Creating systems engineering products with executable models in a model-based engineering environment

Robert Karban*, Frank G. Dekens*, Maged Elaasar*, Nerijus Jankevičius*

*Jet Propulsion Laboratory California Institute of Technology Pasadena, CA 91109, USA;
No Magic, Inc. Kaunas, LT-51480, Lithuania

*robert.karban@jpl.nasa.gov

Abstract

Applying system engineering across the life-cycle results in a number of products built from interdependent sources of information using different kinds of system level analysis. This paper focuses on leveraging the Executable System Engineering Method (ESEM) [?], which automates requirements verification (e.g. power and mass budget margins and duration analysis of operational modes) using executable SysML [?] models. The particular value proposition is to integrate requirements and executable behavior and performance models for certain system level analysis. The models are created with modeling patterns that involve structural, behavioral and parametric diagrams, and managed by an open source Model Based Engineering Environment (named OpenMBEE [?]). This paper shows how ESEM with OpenMBEE are used to create some required engineering products (e.g. operational concept document) for the Alignment and Phasing System (APS) within the Thirty Meter Telescope (TMT) [?], which is under development by the TMT International Observatory (TIO) [?].

Introduction

The current state of practice in systems engineering reveals many issues with a traditional document-based approach to system design and analysis. In such approach, different stakeholders have their own view (documents) of the mission or system, but they often are inconsistent views of the same system. Each have their own perception of the system, their own “model”. Currently, dependencies between these views are often implicitly represented, and hidden in documents (Figure 1). Additionally, there are often many discipline specific engineering models that also have implicit relationships among them. It is often difficult to trace information across the different sources of information, and verify their consistency. A lot of time is spent by engineers looking for information from other members of the engineering team.

In the rest of this document, we make the case for Model Based Systems Engineering (MBSE), which helps establish the system model as a single source of truth for systems engineering data for a project. We also discuss a method for constructing such models using the standard Systems Modeling Language (SysML), and in such a way that they can be executed by an over-the-shelf SysML execution engine. We show how model executability is crucial in leveraging the model for analysis of requirements satisfaction. Furthermore, we show how such model can serve as a source for producing systems engineering products, in the form of documents that represent different views on the system for different stakeholders. We demonstrate our ideas on a running example that analyzes
various requirements on Alignment and Phasing System (APS) within the Thirty Meter Telescope (TMT) project, which is under development by the TMT International Observatory (TIO) [?].

Figure 1. Implicit dependencies between system engineering documents

Background

Current Practice

Modern engineering relies heavily on models in different domains (e.g. mechanics, optics, software, control). Models (e.g. CAD, FEM, MATLAB) are ubiquitous in domain specific engineering (see Figure 2). However there are many other document based artifacts, which either describe those models or explain how their content is related. The latter kind of artifacts is typically prevalent in systems engineering. The complexity and often sheer amount of information in those documents triggers the need for Model Based Systems Engineering (MBSE), which relies on models, rather than documents, as the source of truth for systems engineering information. These system level models can be used to tie together and federate different sub-system (e.g., mechanical, thermal, electrical) models. It can also be used to generate different views (in the form of documents) that are needed by the stakeholders of a systems project. This paper discusses a method and tooling environment to do all that.
Systems Engineering Process

The international standard, ISO-15288 [?] is intended to harmonise the framework of processes used by any organization or project throughout the full lifecycle of a man-made system. In ISO-15288, systems engineering processes (Figure 3) are organized into five groups; Agreement, Enterprise, Project, Technical and Special. The Technical group of systems engineering processes comprises: Stakeholder Requirements Definition, Requirements Analysis, Architectural Design, Implementation, Integration, Verification, Transition, Validation, Operation, Maintenance, and Disposal. We focuses mainly on the first three activities.

A typical systems engineering process to define and develop a system design can be organized into four categories: 1) requirements development, 2) architectural design, 3) technical evaluation, 4) Synthesis. These four categories are conducted iteratively over several phases, and in parallel to each other, culminating in regular reviews. This framework is also intended to be applied recursively, at each level of the design.

INCOSE [?] adopted a V-model (Figure 4) for its systems engineering process used to define and develop a system. The model describes the activities to be performed and the results that have to be produced during systems
development. The left side of the "V" represents the decomposition of requirements, and creation of system specifications. The right side of the V represents integration of parts and their validation.

![INCOSE V-model of the systems engineering process](image)

Figure 4. The INCOSE V-model of the systems engineering process

Model Based Systems Engineering

Model based systems engineering (MBSE) is the formalized application of modeling techniques to support system requirements, design, analysis, verification, validation and documentation activities, beginning in the conceptual design phase and continuing throughout later life cycle phases. A system model is an abstraction of selected aspects of the structure and behavior of a system that is expressed using a modeling language. SysML [?] is a standard, visual, and general purpose system modeling language developed by the Object Management Group (OMG, [?]) and often used in the context of MBSE. In addition, a method can be applied to the process of implementing MBSE such as Object Oriented System Engineering Method (OOSEM) [?]. MBSE driven method uses models as an integral part of the systems engineering process to:

- Capture requirements and design information
- Integrate with software, hardware, analysis, and test processes
- Accommodate changing requirements and design evolution

In order to correctly use MBSE, the traditional systems engineering’s V-Process needs to be contextualized to the model based paradigm. The suggested JPL V-Process (Figure 5) introduces different models for each step in the process. In this paper we focus on the project and system models (i.e., L2-L3) and associated modeling artifacts that are created early in the development process.
Object Oriented Systems Engineering Method

Object Oriented Systems Engineering Method (OOSEM) provides an integrated framework that combines object-oriented techniques, a model-based design approach and traditional top-down systems engineering (SE) practices. OOSEM is a scenario driven process coupling top-down decomposition with bottom-up design. It provides guidance on building a system model to analyze, specify, design, and verify the system. An example application of OOSEM is also described in detail in chapter 17 of [?]. The following activities are defined in OOSEM: Analyze Stakeholder Needs, Analyze System Requirements, Define Logical Architecture, Synthesize Candidate Physical Architectures, Optimize and Evaluate Alternatives, Manage Requirements Traceability, and; Validate and Verify Systems.

As a set, all of these activities, like their counterparts in ISO-15288, are intended to be performed iteratively and recursively to develop a system-of-systems. ISO-15288 defines 'what' needs to be done, then OOSEM defines 'how' that can be done.

OOSEM defines the architecture in terms of the Domain which provides the context of the solution; the Enterprise which provides the ecosystem of the solution; the System of Interest which is the solution being specified. The System of interest itself is broken down and specified in three ways:

- Black Box: externally visible specification
- Logical: whitebox functional specification
- Physical: whitebox realization specification

Figure 6a shows a recursive V-process and indicates the where the OOSEM activities are applied.
Figure 6a. System Development Process

Figure 6b shows the major and common activities along the lifecycle of systems engineering when applying OOSEM.
SysML

The OMG systems Modeling Language (OMG SysML™) is a general purpose graphical modeling language that supports the analysis, specification, design, verification, and validation of complex systems. These systems may include hardware and equipment, software, data, personnel, procedures, facilities, and other elements of human-made and natural systems. The language is intended to help specify and architect systems and to specify components that can be designed using other domain-specific languages such as UML for software design, or three-dimensional geometric modeling for mechanical design. SysML is intended to facilitate the application of an MBSE approach to create a cohesive and consistent model of the system. In particular, the language provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and parametrics, which is used to integrate with other engineering analysis models.

Executable Models

Most SysML models today are created for documentation purposes where the focus is on syntax and notation. Some SysML models are created to gain system understanding, explore and validate desirable or undesirable behaviors of a system where the focus is rather on semantics. For execution to work, semantics have to be precisely defined, which also helps validate models for correctness. The executability of models also enables debugging them to see if the behavior captured by the modeler is what the modeler actually expects. Executable models require an execution or simulation engine to execute. The purpose of a simulation or solver is to gain system understanding, explore and validate desirable or undesirable behaviors of a system without manipulating the real system. This might be because the real system may not have yet been built or available, or because it cannot be exercised directly due to cost, time, resources or risk constraints. Simulation or solving is typically performed on a model of the system, by visualizing and identifying system defects in early development stages when changes are less expensive to make.

Executable Systems Engineering Method

The Executable Systems Engineering Method (ESEM), a refinement of OOSEM, introduces the next phase of system modeling emphasizing executable models to enhance understanding, precision, and verification of requirements to support requirements analysis and verification. It augments the OOSEM activities by enabling executable models. ESEM produces executable SysML models that verify requirements and includes a set of analysis patterns that are specified with various SysML structural, behavioral and parametric diagrams. It also enables integration of supplier/customer models.

The red encircled portions in Figure 6 show where ESEM injects formal modeling methods.

Systems Analysis

ESEM enables systems analysis by carrying out quantitative assessments of systems in order to select and/or update the most efficient system architecture and to generate derived engineering data. System analysis provides a rigorous approach to technical decision-making. It is used to perform trade studies, and includes modeling and simulation, cost analysis, technical risks analysis, and effectiveness analysis. In particular, it supports requirements verification,
which is a kind of systems analysis assessing whether a system design meets the objectives and satisfies the constraints that are levied by the system requirements.

**Tooling Infrastructure**

As mentioned earlier, engineers live in a landscape of a variety of tools and information models in different domains (e.g. ALM, PLM, CAD), which are often implicitly connected [17]. In order to successfully apply model based engineering, those models and their information must be connected explicitly and managed. The systems engineering data (e.g. architectural models) need to be integrated with the rest of the engineering world. The resulting information structure is a graph, as shown in Figure 7.

![Figure 7. A graph of interconnected engineering models including systems engineering ones](image)

The executable aspect of ESEM system models allows for a much more sophisticated integration with other models, in particular analysis models, due to its well defined semantics and support for co-simulation environments. Such an integration is the goal of OpenCAE [?], the mission engineering environment of JPL. JPL missions are developed using a wide variety of software tools. The OpenCAE system aims to provide a platform for these tools to work together in order to support JPL's various projects. This platform incorporates tooling from systems, software, mechanical, and electrical domains. This means that services and tools must be able to exchange data. Lifecycle support for these tools is also provided, which includes configuration management, archiving, business process implementation, and review support. A proper lifecycle support means that tools can be used with the appropriate level of confidence, whether they are simply exploratory or used to support a critical decision. The "Open" part of CAE is about leveraging the creativity of the JPL technical workforce in order to add functionality to the platform. It emphasizes standards for data interchange such as REST to provide for easier connections.

The environment for systems engineering is a view of OpenCAE with the parts relevant to systems engineering. It strives for a formalized application of systems engineering (leveraging models) 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 by tracking relations between heterogeneous data sources, using a graph database, within a linked data architecture.

OpenMBEE [?] is the open-source portion of OpenCAE providing a platform for modeling that serves SysML clients and web-based view interaction serving as multi-tool and multi-repository integration across engineering and
management disciplines of JPL’s mission environment. It provides basic infrastructure for versioning, workflow, access control, flexibility of content, support for web applications and web-based API access. Figure 8 shows a part of OpenCAE whereas OpenMBEE is encircled in red.

![Figure 8. OpenCAE architecture with OpenMBEE encircled in red](image)

**TMT Application**

In order to help the reader follow our proposed method for model based system analysis, we introduce a current example that we will be referring to. The example is extracted from a production system level model that involves the modeling and analysis of the Alignment and Phasing System (APS) within the Thirty Meter Telescope (TMT) [9], under development by the TMT International Observatory (TIO). The full analysis is not presented here for brevity but is available in [?] and [?].
The Jet Propulsion Lab (JPL) participates in several subsystems of TMT and delivers the complete APS. The TIO is the customer, which provides the APS requirements to JPL, and JPL delivers an operational system to the TIO. The APS team pursues an MBSE approach to analyze the requirements, come up with an architecture design and eventually an implementation. The APS team uses several modeling patterns to capture information such as the requirements, the operational scenarios (use cases), involved subsystems and their interaction points, the information exchanged, the estimated or required time durations, and the mass and power consumption. The focus for this example is on defining executable SysML models to simulate scenarios to produce static mass and dynamic (state dependent) power rollups, and duration analysis. The executable SysML model captures requirements, operational scenarios (use cases with estimated durations of actions, e.g. post segment-exchange alignment where the customer requirement is a maximum duration of 2h and the current best estimate according to system simulation is 1h19m), system decomposition, power and mass characteristics of components, and relationships between subsystems. The goal is to use standard languages and tools, as much as possible, and to minimize custom software development. The associated scenarios (e.g. power up) are analyzed and verified automatically against the system requirements. In addition, the systems engineering team derives requirements for TMT subsystems and develops/refines timing requirements for algorithms, and identifies internal and external interface commands.

Based on the customer supplied requirements and derived use cases, the goals are to a) identify participating subsystems, b) identify interfaces and interactions among subsystems, and c) analyze operation scenarios to ensure that power requirements are always met.

The system model, constructed in the process, is also the authoritative source for the following documents which are produced using the OpenMBEE tooling:

- Requirement Flow-Down Document
- Operational Scenario Document
- Design Description Document
- Interface Control Documents

Following the ESEM method, different levels of abstractions are modeled and analyzed. The so-called black box specification (Figure 9) captures the system of interest in terms of its interactions with other subsystems such as the Telescope Control System (TCS) and M1 Control System (M1CS). The black-box specification also identifies the required operations, top-level behavior and measures of performance (MoP).
Figure 9. The black-box interaction between the APS and other components

The conceptual model (Figure 10) identifies functional components of the APS, which are technology independent and their behavior. This model is used to analyze duration of operational scenarios.

Figure 10. The APS conceptual model

The behaviors of the AP components (Figure 11) are captured with UML state machines and activities which communicate over SysML ports.
Each state in a state machine represents a step in a scenario (Figure 12). The estimated time of the leaf actions is captured in SysML duration constraints.
The conceptual model can be queried for information to produce the Interface Control Documents (ICD) (Figure 13) by querying which information (SysML Signal) is sent by which component over which port to other components. The execution of the model accurately simulates the parallel operations of each of the state machines, such that the execution time estimate is representative of the system to be build.

The conceptual model (Figure 14) serves as a specification for the realization model, which in turn provides the specification for implementation. It is the “as-specified” system.
The analysis of the as-specified system can be triggered by a requirements change (Figure 15) or by a change of the specification at conceptual or realization level. A change of a TMT requirement (kept in DOORS as management tool for textual requirements within OpenCAE) is propagated to the SysML model, managed in a model repository in OpenCAE, where it is formalized into so-called property based requirements, which allow for a formalized trace of requirements into the design.

A change of the system design (e.g. durations, power values) is updated either in MagicDraw [?] (a desktop SysML modeling tool) or in the web client (aka View Editor, which provides a simple interface to edit the SysML models in the model repository). The system model is then simulated (executed) with the Cameo Simulation Toolkit (CST) [?]. CST is a plugin to MagicDraw enabling model execution for early system behavior simulation. The toolkit comes with a built in parametric solver and integrations with popular analysis tools (e.g., MATLAB/Simulink, Maple, and Mathematica). CST provides Functional Mockup Interface (FMI) [?] integration for co-simulation.

The simulation runs a particular system level scenario (e.g. powering up the system) and generates power profiles, as shown in Figure 16. Those power profiles are automatically verified against the requirements (e.g. maximum peak power load) for different parts of the system (e.g. within the telescope dome) and declared passed or failed.
The results are published automatically in the analysis documents (Figure 17) explicitly traced to the as-specified system, and therefore providing the information needed to assess whether the design satisfies the requirement or not.

Future Works

The Functional Mockup Interface (or FMI) [?] defines a standardized interface to be used in computer simulations to develop complex cyber-physical systems (Figure 18). The vision of FMI is to support this approach: if the real
The product is to be assembled from a wide range of parts interacting in complex ways, each controlled by a complex set of physical laws, then it should be possible to create a virtual product that can be assembled from a set of models that each represent a combination of parts, each a model of the physical laws as well as a model of the control systems (using electronics, hydraulics, digital software, etc.) assembled digitally.

Figure 18. Separate model authoring and execution

ESEM plans in the future to allow for integration of system level simulation, as well as, integrated co-simulation. The typical FMI approach is that authoring modeling or simulation tool generates and wraps models into FMI interface (Functional Mockup Units) and saves into standardized *.fmu file format. An FMU may contain models, model description, source code, and shared libraries for multiple platforms. These files can be imported and represented in other modeling and co-simulation environments such as SysML tool in our case. FMI units are represented as black-box blocks with properties in SysML Block Definition diagrams (BDDs) and interconnected with other blocks in SysML Internal Block diagrams for co-simulation (Figure 19). The SysML tool acts as a co-simulation master, controls communication steps and updates slave model input/output parameters after every time step (FMI doStep function) until simulation end time is reached. This approach potentially allows us to integrate existing Simulink, Modelica, Maple and other models in the future.

Figure 19. Example co-simulation of system model with domain-specific model

CONCLUSION

Requirement verification is an important kind of analysis that is often performed in the context of MBSE. In order to carry out such an analysis, the requirements, the executable behavioral and the performance model have to be
integrated in the same model. This is a very important value proposition for the ESEM method described in this paper. ESEM can be used to automate requirements verification with a set of executable SysML modeling patterns. The resulting executable model can be used, along with a documentation tool, such as ViewEditor from our OpenCAE tool stack to produce system engineering products (documents expected by systems engineers). The combination of method and tool infrastructure, proposed in this paper, allows us to integrate ubiquitous models from domain engineering and relate the currently disconnected document based artifacts. This way, requirements can be automatically verified against architecture and design, where as the tool infrastructure allows to produce consistent model based project documentation. Furthermore, we plan in the future to improve the method and the tool infrastructure to tie in system level models with domain specific models (often used for subsystem analysis), leveraging co-simulation (FMI).

Acknowledgements

This research was carried out at the Jet Propulsion Laboratory (JPL), California Institute of Technology, under a contract with the National Aeronautics and Space Administration (NASA), and at NoMagic.

REFERENCES