Emerging Usability Patterns in the Application of Modeling Libraries

Judy Che
Ford Motor Company
jche@ford.com

Bjorn Cole
Jet Propulsion Laboratory
Bjorn.Cole@gmail.com

David Lempia
Rockwell Collins
dllempia@rockwellcollins.com

Ron Lyells
Honeywell Inc. Aerospace Group
Ron.Lyells@honeywell.com

George Allen Sawyer
BAE Systems Inc
george.a.sawyer@baesystems.com

Craig Schimmel
Honeywell Inc. Aerospace Group
Craig.Schimmel@honeywell.com

Scott Workinger
Workinger Consulting
ScottWorkinger@gmail.com

Abstract. This paper summarizes the conclusions of the MBSE Usability Group for 2012. Five perspectives are offered with each perspective grounded by an exemplar from the authors' practice using Model Based Systems Engineering Environments. These exemplars range from very detailed step-by-step descriptions with identification of fine-grained usability issues to system of systems simulations developed by geographically dispersed operations in a global company. Discussion of each exemplar within the Usability Group led to significant lessons learned for the team. This paper summarized the chief lessons learned during this process. Today, the process continues. In the words of Ron Lyells, "Usability is an emergent quality."

1.0 Introduction

In 2011, participants at the INCOSE International Workshop identified potential use cases for MBSE based modeling efforts. When the group voted on the priority of the various use cases, the use of modeling libraries was designated as most important. This natural emphasis arises from two considerations: 1) placing an object from a library is often the first step in building a model and 2) libraries are the natural residing place for a wealth of information related to modeling efforts. In a very real sense, libraries are the repositories of knowledge-rich artifacts and key insights from prior modeling efforts. The model components themselves naturally have associated: 1) names, 2) types, 3) classification semantics, 4) internal structure, 5) a set of problems that they can be used to solve, 6) historical examples of application, 7) a description of how they solve the problem, 8) tailoring information, 9) graphical representation, and 10) associated components and 11) architectural patterns, and 12) best of class algorithms. George Sawyer develops some of these themes with his description of modeling templates in Section 2.4. Organizations with a wealth of experience in modeling have the potential to develop rich modeling libraries that embody their technical knowledge assets. Tool vendors have significant incentives to support this activity. Language designers who want to put power at the fingertips of

modelers have a significant opportunity to support the systems engineering modeling community by addressing the representation of these assets. Systems engineering leaders who wish to streamline project activities need to understand the power of this approach. For instance Sawyer (See Section 2.4.) cites an example of a project that cut integration time in half by careful modeling of interfaces prior to the integration effort. Standard tools, language capabilities, and modeling practices are limited in their ability to support this richness. The role of the MBSE Usability group is to address usability issues in MBSE environments; however, it rapidly became clear as we focused upon the substance of usability issues that many of the practical usability problems that arise using an MBSE environment are associated with basic functionality. For instance, most of the use cases identified at the 2011 INCOSE International Workshop assumed integration and collaboration functions that are not routinely available in today's MBSE environments. We have addressed some of these issues in this paper.

Lempia (See Figure 1.) suggests 3 actors and 5 use cases associated with library usage and offers a detailed exemplar demonstrating creation of a simple product model. A key insight is that the roles of these actors are only partially defined, particularly when we consider the informal aspects of collaboration. Formal roles are only 'the tip of the iceberg' when we consider the evolution of model components and the development of major technical innovations.

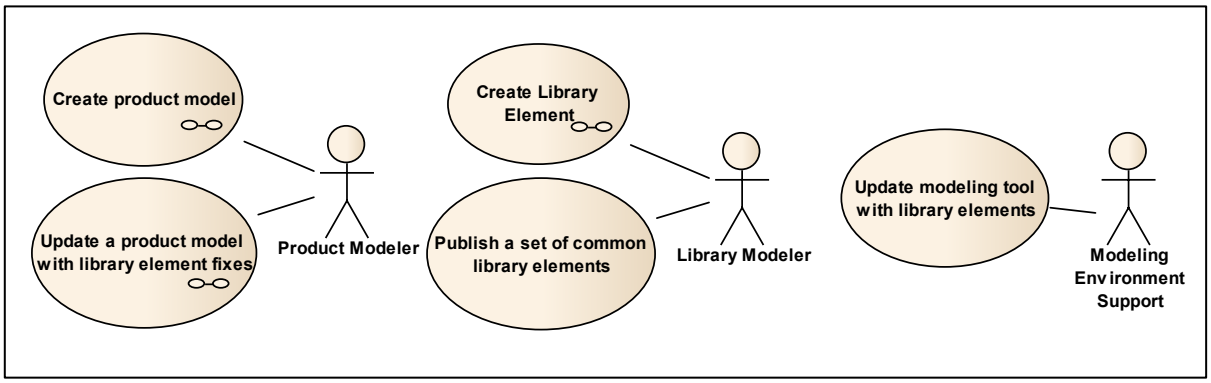

Figure 1: Library Usage Example - Actors and Use Cases

Simulation is a common theme in the exemplars described in this paper. While the development of SysML has accelerated the growth of modeling practice, SysML in its current form mainly supports descriptive models. Yet much of the potential power in an MBSE environment comes from the ability to predict system dynamic behavior. Several of the exemplars in this paper address the practical challenges of creating MBSE simulation environments. There are practical advantages to studying such pioneering efforts; however, it is also clear that these examples merely hint at the spectrum of opportunities in MBSE practice.

2.0 Exemplars of Modeling Library Applications

We begin the examination of Library exemplars with a fine-grained examination of placing components from a library. There is much to be learned from studying 'the humble mouse click.'

2.1 Building a Fine-Grained Simulation Model – David Lempia

This section of the paper focuses on one of these five use cases, the Create Product Model use case. The Product Modeler creates a product model, a behavior or structure model of the

product, to meet a modeling objective. The modeling objective includes both the understanding of what the model needs to do and an understanding of the expected results. To illustrate this use case, a very simple example of a testing model for a low-pass filter is developed. The objective of this example model is to verify that the rise time of the low-pass is consistent with the time constant. The example model is built using a SysML Activity Diagram. The process of creating the product model is described in the use case shown in Figure 2. (In the text below, each Usability Challenge is designated, **UC #**.)

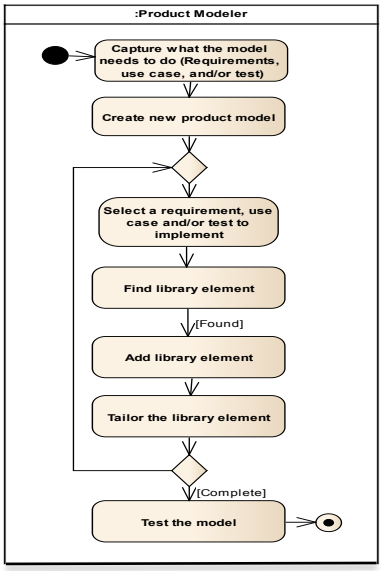

Figure 2: Create Product Model Use Case

The **pre-condition** for this use case is a list of library elements that is available and ready to use. The actions for this use case are listed below:

1. **Capture what the model needs to do (Requirements, use case, and/or test)** – The product modelers require the ability to change the value of Tau for the Low pass filter. The tester must verify that Tau can be changed.
2. **Find library element** - The Step Function, low pass, and Time History Plot library elements are found in the list of library elements. The library elements are implemented as SysML activities. The activities specify the behaviors needed to execute the model.
   a. **UC 1** – It is hard to find, identify, and understand the usage of library elements in a model. Tools must minimize the time it takes to find and understand what the library element is and how to use it. Understanding comes from both an example and a description that includes the environmental constraints, the library element requirements, and the realized tests.
3. **Add library element** - Create an instance of the library element in the editor. In this example, a step function action is placed in the editor. It becomes a call action. The step function has one output and no inputs. The pin placement is visible and is located in the same place as the parameter on the library element activity. The icon draws a picture of a step function and scales the picture to mirror the default value of 1 for the step input and 1 for the step size. (See Figure3.)

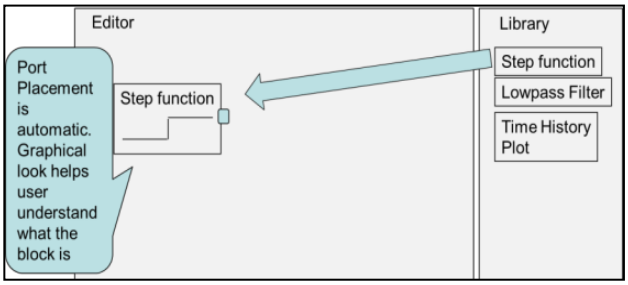

Figure 3: Add Library Element

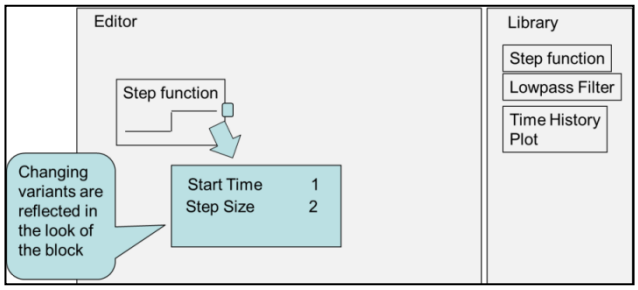

Figure 4: Tailor the Library Element

   a. **UC 2** – It takes a number of steps to re-size the action and to place the pins around the perimeter of the element. Tool vendors need to minimize the number of steps needed to re-size and adjust the location of the pins. As an example of how to do this, some vendors have used a template layout to position the pins and to size the action.
   b. **UC 3** – Some vendors provide no graphical way to indicate which pin is an input

and which pin is an output.

c. **UC 4** – There is no language support for developing custom icons.

4. **Tailor the library element** – Select the variant of a library element to express the characteristics needed for the model. Tailoring customizes form and function of a library element for a specific use in the product model. In this example we tailor the function by setting the step insertion time at 1 second and the step size at a value of 1.

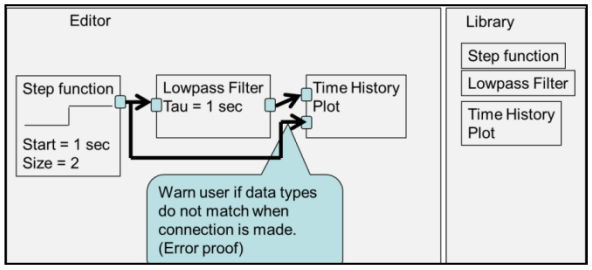

Figure 5: Error Proof

   a. **UC 5 -** It is difficult to look at a diagram containing a library element and see the items that can be tailored on a library element and to see the value chosen for tailoring.

   b. **UC 6** – It is easy to introduce errors in a model when pins are connected together with an object flow. This is because the pins connected together with the object flow may have incompatible types.

5. **Test the model** – Verifying the rise time of the low-pass filter involves executing the model, looking at the time-history plot, and measuring the rise time.

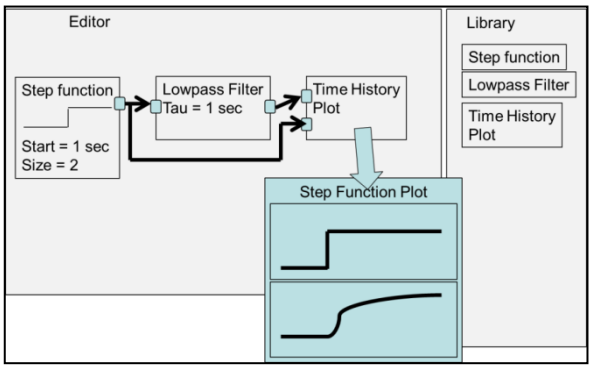

Figure 6: Plot Simulation Results

   a. **UC 7** – It is difficult to execute models. Sometimes executing models involves creating classes, placing the activities below the class, and writing software. This should be easy to setup and understand.

   b. **UC 8** – It is difficult to visualize the results of an executable model. Capabilities such as the ability to single step, watch the value of a pin, and plot the time-history of a pin help understand the dynamics of a simulation and aid in testing.

Each of the remaining use cases un-cover additional usability issues. This is a subject for a future paper. These additional use cases are summarized, below.

Update a product model with library element fixes - Library elements will change over time. Changes may be driven by events such as feature enhancements, dependencies, or bug fixes. The product modeler needs a way to update the product model with changes.

   a. **UC 9** – It is difficult to identify the dependencies associated with a change. Dependencies may include new model elements that need to be loaded into the model because of an update and assumptions for the model environment.

Publish a set of common library elements - Common library elements only have value in an organization if they can be found and maintained over time. The Library Modeler publishes common library elements and notifies the product team (product support and product modelers).

   a. **UC 10** – It is difficult to package and share multiple elements. Aspects include requirements, the design, the implementation, the tests, and the test results.

   b. **UC 11** – It is difficult to document the help on a library element.

Create library element - Creating library elements starts with creating a product model. Some of the model elements used to create a product model are usable across multiple

products in a product line, across all models in an enterprise or across all models in a modeling discipline. The Library Modeler considers the requirements of the user group and updates the library element, documents the library element, and tests the element to show that it meets the requirements.

Update modeling tool with library elements - Finally, the Modeling Environment Support person updates the modeling tool when new or changed library elements are available. If desired, the product team also needs a way to update the existing models to use the new and improved library elements.

a. **UC 12 -** It is difficult to configuration manage multiple versions of a library element in a model. Managing includes capabilities such as selecting only one version of a library element for use in a model, upgrading to a new version of a library element, and differencing one version of a library element with another.

Creating product models from library elements is the most important use case selected by the MBSE Usability Group. This paper details the use case steps needed to build and manage a model from library elements and outlines the activities in other use cases. These use cases revealed a number of necessary features. For example, there needs to be a way to tailor library elements, to capture environmental dependencies, to capture dependencies on other model elements, and to package numerous artifacts together (requirements, tests, test results, etc.). These features need to be supported by the tools, the tool environments, and the underlying modeling languages.

2.2 Libraries and Domain-Specific Modeling – Bjorn Cole

Building upon the insights from usage in a single domain, we consider the challenges of incorporating many specialized domains into a single modeling environment.

For descriptive modeling frameworks such as SysML, there is an interesting quandary raised by those more familiar with traditional programming libraries. The traditional libraries are executable – they confer desired functionality directly to a new codebase. SysML models are not executable by themselves, but rather provide a regular structure for further computation. Thus, one can easily ask: if the language is not an executable one, what is the value in libraries?

The answer to this is that while SysML itself is not executable, it does provide a platform for steadily refining semantic definitions until other programs can execute them via "instructions" captured or constrained by the model. The general SysML structure provides a great deal of guidance for execution. For example, the behavioral constructs have many semantics for general execution according to a state machine, Petri net, or message-passing set of semantics. [Jankevicius 2011] OMG provides standards for further refinement to aid execution [Open Modeling Group 2011]. The classification system provides guidance on using object-oriented concepts to run instances of generic classes through these behaviors, and SysML Parametrics expand this into solving static equations [Bajaj 2011]. So, if this comes "out of the box" so to speak, why are we further enhancing the language with libraries?

This is the core of the issue. While SysML provides a generic set of semantics, various engineering domains have very specific semantics that need to be respected. An excellent example is considering the use of a power bus or a junction in hydraulics. In basic SysML, a modeler would typically draw on an Internal Block Diagram (IBD) a series of components surrounding a part called "bus" with connectors linked to it. The power bus or hydraulic junction's purpose is to deliver all attached components to the same potential, whether it be voltage or pressure, for well-understood distribution. Analysis tools can then be assured a

uniform description of this engineering element and be instructed in the proper way of incorporating that model datum into the execution of the algorithm. The key point here is "uniform description." The advanced members of a given institution can discuss and argue about the appropriate or most elegant SysML elements to use for encoding a given semantic and then render the decision into a library.

With the "best of class algorithms" notion discussed, it is time to turn to the more generic purposes of a library. Providing an easy-to-access repository of modeling knowledge is of course of high value when one considers a real modeling team. A real modeling team on a project is likely, even by project intention [Bayer 2012], to have a wide variety of skill levels in its members. There will be an important subset of this team that does not have deep SysML modeling knowledge, but brings domain or general expertise to be encoded into the model.

There have been calls and even a proposed implementation [Friedenthal 2011] of a so-called "SysML Lite" to help new modelers get a foothold in the language. This is understandable, since SysML has many dozens of concepts that a modeler can use, and at least two dozen of them are necessary to cover a system in both its structural and behavioral aspects. However, SysML Lite is generated by more or less reducing the depth of modeling across all aspects of the language. This may or may not conform well to the needs of organizations that heavily emphasize behavioral representations or structural ones and need the more specialized types of full SysML.

There are several core concepts in each domain that are brought together to build functional systems. These concepts can be mapped to the concepts within SysML since it is a system-oriented language, and often in a straight-forward way. The first stages of constructing the library are about formulating these mappings. Once complete, the engineers can be assured that when they add a "propellant tank" or a "valve" to their model, it will be compatible with the modeling framework.

Further, libraries support orienting new modelers to the object-oriented framework of SysML. The Meta-Object Facility is built on a core of elements, element classification, relationships, and inheritance. Elements and relationships are easily understood. The more novel concepts are classification and inheritance of elements. A library will typically support these naturally. Using this library will walk a user through selecting the type and then selecting a specific part. Inheritance among classifications is also natural. For example, an engineer should be able to use a generic tank, or a tank with a specific material, or a tank with a specific material and propellant management device.

This also highlights an important usage aspect of system modeling in general. Systems engineering practice requires the breakdown of complicated machines into smaller pieces. Preferably, this breakdown enables the smaller pieces to be contained comfortably within an engineering specialty with trained engineers that are practiced at handling similar development problems. This means that at some point in model-based practice, the domain engineers will have to make some inputs at the boundaries of the system model (and that the system modelers have sufficient cross-training to project some of their work into domain models). Adding domain-specific libraries is not simply about helping new modelers into the enterprise; they provide a way to capture knowledge projected into the system model.

A problem in discussing domain-specific icons is that the machine-readable part of the UML / SysML specification currently does not include the graphical elements of diagrams. At least one authoring tool currently has a customization capability that allows for experimenting with custom diagramming. MagicDraw has served as the experimental platform at JPL for working with domain-specific libraries. It has done so without a standard so far, but it has provided the chance to work out use cases and inputs from potential users regarding look and feel.

In Figure 7, the MagicDraw tool is shown to emphasize the relevant parts of the library. First is the custom icon itself, which is the most visible. The toolbar of concepts for the custom diagram serves two purposes. It provides access to the custom icons, but also serves to apply a Stereotype matching the component type directly to this part. The next step is not yet automated, but is relatively straightforward. The element has been stereotyped, but not yet classified. In the Containment Tree, a set of SysML Blocks (eligible to be Classifiers) is housed to have properties for a specific component. In this example, the library has been a source for directly applying domain-specific iconography, loading the type (and its associated properties like mass and cost) into the new model, and restricting semantics of SysML as appropriate for the kind of thing being placed on the diagram via its stereotype.

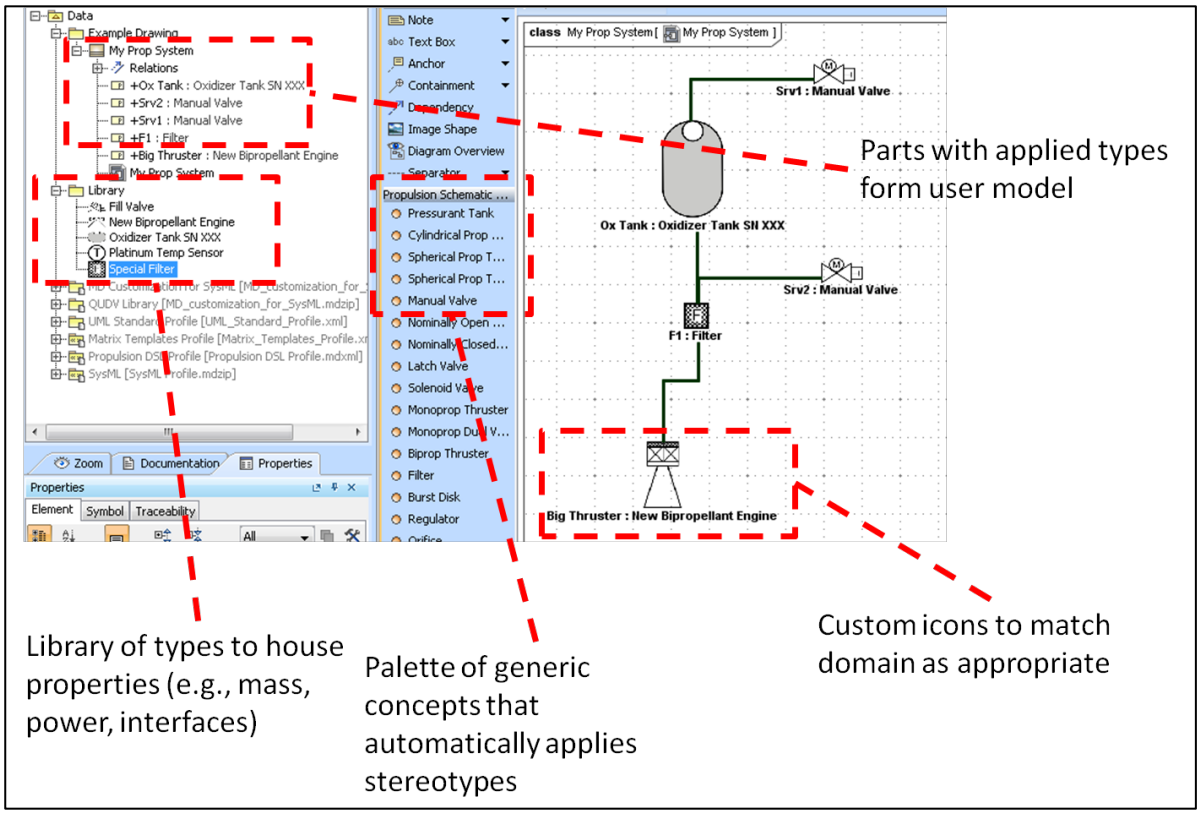

Figure 7: Customized propulsion schematic in MagicDraw

A standard is currently being developed to help improve the specification of custom diagrams. It is an OMG product called Diagram Definition (also called Diagram Definition and Interchange) [OMG 2007]. Hopefully this standard will mature quickly, and interested readers at OMG member organizations are encouraged to investigate the status of this standard and contribute via knowledgeable representatives. The development of this standard will also likely have the benefit of supporting not just libraries of custom diagrams, but libraries of diagrams of standard elements available in all authoring tools.

2.3 MBSE Libraries for Automotive Engineering – Judy Che

We build upon the detailed insights into collaboration between specialists by examining an example of practical architecture that integrates specialized domain contributions.

The complexity of designing today's sophisticated vehicles requires a model-based development process supported by a set of robust vehicle system models. The goal is to use vehicle system models earlier in the design process to develop and trade-off vehicle level attributes such as:

- Fuel economy
- Performance
- Drivability
- Cabin comfort
- Safety

The vehicle system models described here include representations of both hardware and controls elements and are intended to simulate the functional behaviors of complex mechatronic systems. These vehicle system models can be built up by assembling a set of domain or component models using a Vehicle Modeling Architecture (VMA) as shown in Figure 8. This architecture and typical applications have been described in previous publications [Belton 2003].

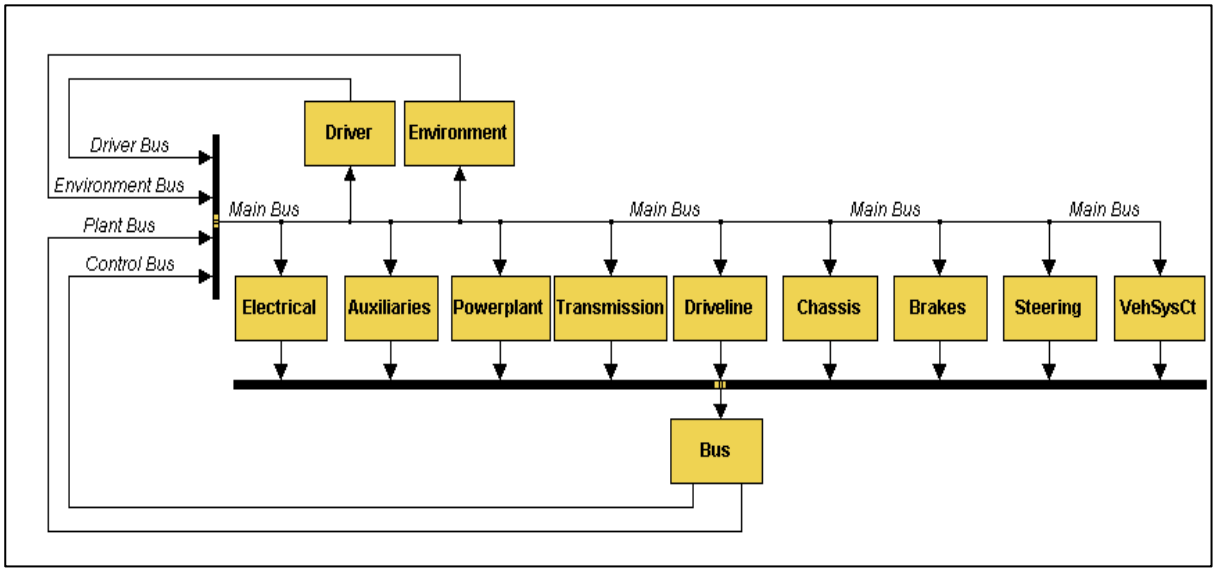

Figure 8: Vehicle Model Architecture

The VMA provides a well-defined high-level modular structure for dynamic vehicle modeling with key vehicle subsystems represented as distinct elements. Subsystem connections are specified through well-defined interfaces. While the structure and interfaces are fixed, model content is not. The vehicle model architecture provides a way to manage model requirements for each component for simulation use cases such as vehicle performance or fuel economy analysis.

Our goal is to establish a well-defined process by which Subject Matter Experts (SME's) in each domain create and verify models for their domain such as engine, transmission, etc. These domain models can then be integrated to build up a set of vehicle system models as seen in Figure 9. These sets of domain models can be managed through a set of re-usable model libraries of component models with varying levels of fidelity. In many cases, parameter data sets can be managed separately from the models so that a single engine model can be made to represent a 1.5L or a 2.0L engine with a simple data set change. In order to support such a model-based vehicle model creation process, tools and utilities are needed to help specify, assess compatibility, edit, save and apply model variant and data set choices. These steps include:

- Specify a set of model domain choices (both hardware and controls) for a particular analysis such as fuel economy or performance simulation as shown in Figure 10.
- Specify data sets for each domain model – i.e. 1.5L or 2.0L engine maps
- Assess model choice compatibility – i.e. between plant and controls models
  - Use meta-data or other mechanism to associate "compatible" plant & control models

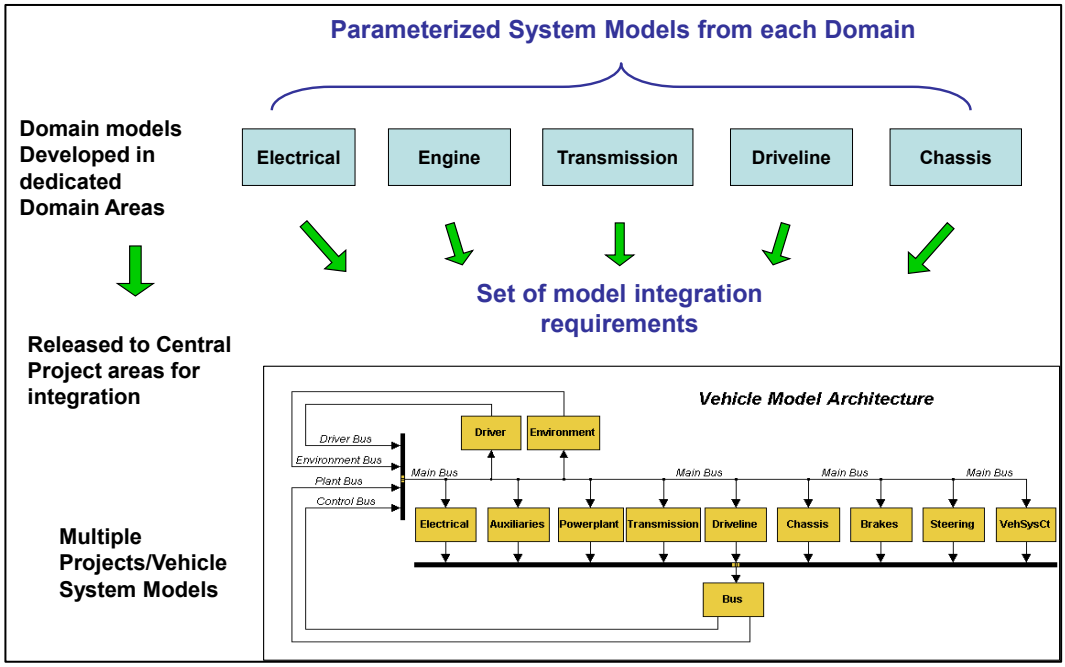

Figure 9: Building Vehicle System Models by Assembling Domain Models

- Verify interfaces line up for hooking up models
  - Signal definition (e.g. Engine Brake Torque vs. Indicated Torque)
  - Units (Nm or Ft-lbs)
  - Data types (real, integer, etc.)
- Allow edit and save of domain model choices and corresponding data sets
- Provide automated way to build up vehicle model from domain model choices by auto-wiring signals (e.g. Simulink) or physical connections (e.g. Dymola flanges)
- Ensure appropriate versions of models and data sets are selected

Some previous work in using SysML to support Multidisciplinary vehicle system modeling can be found in [Branscomb 2013].

Model Coordination Matrix											
	Package Desc	ELE PLNT	ELE CTRL	AUX PLNT	AUX CTRL	PWP PLNT	PWP CTRL	TRN PLNT	TRN CTRL	CHA PLNT	CHA CTRL
1		Analysis A [Fuel Economy]									
	Analysis A	Low	Low	Null	Null	Med	Med	Med	Med	Low	Low
2		Analysis B [Performance]									
	Analysis B	Low	Low	Null	Null	Low	Low	Med	Med	High	High
3		Analysis C [Climate Control]									
	Analysis C	Low	Low	Med	Med	Med	Med	Med	Med	Low	Low
4		Analysis D [Drivability]									
	Analysis D	Low	Low	Null	Null	Med	Med	High	High	High	High

Figure 10: Model Coordination Matrix for Different Analyses

2.4 Library Templates to Support Learning and Modeling Efficiency

– George Sawyer

From an examination of architecture for an integrated modeling environment, we proceed to look at mechanisms for lowering the barriers to using such a complex environment by relying upon infrastructure that communicates the rich context of modeling practice.

2.4.1 Historical Context

BAE Systems has experienced limited success in applying Model-Based Systems Engineering (MBSE) languages, tools and methodologies over the past decade. Several projects have seen very positive results particularly in allowing systems integration to proceed at a more rapid and reliable pace than with similar programs. In one specific program, the final systems integration was completed in just over three months as opposed to the six months originally planned. Despite this and other positive experiences, there has been both organizational and technical opposition to MBSE adoption in most business areas. Overcoming the technical challenges of adopting MBSE was seen as a prerequisite to increased organizational acceptance – one of the primary technical objectives was the development of a robust model template to use as a basis for "jump-starting" MBSE development efforts and allowing adopting projects to make "early gains" that would positively influence their decision to continue forward with the approach. Specifically, the template was aimed at addressing the chief technical difficulties in adopting model development as identified by the SysML RFI [Bone and Cloutier, 2010]:

1. Having to simultaneously learn a language, methodology and tool to be effective
2. Lack of relevant, realistically complex examples with supporting narrative
3. Lack of proven best practices for successful application across domains
4. Difficulty in understanding and applying specific language areas (e.g. ports)

2.4.2 Elements of a Useful Model Template

Recognizing the above problems, we realized that modelers needed a better starting point for their models than was currently available. They needed something they could take with them from their training that would allow them to make rapid progress with their model development and keep their model development efforts going on a path to realize benefit to their projects. The following objectives for a useful model template were identified as follows:

1. Provide an integrated definition of the development methodology
2. Provide an extensible and adaptable structure for the overall model
3. Provide a non-domain-specific infrastructure for developers to build upon
4. Provide relevant examples of model items and diagrams illustrating correct usage
5. Provide embedded directions for extending the model template to a specific solution
6. Provide for robust integration of model elements with other development tools

2.4.3 Integrating Methodology Support

Within BAE Systems, the company-patented Functional Object Decomposition (FOD) methodology is one of the primary ones used however there are a number of widely recognized alternatives as identified by Jeff Estefan [Estefan, 2009]. Regardless of the development methodology (or methodologies) used by an organization, there is benefit to be

gained in making it an integral part of a model template. The following figures show how the methodology description itself can be incorporated as part of a SysML profile. The way in which the methodology description is incorporated takes a page from the development of UPDM in which Mathew Hause identifies the importance of "walking the talk" when it comes to model development [Hause 2010]. This is not done just for convenience but to demonstrate the utility of the language in defining the methodology of "how to use it".

2.4.4 Provide a Flexible and Extensible Model Structure.

Getting the model structure right is one of the key considerations that will "make or break" a model as a useful tool in developing a new (or adapted) system. The model needs to be structured to support a number of critical and seemingly conflicting objectives, such as:

1. Providing an efficient and intuitive means for accessing model information
2. Maintaining consistency with the development methodology
2. Supporting simultaneous, non-conflicting use by multiple developers
3. Integrating with add-on capabilities such as document generation scripts
4. Implementing reuse through import and export of system, subsystems and modules

2.4.5 Providing Modeling Infrastructure to Build Upon

The development of infrastructure is typically very resource intensive, therefore we started by initially developing lower-level modeling libraries that would potentially be useful to system developers regardless of domain. The first effort involved developing a

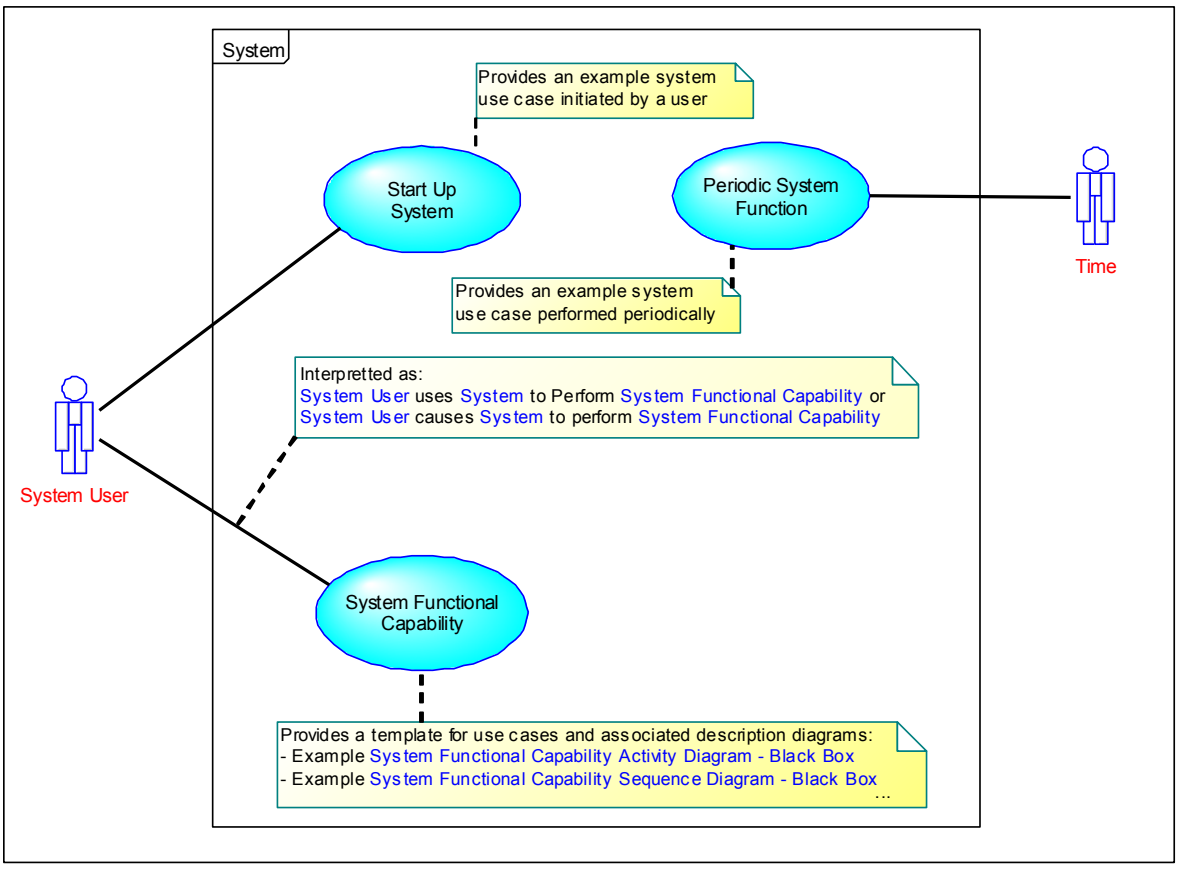

Figure 11. Example Pattern for System Use Cases

comprehensive set of dimensions, units and value types for supporting model constraints

and parametrics. Some modeling tools come with a set of International System of Units (SI) however our standard tool at the time did not. The library of units we developed, including both SI and "English" units, provided users with a common, pre-defined set of types and also laid the groundwork for additional generic infrastructure components such as the pre-defined system environments blocks and the generic shapes library.

2.4.6 Providing Relevant Examples and Directions for Extension

The SysML RFI results serve as a resounding confirmation of the general observations from teaching MBSE across several organizations: that applying the modeling constructs to real-world problems can be exceptionally difficult [Bone and Cloutier, 2010]. This is the rationale for embedding both generic model constructs as well as more concrete examples within the model template.

The generic examples provide a "copy and paste" pattern for system developers to follow which result in more consistency and correctness of language usage. Additionally, the more complex examples serve to guide the development of the system-specific constructs from the basic patterns. Figure 11 provides an example of the type of guidance that is embedded in the model template.

2.4.8 Improving Integration with Other Development Tools

To support the integration with the requirements management tool, the model template includes a number of additions to the basic SysML profile. First, it includes the suggested additions to requirements as identified in appendix C of the SysML standard [OMG 2007]. This permits the stereotyping of requirements by their applicable category and dovetails with the FOD methodology in defining a standard means for identifying satisfaction of each requirement based on their type. The template also contains a set of stereotypes and associated tag types that map to the company-standard set of attributes use for supporting requirements management. By doing this, developers can synchronize all relevant requirements information between the SysML and requirements management tool and perform edits (if authorized) in either tool. Similar integrations have been developed for numerical analysis, configuration management and test automation support tools.

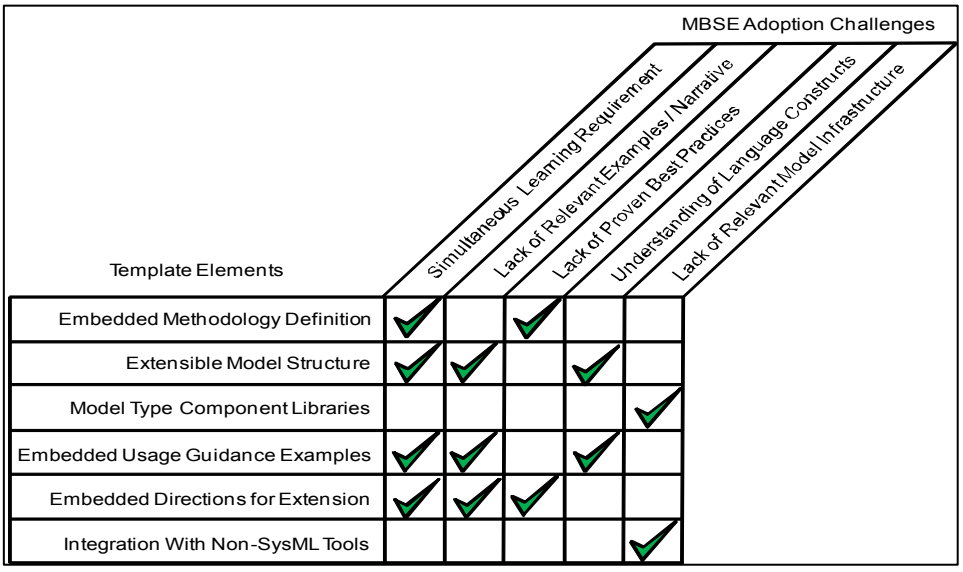

Figure 12. How the Model Template Addresses MBSE Adoption Challenges

2.4.9 Summary on Using Model Templates

In summary, model templates directly address some of the more serious challenges presented by the adoption of MBSE. With experience in using these templates on multiple projects and guidance from the recent SysML RFI, the usefulness of the templates has continued to improve over the past few years. As the following table shows, the current iteration of our model templates provides a level of resolution for each of the primary challenges identified in the RFI. It also addresses the one additional challenge area of infrastructure that we've identified as an additional concern. Our expectation is that the template will continue to improve in response to feedback from projects employing it.

2.5 Practical Application of a Geographically Dispersed Modeling and Simulation Environment – Craig Schimmel & Ron Lyells

At the highest level, we look at effective and efficient architecture for implementing a modeling environment in a large system of systems.

2.5.1 Context

The Problem: The use of simulation is not a new concept and is widely used. The INCOSE Handbook discusses the use of simulation in several system engineering venues; during concept generation, developing performance requirements, and during study of interactions between system elements, and between systems in a system of systems problem space. However, there are issues of scale. Specifically, for organizations like Honeywell, who are combining product portfolio's to build complex systems of systems, how does one effectively leverage simulations that have been independently developed in different geographic locations and in different simulation environments?

The Opportunity: Borrowing from the gaming world; use of simulation environments for exploring concept of operations issues with the customer and then using that same simulation environment to support training of those same customers has proven to be a powerful capability that is especially useful for customers.

A Matter of Persistence: Web based gaming environments provide a persistent environment where gamers can come and go, participating as desired. Using this as inspiration, consider two scenarios where system developers and users come and go for the purposes of exploring concept of operations issues and quantifying needs, system prototyping, and training.

The Persistent Constructive based Simulation Environment: Simulated entities join a persistent environment for purposes of engineering evaluation or ConOps development. The simulations that make up the environment can be swapped dynamically in order to focus on specific evaluation needs. For instance, high fidelity simulations can replace baseline low/medium fidelity in order to explore specific operational or functional detailed aspects of the system.

Such an environment would require the ability to host a large enough set of simulations to satisfy the evaluation needs – albeit with a properly scaled level of fidelity and scope to match the objectives of the evaluation. Such a system would allow alternative concepts to be evaluated in a known, consistent environment. The environment could have the ability to adapt and react dynamically (learn).

The Persistent Live Training Environment: In this scenario, distributed individuals join a persistent environment and train together. The simulation world may exist continuously and trainees can join at any time. Multiple instances of the training environment can exist in parallel. This environment requires the ability to host a simulated world and provide access

for players to join and leave. The world that each player sees must be synchronized and consistent among players. The world must provide sufficient detail to achieve an immersive effect so the trainee acts and learns as if operating in the real world. Such a system can host simulation on on a local scale (from a single laptop to a few workstations) to a much larger scale, operating remotely via the cloud, allowing trainees anywhere to participate in collaborative training. Tactics, techniques and procedures can be updated by the customer as needed.

2.5.2 What Would A Persistent Simulation Environment Look Like?

An example of the kind of persistent simulation environment described in the scenarios above is depicted in the concept of operation diagram in Figure 13. The REAL (Reactive Environment for Active Learning) framework has been developed by Honeywell's SMARTLAB organization to leverage its own simulation capabilities that are globally distributed.

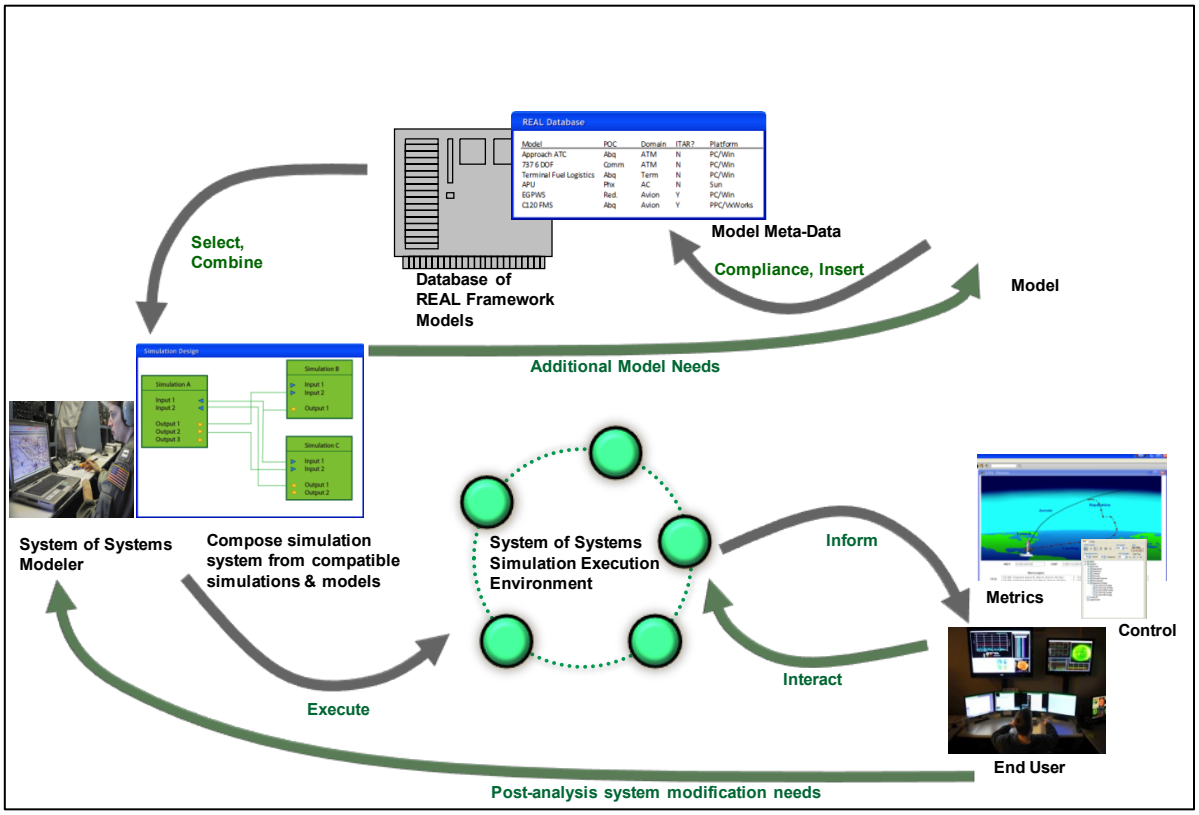

Figure 13. Operational Scenario Using REAL Framework

Key elements of the environment include:

1. **The Simulation Set** is a meta-data database identifying REAL Framework compatible simulations. It allows the system-of-system modeler to identify the simulations to be combined into a simulation environment and operational conditions.
2. **The Tool Set** is set of common tools and libraries (including COTS components) that aid the creation of REAL Framework simulations and provide readymade, frequently used capabilities. Eg. Simulation control, visualization, data capture and metrics.
3. **The Simulation Execution** is the selected set of system of systems simulations and tools, combined with the run-time middleware into a complete simulation environment useful to the end user.
4. **The Middleware** is the flexible mechanism of system model integration that ties all

of the other elements together. A model needs only be integrated to the middleware once, yet the system modeler has the ability to change the apparent model behavior and reuse the model in any other simulation environment.

But you say, "Simulation environments and standards are already defined… What's the problem?" The problem is that it takes a lot of work to adapt a simulation from one framework to another. (e.g HLA to DIS) The other problem is that environments like HLA are complex in themselves; developed to support very diverse simulation problems. So, the very breadth and capability come with a usability complexity that drives the learning curve for a modeler to levels that program timelines or organizational resources cannot support effectively. Because it takes so much work, many don't try to leverage potentially available simulations. Groups sometimes will invest the time to create their own new simulations, build a work around, or ignore simulation as a means altogether. An internal study within one of Honeywell's advanced modeling and simulation organizations, where HLA, DIS, and Honeywell-developed distributed simulation protocols are used, revealed that only about 40% of the simulation applications actually contained interfaces where data was shared. Of those, 75% used custom messages over sockets and just sent data.

2.5.3 A Theory For A Workable Countermeasure:

If one can build a wrapper around existing simulations that removes the simulation environment attributes and complexity from the simulation, then one can focus on the management of data that is consumed and produced by a given simulation, thereby simplifying the simulation interface and improving the usability and availability of a given simulation. This theoretical "wrapper" is shown as the SimAPI in Figure 14.

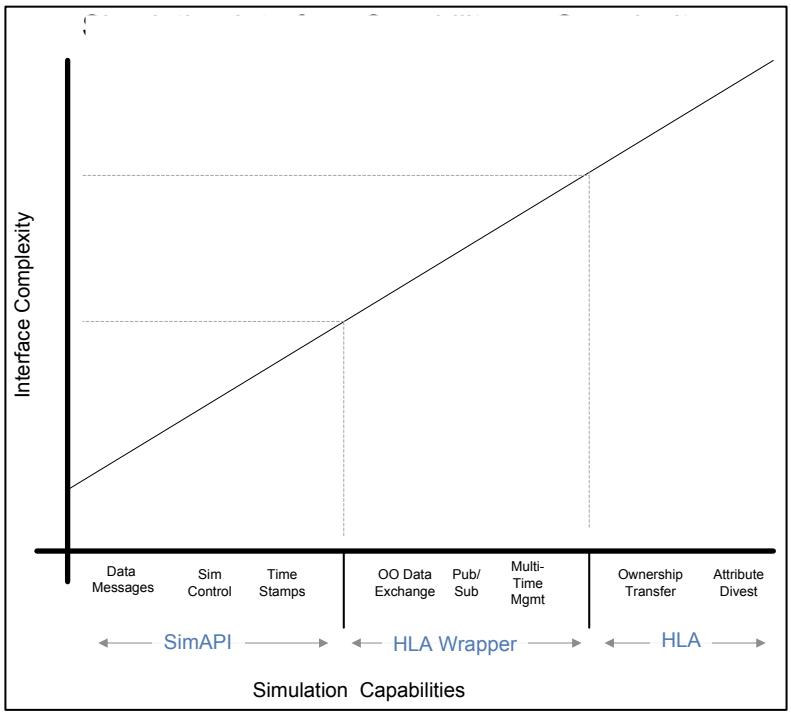

Figure 14. Simulation Interface Capability vs. Complexity

In this notional example using HLA, the wrapper would provide a significant reduction in needed interface complexity for those simulation problems that do not require the full suite of simulation capability that HLA affords.

The Model - A Modified Framework Architecture Is Required: Figure 15 shows a simulation architecture created by Honeywell that utilizes a "SIM API" interface that effectively allows the removal of simulation interface complexity from the simulation itself The SIM API frees the simulation developer to focus solely on the simulation model and the simulation integrator to focus on the data needed for the simulations to interact. The SIM API is based upon a simulation interface specification that also allows simulations designed with it to be easily reused between simulation architecture environments (e.g. DIS vs HLA).

Legacy simulations need to be updated to meet SIM API specification. This is not a labor intensive effort.

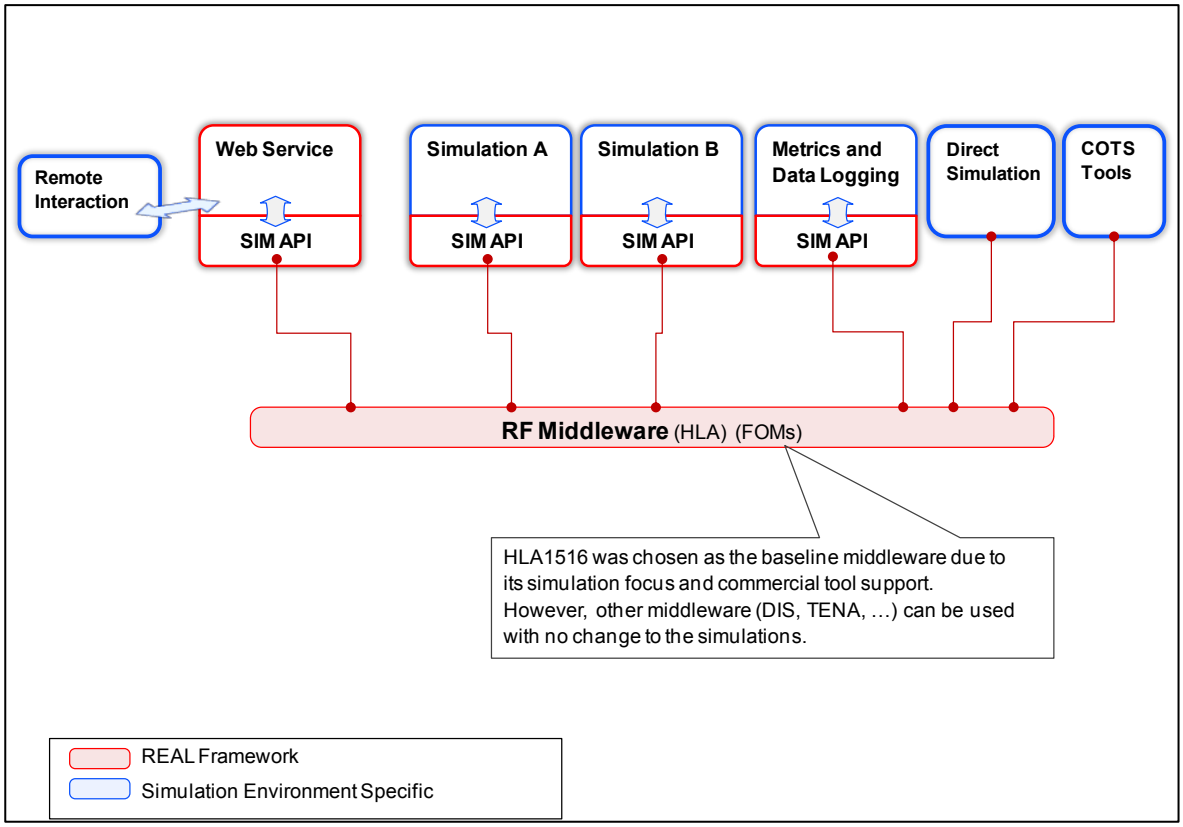

Figure 15. REAL Framework Run Time Architecture

Once done the model transfer effort is reduced from several weeks to less than a day. This reduction in effort is possible because interfacing with existing HLA or DIS standards usually requires the simulation developer to modify or transform data between the internal simulation representation and a representation defined by the standard. The SIM API allows the developer to use any representation, minimizing the translation effort within the individual simulation. Instead, the simulation integrator (system of systems modeler) defines the transformation between the simulation data and the standard in use, which is applied automatically within the SIM API.

2.5.4 The Method - Development Objectives For The SIM API:

The team, that developed the SIM API interface specification, focused upon making the simulation developer and integrator as effective as possible. Effectiveness here being defined as giving the simulation team the ability to realize results from its simulation activity as quickly as possible. Some of the objectives considered include:

- Make it convenient to use:
  - The SIM API should be similar in effort to low cost ad-hoc approaches, such as a simple socket based interface.
  - Allow the simulation to use its own data representation rather than an externally imposed representation.
- Make it ‘write once’ for the simulation modeler
  - Many simulations are available in organizations without the interest or resources to integrate a variety of interfaces
  - Give the system of systems modeler the ability to change the underlying distribution technology (changing FOMS, HLA->DIS, etc) without flowing changes back into the simulations.
- Keep the data formats concise
  - Let the simulation modeler define the data in simulation terms,
  - Recognize that any simulations have limited data needs,

- Let the SoS modeler perform the data connection and translation.

2.5.5 Current progress and Conclusions:

The REAL Framework environment has been developed and is currently dispersed to several locations across the globe in support of aerospace avionics related human factors studies in simulated airspace conditions. The simulation environment itself is working as intended with the expected reduction in simulation integration times being realized. There are no anticipated technical obstacles to further deployment and usage. However, there are cultural and awareness issues that need to be dealt for each engagement or deployment. These issues and concerns usually arise from uncertainty within the developer and stakeholder community as to the cost / benefit of engaging in systems of systems simulation effort when they have not been through the experience. The approach discussed here provides a way to quickly show progress and ease of use for those new to simulation or new to using simulation for exploring system concept of operations or training provisions.

Definitions:

DIS – Distributed Interactive Simulation: A standard networking protocol for exchanging information between various simulation applications. Reference IEEE 1278.1.

HLA – High Level Architecture: An architecture framework to support distributed modeling and simulation. Reference IEEE 1516.1

REAL: Reactive Environment for Active Learning

SMART: Simulation and Modeling for Acquisition, Requirements and Training

SMARTlab: Honeywell Advanced Technology modeling and simulation lab, which applies the SMART process for the entire product lifecycle

3.0 Conclusions and Lessons Learned

This paper has summarized five exemplars of usage for modeling libraries based upon the practical experience of the authors in varying contexts. To some degree this is an eclectic mix of experiences; however, there were common themes:

Experience with MBSE environments suggests that the modeling libraries of the future will contain knowledge-rich objects. Modeling objects will contain and be associated with many types of information, including architectural and design patterns, tutorial examples, context of applicability, and integration data. A significant segment of the modeling community wants to go beyond basic descriptions of designs. They want to predict system behavior using overall system models that are integrated with specialized, discipline-specific models. This creates several challenges:

- Most integrated MBSE simulation environments rely upon significant customization effort. General APIs for model interfaces tend to be complex. However, as Lyells describes in Section 2.5, above, there have been successful large-scale implementations that reduce complexity by using specialized APIs such as the SIM API shown in Figure 15. Such middleware can significantly reduce the workload for individual modelers who need to create a system level simulation. We perceive such efforts as one necessary aspect of modeling environments that are scalable to large projects.
- As Cole observes in Section 2.2, in an integrated modeling and simulation environment, discipline specific models need to "project meaning" into the system simulation and vice versa. This semantic interoperability is one key to integrated functioning. There are significant efforts in progress to address these issues. See, for instance, the work of Henson Graves as part of the INCOSE Model Based Systems Initiative.

- There is a delicate balance between standardization and experimentation. We believe that standards work is particularly valuable when it addresses interfaces and interoperability challenges. An example of the power of this approach is the growth of the Internet as supported by the existence of TCP/IP. However, we see the current period as a time of many experiments by many individuals and organizations. Attempts at standardization should honor this process and attempt to support it using a 'design for emergence' philosophy. If we are successful with striking the right balance, we could see a growth in system modeling that rivals the growth of the Internet.
- Discipline-specific models imply specialized views, where each view exposes the objects, features and attributes that support a specialized set of modeling tasks and are represented using the symbology of the domain. Such capabilities are not well-supported at present. Some tools offer capabilities to create customized graphics and there is a preliminary OMG effort to create a diagramming standard. We see this as a positive trend. The nature of the tasks supported dominates the creation and usage of specialized views. A point that Lyells makes strongly is that we need to be clear whether a task is directed toward communicating with humans or communicating with machines. Eventually, it is expected that modeling environments will offer ubiquitous end user programming capabilities, as straightforward to use as spreadsheets; however, this is a challenge that will require significant development effort.
- MBSE environments that incorporate multiple specialized modeling domains tend to use layered architectures. (See, for example, the architectural features displayed in the diagrams offered by Che (Section 2.3) and Lyells (Section 2.5).) We expect these modeling environments to evolve toward Service Oriented Architectures with "plug and play" capabilities for adding discipline-specific modeling capabilities. The experience of working in such an environment will be one of interacting with a large, collaboratively developed model with many specialized views that support the efforts of many experts.

Overall, we see the beginning of a trend to vastly extend the scope of information in MBSE libraries. For language designers, this implies a need for significant extension of current representation capabilities. For tool designers, it means a significant extension of tool capabilities. For users, it implies a great expansion in modeling capabilities and a period of time where it is likely where usability will be severely impacted by increasing complexity while tool and language developers struggle to accrue the insights that they need to create simpler modeling environments.

Lempia, in Section 2.1, offered a perspective on the actors and use cases that arise when human beings interact with libraries. However, the way that humans interact with libraries goes beyond these formal interactions. We expect that the informal interactions will be at least as important as the formal interactions. These informal interactions include modelers borrowing from the work of colleagues and multiple creative interactions that generalize the scope of model components and other knowledge artifacts. We expect such roles and patterns of interactions to continue to evolve.

4.0 The Path Forward

The five perspectives in this paper, based upon five sets of experience with industrial strength modeling activities suggest that the systems modeling community is in a period of rapid experimentation and creative activity. The MBSE Usability Group within the INCOSE Model Based Systems Engineering initiative will continue to support this effort in the

following ways:

- Work on the library usability will continue in the coming year. We have only begun to address the challenges in this area. We welcome participation from the larger modeling community, including modelers, tool vendors and language developers.
- Work on other use cases, such as those identified in the 2011 INCOSE International Workshop will be addressed in the coming year. A specific focus in 2013 will be another high value use case focusing upon the development of model structure and making assertions about particular model structures.

As Steve Jobs observed, "Creativity comes from connections." In this time of rapid experimentation, it is important that individuals and organizations that are experimenting with modeling infrastructure share their insights, so that all of us can benefit from creative interaction. As we build modeling artifacts and interact creatively with our colleagues, each new development becomes a communication on the path of a shared spiral of meaning. Today, many of these processes are occurring using pre-Internet models of interaction, such as sharing papers at yearly seminars. Although such traditional approaches are valuable, we need to move forward with this process of dialog, discussion, and development at Internet speed. We need a business model that supports this accelerated evolution.

5.0 References

Bajaj, M., Zwemer, D., Peak, R., Phung, A., Scott, A., Wilson, M., "Satellites to Supply Chains, Energy to Finance — SLIM for Model-Based Systems Engineering Part 1: Motivation and Concept of SLIM." INCOSE International Symposium, 20-23 June 2011. Denver, CO.

Bayer, T.J.; Seung Chung; Cole, B.; Cooke, B.; Dekens, F.; Delp, C.; Gontijo, I.; Lewis, K.; Moshir, M.; Rasmussen, R.; Wagner, D.; , "Model Based Systems Engineering on the Europa mission concept study," *Aerospace Conference, 2012 IEEE* , 3-10 March 2012.

Belton, C., Bennett, P., Burchill, P., Copp, D., Darnton, N., Butts, K., Che, J., Hieb, B., Jennings, M., Mortimer, T., "A Vehicle Model Architecture for Vehicle System Control Design," 03AE-176, *2003 International Congress of the Society of Automotive Engineering*.

Bone, Mary and Robert Cloutier 2010. "The Current State of Model Based Systems Engineering: Results from the OMG SysML Request for Information 2009." Paper presented at the 8th Conference on Systems Engineering Research, Hoboken, New Jersey, 17-19 March 2010.

Branscomb, J.M., Paredis, C.J., Che, J., Jennings, M.J., "Supporting Multidisciplinary Vehicle Analysis Using a Vehicle Reference Architecture Model in SysML," *Conference on Systems Engineering Research (CSER'13)*, Atlanta, GA, March 19-22, 2013.

Estefan, Jeff A. 2008. "Survey of Model Based Systems Engineering Methodologies (MBSE)." INCOSE Technical Paper INCOSE-TD-2007-003-01, June 10, 2008.

Friedenthal, S., Moore, A., Steiner, R., 2011. *A Practical Guide to SysML, 2nd ed.*, Burlington, MA (US): Morgan Kaufman.

Hause, Matthew 2010. "An Overview of UPDM, a Unified Profile for DODAF and MODAF." Paper presented at the 20th Annual International Council on Systems Engineering (INCOSE) International Symposium, Chicago, Illinois, USA, 12-16 July 2010.

Jankevicius N. 2011. "Executable UML with MagicDraw: Cameo Simulation Toolkit." Presentation to the Open Modeling Group Technology Committee. http://www.omg.org/news/meetings/tc/agendas/va/xUML_pdf/Jankevicius.pdf.

Open Modeling Group. 2011. FUML. *Semantics of a Foundational Subset for Executable UML Models (fUML), v1.0*..

___. 2011. DD. *Diagram Definition (DD)*.

OMG Systems Modeling Language (OMG SysML™), V1.0, 2007b, OMG Document Number: formal/2007-09-01, URL: http://www.omg.org/spec/SysML/1.0/PDF, Accessed November, 2007

Biographies

Judy Che currently works in the Vehicle System Analysis and MBSE department at Ford Research & Advanced Engineering. She specializes in methodologies for simulating vehicle level attributes using full vehicle system models. She joined Ford Motor Company in 1999 after receiving her doctorate from University of Michigan in Mechanical Engineering.

Bjorn Cole, Ph.D. is a systems engineer at the Jet Propulsion Laboratory. He has expertise in Systems Modeling, Systems Engineering, Strategic Technology Planning, Advanced Analytics, and SysML.

David Lempia is a Principal Systems Engineer in the Engineering Infrastructure Development & Lean organization of Rockwell Collins and has over 20 years of experience in systems development. He received his M.S. in electrical engineering from the University of North Dakota in 1987 and his M.S. in Business Administration from the University of Iowa in 2007.

Ron Lyells has been a member of INCOSE for over 6 years. Currently employed by Honeywell's Aerospace Group, Ron is part of a team responsible for improving product development effectiveness across the Aerospace organization. Ron is specifically responsible for developing and promoting a common product development enterprise model to help support integration of all the aerospace product lines. In addition he has also been responsible for developing and promoting a common system engineering design method across the Aerospace organization. Ron has worked in the Aerospace industry 34 years in various leadership positions involved in product development lifecycle stages ranging from proposal to production support. He holds a B.S degree in electrical engineering from Arizona State University.

George Sawyer is a Principle Systems Engineer for BAE Systems, a world-wide defense, aerospace and technical services company. George is one of the company's leading practitioners of Model-Based Engineering and works with a number of divisions to support their adoption of this technology. He is actively involved in using MBSE in the development of rotary and fixed-wing self-protection systems.

Craig Schimmel is a staff engineer working at the Honeywell SMARTLAB facility in Albuquerque. Craig has been involved in numerous modeling and simulation efforts, generally in support of operational analysis with various customers within the U.S. aerospace community.

Scott Workinger, Ph.D., has 35 years experience leading people who create innovative, practical solutions to business problems and field working systems in a broad spectrum of industries. His research at Stanford focused upon usability issues pertaining to the integration of design models and engineering processes. He serves on the Board of Directors for the Silicon Valley Chapter of INCOSE. Since its founding in 2010, Scott has served the MBSE Usability Group as their leader. Today he has an international consulting practice teaching systems engineering, test engineering, architecture, and critical thinking skills. Scott has a passion for empowering his students.