand the SysML suite of visualization models.

Table 3. Mapping of SysML Diagrams to LML Diagrams and Entities.

SysML Diagrams	LML Models	LML Entities
Activity	Action Diagram	Action, Input/Output
Sequence	Sequence	Action, Asset
State Machine	State Machine	Characteristic (State), Action (Event)
Use Case	Asset Diagram	Asset, Connection
Block Definition	Class Diagram, Hierarchy Chart	Input/Output (Data Class), Action (Method), Characteristic (Property)
Internal Block	Asset Diagram	Asset, Connection
Package	Asset Diagram	Asset, Connection
Parametric	Hierarchy, Spider, Radar	Characteristic
Requirement	Hierarchy, Spider	Requirement and related entities

Table 4. Key Stakeholder Information Needs and Corresponding LML Entities.

Systems Engineering	Architecture	Program Management	LML Entity
Cost	How (much)	Cost	Cost
Schedule	When	Schedule	Time/Action
Performance			Equation
Form	Who	Organization	Asset
	What	Resource	Resource, Asset
	Where	Location	Location (Physical, Orbital, Virtual)
Requirement	Why	Goal, Objective, & Decisions	Decision, Statement (Requirement)
Function	How	Task	Action
Metric		Metric	Characteristic (Measure)
Interface			Connection (Conduit), Input/Output
Risk		Risk	Risk
		Artifact	Artifact

Conclusion

Model-Based Systems Engineering currently focuses on the information needs of the systems engineering and architect, primarily in the conceptual and preliminary design phases. These processes are used to pass the system specification models (i.e. requirements) to the detailed design phase, but other models generated in the early lifecycle phases (i.e. capability, operational, and system models) are often not used, or conveyed during the detailed design, or other later phases of the system's lifecycle. The failure to use these models deprives engineers working in the later phases of lessons learned by the engineers earlier in the lifecycle.

For MBSE to be effective, model-based processes must replace traditional systems engineering processes. This does not mean that system engineering will be supplanted by MBSE, because MBSE is an enabler for better systems engineering. As with any new methodology, organizations must adopt new processes to be truly effective. This does not only include a fundamental change to systems engineering processes, but also to how authoritative data is configuration controlled, and how gate reviews are conducted,

The systems engineering community has put a lot of stock into incorporating MBSE into the discipline, with moderate success. It is time to further evolve MBSE by joining the traditional visual-based modeling languages with an ontology to provide a better definition, structure, and concordance within the virtual representation of the system. The community has long-envisioned an ontology for SysML. This ontology is one of the key upgrades that is being sought with SysML 2.0. However, an ontology is already possible by relating SysML to the LML ontology. This has already been implemented with SPEC Innovations Innoslate tool, and could be easily achieved with other MBSE tools. While this may not be the solution envisioned for SysML 2.0, it does make a major step towards achieving the goals of the next generation MBSE language as espoused by Friedenthal and Burkhart (2015).

References

Friedenthal S. and R. Burkhart, 2015, 'Evolving SysML and the System Modeling Environment to Support MBSE', *INSIGHT* 18 (2), 39-41.

International Council on Systems Engineering., 2007, 'Systems Engineering Vision 2020', INCOSE, San Diego, CA (US).

International Council on Systems Engineering., 2014, 'A World in Motion: Systems Engineering Vision 2025', INCOSE, San Diego, CA (US).

Lifecycle Modeling Language (LML) Steering Committee, 2015 'Lifecycle Modeling Language (LML) Specification', http://www.lifecyclemodeling.org/spec/1.1.

Long, D. and Z. Scott 2011, 'A Primer for Model-based Systems Engineering (2nd Edition),' Vitech Corporation, Blacksburg, VA (US).

Vaneman, W. K. , 2016, '*Enhancing Model-Based Systems Engineering with the Lifecycle Modeling Language*', Proceedings of the10th Annual IEEE Systems Conference, Orlando, FL (US).

Biography

Warren Vaneman is a Professor of Practice in the System Engineering Department at the Naval Postgraduate School. He has more than 30 years of leadership and systems engineering experience from various positions within the Intelligence Community and Department of Defense. He is a Retired Navy Reserve Captain, and has a B.S. from the State University of New York Maritime College, a M.S. in Systems Engineering, and a Ph.D. in Industrial and Systems Engineering from Virginia Tech. He is a Certified Systems Engineering Professional.