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

Robert Karban*[a], Frank G. Dekens[a], Sebastian Herzig[a], Maged Elaasar[a], Nerijus Jankevičius[b]
[a]Jet Propulsion Laboratory California Institute of Technology Pasadena, CA 91109, USA;
[b]No Magic, Inc. Kaunas, LT-51480, Lithuania
*robert.karban@jpl.nasa.gov


## ABSTRACT

Applying systems 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) [1] [2], which automates requirements verification (e.g. power and mass budget margins and duration analysis of operational modes) using executable SysML [3] models. The particular value proposition is to integrate requirements, and executable behavior and performance models for certain types of system level analysis. The models are created with modeling patterns that involve structural, behavioral and parametric diagrams, and are managed by an open source Model Based Engineering Environment (named OpenMBEE [4]). This paper demonstrates how the ESEM is applied in conjunction with OpenMBEE to create key engineering products (e.g. operational concept document) for the Alignment and Phasing System (APS) within the Thirty Meter Telescope (TMT) project [5], which is under development by the TMT International Observatory (TIO) [5].

**Keywords:** MBSE, SysML, Verification, Requirements

## 1. INTRODUCTION

In current systems engineering practice, system design and analysis are traditionally performed using a document-centric approach. Document-centric approaches lead to stakeholders producing a number of documents that represent their respective views on the mission or system under development. Given the ad-hoc and informal nature of natural language documents, and lack of integration, these views can quickly become inconsistent in practice. Currently, dependencies between these views are often only implicit, or informally defined in documents (Figure 1). Additionally, documents are often accompanied by a variety of discipline-specific engineering models that also have only implicit relationships among them. It is often difficult to trace information across the different, distributed sources of information and verify their consistency. A lot of time is spent by engineers looking for up-to-date information from other stakeholders.

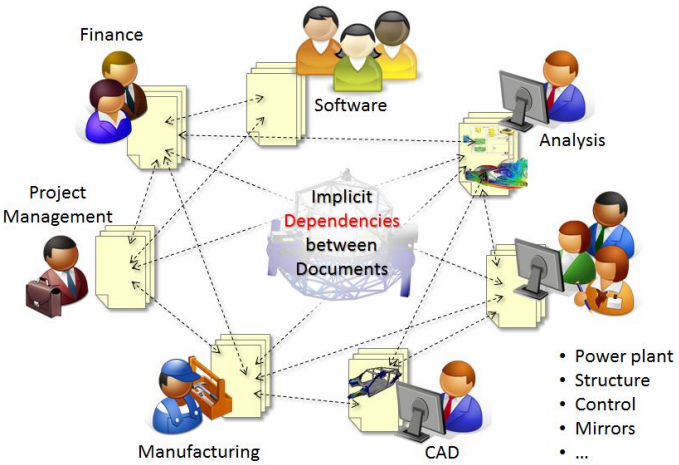



Figure 1. Implicit dependencies between system engineering documents

In the remainder of this document, we make the case for Model Based Systems Engineering (MBSE), which advocates the development of a system model to provide a single source of truth for systems engineering data for a project. We also discuss and put into context a methodology for constructing system models using the standardized Systems Modeling Language (SysML), and in such a way that the specified behavior can be executed by an off-the-shelf SysML execution engine. We show how model executability is crucial in leveraging the model for analysis of requirements and their verification through simulation. Furthermore, we show how such model can serve as a source for semi-automatically generating and maintaining systems engineering deliverables in the form of documents that represent a consistent set of different views on the system for different stakeholders. We demonstrate our ideas using a running example that analyzes various requirements on the Alignment and Phasing System (APS) of the Thirty Meter Telescope (TMT), which is under development by the TMT International Observatory (TIO) [5].

# 2. BACKGROUND

## 2.1 Current Practice

Engineering of complex systems relies heavily on the use of models addressing concerns from different domains (e.g. mechanics, optics, software, control). Models (e.g. CAD, FEM, and MATLAB) are ubiquitous in domain-specific engineering (see Figure 2). However, there are many other document-based artifacts, which either describe these models or explain how their content is related. Typical in systems engineering are the latter. The complexity of, and often vast amount of information contained in such documents motivate the need for Model Based Systems Engineering (MBSE). MBSE relies on formal models rather than documents to act 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, and can also be used to generate different views (in the form of documents) that are needed by the stakeholders of a systems engineering effort. This paper discusses a supporting methodology and tooling environment.

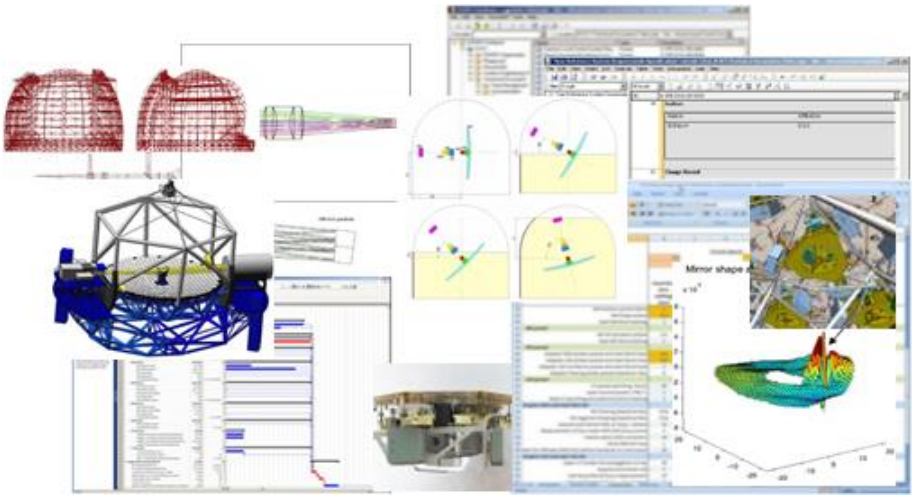


Figure 2. Landscape of engineering models

## 2.2 Systems Engineering Process

A typical systems engineering process (Figure 3) that follows the international standard ISO-15288 [6] to design and develop a system can be divided into four key processes: 1) requirements development, 2) architectural design, 3) technical evaluation, and 4) synthesis. These four processes 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 stage of design.

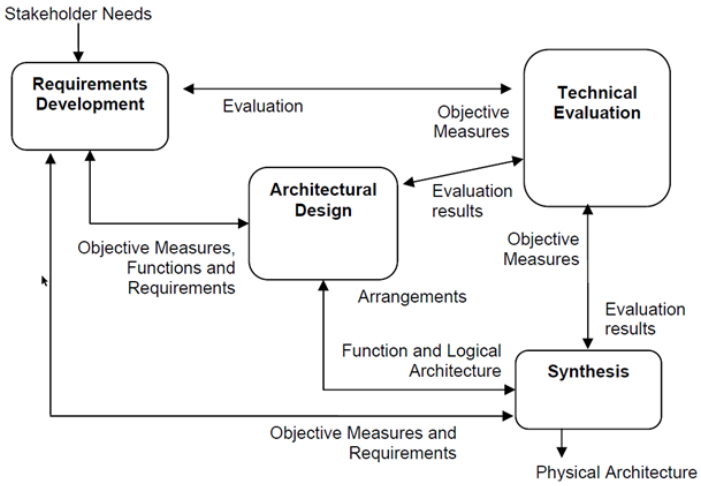



Figure 3. The ISO-15288 systems engineering process [11]

INCOSE [7] adopts a V-model (Figure 4) for its advocated systems engineering process that is to be used for defining and developing arbitrary systems. The model describes both design and development activities to be performed, and the results that should be produced. The left side of the "V" represents the decomposition of requirements and creation of specifications. The right side of the V represents testing and integration of parts, and validation of the system.

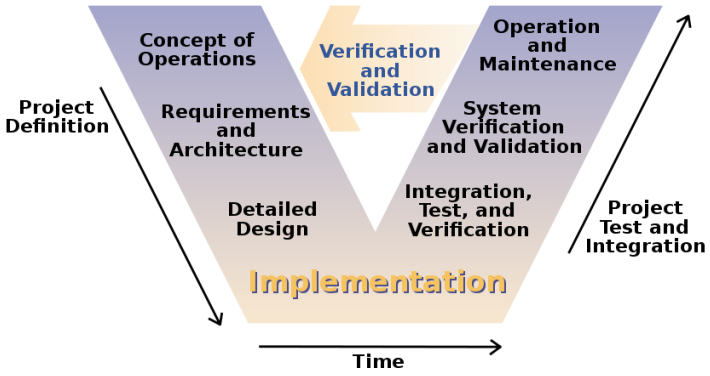



Figure 4. The INCOSE V-model of the systems engineering process [7]

### 2.3 Model-Based Systems Engineering

Model-Based Systems Engineering (MBSE) is the formalized application of modeling 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 and is expressed using a modeling language. SysML [3] is a standard, visual, diagrammatic, and general purpose systems modeling language developed by the Object Management Group (OMG, [8]) and is often used in MBSE applications. There are a number of guiding methodologies that can be applied for creating system models in a structured and methodological fashion, a well-known example of which is the Object Oriented System Engineering Method (OOSEM) [9]. Such methodologies use models as an integral part of the systems engineering process to:

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

In order to correctly apply 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

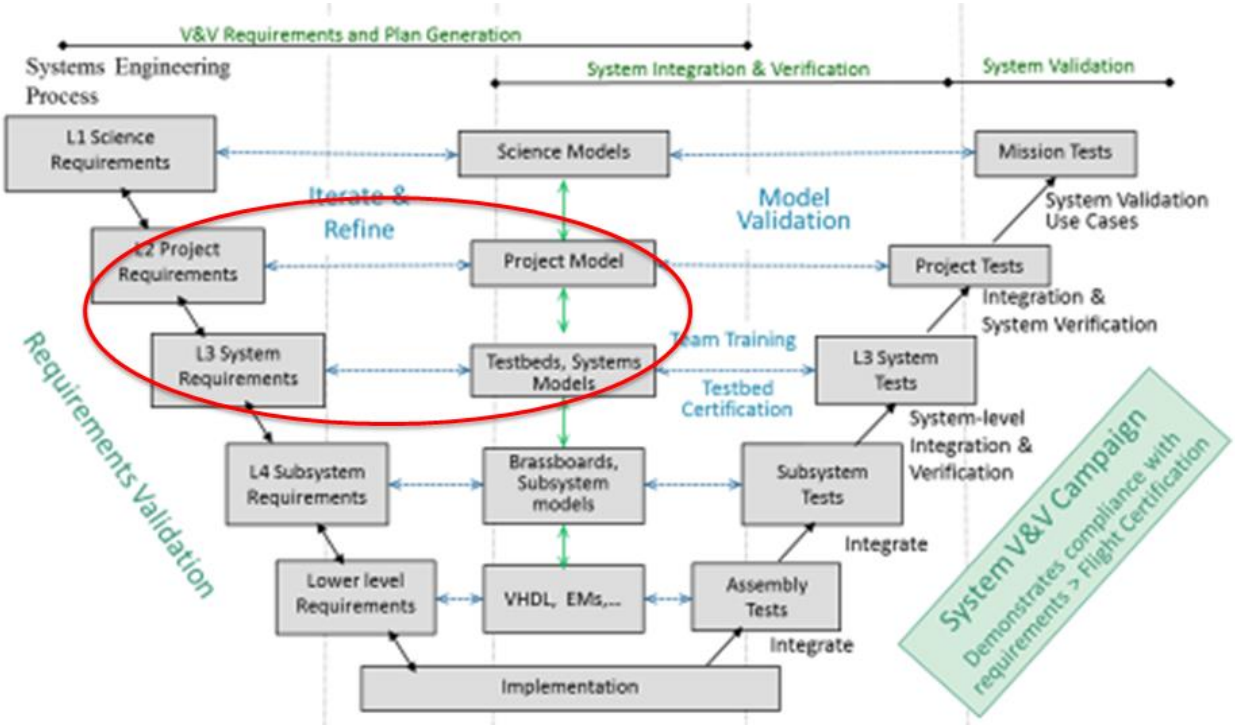



Figure 5. The V-model of the JPL systems engineering process

### 2.4 SysML

The OMG systems Modeling Language (OMG SysML™) [3] 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.

### 2.5 Object Oriented Systems Engineering Method

The 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 [9]. 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. 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 necessary context to the system: the *Enterprise* which provides the ecosystem of the solution, and 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, with top level behavior and system interfaces only
- Logical: white box functional specification
- Physical: white box realization specification

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

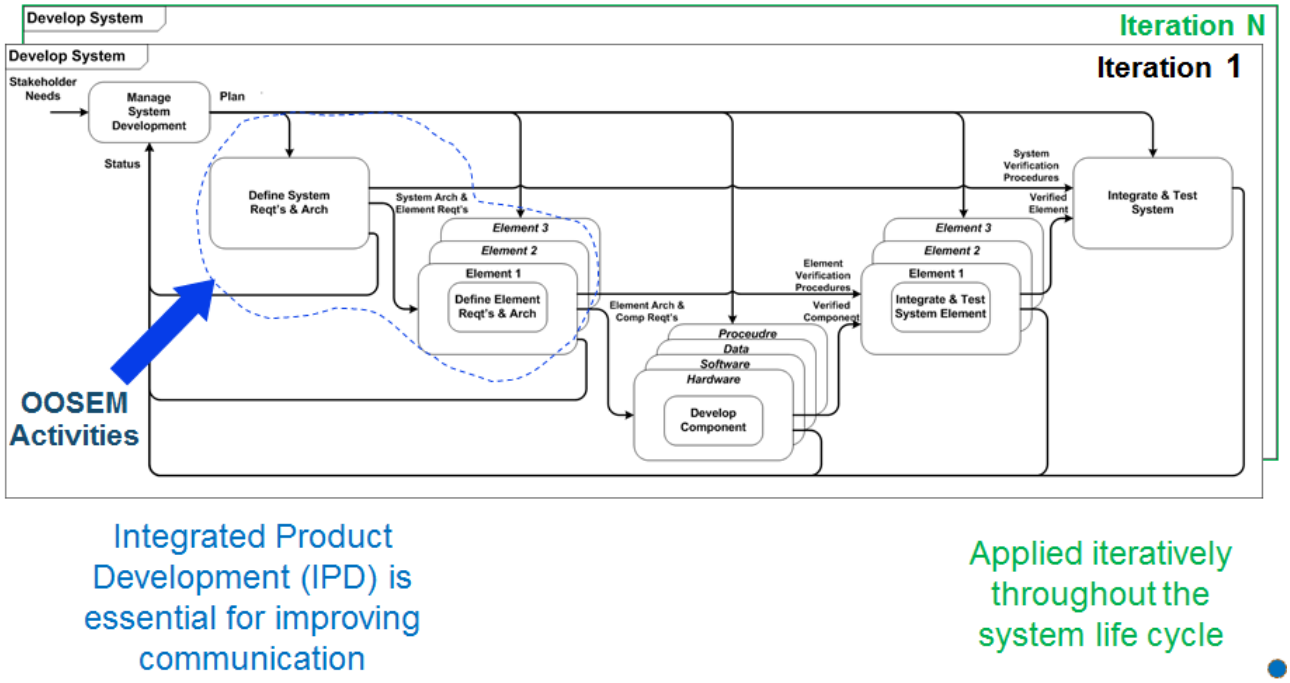



Figure 6a. System Development Process [9]

### 2.6 Executable Models

Most SysML models today are created for documentation purposes where the focus is on syntax and notation. Some SysML models are created for deepening understanding of a system, and exploring and validating desirable or undesirable behaviors of a system, where the focus is rather on execution semantics. For execution to work, execution semantics have to be precisely defined, which also helps validate models for correctness. The executability of models also enables debugging the specified behavior to check whether what is modeled is what the modeler intended to capture and the outcome of simulating its behavior is as expected.
Executable models require an execution or simulation engine that is capable of interpreting the specific execution semantics. The purpose of simulation 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 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. Simulations also have their limitations, especially at early stages, the uncertainty is very high, and the potential for emergent behavior exists; this cannot be accounted for by all formalisms that support simple linear or non-linear, equation-based simulations.

### 2.7 Executable Systems Engineering Method

The Executable Systems Engineering Method (ESEM) [1], 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.
Figure 6b shows the major activities common to systems engineering activities in the lifecycle of a system when applying OOSEM. The red encircled portions in Figure 6 show where ESEM injects formal modeling methods.

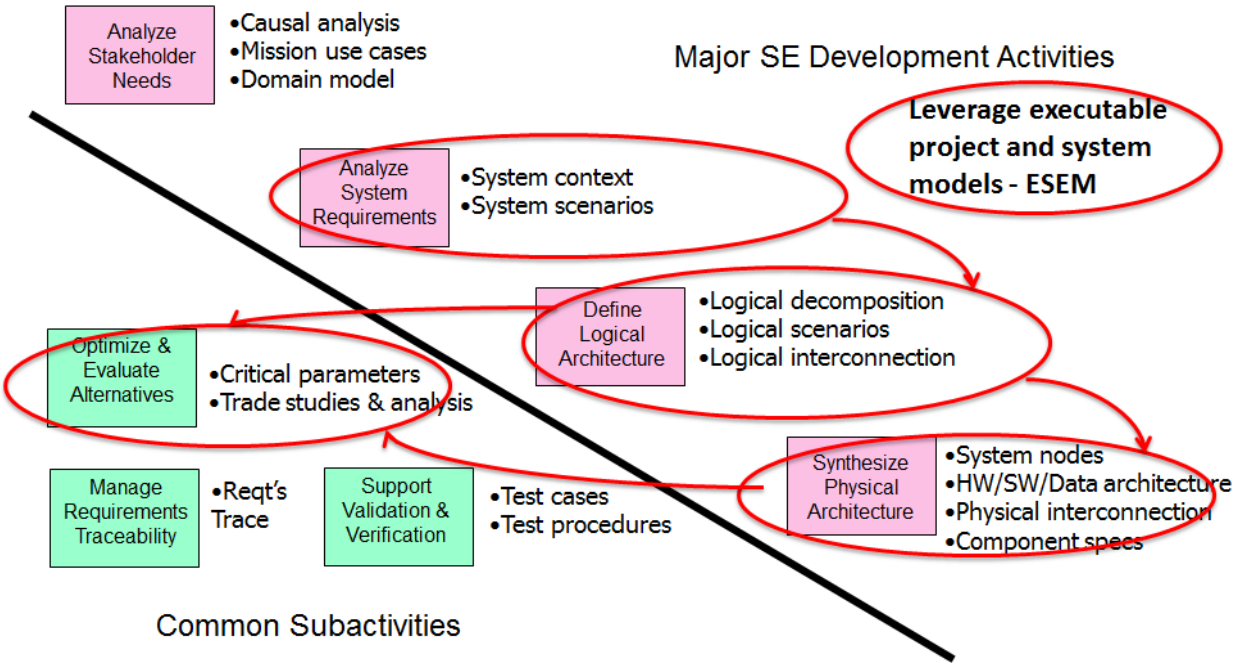



Figure 6b. Activities performed in OOSEM [9]

### 2.8 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.

### 2.9 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 [10]. In order to successfully make use of a model-based engineering paradigm, those models and their information must be formalized, and connected explicitly and managed. The systems engineering data (e.g., architectural models) need to be integrated with the remaining engineering artifacts. The resulting information structure is a graph, as shown in Figure 7. Note, that there are probably also connections between for example ALM, Simulation, Requirements as a complete separation of concerns is not possible.

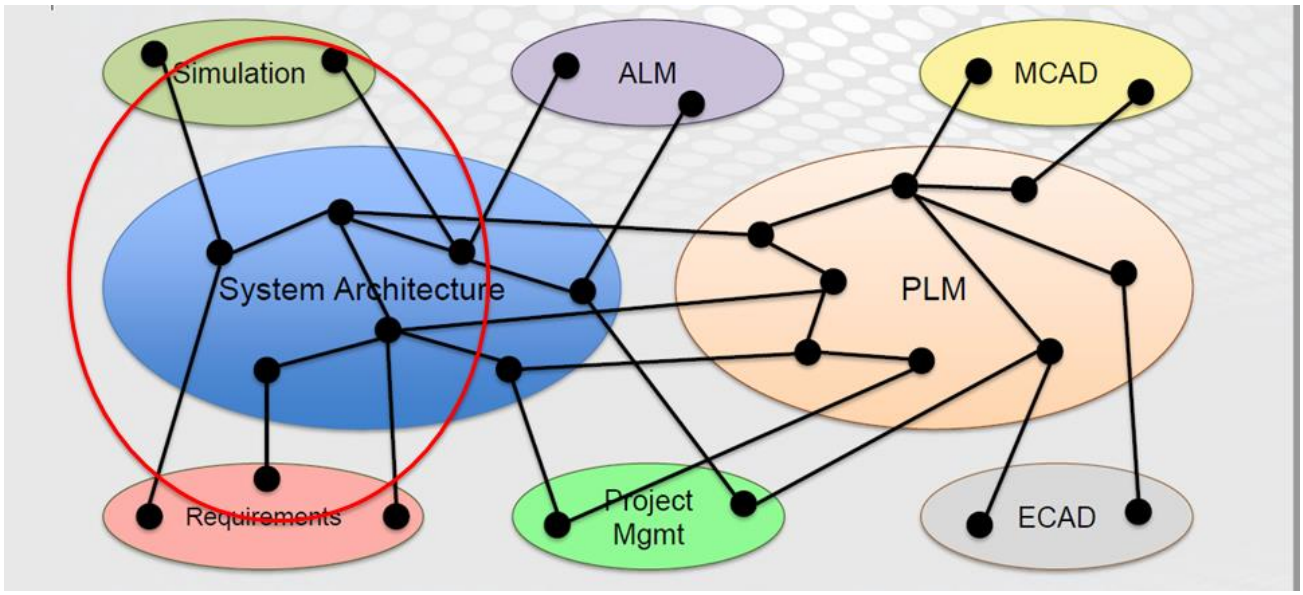



Figure 7. A graph of interconnected engineering models including systems engineering ones [10]

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. Figure 8a shows indicatively a part of OpenCAE software system whereas OpenMBEE is encircled in red.

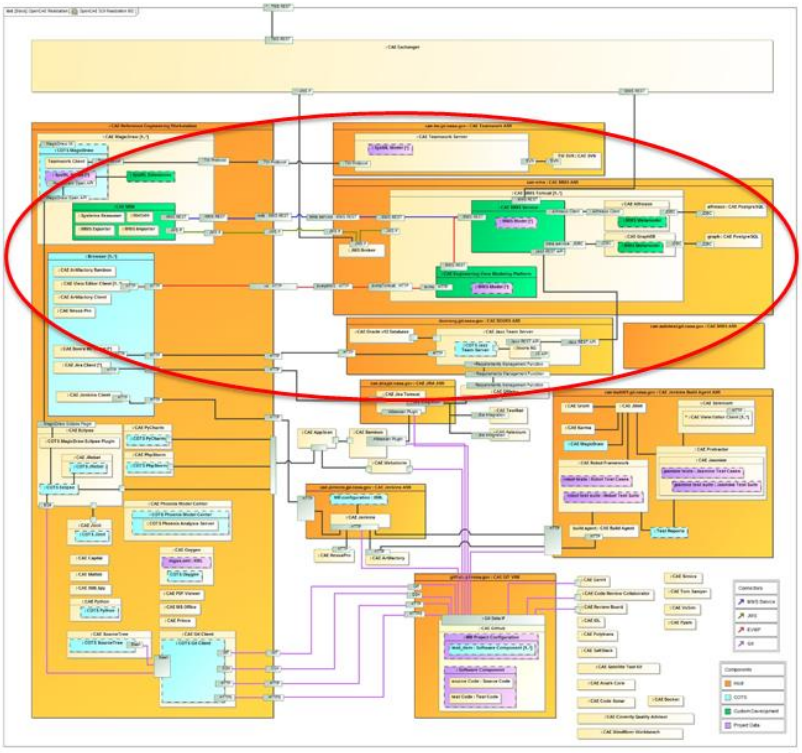


Figure 8a. OpenCAE architecture with OpenMBEE encircled in red

The diagram is meant to give only an impression of the involved infrastructure without going into details. 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.
The environment for systems engineering is composed of OpenCAE applications and services 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. OpenMBEE [4] is the open-source portion of OpenCAE providing a platform for modeling that serves SysML modeling tools acting as clients and a web front-end serving as a multi-tool and multi-repository integration platform across engineering and management disciplines of JPL's mission environment. It provides a basic infrastructure for versioning, workflow, access control, flexibility of content, support for web applications and web-based API access.

Fig. 8b shows example model artefacts produced by OpenMBEE and how they relate and integrate. System models are constructed, queried and rendered following the view and viewpoint paradigm [16] from a repository (Model Management System – MMS). The model can be accessed through a sophisticated modeling tool like MagicDraw or through a web interface (exposing a subset of model operations).

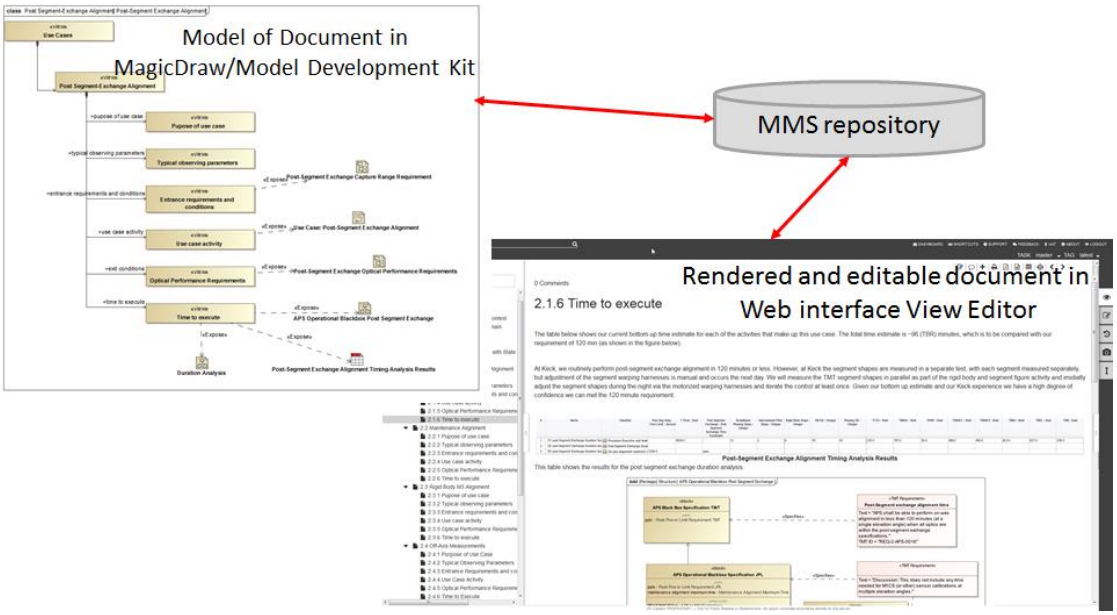



Figure 8b. Interaction between core elements of OpenMBEE

## 3. TMT APPLICATION

In order to help the reader follow our proposed method for model-based systems analysis, we first introduce an example system. The example is extracted from a production-level system model that involves the modeling and analysis of the Alignment and Phasing System (APS) for the Thirty Meter Telescope (TMT) [5], under development by the TMT International Observatory (TIO). The full analysis is not presented here for brevity but is available in [1] and [2].
The Jet Propulsion Lab (JPL) participates in the design and development of 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 using several modeling patterns as described in detail in [1]. Those patterns are only briefly described for brevity. 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).

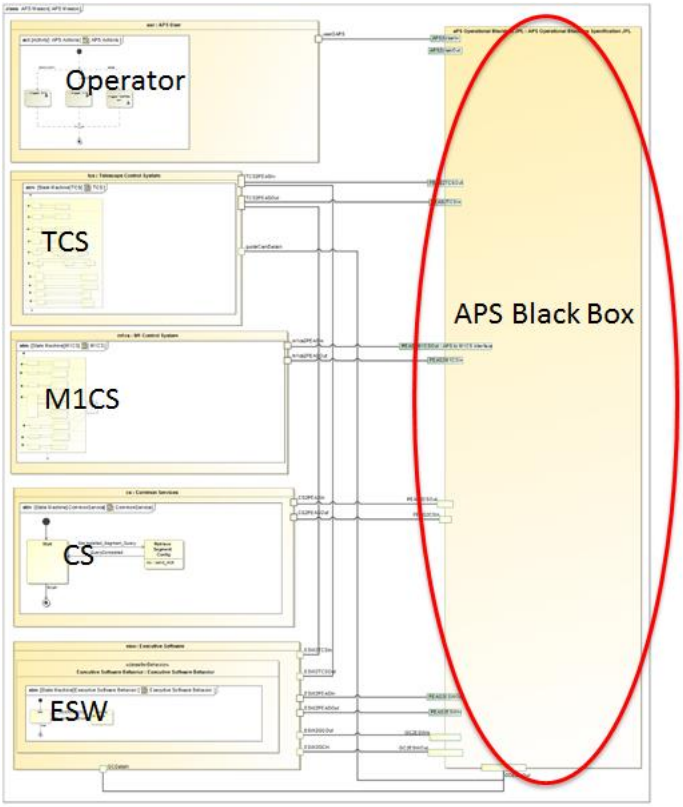



Figure 9. The black-box interaction between the APS and other components

Interactions are modeled using proxy ports that are typed by SysML interface blocks, where the interface blocks specify operations and flows at the interface of APS. The black-box specification also identifies the required operations, top-level behavior and measures of performance (MoP). The conceptual model (Figure 10) specifies functional, technology-independent components of the APS and captures their specified behavior. This part of the model is used in the process of analyzing characteristics such as the duration of certain operational scenarios.

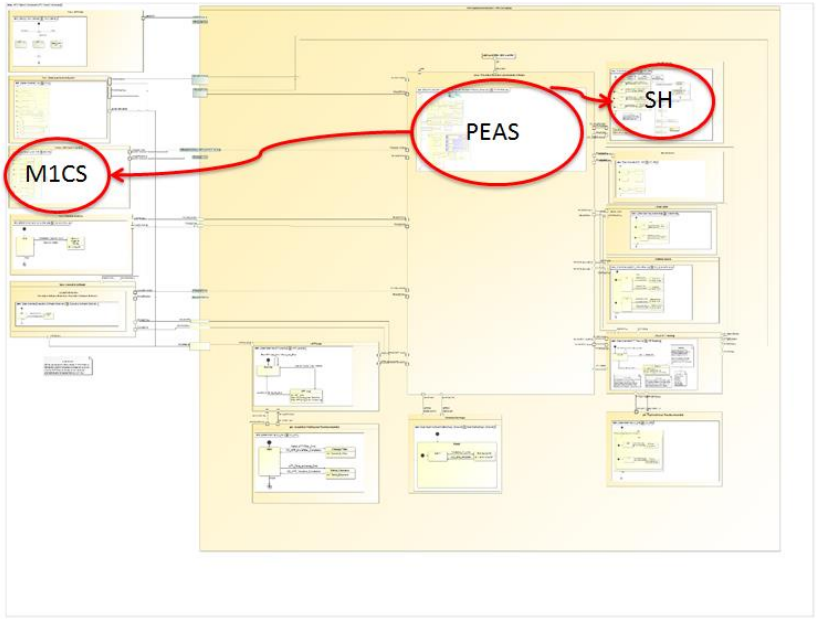
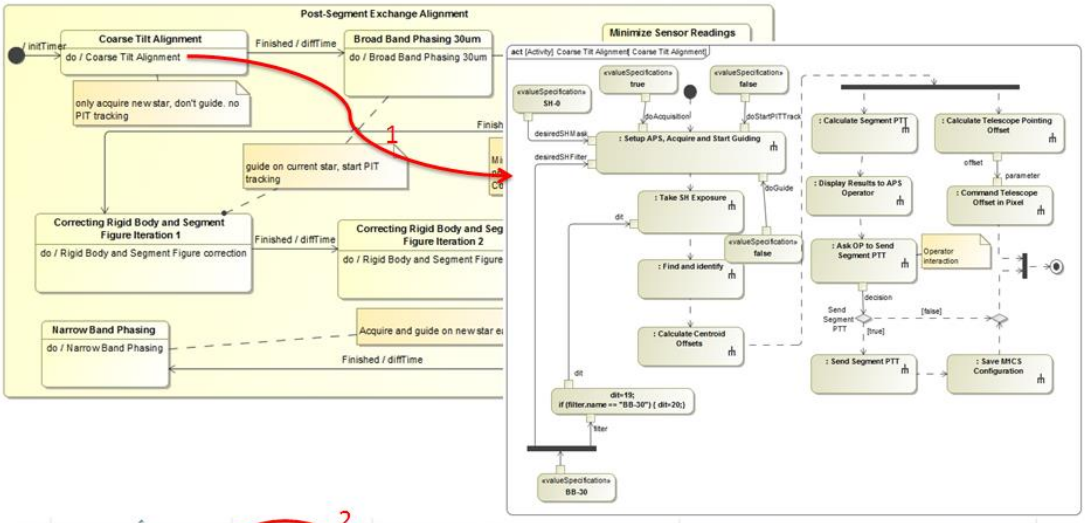



Figure 10. The APS conceptual model

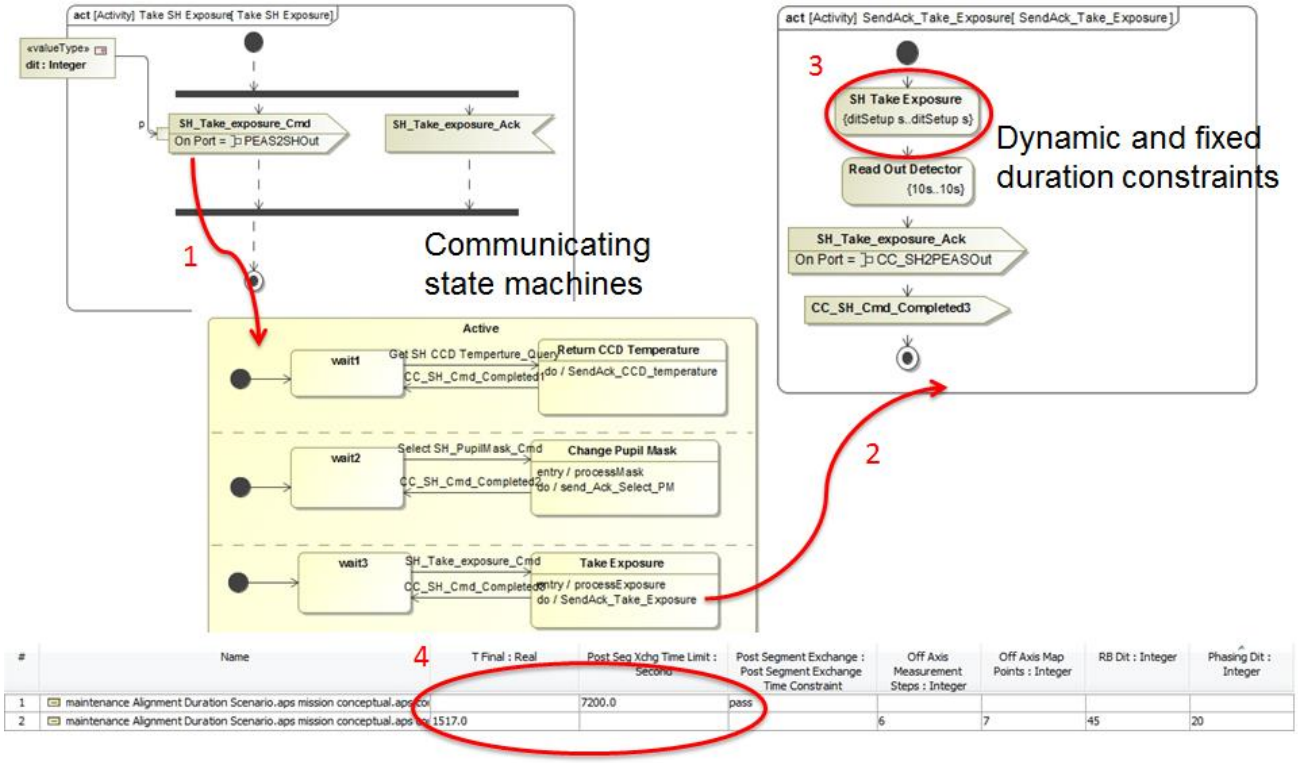
The behavior of the various APS components (Figure 11) is captured using UML state machines and activities (1). Communication across APS components and to components outside of APS is handled by sending signals over SysML ports. Duration constraints (minimum and maximum time) are captured as variables or constants in a table (2).




| # | Name | Specification | Constrained Element | Owner Of Constrained Element |
|---|---|---|---|---|
| 1 | dAPTMoving | 10s..10s | Moving | SendAck_Filter |
| 2 | dAPTTakeExposure | ditSetup s..ditSetup s | APT Take Exposure | Send_Exposure |
| 3 | dCalcCentroidOffsets | 5s..1s | Execute Centroid Offset Calculation | Calculate Centroid Offsets |
| 4 | dCalcPupilRegistrationOffset | 0.5s..1s | Calculate Pupil Registration Offset | Calculate Pupil Registration and Image Offset |
| 5 | dCalcRMSForM2AndSegmentPTT | 1s..3s | Calculate RMS for M2 and Segment PTT Cmds | Calculate RMS for M2 and Segment PTT cmds |

Figure 11. The behavior of some APS components

Each state in a state machine represents a step in a scenario (Figure 12). In a step the activity eventually communicates with another component (1) which triggers its own behavior in a state (2). The estimated execution time of the leaf actions is captured using SysML duration constraints (3).




| # | Name | T Final : Real | Post Seg Xchg Time Limit : Second | Post Segment Exchange : Post Segment Exchange Time Constraint | Off Axis Measurement Steps : Integer | Off Axis Map Points : Integer | RB Dit : Integer | Phasing Dit : Integer |
|---|---|---|---|---|---|---|---|---|
| 1 | maintenance Alignment Duration Scenario.aps mission conceptual.aps | | 7200.0 | pass | | | | |
| 2 | maintenance Alignment Duration Scenario.aps mission conceptual.aps | 1517.0 | | | 6 | 7 | 45 | 20 |

Figure 12. Detailed behavior of some of the APS components

The conceptual model can be queried for information to produce the Interface Control Documents (ICD) (Figure 13) by querying which information is sent by which component over which port to other components. This kind of information that flows is specified using SysML signals and any attached payload. The execution engine accurately simulates the parallel operations of each of the state machines in the model, such that the execution time estimate is representative of the system to be built.

| # | Signal | Signal attributes | On Port | Owner of Port | Owner of Signal Send |
|---|---|---|---|---|---|
| 1 | Post-Segment Exchange Alignment | | user2APS | APS User | Trigger_CTA |
| 2 | Start Fine Tilt Alignment | | user2APS | APS User | Trigger_FTA |
| 3 | Start Get Reference Beam | | user2APS | APS User | Trigger_GetRBeam |
| 4 | Get ShearPlatePos_Ack | pos : TipTilt | SP2PEASOut | Shear plate | SendAck |
| 5 | Set_ShearPlate_Pos_Ack | | SP2PEASOut | Shear plate | SetPosition |
| 6 | PIT_Take_exposure_Cmd | dit : Integer | PITT2PITAOut | PEAS PIT Tracking | Take PIT Exposure( dit : Integer ) |
| 7 | PIT_Update_Camera_Params_Cmd | p : Integer | PITT2PITAOut | PEAS PIT Tracking | Update PIT Camera Params( camParams : Integer ) |
| 8 | SPDataMessage | deltaPos : TipTilt | PITDataOut : PITDataProducerIF | PEAS PIT Tracking | Do PIT Tracking |
| 9 | Get PITStatus Ack | | PITA2PITTOut | Pupil and Image Tracking Assembly | SendAck_Status |
| 10 | PIT_Take_exposure_Ack | | PITA2PITTOut | Pupil and Image Tracking Assembly | SendAck_Exposure |
| 11 | PIT_Update_Camera_Params_Ack | | PITA2PITTOut | Pupil and Image Tracking Assembly | UpdateParams |
| 12 | Update APT Pixel Offset Cmd | offset : APT Pixel Offset | PIT2TCSOut | PEAS PIT Tracking | Command Telescope Offset from PIT( parameter : APT Pixel Offset ) |
| 13 | Set_ShearPlate_Pos_Cmd | tilt : TipTilt | PIT2SPOut | PEAS PIT Tracking | Adjust Shear Plate PIT( parameter : TipTilt ) |
| 14 | Start PIT Tracking_Ack | | PIT2PEASOut | PEAS PIT Tracking | Send_Ack_Start |

Figure 13. An example of ICD for APS

The conceptual model (Figure 14) serves as a basis for the specification for the realization (physical) model, which in turn provides the specification for implementation. It is a representation of the “as-specified” system.

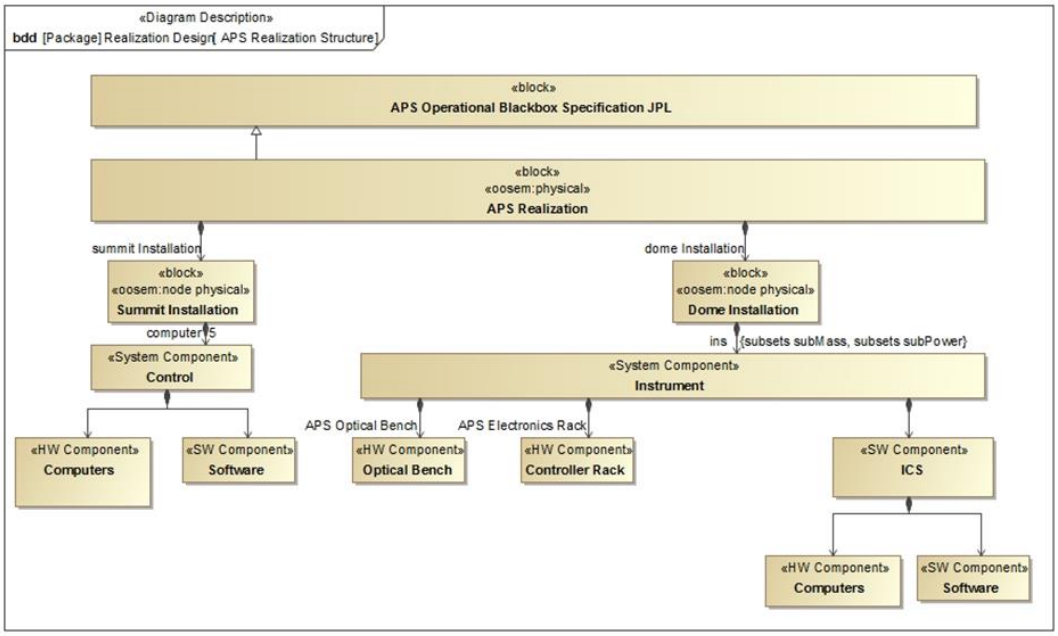



Figure 14. The realization specification for APS

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.

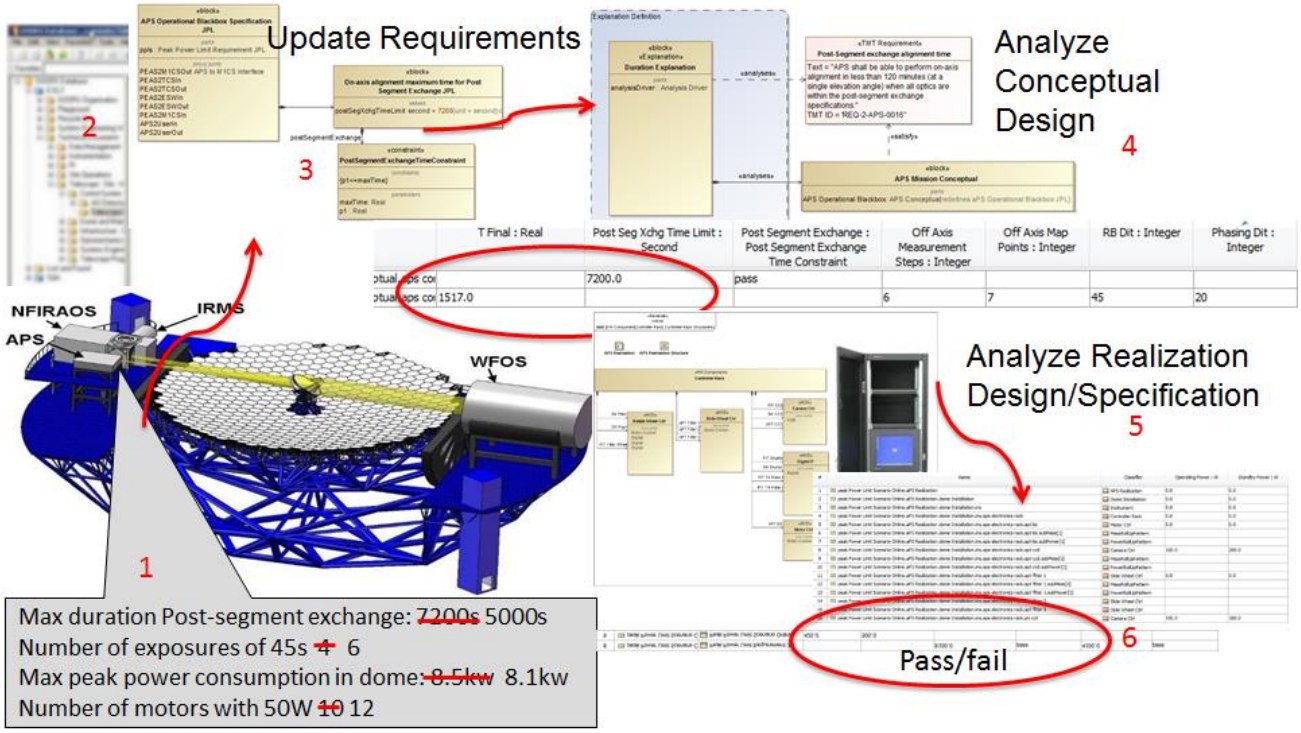



Figure 15. The propagation of a change in the requirements of APS

A change (1) of a TMT requirement (kept in DOORS (2) 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 (3), which allow for a formalized trace of requirements into the design. A property-based requirement allows to capture textual content (e.g. values, constraints) of requirement as distinct SysML model elements and enables formal evaluation of those properties. The as-specified conceptual design (4) and/or the realization design (5) are verified against the changed requirement, resulting in pass or fail (6).
The workflow can also be reversed where a change of the system design (e.g. durations, power values) is updated either in MagicDraw [12] (a desktop SysML modeling tool) or in the web client (called *View Editor*, which provides a simpler interface to edit portions of the SysML models in the model repository). The system model is then simulated (executed) with the Cameo Simulation Toolkit (CST) [13]. CST is a plugin to MagicDraw enabling model execution for early system behavior simulation that natively supports (extended) token flow semantics for Activity Diagrams and State Machines. The toolkit comes with a built-in parametric solver and integrations with popular analysis tools (e.g., MATLAB/Simulink, Maple, and Mathematica). CST also provides Functional Mockup Interface (FMI) [14] 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.

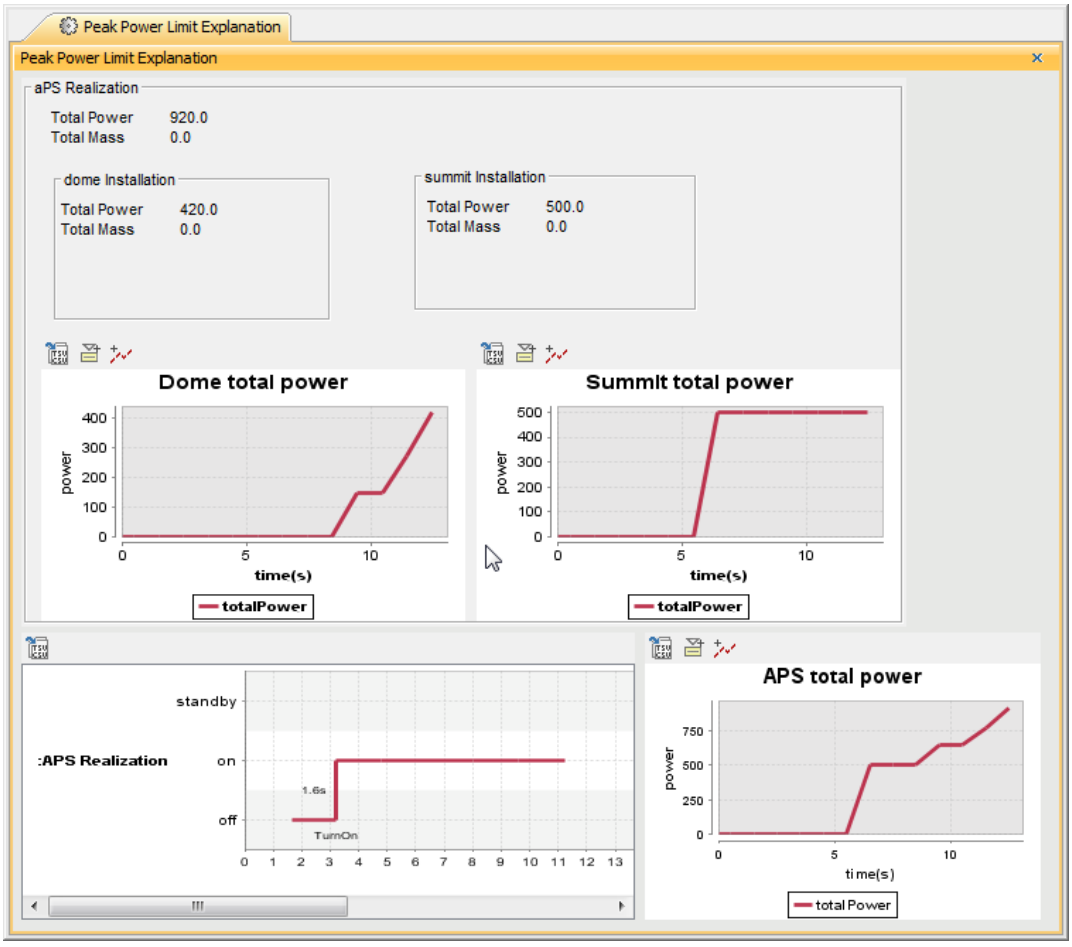



Figure 16. The APS power profiles produced by the Cameo Simulation Toolkit

The results are published automatically in tables embedded in the analysis results documents (Figure 17), and contain explicit traces to components of the as-specified system, and, therefore, provide the information needed to assess whether the design satisfies the requirement or not. The TMT model is available open-source [15].

| # | Peak Power Enc : W | Peak Power Facility : W | Name | Classifier | Power Peak Limit Enclosure : W | Enc : Peak Power Load Constraint | Power Peak Limit Summit Facility Buildings : W | Facility : Peak Power Load Constraint |
|---|---|---|---|---|---|---|---|---|
| 1 | | | peak Power Limit Scenario | Peak Power Limit R | 8100.0 | pass | 4100.0 | pass |
| 2 | 420.0 | 500.0 | peak Power Limit Scenario | Peak Power Limit S | | | | |

Figure 17. The power analysis results shown in an analysis document

## 4. FUTURE WORK

The Functional Mockup Interface (or FMI) [14] 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 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.

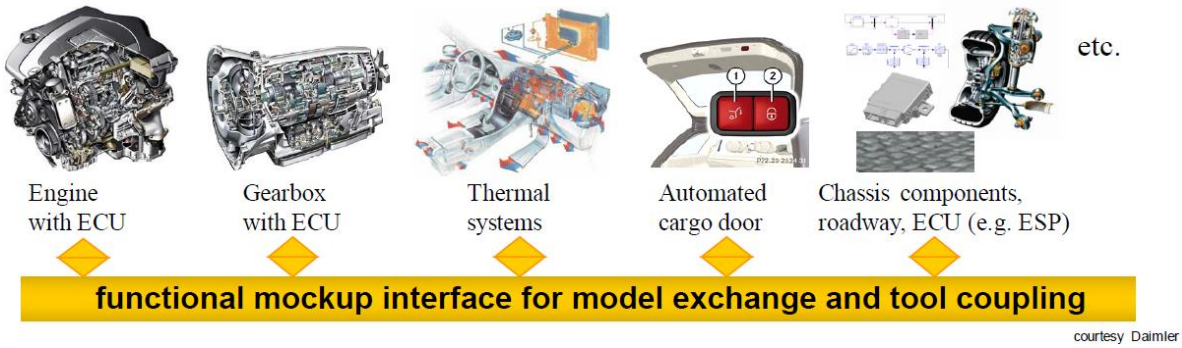



Figure 18. Separate model authoring and execution

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 the authoring modeling or simulation tool generates and wraps models into an 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).

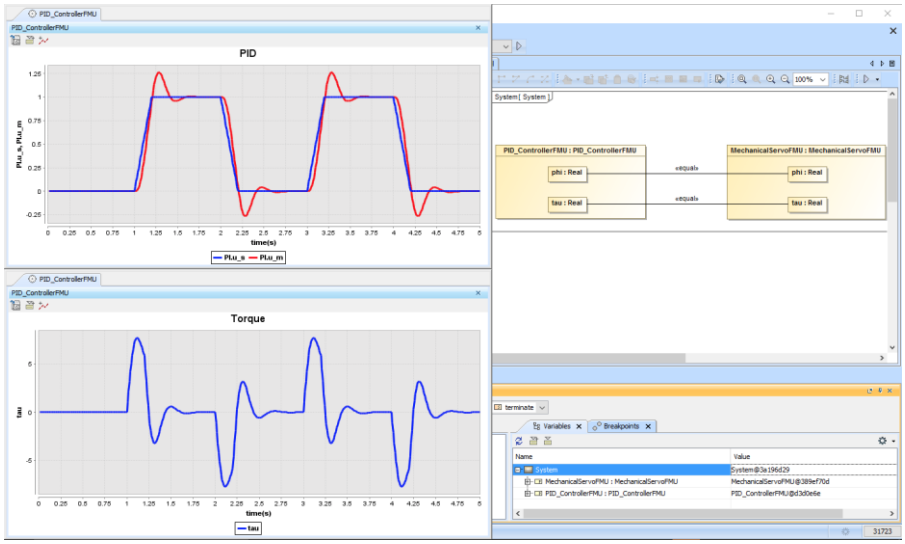



Figure 19. Example co-simulation of system model with domain-specific model

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.

## 5. SUMMARY

Requirement verification is an important kind of analysis commonly performed in the context of MBSE. In order to carry out such an analysis, the requirements, executable behavior, and the models predicting the performance of the specified system have to be integrated. In ESEM, this is done by using standard SysML modeling constructs. The ability to integrate requirements, executable behavior and performance models is a very important value proposition for the ESEM methodology described in this paper: ESEM can be used for automating requirements verification using an executable SysML model and proposes a set of modeling patterns for this. 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 the methodology and supporting tool infrastructure proposed in this paper allows one to integrate engineering models from various domains and relates the currently disconnected document based artifacts formally. This way, requirements can be automatically verified against specified architecture and design, and the tool infrastructure allows to produce consistent model based project documentation. In future work, we plan 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).

## 6. 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.
The TMT Project gratefully acknowledges the support of the TMT collaborating institutions. They are the California Institute of Technology, the University of California, the National Astronomical Observatory of Japan, the National Astronomical Observatories of China and their consortium partners, the Department of Science and Technology of India and their supported institutes, and the National Research Council of Canada. This work was supported as well by the Gordon and Betty Moore Foundation, the Canada Foundation for Innovation, the Ontario Ministry of Research and Innovation, the Natural Sciences and Engineering Research Council of Canada, the British Columbia Knowledge Development Fund, the Association of Canadian Universities for Research in Astronomy (ACURA) , the Association of Universities for Research in Astronomy (AURA), the U.S. National Science Foundation, the National Institutes of Natural Sciences of Japan, and the Department of Atomic Energy of India.

## 7. REFERENCES


[1] Karban, R., Jankevičius, N., Elaasar, M. “ESEM: Automated Systems Analysis using Executable SysML Modeling Patterns”, INCOSE International Symposium (IS), Edinburgh, Scotland, 2016

[2] Karban, R. “Using Executable SysML Models to Generate System Engineering Products”, NoMagic World Symposium, 2016

[3] OMG SysML, [Systems Modeling Language (SysML) Version 1.4], OMG, 2014

[4] OpenMBEE https://github.com/Open-MBEE

[5] TMT, “Thirty Meter Telescope.” http://www.tmt.org

[6] ISO/IEC, ISO/IEC 15288:2008(E), Systems and Software Engineering – System life cycle processes, International Organisation for Standardisation/International Electrotechnical Commission, February 1, 2008

[7] INCOSE “International Council on Systems Engineering.” http://www.incose.org

[8] OMG, “Object Management Group.” http://www.omg.org

[9] Friedenthal S, Moore A., and Steiner R., [A Practical Guide to SysML 3rd Ed.] Morgan Kaufmann OMG Press, 2014

[10] Zwemer, D., “Connecting SysML with PLM/ALM, CAD, Simulation, Requirements, and Project Management Tools”, Intercax LLC, May 2011

[11] Pearce P., Hause M., “ISO-15288, OOSEM and Model-Based Submarine Design”, SETE, APCOSE 2012

[12] NoMagic “MagicDraw” http://ww.magicdraw.com

[13] NoMagic “Cameo Simulation Toolkit” http://www.nomagic.com/products/magicdraw-addons/cameo-simulation-toolkit.html

[14] FMI, https://www.fmi-standard.org/

[15] TMT model https://github.com/Open-MBEE/TMT-SysML-Model

[16] ISO/IEC, ISO/IEC 42010:2011, Systems and software engineering - Architecture description

[17] Troy, M. et. al., “The Alignment and Phasing System for the Thirty Meter Telescope: Risk Mitigation and Status Update”, SPIE, Edinburgh, Scotland, 2016