2014 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
SYSTEMS ENGINEERING (SE) TECHNICAL SESSION
AUGUST 12-14, 2014 – NOVI, MICHIGAN

MODEL-BASED PRODUCT LINE ENGINEERING – ENABLING PRODUCT FAMILIES WITH VARIANTS

Matthew Hause
Atego Chief Consulting Engineer
Camelback Center, 2355 E. Camelback Rd. Suite 615, Phoenix, AZ 85016, USA

ABSTRACT

Product Lines are a group of related products manufactured or produced within or between collaborating organizations. To effectively manage a product line, one needs to understand both the similarities and differences between the different products and optimize the development lifecycle to leverage the similarities, and concentrate development on the differences. ISO 26550:2013 Software & Systems Engineering – Reference Model for Product Line Management & Engineering provides a standard for defining these similarities and differences as well as the choices between them. Model-Based Systems and Software Engineering (MBSE) using the Systems Modeling Language (SysML) and the Unified Modeling Language (UML) provide a means of modeling systems and software. Bringing the two together allows users to model product lines in industry standard formats. Combining these with an execution engine means that product models can be created for specific products, whilst maintaining the original product line model. This provides significant ROI for ground vehicles.

INTRODUCTION

Product lines have existed since the industrial revolution. Manufacturers have long employed Product Line Engineering (PLE) techniques to create a product line of similar products using a common factory that assembles and configures parts designed to be reused across the product line. Automotive manufacturers create unique variations of a car model using sets of carefully designed parts and a factory specifically designed to configure and assemble those parts. Henry Ford was one of the first manufacturers to do this on a grand scale using assembly lines as well as interchangeable parts. However, this capability evolved over a period of time. Manufacturers would create a single product for a specific purpose or customer. Variations of the product would be created when customers' needs changed or to improve production. Eventually, these would evolve into product lines. Often the management of the product line depended on the skill and memory of the chief production engineer. Over time, engineering techniques would be employed to create lines of similar products by allowing for specialization and customization as well as leveraging interchangeable parts. This helped to drive down manufacturing costs and increase customer choice. However, component dependencies, mutually exclusive components, component trade-off studies, etc. can be difficult to manage, maintain, and document. These challenges increase complexity and diminish the significant ROI of PLE. Systems and Software Model-based Product Lines are a similar paradigm. However, software product variants were normally created by using conditional compilation and similar build and runtime techniques. This was error prone and difficult to visualize. It was also too late in the process as the earlier you consider commonality and variation in the Product lines lifecycle the greater the ROI. Model-based techniques will be necessary to alleviate these issues in the same way that they are revolutionizing other aspects of systems and software engineering. Using automotive examples, this paper will describe Model-based Product Line Engineering, the process for creating product lines, the 150% model, variant modeling and mapping variation systems. Finally the paper will describe software analysis, variant feature selection, product model creation, and the benefits of this approach as applicable to the military ground vehicle domain.

The Shift towards Models

Engineers prefer to create models of systems to visualize systems. Model-Based Systems and Software Engineering (MBSE) techniques have become the industry norm for expressing systems and software architectures. The INCOSE

SE Vision 2020 [9] defines 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 development and later life cycle phases. MBSE is part of a long-term trend toward model-centric approaches adopted by other engineering disciplines, including mechanical, electrical and software. In particular, MBSE is expected to replace the document-centric approach that has been practiced by systems engineers in the past and to influence the future practice of systems engineering by fully integrating into the definition of systems engineering processes." Applying MBSE typically provides significant benefits over document centric approaches by enhancing productivity and quality, reducing risk, and providing improved communications among the system development team. [9] Systems of systems are defined using Architecture frameworks such as the Department of Defense Architecture Framework (DoDAF) [1], systems architectures using the Systems Modeling Language (SysML) [2], and [3] and software architectures using the Unified Modeling language (UML) [4].

ELEMENTS OF SYSML

SysML is more than a diagramming notation. It also defines relationships between and properties of the elements which are represented on those diagrams. While it is useful to start by using a whiteboard or 'drawing' tool such as Visio, to reap the rewards you will need to use a tool which provides both the underlying database and the diagrams which provide views onto that data. The SysML diagrams can be used to specify system requirements, behavior, structure and parametric relationships. These are known as the four pillars of SysML. The system structure is represented by Block Definition Diagrams and Internal Block Diagrams (Blocks are defined later in this paper). A Block Definition Diagram describes the system hierarchy and system/component classifications. The Internal Block Diagram describes the internal structure of a system in terms of its Parts, Ports, Interfaces and Connectors. Parts are the constituent components or "Parts" that make up the system defined by the Block. Interfaces define the access points by which Parts and external systems access the Block. Connectors are the links or associations between the Parts of the Block. Often these are connected via the Ports.

The behavior diagrams include the Use Case Diagram, Activity Diagram, Sequence Diagram and State Machine Diagram. A Use Case Diagram scopes the context and provides a high-level description of the system functionality. A Sequence Diagram represents the multiple interactions between collaborating Parts of a system. The Activity Diagram represents the flow of data and control between Activities. Activities represent behaviors or functionality in the system. This is similar to function block diagrams. The State Machine Diagram describes the state transitions and actions that a system or its parts performs in response to events. The Four Pillars are shown in Figure 1

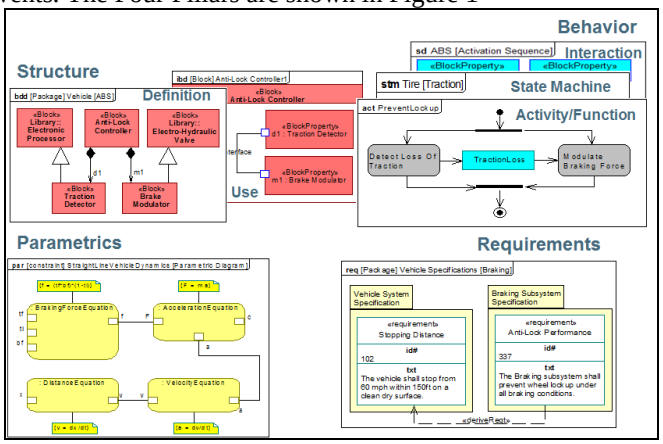

Figure 1. The Four Pillars of SysML

The Requirement Diagram captures requirements hierarchies and the derivation, satisfaction, verification and refinement relationships. The relationships provide the capability to relate requirements to one another and to relate requirements to system design model elements and test cases. The requirement diagram can provide a bridge between typical requirements management tools and the system models or be used for model based requirements engineering independently. The parametric diagram represents constraints on system parameter values such as performance, reliability and mass properties to support engineering analysis. Finally, the Package Diagram is used to organize the model. SysML includes an allocation relationship to represent various types of allocation including allocation of functions to components, logical to physical components and software to hardware.

Structural Elements of SysML

The major structural element in SysML is the «block» which extends the UML Structured Class. It is a general purpose hierarchical structuring mechanism that abstracts away much of the software-specific detail implicit in UML structured classes. Blocks can represent any level of the system hierarchy including the top-level system, a subsystem, or logical or physical component of a system or environment. A SysML block describes a system as a collection of parts and connections between them that enable communication and other forms of interaction. Ports provide access to the internal structure of a block for use when the object is used within the context of a larger structure. Two diagrams are used to describe block relationships. The Block Definition Diagram (bdd), similar to a UML class diagram, is used to describe relationships that exist between blocks.

The Internal Block Diagram (ibd) is used to describe block internals.

Modeling Systems

These modeling languages provide a means of expressing systems and software architectures at all virtually of levels of detail and abstraction. However, they lack a means of expressing product lines and the variations between them. Techniques such as inheritance and constraints have been attempted, but they can only provide a set of variants to one level, or quickly become too complex and complicated. To express these concepts properly, it is necessary to integrate a set of product line constructs into the model that are specifically aimed at providing these capabilities. These models can also become overly complex and large, defeating key modeling objectives of abstraction and simplification. The best solution to this problem is to break up the models so that they represent individual sub-systems and sub-sub-systems. The models can then be linked together to define the whole system of interest. Two standards can help here: the OMG Reusable Asset Specification (RAS) and the OASIS OSLC Asset Management standard. This topic is covered later in this paper.

Product Line Engineering

Product Line Engineering (PLE), also known as Product Family Engineering (PFE) is a method that defines the underlying architecture of an organization's product platform. When applying a model-based approach to PLE, variability modeling must be included.

Orthogonal Variability Modeling (OVM) provides the ability to model systems and software products lines, their variation points, variant diagrams, variants and their variability relationships such as mutual exclusions and product dependencies. OVM was developed by the University Duisburg-Essen, PALUNO Institute [5] (K. Pohl et al, 2005) and is now an ISO standard (ISO 26550: 2013, Reference Model for System and Software Product Line Engineering and Management) [7]. See also [12] and [13]. Through this modeling technique, product line engineers have the ability to design product line variability options, constraints and conflicts, (if any exist), and to pick their desired end product. After modeling the variability in the product line model, the engineer can create decision sets and then choose to include or exclude variants for those decisions sets.

Variant Modeling

The following variability elements comprise the variability model.

- Variation Point - is a variable product line feature whose options are defined through Variants.
- Variant -is an option that can be chosen for a Variation Point.
- Dependency
    - Variability Dependency - specifies that a Variant is an option for a Variability Point.

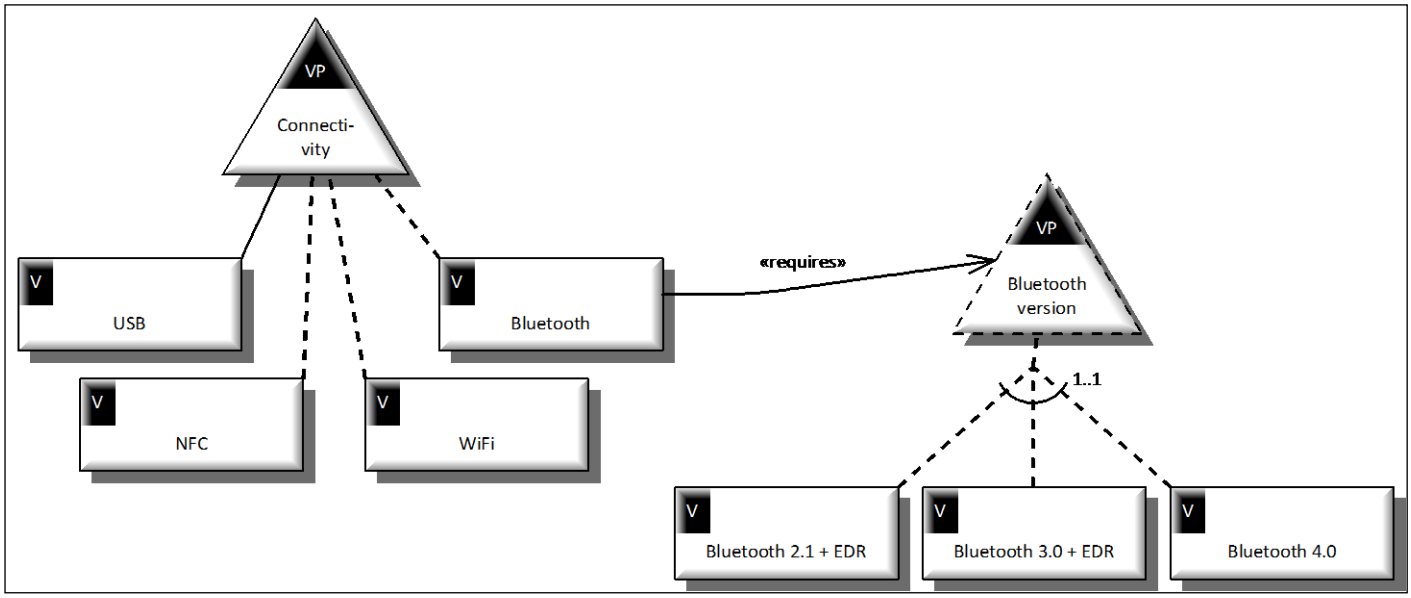

Figure 2. Variability Diagram

  - Excludes Dependency - specifies that the inclusion of a Variant or Variation Point requires the exclusion of another Variant or Variation Point.
  - Requires Dependency - specifies that the inclusion of a Variant or Variation Point requires the inclusion of another Variant or Variation Point.
- Alternative Choice - groups a set of Variability Dependencies and specifies the number of Variants that need to be included.
- Artifact Dependency - this is a special Dependency which specifies that an artifact (any base model item) is associated with a Variation Point or Variant. It is the link between the Variant Model and the System or Software Model.

Figure 2 is an example of the notation making use of several of the notational features.

In this example, all of the variants from the connectivity variant point are optional, but for the USB, which is mandatory. Optional Variability Dependencies can be constrained by a minimum and maximum number of possible choices. The syntax is <min>..<max> next to an arc connecting the variability dependencies. In this case the different types of Bluetooth. Variable Elements can be linked to express:

- That the selection of one requires the selection of another
- That the selection of one excludes the selection of another

The scope can be from:

- variant to variant
- Variant to variation point
- Variation point to variation point

In Figure 2, the selection of the Bluetooth connectivity requires the selection of the Bluetooth version.

Integrating OVM and SysML

The Variant Model and the Base System or Software Family Model together represent the Product Line Model, also frequently referred to as the 150% Model or the Overloaded Bill-of-Materials (BoM). This is a full representation of the product line, with all of its commonality and variation.

To enable this, OVM elements can be integrated into SysML or UML (the 'Base' Family Model) and then linked with any other models elements. Connections between Variable Elements and the Base Family Model allows engineers to model which model elements are in the product family model due to a specific variant or variation point. Artifact Dependencies can be created to all types of base model elements:

- Structural such as UML classes, SysML blocks or parts
- Behavioral such as Use Cases, Transitions or States

In order to express this dependency, base model elements can be shown on Variability Diagrams and Variable Elements can be shown on Base Model Diagrams. Figure 3 shows a simple model of the options for a car engine. The triangle is the variant point representing the engine type. The alternate choices (dashed lines) link to the two engine variants as Efficient and Fast. The multiplicity of "1..1" means that at least one engine and no more can be chosen. The artifact dependencies are linked to the diesel and gasoline engines modeled as SysML blocks.

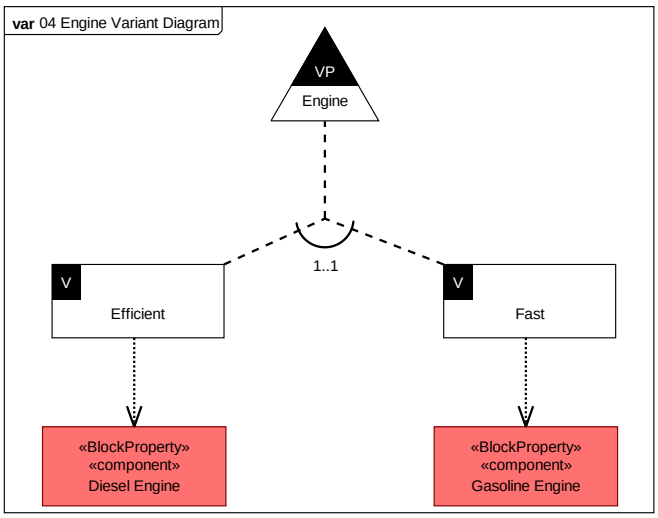

Figure 3. Engine Type Decision Tree

This Variation differs from SysML inheritance in that it not only indicates the choices which can be made but it also allows engineers to use a separate (or orthogonal) nomenclature for the variations and choices to that used in the more technical Base Model. This is particularly useful when customers, managers or marketing teams will make the product decisions, based on the rules encoded by the product line engineer. Also, complex multi-level decision sets are impossible to model in the base modeling languages. In order to properly express the model variability as opposed to the model structure, an orthogonal modelling construct is necessary. This is the purpose of OVM.

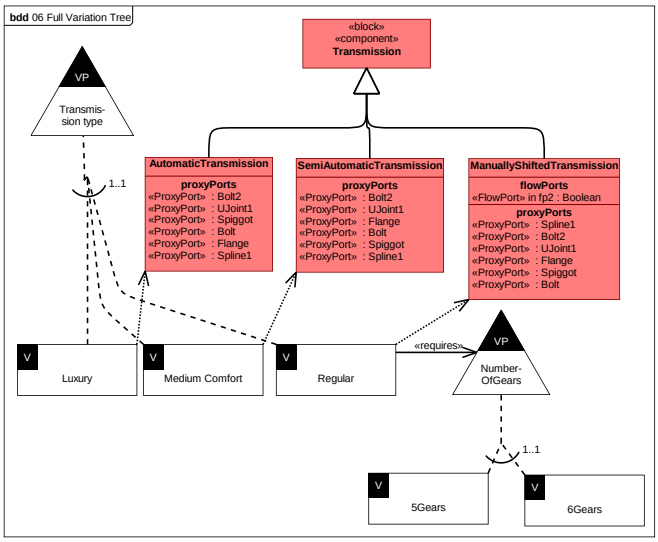

Figure 4. Transmission Selection

Figure 4 shows the transmission with subtypes of automatic, semi-automatic, and manual. The variation point is transmission type and the variations are luxury (automatic), medium (semi-automatic), and regular (manual). If regular is chosen, then the additional variation point of number of gears is required. The user must choose between 5 or 6 gears. This chain of decisions can become quite complex, and is only possible by using a set of constructs that is orthogonal to the MBSE language and fully integrated with it. Together, these form Model-Base Product Line Engineering or MB-PLE.

MBSE + PLE = MB-PLE

In order to define the product line and its various options, it is necessary to define a model called the 150% model. This is the case in Figures 3 and 4 as no car contains two engines and three different types of transmission. The 150% model contains the system along with all of its possible sub-system components (possibly from separate but connected models), interfaces, behavior, requirements, etc. OVM provides the ability to define a variation point of Engine, and then define that one and only one of the possible engines above can exist in any actual product. Dependencies between engine type and transmission type, exclusive or required relationships, etc. can also be defined. In addition, variants between requirements can also be defined. A particular system choice will have specific requirements corresponding to each variant. By linking the requirements in this way, the resulting requirements traceability and compliance can be maintained. In other words, product line feature selection will not only result in a 100% product model but also the 100% subset of the product line's 150% of requirements. Test scripts and sequences can also be included. This significantly changes the paradigm, enabling MB-PLE to cover the complete range of systems engineering concerns. However, in order to take advantage of the product line model, the variability needs to be "executable" resulting in a product model.

Executable Variability

Executable variability involves navigating through the variation points and selecting the desired elements. Figure 5 shows an example variant selector interface.

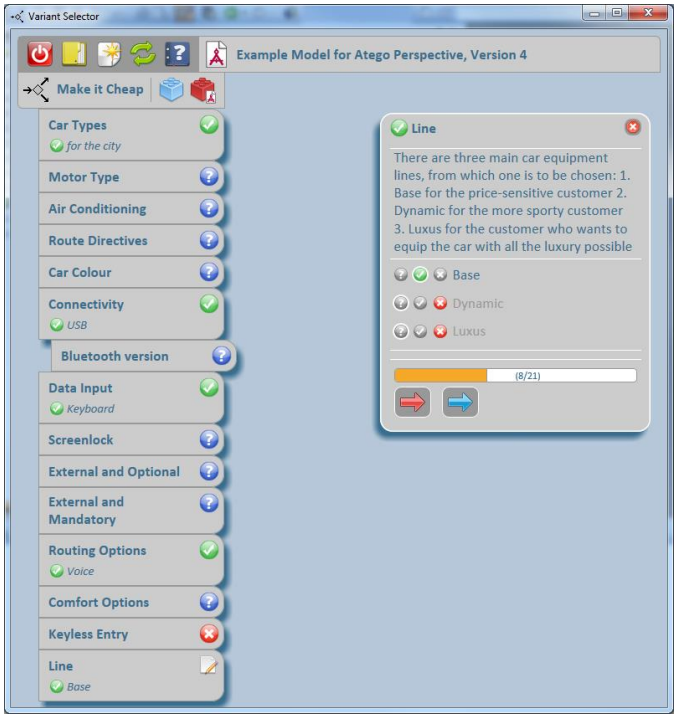

Figure 5. Example Variability Choices for Vehicle

Figure 5 shows the decisions made by the product customer, manager or stakeholder for the required product model. The decision making process shown in Figure 5 is in progress with decisions such as car type and Connectivity having been made and Screenlock and others as undecided. Having selected the required elements, a product model can be automatically generated from the product line model. The 150% model is used to contain the base model and all the variants. Product models can then be created from this model and follow their own natural lifecycle. To maintain consistency, the 150% model should always be viewed as the "master" model. If new assets are created for the product model, these can be incorporated into the 150% model for use in future product lines. Keeping track of the different assets and determining which models use which assets can become problematic and complex. In order to facilitate this,

there needs to be a means of defining, reusing and sharing the assets between models.

Reusing Assets

Each of these components in a system can be a complex system of systems in and of itself. However, often the internal details of these systems are not pertinent or can increase the size of the model. In addition, it can be advantageous or even mandatory to reuse the components without changing them. There may be several different versions of evolutions of the systems as well, making the 150% model overly complex. Consequently, a mechanism is required to manage and reuse the model assets as necessary.

The Reusable Asset Specification

The OMG Reusable Asset Specification (RAS) is used for defining reusable assets, their interfaces, characteristics and supporting elements [8]. There are three key dimensions that describe reusable assets: granularity, variability (and visibility), and articulation. The granularity of an asset describes how many particular problems or solution alternatives a packaged asset addresses. The visibility can vary from black-box assets, whose internals cannot be seen and are not modifiable, to white box assets which are visible and modifiable. The articulation dimension describes the degree of completeness of the artifacts in providing the solution. Asset specifications can also include supporting documentation, requirements addressed, interfaces, etc. Instead of a "mega-model" approach, a standards-based "model of models" approach is what is necessary.

Reusable SysML

Combining SysML and RAS provides a Model of Models approach with the main model specifying the system of systems and referencing assets in various levels of detail. The models specified by these assets can be referenced when detailed analysis is required, or hidden when a SoS viewpoint is required, allowing the analyst to see the forest through the trees. To extend the metaphor further, details of the individual trees can also be examined when necessary. The variability of the assets in the library is normally at the black-box level. Included with the asset is a description, interface, references, values, etc. This level of detail is appropriate for reuse of the system as a black box component in a system of systems architecture. However, since the requirements, interfaces, behavior, published and consumed events, included parts, references and parametric characteristics are also included, quantitative as well as qualitative analysis can be done on the asset to determine if it is the best fit for the problem at hand. The inclusion of these features increases the articulation or degree of completeness of the artifacts. Included with the asset is the specification of the source model from which the asset definition was taken. Finally, the individual assets can contain variability elements as well. For example, a separate model could be created containing all the engine variants and choices between them. These can be reused in the SoS model along with the lower level models. Figure 6 shows an example structure of a reusable asset library, a set of asset models, a 150% model and a set of product models.

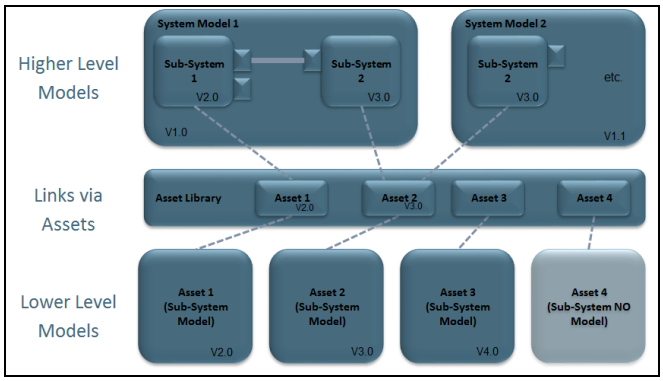

Figure 6. Asset Reuse.

In Figure 6, the models at the top of the figure, the higher-level models are the 150% and product models. These models reuse the assets from the asset library shown in the center section Links via Assets. The Lower Level Models contain one or more assets that have been shared in the asset library. Additionally, these assets can also contain variability. Consequently, the asset itself can contain a set of choices to define the correct component to meet the requirements for a specific solution space.

Leveraging Standards

Together, these standards and approaches provide the ability to implement Model-based Product Line Engineering (MB-PLE) at all levels of architecture and throughout the various phases of the development cycle. Independent survey results have shown that applying MB-PLE approaches can reduce total development costs by 62% and deliver 23% more products on time. In today's budget constrained world these are numbers that demonstrate a return on investment that is worth investigating [10], and [11].

Conclusion

As systems and models of systems become increasingly complex, we need to discover new ways of organizing the models and the decisions made while creating them. The combination of SysML, Product Line Engineering using the Object Variability Modeling and the Reusable Asset Specification provide Model-Based Product Line Engineering. This enable engineers a means to reuse assets while making value-based decisions on system

configuration. Together, these provide a demonstrable ROI that will reduce development time and costs and help automotive engineers build better systems.

REFERENCES

[1] DoDAF DoD CIO, 2012, DoD Architecture Framework Version 2.02, DoD Deputy Chief Information Officer, Available online at http://dodcio.defense.gov/dodaf20/dodaf20_pes.aspx, accessed June, 2014.

[2] Friedenthal, S., Moore, A., Steiner, R. Practical Guide to SysML: The Systems Modeling Language Second Edition, Morgan Kaufman, Oct 31, 2011

[3] Object Management Group (OMG), June, 2012, OMG Systems Modeling Language (OMG SysML™), V1.3, OMG Document Number: formal/2012-06-01, http://www.omg.org/spec/SysML/1.3/PDF/, Accessed November, 2013

[4] Object Management Group (OMG), 2007a. Unified Modeling Language: Superstructure version 2.1.1 with change bars ptc/2007-02-03. [online] Available from: http://www.omg.org [Accessed September 2007].

[5] PALUNO, The Ruhr Institute of Software Technology Software Product Line Engineering (Pohl et al - Springer 2005)

[6] ISO 15288:2008 Systems Engineering standard covering processes & life cycle stages.

[7] ISO 26550:2013 for Software & Systems Engineering – Reference Model for Product Line Management & Engineering.

[8] OMG, 2005, Reusable Asset Specification (RAS), Version 2.2, http://www.omg.org/spec/RAS/2.2/PDF, formal/05-11-02

[9] INCOSE SE Vision 2020, September 2007, Available at http://www.incose.org/ProductsPubs/pdf/SEVision2020_20071003_v2_03.pdf, Accessed Nov 2013

[10] EMF 2013 Independent Survey Results from 667 Systems Engineering respondents, http://www.embeddedforecast.com/EMF_freewhitepapers43.php

[11] Linda Northrop, SEI SSPL 2008-2012), http://www.sei.cmu.edu/library/assets/spl-essentials.pdf

[12] Seven Years of Orthogonal Variability Model - Experiences and Insights" (A. Heuer et al. 2010)

[13] Intro to OVM at the SysML RTF" (V. Stricker, 2012)