Conceptual Model for Space Mission Systems Design

Sanda Mandutianu[1], Mehrdad Moshir[1], Kenneth Donahue[2]
Jet Propulsion Laboratory[1]
California Institute of Technology
4800 Oak Grove Dr., Pasadena, CA91107
Massachusetts Institute of Technology[2]
sanda.mandutianu@jpl.nasa.gov

Abstract.

Modelling tools and methodologies without concerns about human understanding may not be suitable for communicating with stakeholders, customers, or management. There are many factors that can increase understandability. Using a generic systems engineering modelling language certainly helps but is not always enough. The understandability of modelling can be improved by using better modelling techniques and strategies, which might be part of a coherent methodology with a deliberate emphasis on communicability. The paper describes the application of Systems Modeling Language (SysML) to the design of a JPL mission using the Object Oriented System Engineering Method (OOSEM). Complementary formal modelling using the Web ontology language (OWL) has also been investigated. Using only traditional means, under increased demands on productivity and decreased resources, system integration may become high risk, and lifecycle support is expensive and may result in rapidly outdated components. Model-based systems engineering (MBSE) is showing promise as a solution to this problem in addition to and enhancing more traditional approaches. The space mission systems models represent sciences and technologies including astrophysics, physics, electronics, mechanisms, hardware and software. Representations of system architecture specification, requirements, evaluation of alternative architectures and trade studies, including static simulation of the performance are grouped into views that represent different stakeholder viewpoints. The human understandability of the resulting models and how the modeling process can increase communication and understandability has also been addressed and observations are reported.

Introduction

When solving problems or designing systems which might be considered as a problem solving situation, humans are forming mental models and then they are transposing their mental models into material models. They model both the problems they are solving and the projected solutions as well. The models are intended to be used for communication between people or as material for logical or mathematical computations. Modelling may or may not be formal. Formal modelling implies using a formal modelling language that has the syntax and the semantics formulated in a

logical or mathematical language, such as for instance OWL [OWL]. The modelling can as well be informal, such as drawings on paper or in an editing computer tool, or semi-formal such as SysML [SysML] diagrams. The models can be translated from one representation to another, usually from a less formal representation to a more formal one. Attempts have been made to follow the reverse route, but some aspects can be lost in the translation [ODM].

While formal models are created specifically to be "understood" either by abstract automata or by computing machines, they still need to produce a human-understandable description of the system. The human-comprehensibility of a formal model can be evaluated by its similarity with the mental models of the designer, while its machine representation is preventing ambiguity and omissions. The issue is how to handle this dual nature of modelling: the models can be in the same time abstract mechanisms and still be understood by their human users.

Systems engineers design systems by abstracting their design and domain knowledge into an ordered set of concepts at different levels of abstraction. A model can be defined as an abstraction of some aspect of a system. The modelling activities and the resulting models support fundamental systems engineering activities such as requirements analysis, architecture, trade studies and analysis.

The models are used to evaluate alternative systems architectures; they are refined and iterated until the desired result is obtained. The model based approach to system engineering (MBSE) brings formal system models as the preferred way to represent systems, systems engineering activities and the resulting artifacts of these activities. Formal systems models offer many advantages, because they introduce additional rigor and flexibility, and because they can be both human and computer understandable. The information communicated by the models can be packaged at different system levels, or levels of abstraction to increase stakeholder understandability at each level.

The generic formalism offered by systems engineering modeling standards such as SysML [SysML] can even further be enhanced by domain semantics such that the models can be easily and unambiguously understood by the domain experts. Using the basic SysML graphical symbolism which has been designed to represent familiar systems engineering concepts is the first step in this direction. SysML extensibility represents a powerful vehicle to further increasing human understandability of the resulting models. The pilot touched on increasing human understandability in several ways including the use of abstractions layers, domain profiling, modularity, defining viewpoints for different stakeholders, performance views for trade studies, Customized visual representations have been used whenever possible to produce visual model representations as close as possible to the familiar graphical representations currently used by the project, such as those used in the computer graphical editors (Visio, PowerPoint, pictures).

Mission Overview

The JPL mission study team has developed engineering models that were used to trade options for the mission architecture. These models are at the heart of early formulation phases and constitute the basis for all further design and development. They embody invaluable engineering talent and expertise. The way they are represented, integrated and used to evaluate mission merit and lifetime performance is critical and ultimately determines the very existence of the project.

These models are of great complexity, and are inherently different by their very nature (astrophysics, physics, mechanics, electronics, software, and telecommunications, to name a few). While some of the models have been adapted from previous similar missions, some of them had to be developed specifically for this mission. The resulting engineering artifacts, besides the actual models, include extensive reports, either generic, or targeting specific domains or problems, such as radiation issues, data return, etc.

The approach of this pilot has been to start with the existing level of model consistency that has been achieved by the JPL mission team study using traditional systems engineering process, and attempt to tighten this consistency in a more timely, complete, and automate manner.

This pilot investigated just a few of these models considered to be a relevant slice through the mission architecture: (1) Key mission and system parameters, (2) Lifetime in a radiation environment, (3) Science merit dependencies on instrument, mission, and system characteristics, and (4) Science data collection and downlink strategies.

Why Models May Become Incomprehensive?

While formal models can be, by their nature evaluated using precise methods, understanding a model is not a formal process. The mental model created by the model user while modelling or interpreting the model is in the user's mind. Understanding implies using the domain information that is communicated through models, the presentation of the models, and may be influenced by model users' personal characteristics. The model presentation may use textual and visual languages, and necessarily includes the media supporting them. Model understanding is also affected by the users' personal characteristics such as their knowledge and skills' level of both domain and modelling techniques.

During the pilot development, and especially when the models had to be communicated to the stakeholders, it became apparent that some of the models were easier to communicate while others were not. Moreover, even the modelers seemed to have trouble understanding some of their own models. Given the short period of time at our disposal the problem has only started to be recognized rather than addressed. It seems that there are not too many reported attempts to address the understandability of the models, and what makes a good model for humans, and even less in a rigorous way.

To constitute effective design means, models must not only satisfy formally defined criteria such as correctness or completeness, they also need to be understood by humans, and even appeal to them. If the models are not well understood, the formal aspects may be impacted as well. The

models may become incomprehensible for humans for various reasons. Some empirical research [Mendling 2007] reported that larger models tend to have more formal flaws, hypothetically due to the limited capabilities of humans to keep track of a large number of interrelations. It seems that the model size is of dominant importance on model understandability.

There are many other different factors that can affect understandability, such as how the abstractions are chosen and used, the clarity and suggestiveness of presenting the models, selecting the information to be presented to various audiences, choosing consistent modelling conventions such as naming conventions and standards, and probably many others. In some cases, there might be too many arrows in the same diagram; in other cases the direction of the arrows might be wrong. It also seems that naming conventions play an important role in conveying the meaning of the model elements.

A distinction can be made between the model understandability and the modelling language understandability. While factors such as level of abstraction, number of embedded structures, the degree of modularization, simplicity, or model presentation are related to the act of modelling, the primary building blocks capacity of conveying meaningful information is mostly related to the modelling language understandability.

The personal related factors are also very important in understanding models and languages. Modelling experience and familiarity with the modelling techniques are helping factors. In the case of visual models visual perceptiveness has a positive influence. The personal domain knowledge is also relevant for the understanding of the system models.

Model-related factors include: use of abstractions, un-ambiguity, simplicity, good structure, modularity, and model presentation. Besides visual renditions, textual support or self-documenting using annotations and descriptions can add to the understandability, if applied consistently. The way the abstractions are used is one of the most important factors in dealing with system complexity. This can be achieved by defining several levels of abstractions, using decomposition, viewpoints or other specific means offered by the modelling languages and by the modelling methodologies. Modelling conventions and standards [Friedenthal 2008] can also significantly increase the understandability of the models.

A Model Integration Environment

To understand how the models are used, the context in which they are used needs to be outlined. Applying MBSE in practice requires a development environment or a framework that enables tool and model interoperability. An MBSE environment may include a wide variety of tools to support different models or other associated modeling artifacts. The tools and the models have to be integrated, in other words they have to interoperate. Information is propagated among models and tools. To increase understandability the opportunities for different interpretations of the exchanged information needs to be minimized.

Traditionally the model integration has been achieved by point-to-point interoperability solutions that are generally unique and *ad hoc*. An example of an ad-hoc but still often used interoperability solution is manually transferring data (cut and paste, re-keying) between tools.

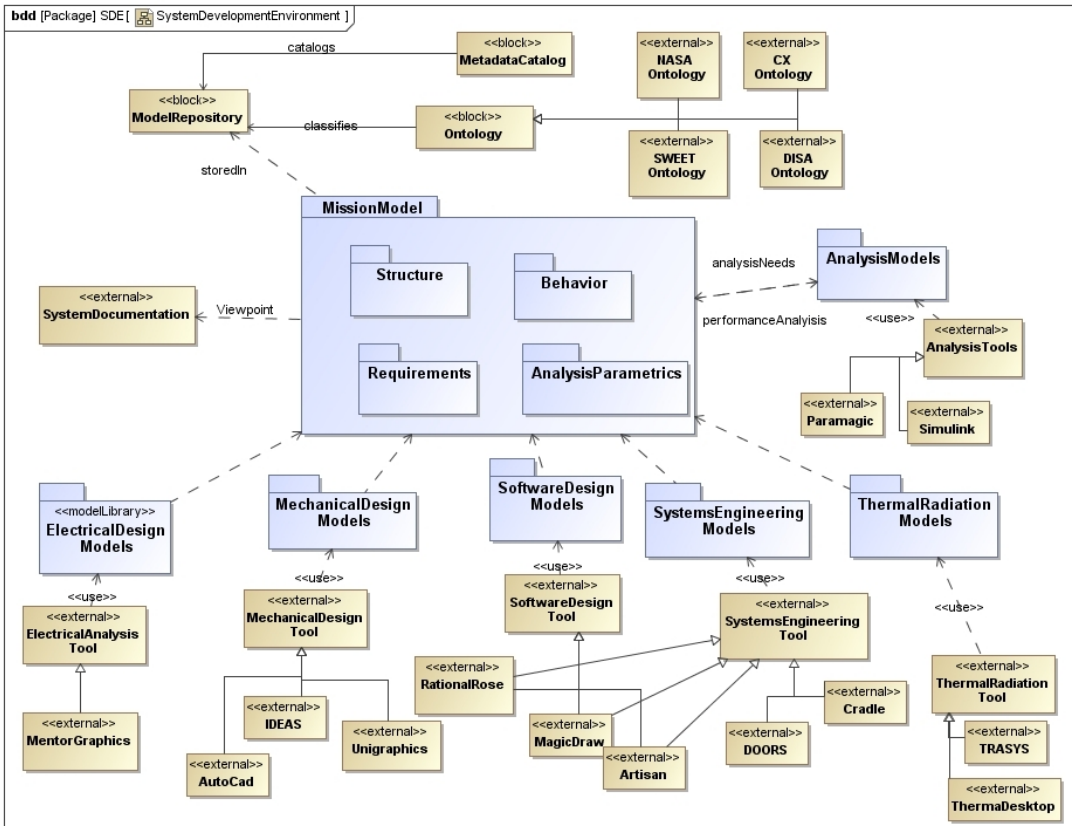

Figure 1 Model Integration Environment

The integration of different models within the MBSE development environment can be achieved by different interoperability techniques, all of them based on the compatibility between data models.

The simplest way to achieve interoperability is to enforce a new, unique data model for all tools; this is a static solution in the sense that always needs human intervention to change data model elements. Another approach is to use a mediator which translates data between two systems with different data models; this approach has the advantage that while still being static, reduces the number of peer-to-peer-connections. A different approach is to define a central information model that represents a neutral and unifying view of a group of data models and each of the resources can be mapped to the central model.

The spectrum of the data incompatibility problems may vary from using different data formats, to those that arise from semantic differences in the structure or schemas of data. A semantic approach may save time by ideally capturing the meaning of data once. While a common data format may never be achieved, this approach has the goal to establish a common understanding. An ontological information model is potentially richer than a traditional data model, including more

levels of generalization, business rules, etc. Its generality allows the information model to serve as an authoritative reference for multiple data sources, regardless of format or technology.

The semantic integration of different models requires that each model element from a particular model be mapped to the appropriate concepts in the reference model. That reference source contains terms in use by the SE community, and it is intended to facilitate interoperability, data sharing, and semantic integration between models representing different systems. Models are annotated with terms from the reference source – these are seen as metadata describing the models. The annotated model can be stored in a catalog (metadata catalog or metadata registry). Semantic integration mechanisms ensure the proper mapping of the terms to unique concepts.

The integration framework can be considered as a project in itself and its architecture can be represented using MBSE techniques (see Figure 1).

Systems Engineering Conceptual Modelling

The purpose of creating conceptual models for a particular domain is to share and reconcile different representations of the same reality, in our case different representations or models of the same system. One of the important aspects of sharing and reconciling is that the concepts are unambiguously defined, understood and agreed upon. The concepts can then be used as controlled language elements and rules of composition can be defined that prescribe how primary conceptual elements can be used to derive more complex structures.

Systems engineering as an interdisciplinary domain covers technical as well as management aspects of engineering complex systems. A conceptual model must ensure the complete coverage and integration of domain engineering disciplines including electrical, mechanical, software, communication, control, and operations. Additionally, the conceptual model has to integrate the planning and managing of the overall technical effort, the systems engineering processes in all phases, and design and verification including analysis and trade-offs.

An important factor in increasing understandability is using a standard way of categorizing the systems engineering concepts. Such a standard is AP233 [AP233] which served as a reference for the pilot conceptual modelling. According to AP233 the conceptual coverage of the systems engineering domain can be roughly grouped into two categories: technical models and project management models. The project management models recommended by AP2333 have not been addressed by this pilot.

The technical models include requirements models, structure, behaviour, allocation, risk analysis, validation and verification. Any conceptual system model has at its core a structure or a way to represent the topology of the system, or in other words the static relationships among systems, subsystems, and components. The structure can be represented using levels of abstractions, composition, and other relationships. The behaviour models describe how the system acts or performs over time. The modelling approach can be function-based or state-based. Other important models are requirements models. A fundamental aspect of requirements modelling is

transposing textually represented requirements into more formal, eventually computer represented format, that allow for their derivation and the representation of their impact on the system design. The risk analysis models identify risks and their status, likelihood, mitigation, etc., and their relationship with other models.

The conceptual models can be further enhanced by the use of controlled vocabularies, or ontologies. By analogy, the ontology acts as a formal language or a grammar that the modellers use to express their domain models. The vocabulary of such a language offers the primary elements such as: Requirement, System, or Function. The syntax of this language gives the rules of composing the elements of the vocabulary: a System is composed of Subsystems, a System has a Function; a System satisfies a Requirement, etc.

The conceptual modelling described succinctly above has been used to represent a JPL mission in its formulation phase at the time of the pilot development. In early stages of design, the models are low fidelity but cover a deep while quite narrow slice through the whole set of systems engineering activities. The models have been used in support of trade studies and high-level decision making processes. Using standard conceptual models has an essential role not only in ensuring computer models interoperability but also in increasing human comprehensibility. Models can be better understood if the modellers use the same vocabulary, rules and the same meaning for the modelling concepts.

Methods and Tools

The method can be seen as the set of related activities, techniques, and conventions used to produce a system model. See [Estefan 2008] for a detailed study of SE methodologies. The modeling languages under consideration have been SysML and OWL [OWL]. Some of the criteria used for selection are shown in Table 1.

SysML extends UML 2 and combines visual diagrams for human communication with metamodels that can be exchanged in digital form across modeling tools. UML/SysML provides standardized behavior models and abilities to describe hierarchical system structure and interconnection between system elements. SysML is intended to be compliant with the ISO AP233 [AP233] standard. The intent of AP233 is to support the whole system development life cycle ranging from requirements definition to system verification and validation. OWL is a logic based language that is neutral to respect of ontology. The ontology is a system of concepts used to describe a domain of interest. The main difference between the two languages is that SysML lacks formal semantics, which has impact on the ability to integrate information from multiple languages and tools. OWL provides the logical mechanisms for semantic integration such that the meaning of the concepts is captured independently of the domain of interest, and supports automatic reasoning.

From modeling perspective, SysML and OWL are overlapping and can be considered as complementary. The use of OWL is not intended to replace systems engineering languages such as SysML, or other specific modeling languages (ex: Modelica [Modelica]) but to enhance these languages and serve as an integration vehicle in an integrated MBSE framework. On the other hand, a SysML model is comparable to a schema, or ontology. A desirable solution is to combine

the two languages in a coherent and meaningful way within the conceptual systems engineering framework offered by standards such as AP233.

Table 1

Language Selection Criteria	SysML	OWL
General purpose SE language	yes	no
Standard information modeling concepts: classes, relations, individuals	yes	yes
Expressing behavior	yes	no
Meaning for subject matter concepts, rather than generic information technology	yes	no
Automated reasoning	no	yes
Standard domain ontology	no	yes
Support conceptual phase of the lifecycle	yes	yes
Support of a SE modeling tool	yes	no
Support for engineering analysis	yes	no
Commitment to basic concepts of SE	yes	no
Capture meaning of concepts independently of interpretation of subject matter experts	no	yes
Web-based representation	no	yes
Formal semantics	no	yes

The pilot has used SysML as the primary model capture language considered as best fitted for the SE domain, and has explored populating an OWL model for automated reasoning. This may involve bidirectional mappings between the languages and extensions of each language (e.g. profiles for both languages, SysML stereotypes). Connecting the two languages at the metamodel level is still under investigation by the OMG [ODM 2005].

The modeling tool selection criteria for the pilot included: SysML compliance, model library support, interoperability with other modeling tools, XMI[1] serialization of models, simulation capabilities, and static simulation plug-in for trade studies modeling (Paramagic, Mathematica).

The system model has been initially represented in SysML using MagicDraw™ [MagicDraw]. The model generated via MagicDraw has been successfully exported into an OWL modeling tool, TopBraid™.

SysML and OWL are neither a methodology nor do dictate any particular development process. They provide modeling means to enhance communication of the design intent, but this doesn't prescribe how these can be best applied within a design flow. That is the role of a methodology. A sound methodology which suits the peculiarities of the JPL MBSE is necessary to complement the

[1] XMI serialization: XML Metadata Interchange (XMI) is an XML-based encoding standard for UML models. Object Constraint Language (OCL [OCL]) is a formal language used to specify well-formedness rules for models. OCL can be compared in style with XML DTDs. DTDs constrain the number of possible valid instances of XML documents, whereas OCL constrains the number of possible valid instances of UML models

use of SysML for modeling. We also argue that the methodology, by prescribing guidelines for the act of modeling can considerably increase the degree of understandability of the models.

Methodology selection criteria included: support for the transformation and transition of the traditional SE processes to MBSE, support for the conceptual (formulation and pre-formulation) mission development phase, modeling traceability, and modeling verification

Object Oriented Systems Engineering Methodology [OOSEM], the selected methodology, enhances object-oriented methodologies that are directly applicable to SysML to support the analysis, specification, design, and verification of systems. OOSEM includes system engineering activities such as needs analysis, requirements analysis, architecture, trade studies and analysis, and verification. Across the product lifecycle, one of the areas that usually lack efficient support is the conceptual stage, during which the functional architecture is decided upon. The conceptual stage follows the transformation of the customer needs into product functions and use cases, and precedes the design of these functions across engineering disciplines such as mechanical, or software. OOSEM recognizes these problems and offers solutions to it.

Models

In the following only the modeling activities that were actually achieved in the pilot are presented.

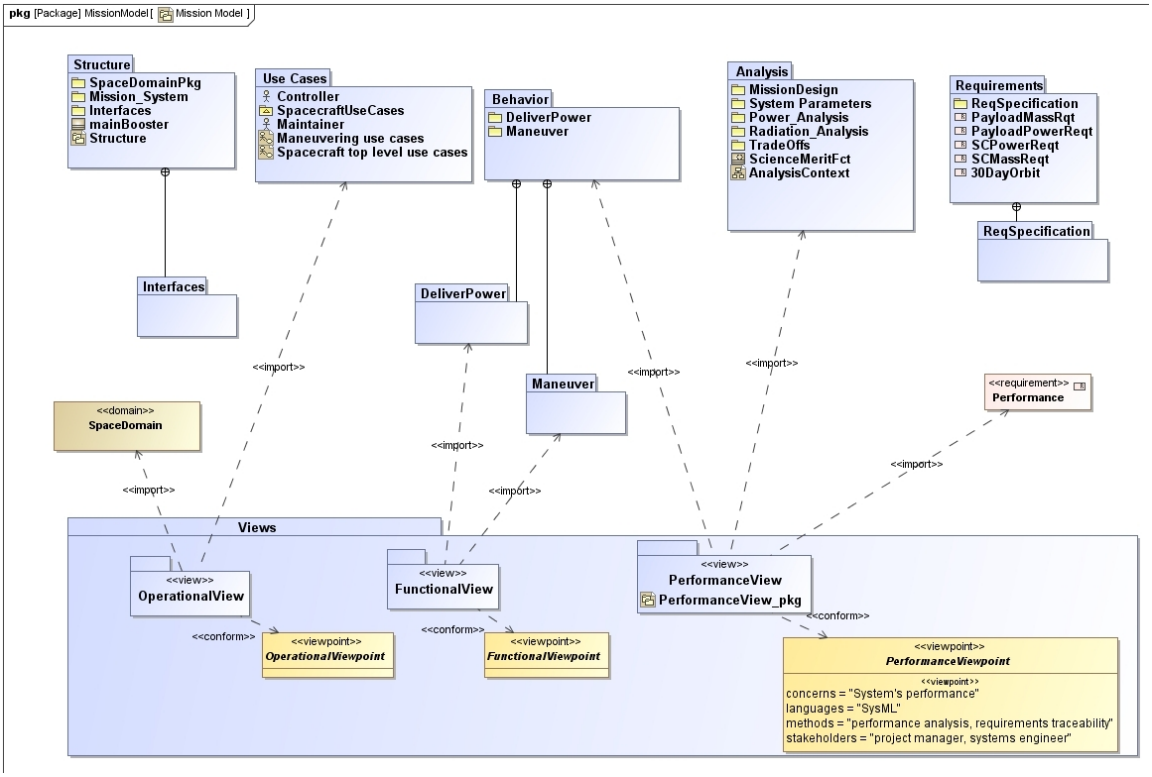

Figure 2 Model Organization

The model has been structured as a recursive package structure. The top model SpaceMission contains the MissionModel and the ModelingDomain. The MissionModel contains packages: Analysis, Behavior, Requirements, Structure, UseCases, and Views. The package structure is partially represented in the package diagram in Figure 2.

Use Case Model

The use case modeling provides a means to describe the functionality of a system. The use case model is developed during the early phases of requirements analysis. Use cases communicate and help the understanding of the functional requirements of the system, without any requirements implementation considerations. A use case model consists of use cases, actors, and their relationships. Actors are external entities interacting with the system, such as users or other systems. A use case describes some functionality of the system and represents the set of actions initiated by an actor.

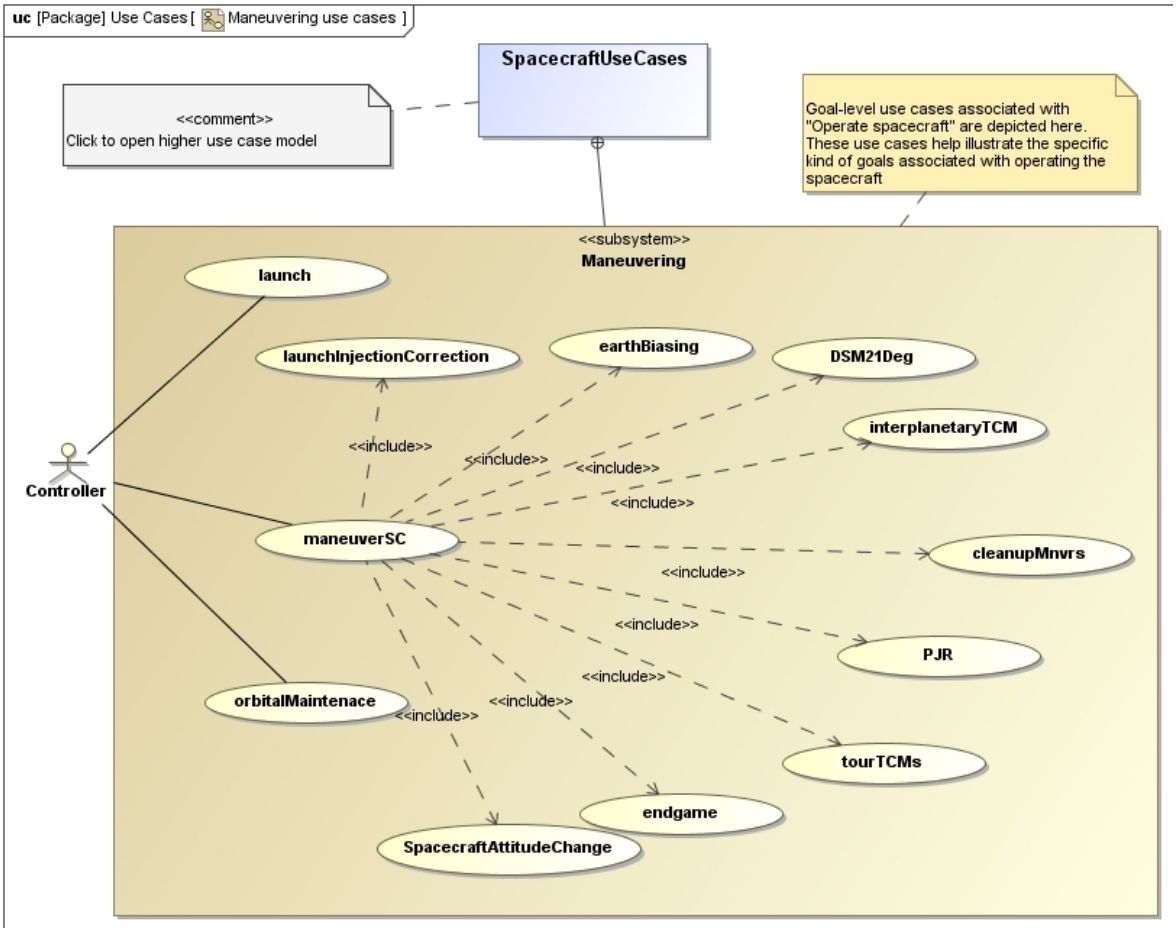

Figure 3 Maneuvering Use Cases

To increase understandability, a main use case model has been first developed, followed by more specific use case models. An example of a specific use case model is represented in Figure 3. It represents a spacecraft maneuvering use case model. The main intent of grouping the functional requirements for maneuvering the spacecraft on a single diagram is to better communicate with specific user categories (operators or controllers) without interfering with other functionality that might not be relevant to them.

Logical Architecture Model

The activity of defining a logical architecture includes decomposing the system into logical components that satisfy the system requirements. The logical components are implementation independent abstractions of the system components, which perform the system functionality. For each function or operation that the system is required to perform there should be a scenario that describes the interaction among the implied logical components.

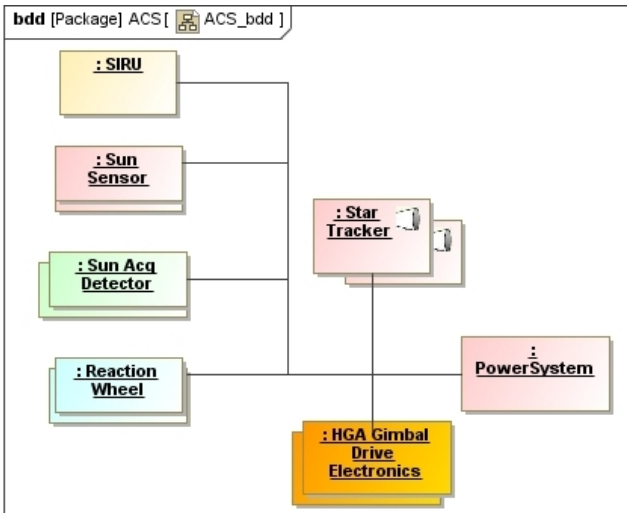

Figure 4 Logical Architecture of a Spacecraft Subsystem

Figure 4 shows the logical decomposition of the attitude control subsystem into logical components. The whole diagram represents the system. The boxes represent different logical components of the system. The arrows represent relationships between components such as aggregation or composition. As a note, a substantial number of details describing each component (such as attributes describing components in detail) have been omitted from the graphical rendition so that the diagram can be more easily communicated to stakeholders not interested in such details.

The logical architecture represents an intermediate level of abstraction between the requirements and the implementation models. Separating these two levels is a first step in increasing understandability. Next, the logical decomposition can be iteratively refined starting from some initial generic components and adding more related components with increased details. This separation of concerns has the potential to considerably maximize models' understandability.

Optimize and Evaluate Alternatives

Designing involves performing various engineering analyses to support trade studies for alternatives optimization. This modeling activity is meant to support engineering analysis and trade studies. It can be achieved in conjunction with any other modeling activities and includes: identifying what types of analyses are needed, defining the analysis context, and performing the engineering analysis
The main objective of an engineering analysis is to characterize some aspect of the system such as its performance, physical properties or cost. The optimal or preferred design solutions can be evaluated, verified, and selected. Many types of analyses can be identified throughout the design process.

Although this activity can be defined for any design in general, the following sub-activities have been designed with a particular modeling language in mind, i.e. SysML, hence the need to explain some of its modeling specifics.

The analysis modeling is supported in SysML by the parametric models. SysML parametric models capture the constraints on the properties of the system, leaving their evaluation for the appropriate analysis tool that may interface with the model for this purpose. A constraint is represented as an equation statement (formula) that can be eventually evaluated by an analysis tool. The variables of the equations are represented as external bindings or in other words ends of a yet to be defined relation (similarly to the valences of a chemical formula) to properties of the system. This way the equation specification is decoupled from its actual usage. To actually evaluate the constraint (i.e. to solve the equation) the constraint usage needs to be defined in the model. For instance one can define a generic equation such as F=m*a, and bind (i.e. define a relation) ‘F’, ‘M’, and ‘a’ to parameters belonging to different system components.

In SysML a constraint can be a part of an element definition. This is an adequate representation if the constraints are always related in this way in all contexts of usage. A more flexible approach is to decouple the constraint definition from the model element that is constrained by it.

Although not very intuitive at first sight, this decoupling of the definitions of the equations from their usage, follows one of the most important principles of increasing understandability: using levels of abstraction.

Analysis Models

For the purposes of our pilot we selected just a few of the analyses that have been identified by the project such as mass analysis, power analysis, radiation analysis, and science merit analysis. These analyses are basically executable mathematical models used by the systems engineers to predict system parameters values. These values are traded against each other and the preferred combination is selected. Selected combinations of systems parameters determine alternative

system architecture solutions. The selection criteria are defined based on previous missions and personal experience.

Analysis Context

Defining analysis context is a modeling technique that allows the association of all the constraints that are used for a particular analysis including the model element for what that analysis is performed. This technique is in fact a refinement of the basic decoupling of the definition of parametric models from their usage.

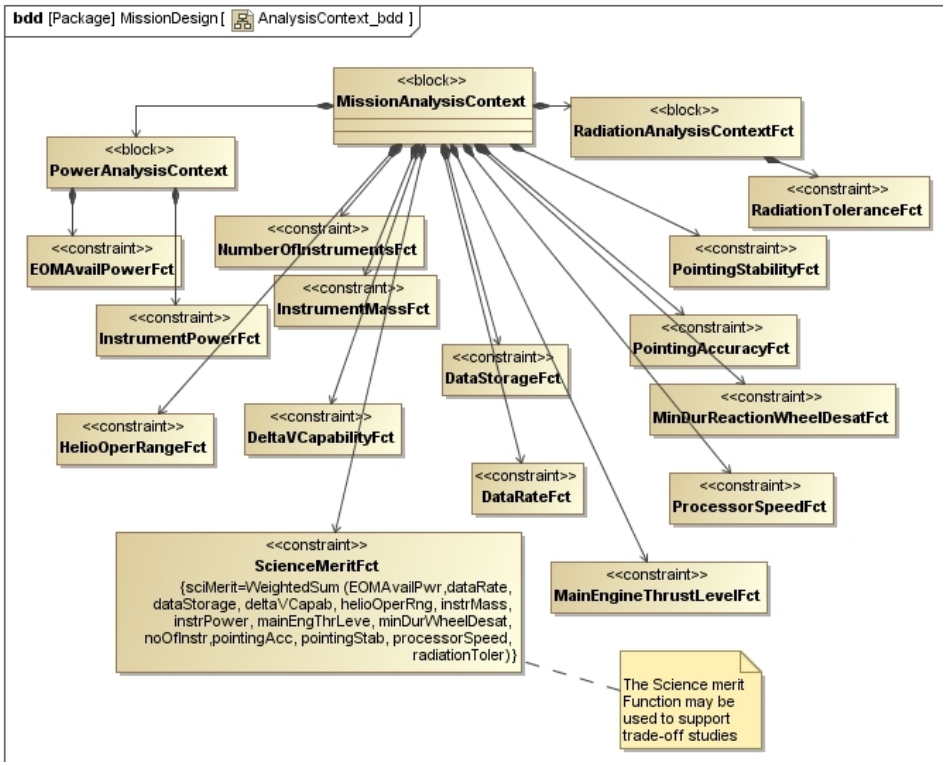

Figure 5. Mission Analysis Context

Defining the analysis context is a modeling technique by which the constraints used in the analysis process are represented in the same context with the model elements that are constrained by them. Libraries of constraints may already exist (in SysML those would be groupings of constraint blocks).

Trade studies can be represented as analysis contexts. In fact a trade study is a type of analysis context, which contains the model elements that represent different alternatives, and the constraint used to determine the evaluation criteria. The constraint is evaluated using a criterion, usually represented as a mathematical function, sometimes called objective function. The objective function specifies a way to relate system cost effectiveness to the measures of effectiveness for the key performance parameters of the system.

The constraints that have been used to define the analysis at the mission system level for the pilot are represented in the mission analysis context shown in Figure 5. The analysis context has been further modularized to represent different aspects of the analysis such as Power Analysis, Radiation Analysis, etc.

Constraints models

The specification of the constraints enables the integration between the design and analysis models. Associating these models can be done in various ways, depending on the representation formalism. SysML has the parametric diagrams to achieve this. A parametric diagram allows the binding of the parameters of the analysis equations that are defined in the analysis context to the properties of the system being analyzed.

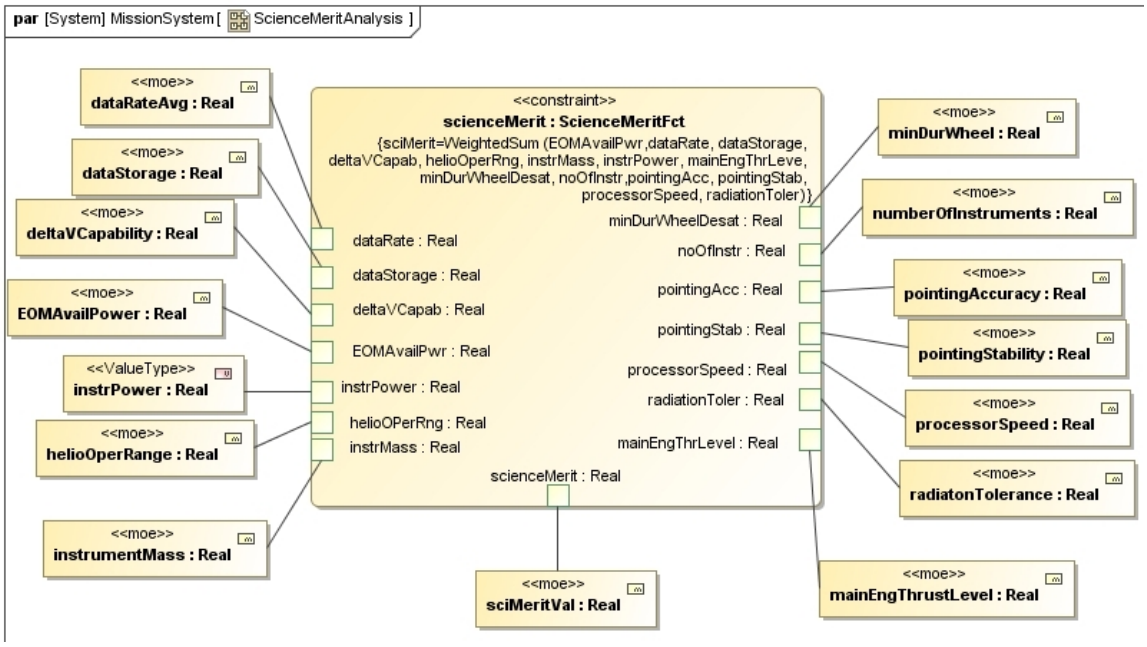

Figure 6 Science Merit Function captured in a parametric diagram

The parametric diagram given as example in Figure 6 uses the equations defined in the Mission Analysis Context and shows the bindings of the parameters of the function that calculates the science merit to the measures of effectiveness of the mission system.

Engineering Analyses Models

The equations are represented as declarative knowledge. There is no executable aspect to the information models. To solve the equations an executable capability is required. This is usually an engineering tool (Simulink, SOAP, Excel, and Paramagic).

No matter how the analysis results are obtained, the results represent the specific values or the range of values of the system properties that satisfy the constraints. The results can be included in the information model. In the example of the science merit, the result shows the extent to which the system satisfies its science merit requirements.

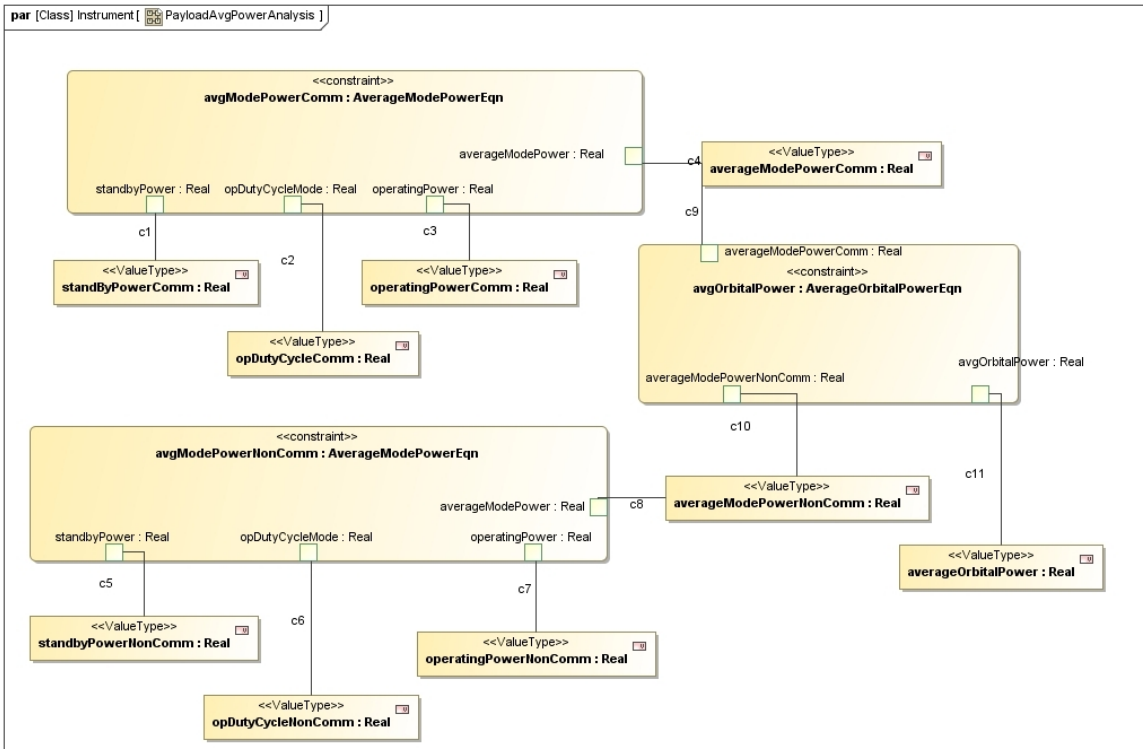

Figure 7. Payload Power Analysis Model

Before performing the actual engineering analysis, the trade studies can be modeled and these models passed onto the analysis engines.

The engineering analysis or a trade study is used to compare some alternative solutions and compare them with a given requirement. The solutions are characterized by a set of “measures of effectiveness” (moes) that may have a calculated value. The moes are calculated using a function

(utility, or cost, or merit function) and then compared so that the best (optimal) solution can be selected (or used in support of the decisional process).

Parametric models capture the constraints on the properties of a system- these can be evaluated by an appropriate analysis tool.

Executing Models

The model represents a declarative way of specifying the system. As being captured in a repository, that information can be queried, analyzed for consistency or passed to be used by different engineering tools. For instance, one can describe behaviors in the model, but not "execute" them. One can describe what needs to be done in each step, what are the input and output, but one cannot really do the steps. One can describe what the equations are, and the provenance of the parameters, but one cannot solve the equations and get actual results.

These aspects are outside the scope of the modeling itself, but the modeling can capture what is necessary to trigger execution of the models, including solving equations.

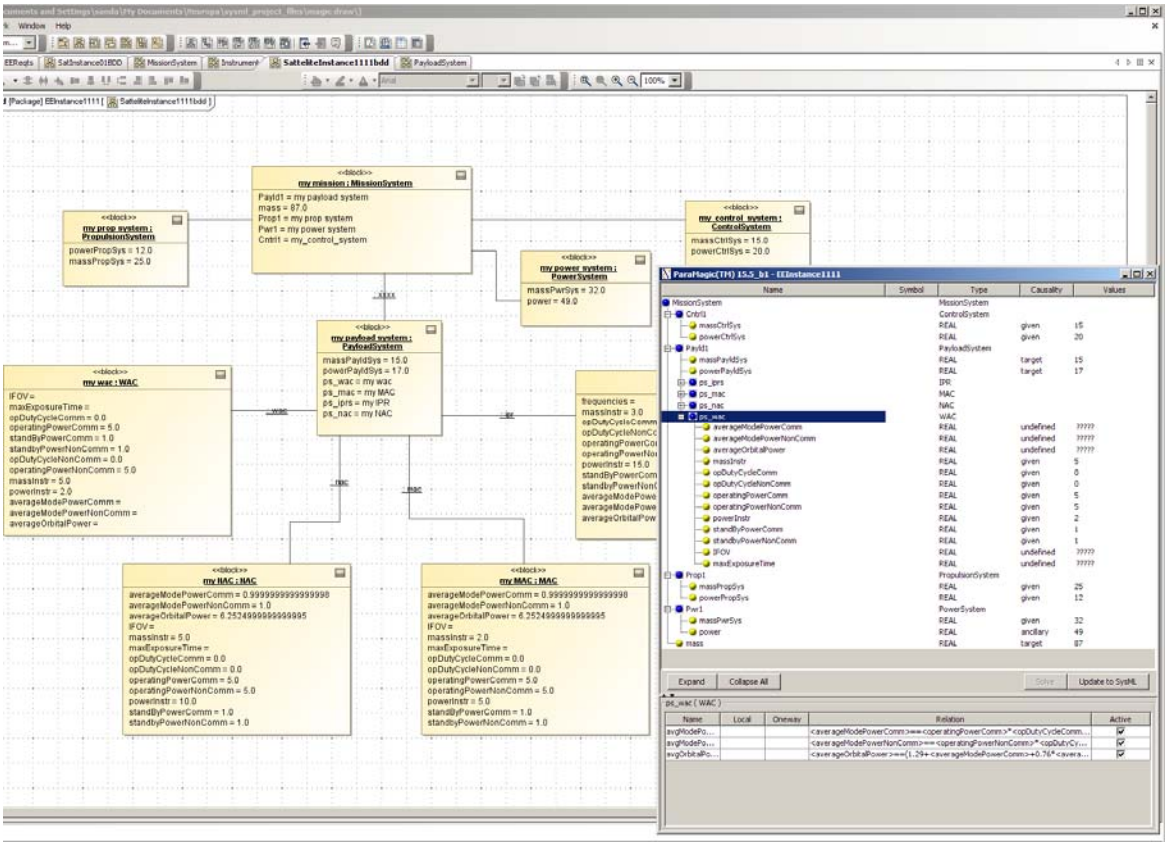

Figure 8. Static Constraint Solving

The model may contain descriptions of equations that can be understood by computer systems that are specifically designed to solve equations. The model may contain behavior descriptions that can

be understood by computer systems that simulate behavior, sometimes including solving equations. The executive computer system needs to "understand" the syntax and semantics of the declarative model, generate code, and execute it.

The system can be described by various types of modeling artifacts, including behavior models and structure models. By combining these models one can produce a system model that can be used by the adequate executive software. The "executed" model has the potential to provide better understanding about how the system "works" prior to be actually built. By analyzing the results of the execution, the dynamics of the system can be analyzed and if necessary, adjusted.

Executable system models [Friedenthal 2008] may be grouped into categories such as dynamic system model, performance simulation model, and analytical models.

A dynamic system model is represented as a set of static "declarations' of behaviors as sequences of "actions", input and output messages or signals, events, states and state transitions, etc. This could be a graphical rendition (as in SysML activity diagrams, sequence diagrams, or state machine diagrams) or a description in some other information description language (rules, frames, etc).

A performance simulation model is a continuous simulation model that extends the dynamic simulation model by capturing the underlying mathematical relationships necessary to analyze the system performance. The execution environment must also be able to simulate the underlying mathematics (solve) such as numerical solutions to differential equations.

An analytical model addresses a specific type of analysis such as mass properties, radiation lifetime, or thermal characteristics. The executable analytic model can be used either by a dynamic simulator or a static constraint solver that solves a set of simultaneous equations.

Static Constraint Solving

A set of mathematical relationships has been modeled to describe the application domain that has been the target of this pilot:

- power, mass, data return parametrics
- radiation
- science merit return

The static constraint solver [Paramagic] simulated and thus solved the set of equations that has been defined (Figure 10). The results have been imported into the model for further analysis and as a support for decisions involving tradeoffs.

Semantic Level - Using formal modeling to validate models

The SysML models described in the previous section can be considered as semi-formal in the sense that the semantics of the language in not formally defined. If more rigorous model

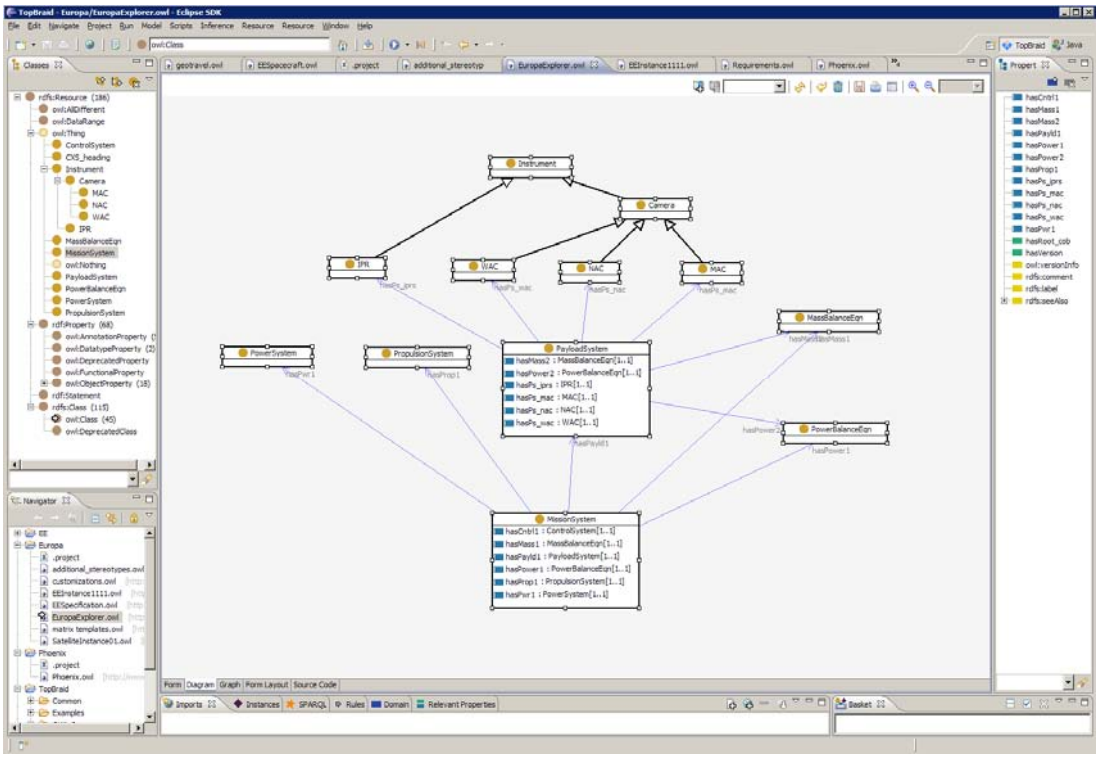

Figure 9. System Model Represented as OWL Ontology

consistency and validity is desired, then it is necessary to translate the SysML models into formal models. This means that a translation is needed between modeling languages. The formal language that has been considered for this pilot is OWL [OWL].

In systems engineering it is a common practice to build a simulation model to numerically study the system. The system characteristics are simulated using a simulation model. We did that by statically solving some of the performance models reported in the previous section. In this section we show how we used the OWL language as a modeling tool with the goal of analyzing the model properties in more rigorous ways. This approach is complementary, rather than alternative to simulation. It does allow for a deeper analysis of certain properties of the system, and most importantly, gives certain, i.e. formally provable results.

The semantics of the information models can be further refined by using external classification schemes, such as controlled vocabularies, ontologies, or other techniques (for instance an external class library, if the implementation approach is object oriented). Models can be annotated with concepts and definitions from the classification source- these are seen as the metadata describing the models. The annotated model can be stored in a catalog (metadata catalog, metadata registry).

Stereotyping in SysML is an example of an annotation technique used to semantically enhance the language core vocabulary. For instance, a “Block” can be stereotyped (annotated) as a “System”. Libraries of stereotypes (known as “Profiles” in SysML) are sometimes provided by the tool vendors, to enhance the user experience of different communities of users- in this case the SE community.

The semantic integration of different models requires that each model element be mapped to the appropriate concepts in the controlled terminology. That “golden source” contains terms in use by the SE community, and it is intended to facilitate interoperability, data sharing, and semantic integration between models representing different systems. The semantic integration is provided by mapping the terms to unique concepts.

If the controlled terminology is represented using a language that has formally defined semantics, the mapped models can be automatically validated by computer programs known as reasoners. The reasoners can also identify conflicting or inconsistent concepts. See [Jenkins] for an illustration of this technique.

Conclusion and Future Work

We have presented a model-based system engineering pilot for a JPL mission in its early stages of formulation and design. We have argued that this approach is feasible and advisable starting from such an early stage. We also argued that the modeling challenges, including human comprehensibility can be alleviated by using a conceptual model. We have described how the modeling act can be improved by using conceptual models and how a good methodology can increase the quality of the modeling results. We illustrated that by using OOSEM.

We started to address the understandability by focusing on the resulting models, and what a good process of coming up with good models could be, rather than on the understandability of languages that we used, SysML or OWL.

In particular we have discussed the central role of a conceptual model and how ontology can increase the quality of the modeling. We argued that enduring numerical validation by simulating models is complementary to ensuring logical consistency offered by formal languages such as OWL and reported related experiments. This should provide a good starting point for more detailed, mission-specific exploration of modeling techniques.

The main potential benefits of the conceptual modeling as illustrated by the results of this pilot include: (1) improved communications among model designers and stakeholders, (2) consistent and complete representation of system models across missions and phases of missions, (3) reduced errors and ambiguity, (4) reduced design and maintenance cost, (5) increased overall management of system complexity, (6) saving time and resources. These benefits need to be validated by more comprehensive implementations and metrics needs to be developed to assess their value.

One of the longer term objectives is to develop a more comprehensive mission conceptual model that can be used across missions and mission phases with minimal changes. This should start with a comprehensive domain analysis of the formulation phase and beyond. The real benefits of the model-based approach to system engineering need to be still determined after a critical mass of applications will be available for analysis.

References

[AP233] Application Protocol 233, http://www.ap233.org/

[Estefan 2008] J. Estefan, Survey of Model-Based Systems Engineering (MBSE) Methodologies, INCOSE MBSE Focus Group, 2008.

[Friedenthal, 2008] S. Friedenthal, A. Moore, R. Steiner, A Practical Guide to SysML, Elsevier press, 2008.

[Jenkins 2008] Model Development for efficient Project Formulation, JPL http://wikidev.jpl.nasa.gov/index.php/ModelDevelopmentforEfficientProjectFormulation

[MagicDraw] http://www.magicdraw.com/

[Modelica] The Modelica Association. The Modelica Language Specification Version 2.2.

[Mendling 2007] Mendling, J., Reijers, H. A., Cardoso, J., What Makes Process Models Understandable?, in Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Volume 4714, 2007.

[OCL] OMG, UML 2.0 OCL Specification, 2003.

[ODM 2005] OMG, Ontology Definition Metamodel, www.omg.org/docs/ad/05-08-01.pdf

[OOSEM] OOSEM – Object Oriented Systems Engineering Methodology Tutorial, https://bravo-lib.jpl.nasa.gov/docushare/dsweb/View/Collection-83454

[OWL] OMG, Web Ontology Language (OWL) http://www.w3.org/2004/OWL/

[Paramagic] InterCAX, Paramagic - SysML Parametrics Solver http://www.intercax.com/sysml

[Protégé] National Library of Medicine, Standford Center for Biomedical Informatics Research, Protégé, http://protege.stanford.edu

[SysML] OMG, OMG Systems Modeling Language (OMG SysML™), OMG Available Specification, September 2007.

[TopBraid] TopBraid Composer http://www.topquadrant.com/topbraid/composer/index.html

ACKNOWLEDGEMENTS

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

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

The pilot team would like to thank Bob Rasmussen, JPL Division 31 Chief Engineer, the initiator of this pilot for his vision, directions, insights, and valuable advice and contributions. This pilot wouldn't have been possible without all the hard work of the Division 31 in advancing the state of the practice in model based systems engineering and documenting strategy and implementation plans. We particularly thank Jeff Estefan, the Division Chief Technologist and lead of the Model

Based Steering Group for giving directions, advice and valuable contributions. Many thanks go to the mission team, to the study lead Karla Clark for her support and very straightforward comments and suggestions, and to Rob Lock, Grace Tan-Wang, and Tracy Van Houten who allowed us to constantly bug them and still provide us with many valuable and critical information and feedback. We thank our sponsors the JPL System Modeling and Analysis Office led by Steve Prusha and valiantly represented for this effort by Steve Wall who closely coordinated with us. Thanks are also extended to Chi Lin, Manager of the Division 31 Engineering and Development Office for her understanding, encouragement and support, and to Steve Jenkins for his expert advice on systems engineering modeling and support. It has been a true pleasure and inspiration to work with such talented individuals.

BIOGRAPHY

Sanda Mandutianu has been the task lead and modeller on the Europa Explorer Information Modelling pilot. She has joined Jet Propulsion Laboratory 15 years ago, as the Technical Group Lead on a distributed science data project, part of NASA's Earth Observing System Data and Information System (EOSDIS), one of the first attempts at achieving interoperability among geographically dispersed extraordinary large amount of data. Since then she has been working and hold positions as task manager and PI on technology research and development, and on systems and software engineering. Her interests and carrier span a wide area of technology and applications including systems and software architectures, autonomy and control, information architecture, modelling and simulation, artificial intelligence, agent-based technologies and semantic technologies and ontologies. Sanda initiated and pioneered autonomy and agent-based technologies for distributed spacecraft missions at JPL such as Space Technology 3 (ST3) and Terrestrial Pathfinder (TPF), and worked on NASA wide projects such as Constellation, the return to the Moon NASA project. She holds a MS in Physics from the University of Bucharest. Sanda is interested by the innovative and conceptual aspects of her work and how they are applied in practice. She published over 30 peer reviewed papers, book chapters and conference proceedings in the area of software and systems engineering, agent-based technologies and semantic technologies and natural language understanding. Her current work addresses the conceptual and practical aspects of model based systems engineering and information architecture.

Dr. Mehrdad Moshir is a Project Element Manager at the Jet Propulsion Laboratory focusing on the Project Modeling of Space Interferometry Mission (SIM-Lite), Soil Moisture Active Passive (SMAP) Mission and the Jupiter Europa Orbiter (JEO) mission concept. He holds an A.B. in physics from UC Berkeley as well as an M.A. and Ph.D. in physics from Princeton University. As a graduate student he used to build particle detectors and electronics for high energy physics experiments at Brookhaven National Laboratory and gradually migrated towards computational particle physics. His career spans university professorship; lead and project management roles in ground processing of IRAS, IRTS, MSX and Spitzer (SIRTF) missions; project systems engineering on Kepler, SIM and SMAP missions with emphasis on bringing innovations into the modelling processes for critical models within those projects. He has published over 40 peer reviewed articles, books and book chapters as well as conference proceedings covering a wide range of subjects such as Systems Design, Data Systems Architecture, Mission Products, Pointing Systems, Statistics, Physics and Astrophysics.

Kenneth Donahue is a Senior at the Massachusetts Institute of Technology studying Electrical

Engineering and Computer Science. He has a wide range of interests including control, artificial intelligence, robotic vision, software architecture, and system modeling. He has been accepted into the Masters program at MIT and is currently doing research on programmable matter and asynchronous logic automata. He was glad to collaborate on the Jupiter Europa Orbiter (JEO).