99Work in Progress

Table of Contents

List of Tables

List of Figures

1 Related documents

1.1 Reference documents

The following documents provide background information as to the present document. Under no circumstance shall the content of reference documents be construed as applicable to the present document, in part or in full, unless explicitly mentioned in the present document. RD1 SysML Spec v1.2 RD2 The Art of Systems Engineering, Maier, Rechtin, CRC Press 2002 RD3 E. Hull, K. Jackson, J. Disk, “Requirement Engineering”, Springer, 2005.

2 Overview of the APE Case Study

Large telescopes pose a continuous challenge to systems engineering due to their complexity in terms of requirements, operational modes, long duty lifetime, interfaces and number of components. A multitude of decisions must be taken throughout the life cycle of a new system, and a prime means of coping with complexity and uncertainty is using models as one decision aid. The potential of descriptive models based on the OMG Systems Modeling Language (OMG SysML™) is shown in different areas: building a comprehensive model serves as the basis for subsequent activities of soliciting and review for requirements, analysis and design alike. Furthermore a model is an effective communication instrument against misinterpretation pitfalls which are typical of cross disciplinary activities when using natural language only or free-format diagrams. Modeling the essential characteristics of the system, like interfaces, system structure and its behavior, are important system level issues which are addressed. Also shown is how to use a model as an analysis tool to describe the relationships among disturbances, opto-mechanical effects and control decisions and to refine the control use cases. The Active Phasing Experiment has been supported by the FP6 research program of the European Union. It started beginning 2005 to finish in June 2009. The consortium which participated to this research project is composed of the Instituto de Astrofisica de Canarias in La Laguna in Tenerife in Spain, Fogale Nanotech in Nîmes in France, the Laboratoire d'Astrophysique de Marseille in France, the Osservatorio Astrofisico di Arcetri (INAF) in Italy and of the European Southern Observatory (ESO). Some of the next generation of giant optical telescopes will be equipped with segmented primary mirrors composed of hundreds of hexagonal segments. It is necessary to operate at the diffraction limit of such telescopes if the telescope is to use adaptive optics and be a science driver, and this can only be achieved if the segments are well-aligned both in height, called from now on "piston", and in tip and tilt. The fast control of the rigid body positions of the segments will be based on measurements made with edge sensors. These, however, can only measure differential movements between adjacent segments and therefore have to be supplied with reference values for the absolute measurements of the piston steps at the intra-segment borders. At the moment, such reference values can only be obtained with a precision of the order of a few nanometers by optical measurements, preferably using the light of a star in the field of the telescope. The Active Phasing Experiment has then been proposed by the ESO to demonstrate its capability to align in tip-tilt and piston within few nanometers the segmented primary mirror of a giant telescope, performance which has not been made up to now in Europe. The goal was to simulate a VLT having a segmented primary mirror composed of 61 segments in order to demonstrate the functionality of 4 different optical phasing technologies and also new control principles using theses Optical Phasing Sensors (OPS). The four OPSs have been first tested in laboratory using an atmospheric turbulences generator and VLT simulator. Then it has been shipped and installed on the Nasmyth platform of Melipal (UT3) on Paranal. A total of 30 nights were dedicated for the commissioning and the experiment on sky. APE has been dismounted and shipped back to Garching in June 2009. The essential purpose of the Active Phasing Experiment was to explore, integrate and validate non-adaptive wavefront control schemes and technologies for an European Extremely Large Optical Telescope (ELT). This includes: • Evaluating and comparing the performance of four phasing wavefront sensors, in the laboratory and on-sky; • Integrating segmented aperture control into an active system (including field stabilization and active optics), and driving both the active system and the segments control system from the output of the system. To this end, APE was conceived as a technical instrument fully compliant with the ESO VLT/instrument standard to be installed and tested on-sky at a Nasmyth focus of a VLT unit.: The telescope provided all active functions (field stabilization, focusing, centering and active deformable mirrors) and the APE instrument emulated the optical effect of segmentation only.

Figure 1. Ape fully installed on Melipal Nasmyth A.

In practice, this was realized by re-imaging the telescope pupil onto a small Active Segmented Mirror (ASM), and by directing the output beam(s) of the instrument towards the Optical Phasing Sensors module. The segments of the mirror were activated in piston and tip-tilt with 3 piezo actuators and controlled in closed loop using the Internal Metrology (IM) developed by Fogale Nanotech. The optical phasing sensor module included four OPSs, each based on a different technology and developed by a different partner. IAC had the responsibility to develop a curvature sensor, LAM a phase and spatial filtering sensor, INAF a pyramid sensor, and finally ESO had to develop a new type of Shack-Hartmann sensor and also the Opto-Mechanical Bench (OMB). APE, as any complex system, has a large number of functional, performance, physical and interface requirements which have to be satisfied. This implies the need for a professional requirements engineering and management during the project. This is the first application of SysML during the development. APE has been designed as a modular system on a 3 by 2 m2 optical table. The main subsystems are a turbulence generator (called MAPS), the Active Segmented Mirror, a dual wavelength phase shifting interferometer called Internal Metrology, four optical phasing sensors, the junction boxes and the OMB. The OMB is considered as a sub-system in itself and contains all opto-mechanical components which have not been listed as a subsystem like the derotator, the calibration unit, the off-axis parabola etc. Three electronics cabinets contain the amplifiers, the analogue-to-digital converter cards, the controllers and the nine central processing units required for the control of the electronic components of APE (six CCDs, five translation stages, twelve rotating stages, two fast steering mirrors and the 183 actuators of the 61 segments incorporated in the ASM). The cabinets are linked to the OMB via the junction boxes. The control system alone consists of 12 computing nodes. On sky the telescope replaces MAPS. In total there are 42 optical surfaces (lenses, filters, mirrors, beam splitters, etc.) between the focus of the telescope and the entrance foci of the phasing sensors.

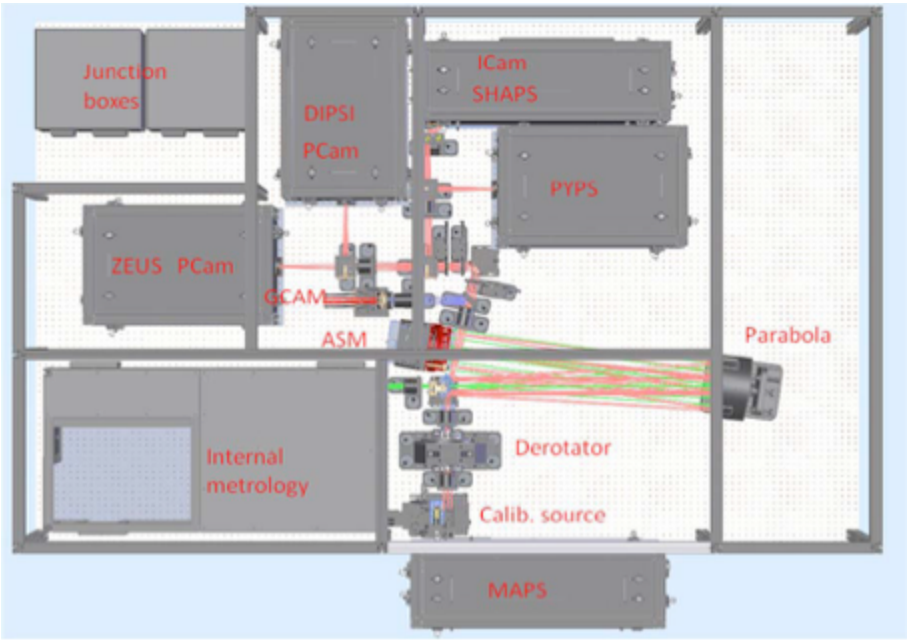


Figure 2. Top view of the APE bench

These elements offer all kinds of optical, mechanical, electronic and software interfaces, both system internal and external to other systems. It also challenges the control, since there are several open and closed loop systems required. A significant amount of data is produced by image processing data flows. Since APE will be deployed in the lab and in an already existing telescope, slightly different functional aspects are active depending on the deployment mode. Therefore different interfaces to existing systems are needed. All these characteristics made APE well suited to evaluate SysML potential in tackling similar issues. SysML is only a graphical language. It defines a set of diagrammatics, modeling elements, a formal syntax and semantics. As any language (formal or informal) it can be used in many different ways and many wrong ways too. Most notably the creation of nonsense models is possible. The SE^2 MBSE challenge team defines as its main goals to Provide examples of SysML, common modelling problems and approaches. Build a comprehensive model, which serves as the basis for providing different views to different engineering aspects and subsequent activities. Demonstrate that SysML is an effective means to define common concepts. Demonstrate that a SysML model enhances traceability The Cookbook covers topics like naming conventions, model annotations, external references; necessary system models, aspects and views; heuristics for modeling system requirements; guidelines for modeling the system structure (product tree, context variants, design variants, re-usable parts, system hierarchies); Interface modeling (logical and concrete, mechanical and flow, ports); allocation strategies; test case tracing; parametrics for performance models; style and layout issues.

3 System Views

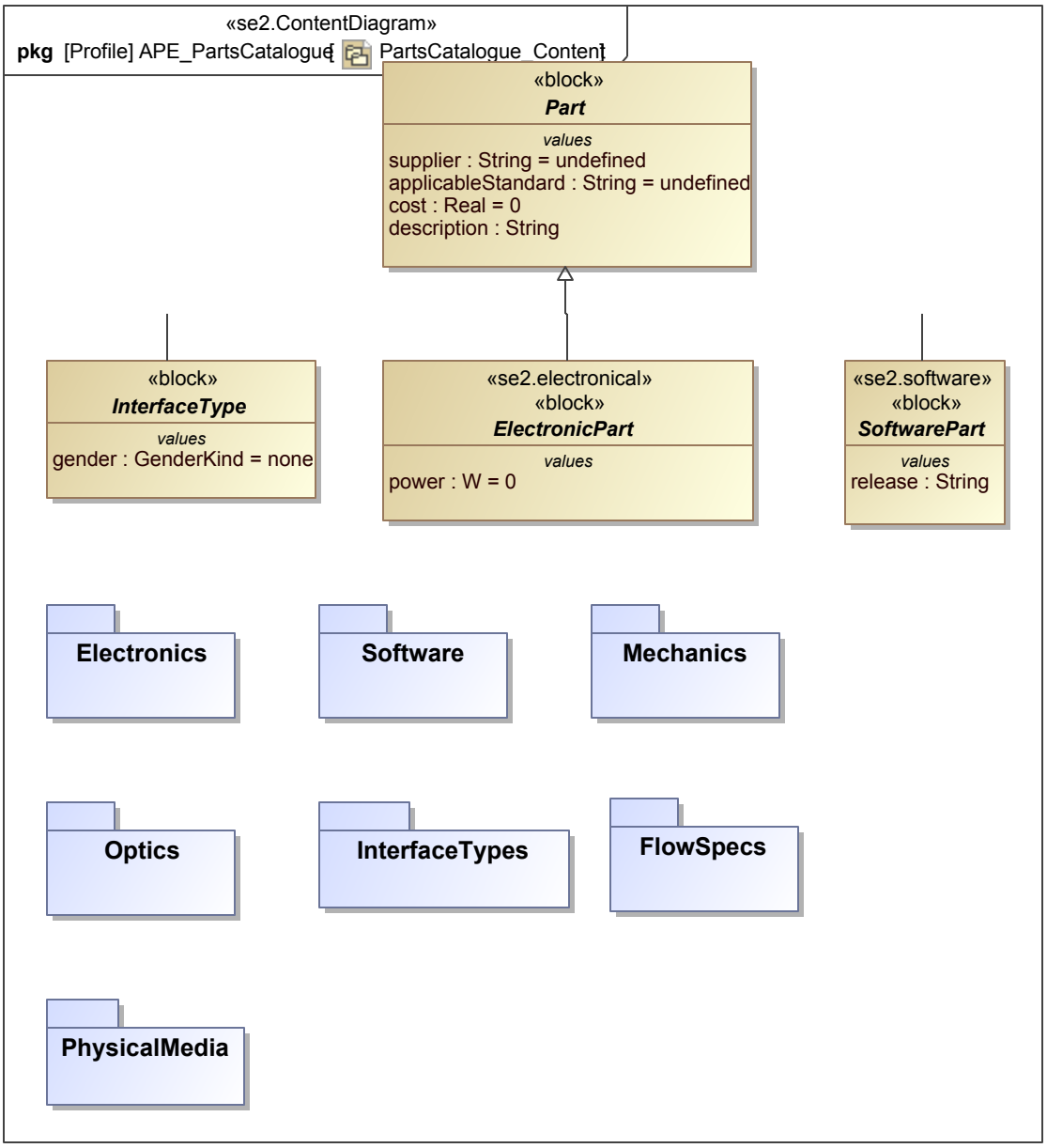


Figure 3. PartsCatalogue_Content

3.1 Guidelines for necessary system aspects and perspectives

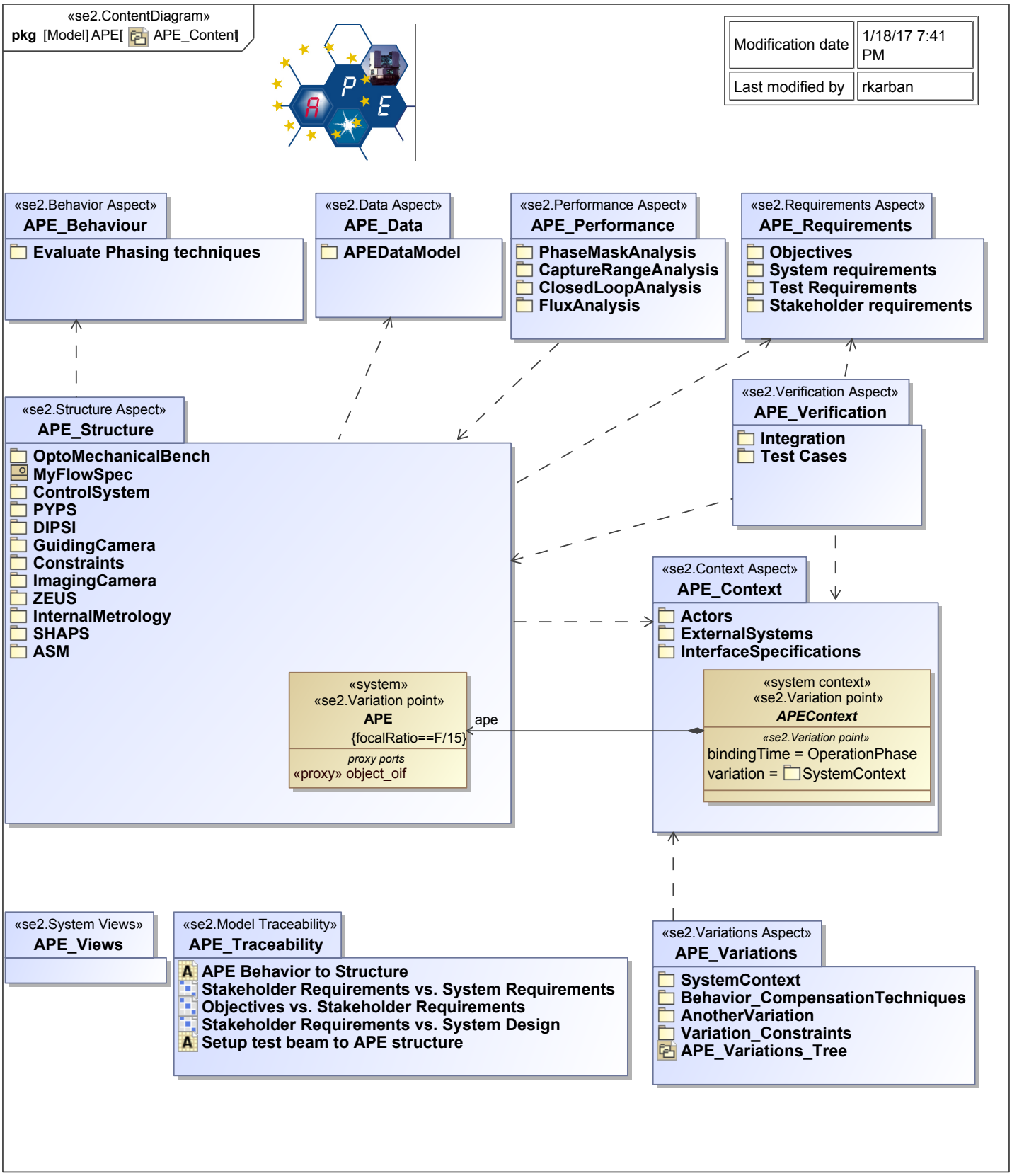


Figure 4. APE_Content

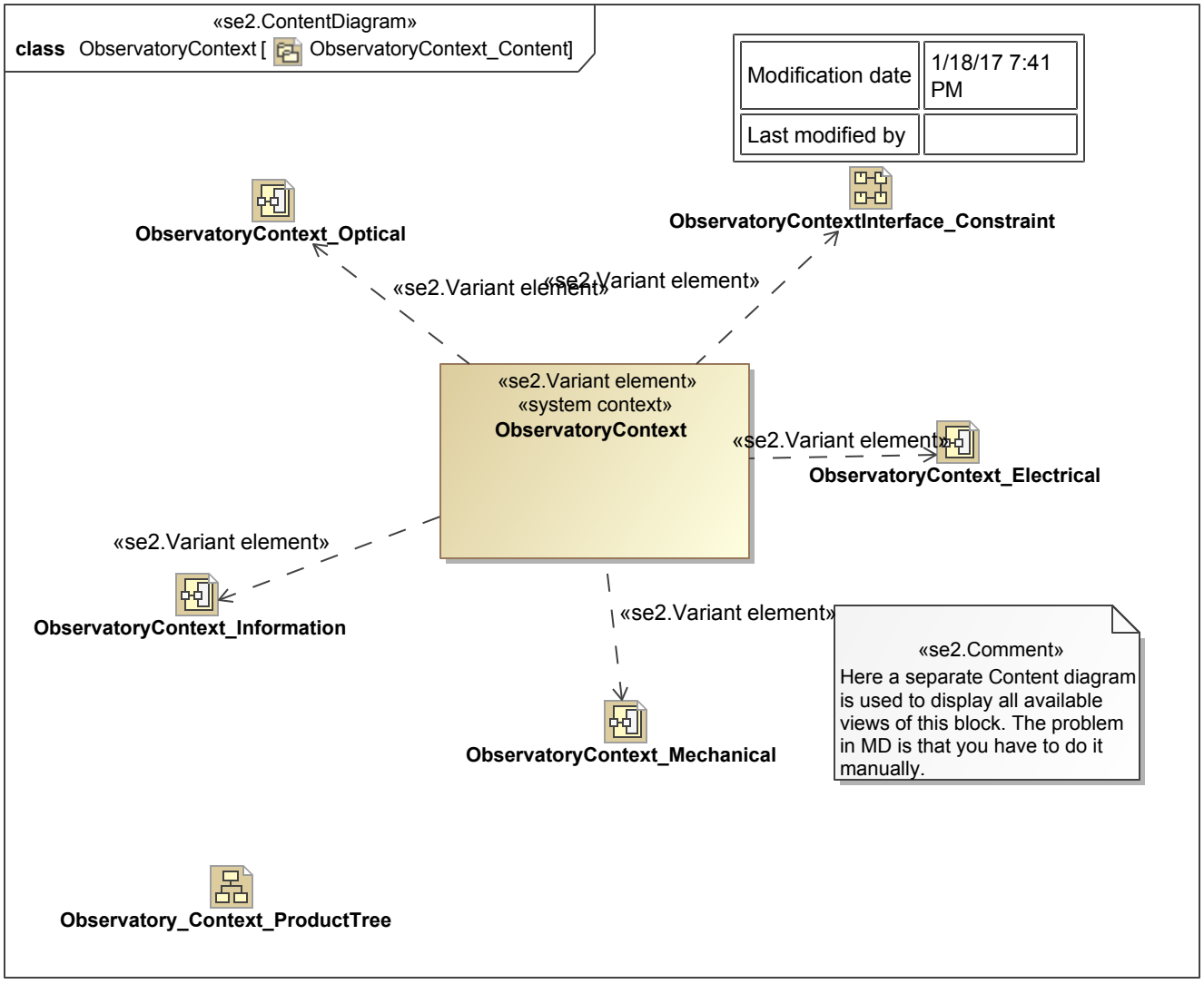


Figure 5. ObservatoryContext_Content

3.1.1 Mechanical Perspective

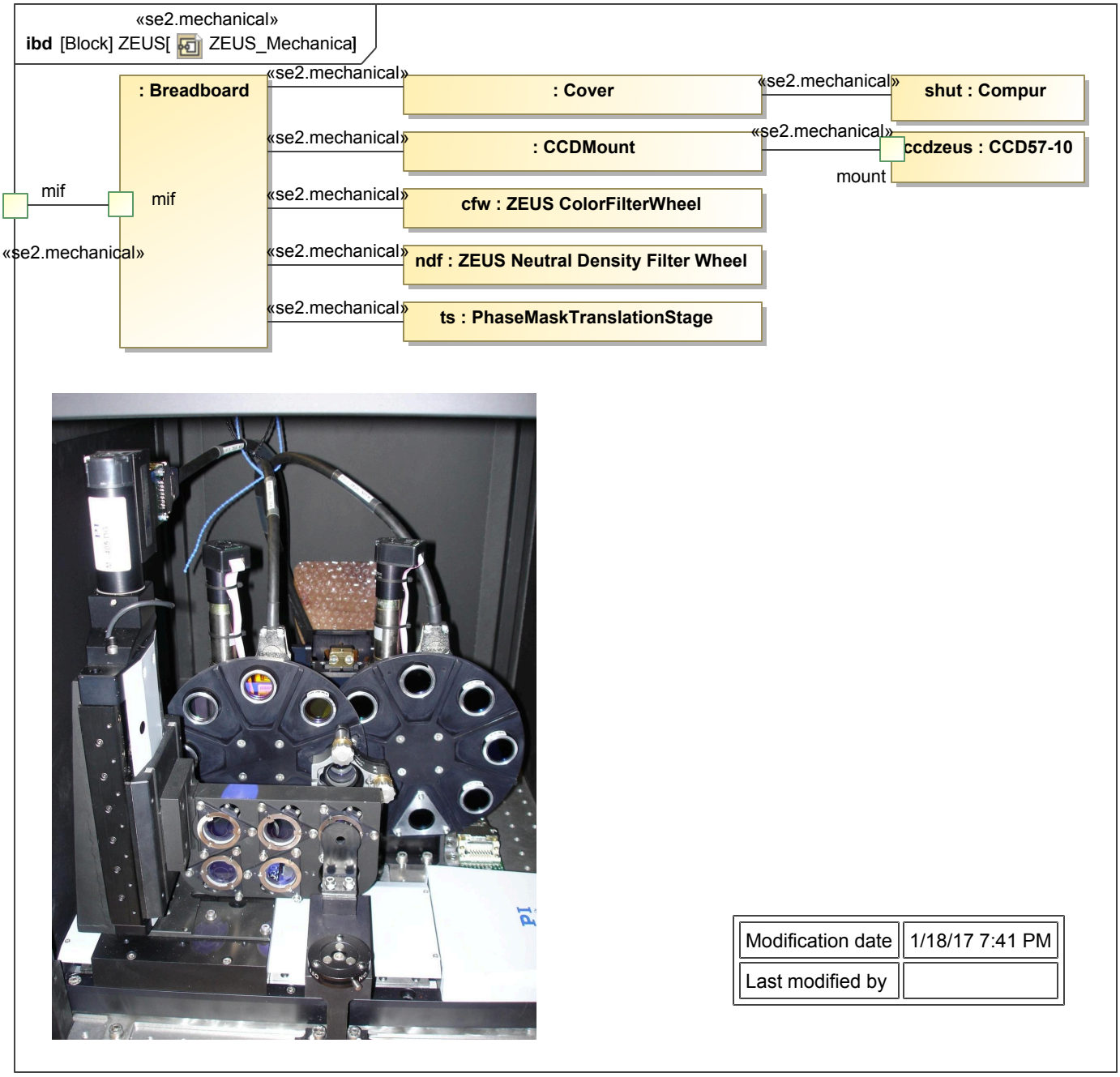


Figure 6. ZEUS_Mechanical

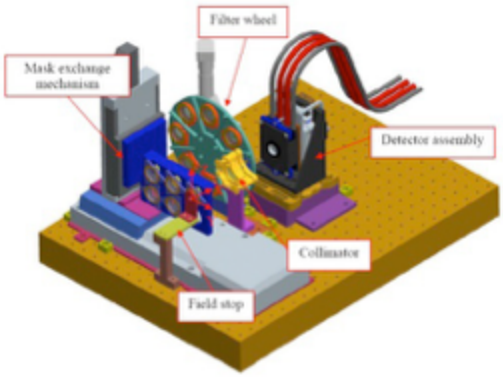


Figure 7. ZUES 3D view of the opto-mechanical concept

3.1.2 Optical Perspective

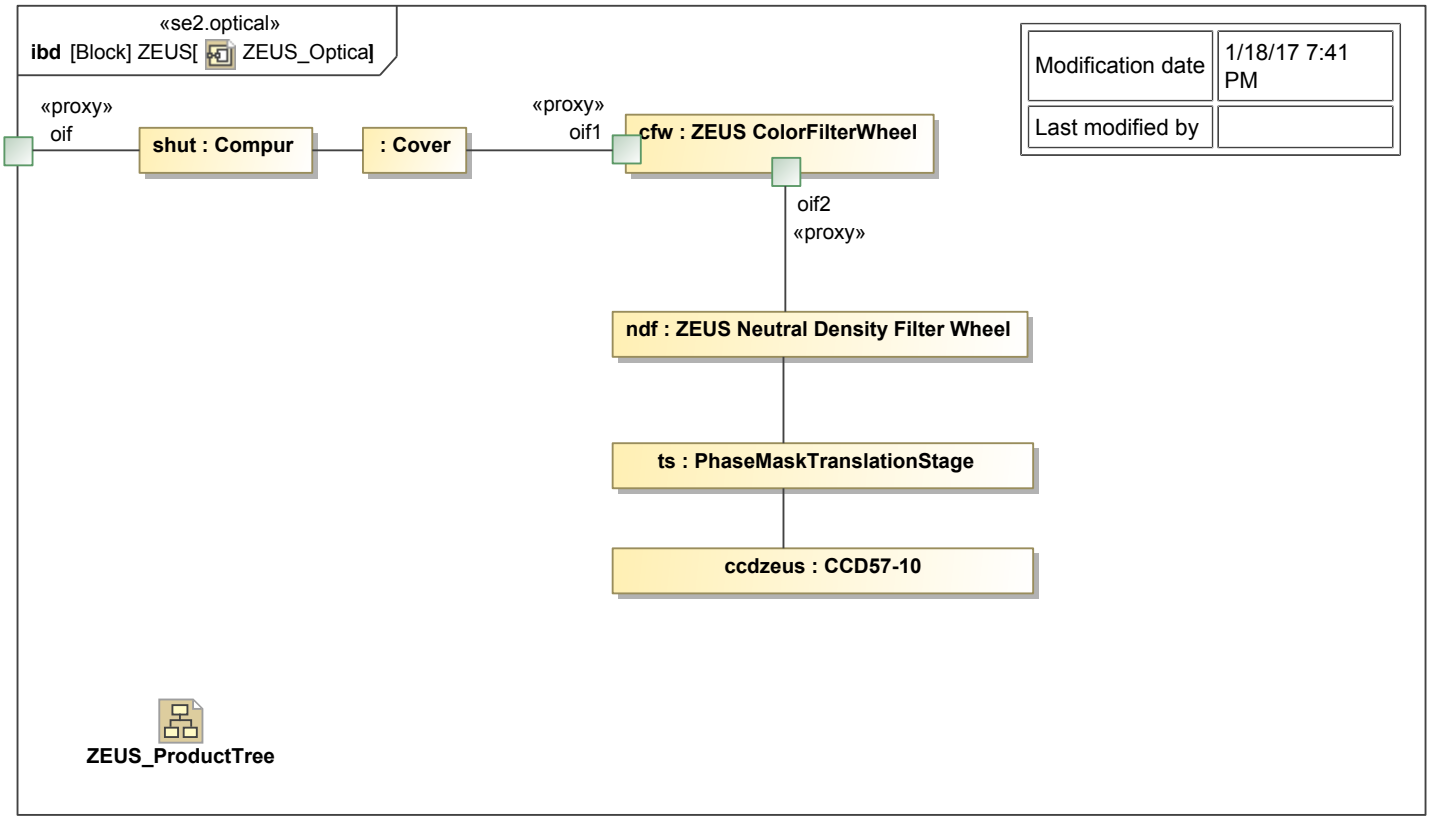


Figure 8. ZEUS_Optical

3.1.3 Electrical Perspective

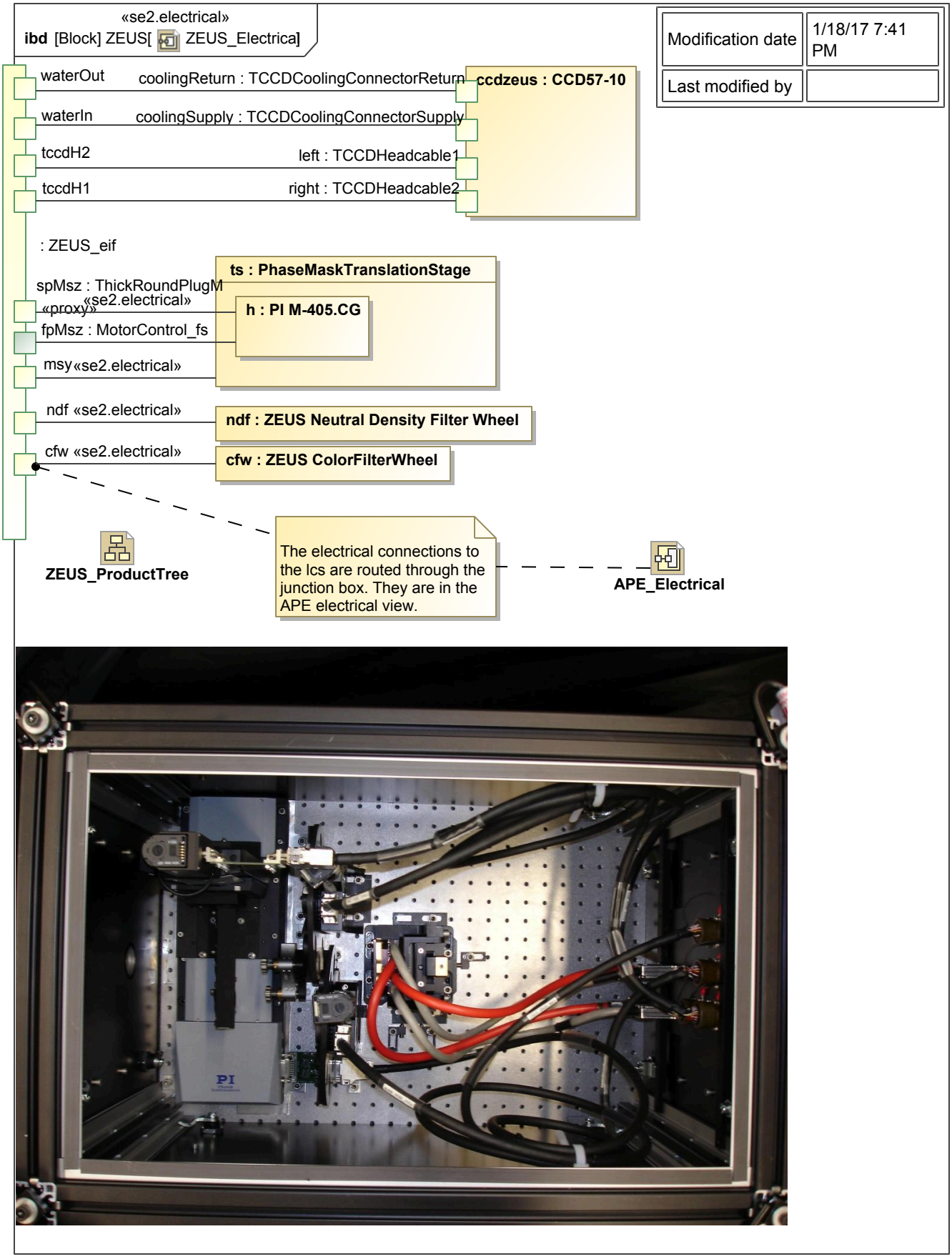


Figure 9. ZEUS_Electrical

3.1.4 Information Perspective

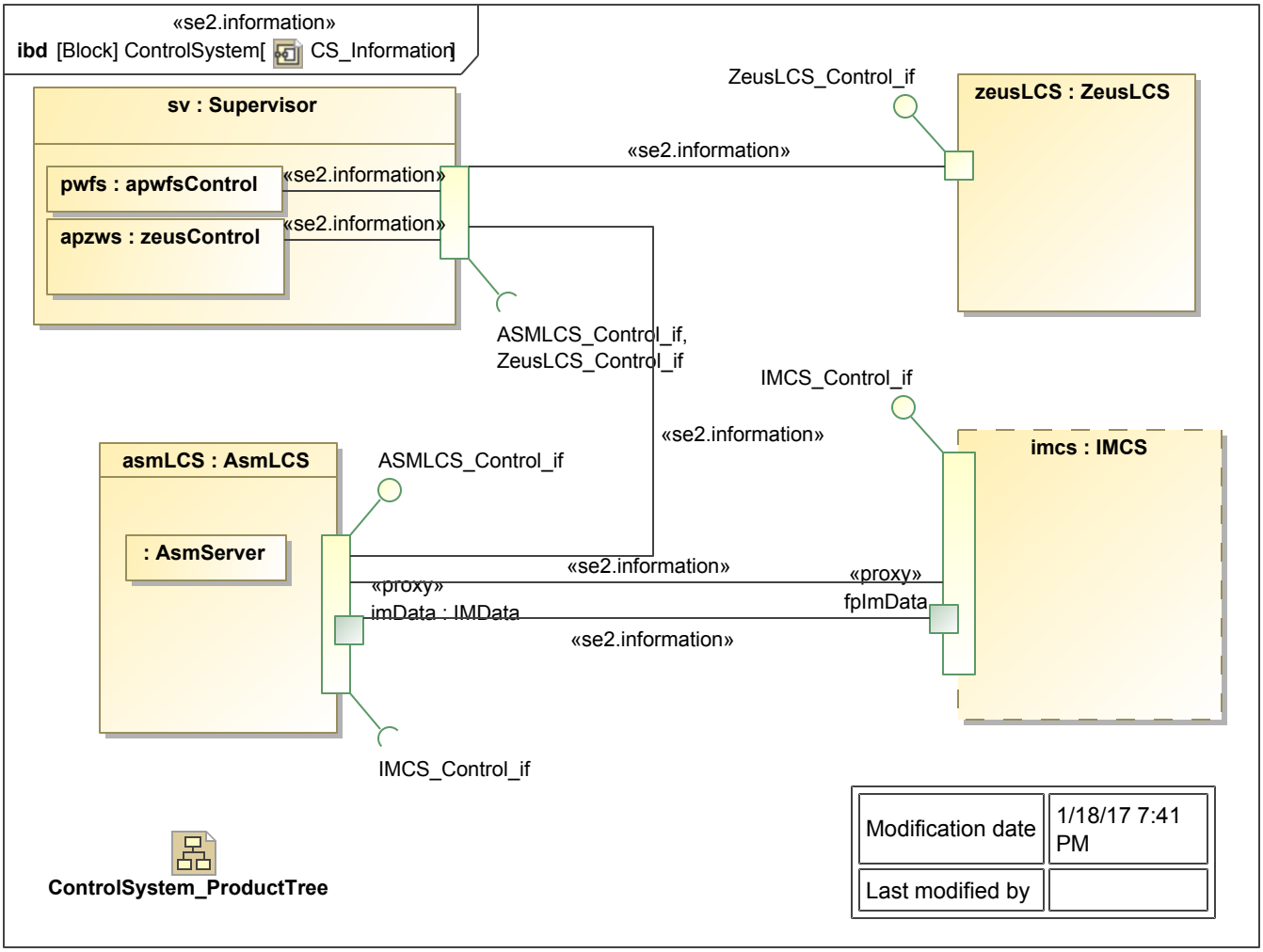


Figure 10. CS_Information

3.2 Relationship between Model Aspects

4 Ontologies

When creating a model it is of utmost important that the semantics of the model elements and their relationships is properly defined. A correct definition of the semantics allows validating models and forces the modelers to model similar things in similar ways, called conceptual integrity. In large multi-user projects a consistent model structure and organization is the foundation on which modeling can take place. Apart from the model structure and organization also the modeling patterns must be defined, e.g. how interfaces are modeled. All this is required to guarantee readability, navigability, and consistent representation of information. An ontology defines formally terms, concepts, and their relationships. There can be general purpose Ontologies to define something like model structure and very domain specific Ontologies like the conceptual elements of a telescope or space craft. The most formal way is to define an Ontology in an ontology language like OWL (Web Ontology Language) and apply model transformation to create a profile with SysML stereotypes, representing those concepts and extend UML meta classes or specialize SysML stereotypes. Most recipes and patterns in this Cookbook are based on Ontologies, e.g. Project Ontology, Variant Ontology, Interface Ontology, Product Tree Ontology, etc. They are defined as a Class model in the SE2Profile. The Classes are allocated to a stereotype model. This is a less rigorous, yet worthwhile exercise to formalize modeling. Knowing the Ontology, its allocation to stereotypes, and the mapping to SysML model elements allows in principle to validate the model against the Ontology and check it for consistency; i.e. determine, whether the representation of the model is consistent with the ontology. Eventually, every model element must have a stereotype applied and its meaning properly defined in an Ontology. In the following some examples are shown. The complete ontology and mapping is defined in the APE model.

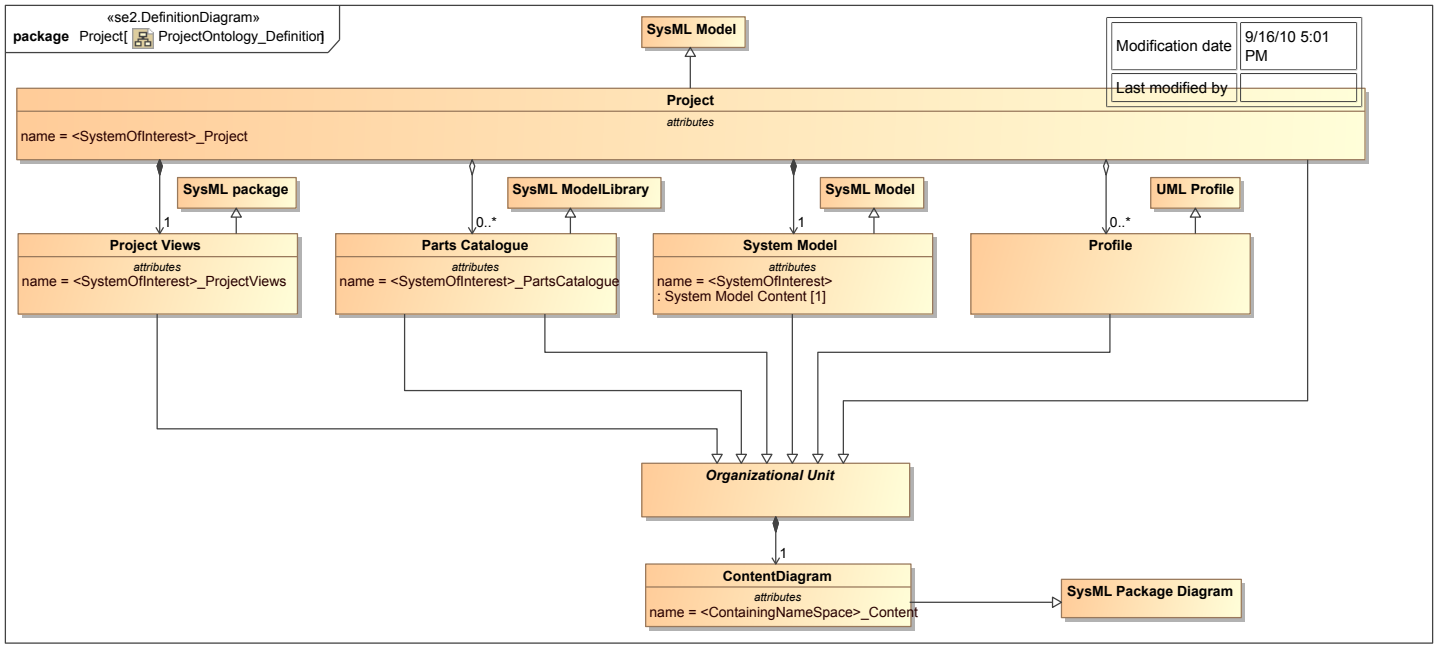


Figure 11. ProjectOntology_Definition

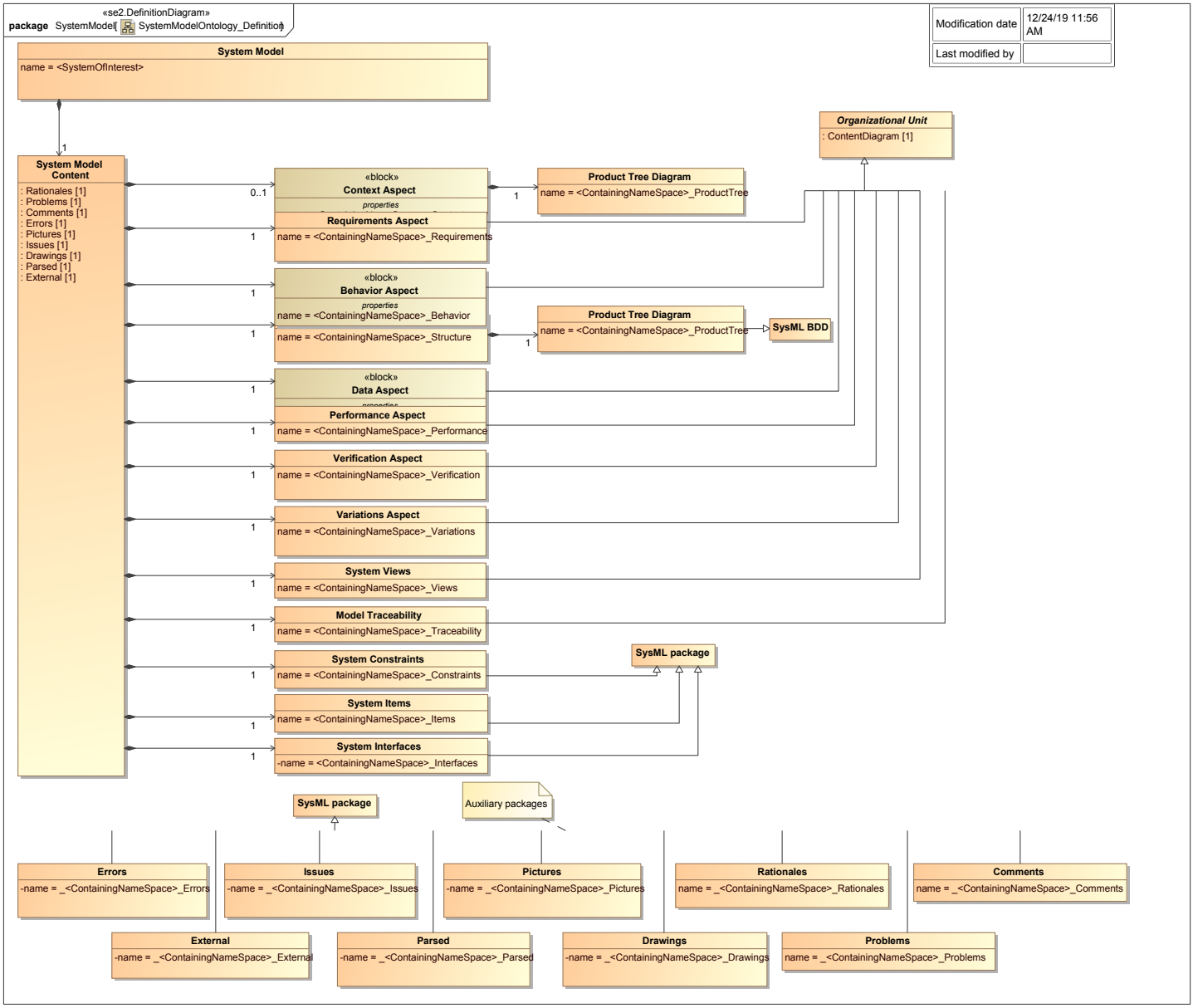


Figure 12. SystemModelOntology_Definition

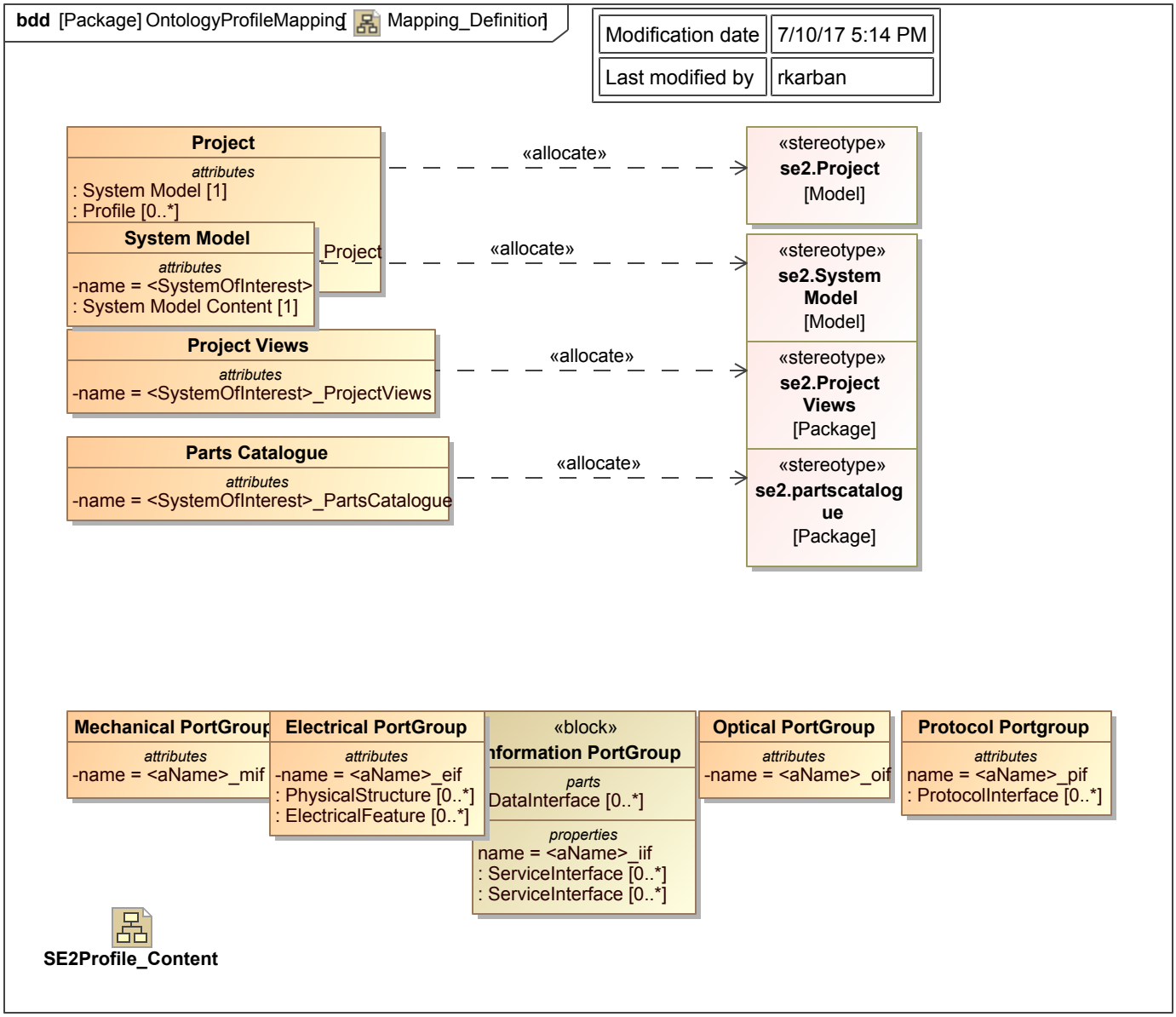


Figure 13. Mapping_Definition

5 Challenges of SysML deployment in an organization

¥ Best practices o Mentor and SysML/Tool confident person o Extend gradually the range, define modeling goals, guidelines, and standards o “Just use it!” (Do not talk about modeling and SysML too much as it raises fear of waste of time) ¥ Observations o (no) support/commitment from management but a necessity for engineering o How is presented to management? How do they see a gain? There is no immediate real-life artifact (no LED blinking, no tangible objects) o Under pressure people fall back to techniques they know o People are often lazy to learn/apply something new o Not modeling means often not understanding and therefore underestimating the problem. o Modeling reveals complexity and people get scared o Contractual problems with models – only text is understood by lawyers

6 Tool (MagicDraw) related guidelines

6.1 Style and Layout

6.1.1 Remove stereotype <<block>> of parts in IBDs to increase readability

Parts in IBDs show by default the stereotype <<block>>. This clutters the diagrams and does not add any useful information. Hide the stereotype (it makes the diagram more compact) unless it adds value.

6.2 Navigation

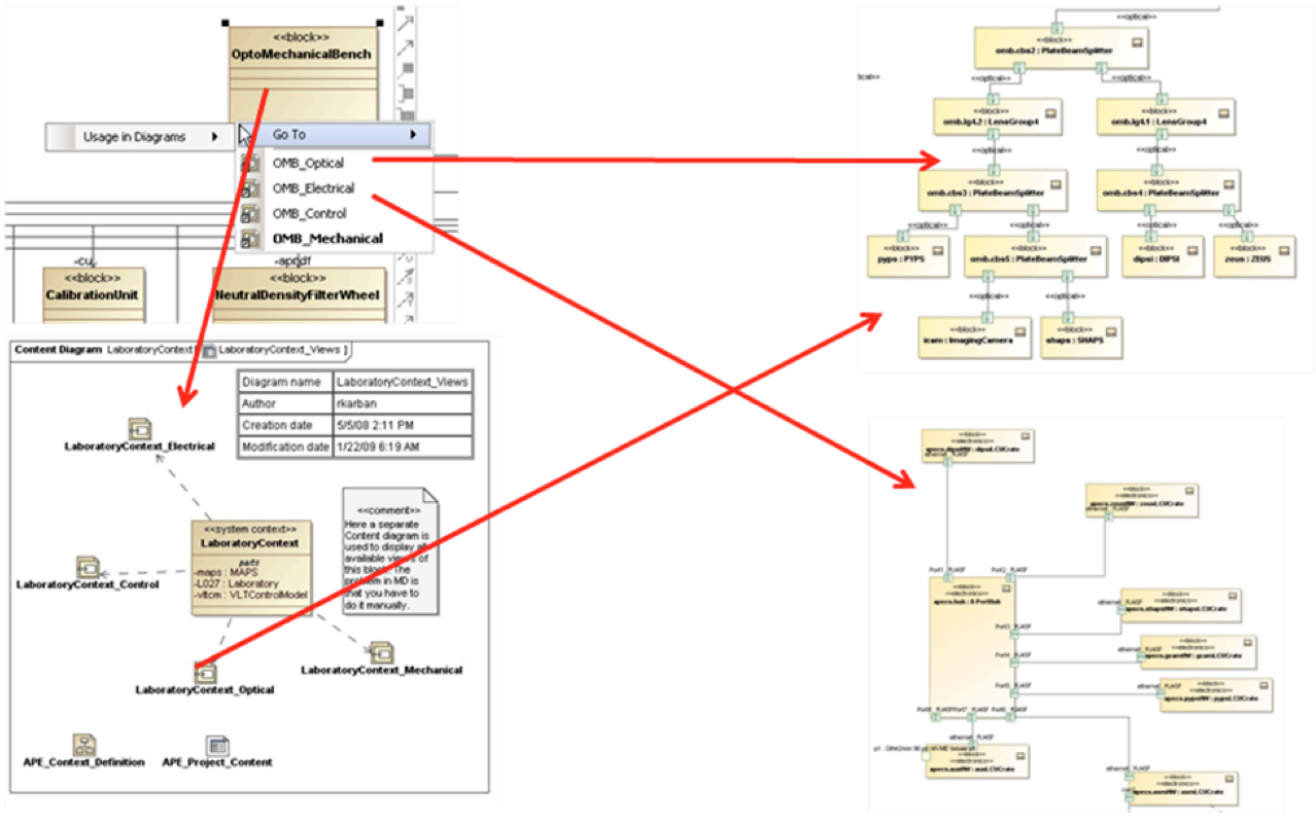


Figure 14. Hyper linking a Model for Navigation

• As soon as you create a new element's symbol, ask yourself: "What can I hyperlink it to ?". • Hyperlink every single model or package to a SysML Package Diagram (or SysML Block Definition Diagram) • Hyperlink every assembly Block to its Internal Block Diagram (IBD) • Place at least one Block Definition Diagram (BDD) diagram icon on every Internal Block Diagram (IBD) • Drag the Internal Block Diagram (IBD) icon of the Type of a Property onto that Property in an IBD so that you can open up a part into its matching IBD. • Hyperlink your top-level SysML ‘system’ and/or ‘system context’ to their IBD or BDD and place them on every diagram possible throughout your project. However, be aware that this can prevent modularization.

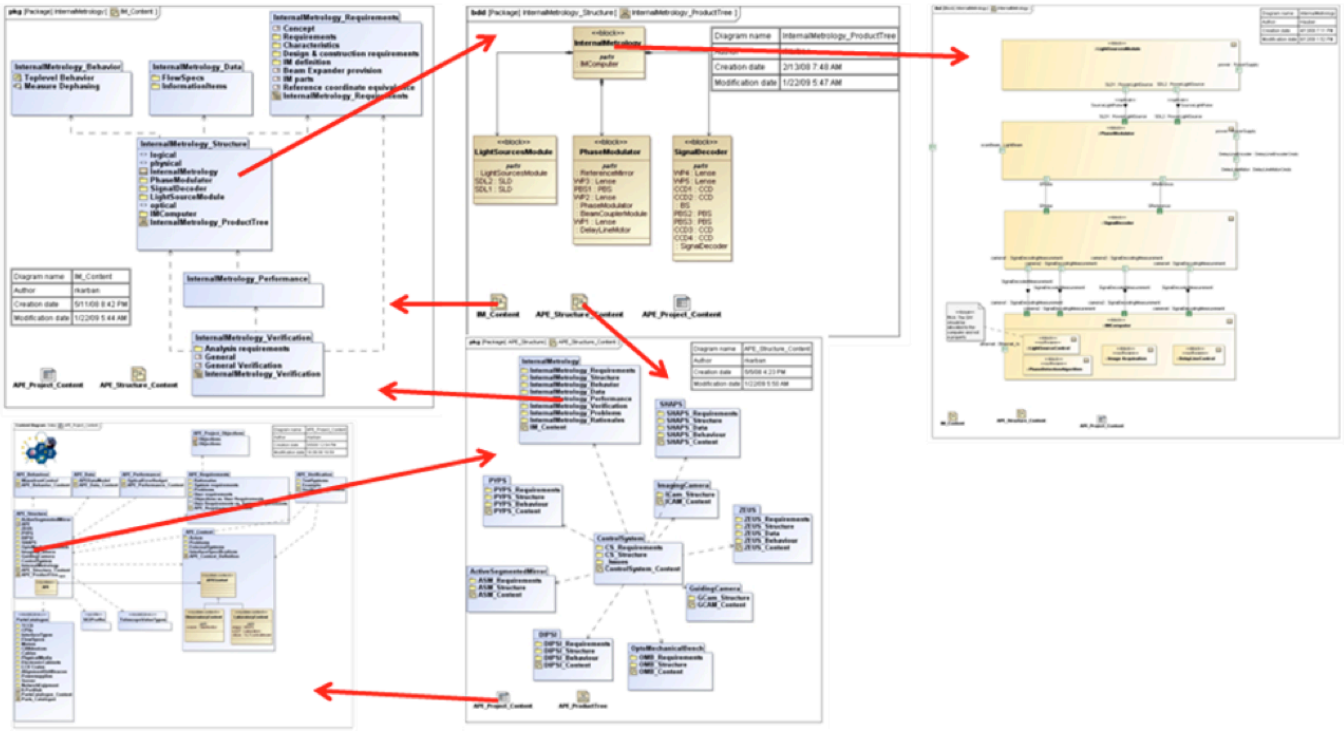


Figure 15. Navigation with hyper links

Navigate on elements. Navigate very little via the browser. You need to make you project completely navigable ON THE ELEMENTS and also ON THE PACKAGES. It is a basic systems engineering idiom. One needs to be able to OPEN UP packages and OPEN UP systems and blocks. Use hyperlinked packages with contents list and dependencies between packages. You can also show (possibly stereotyped) dependencies between packages better to reinforce the sense of systems engineering, You don't have to be too fancy, just reflect the basic sense or process. You need a clearly stated <> (APE) Show BOTH a <> and a <> at the very top level. (You needn't show the parts and properties, however you might like to.) Also, you should link your Context to a diagram (like Context Definition). The hyperlinks to a block lead you to its Content and structure diagrams A block, representing a complete sub-system, is hyperlinked to its _Content diagram and to its IBDs.

7 Introduction

7.1 Scope

This document provides modeling patterns, recipes, guidelines, and best practices for the application of SysML. The presented examples are rather methodology independent and can be used (maybe with some adaption) in any MBSE environment.

Furthermore, it provides additional explanation on SysML syntax, semantics, and concepts. It partially annotates the SysML specification.

This Cookbook gives also some advice on general system modeling strategies and assumes basic knowledge of SysML. The Cookbook is not an introduction to SysML.

Our examples are about the Active Phasing Experiment (APE) project and modeled with the modeling tool MagicDraw from NoMagic. Unless mentioned otherwise everything is independent of these can be applied to other domains and tools.

7.2 Purpose

Each project has to establish a minimal set of rules for using SysML constructs and how to organize the model. SysML is just a language which can be used or abused.

The document provides to the modeler solutions for day to day modeling problems, the modeling “technicalities”. Applying those recipes the modeler can focus more on the content of the system of interest than on modeling problems.

Systems Engineering diagrams always look so simple!

The Systems Engineering exercise is to first make the problem real to the reader/participant, then second to show an 'obvious' solution. To a casual reader these systems decisions should look like no-brainers - that is the value of systems engineering.

This cookbook is organized in this way:

MBSE Goals contains recipes about the organization of models

Chapter 4 proposes style, layout and naming conventions

Chapter 5 gives guidelines about modeling the system views.

Chapter 6 gives guidelines about modeling requirement and use cases.

Chapter 7 contains recipes for modeling the system structure.

Chapter 8 describes the modeling of interfaces.

Chapter 9 deals with the behavior modeling.

Chapter 10 provides guidelines for modeling non-functional aspects.

Chapter 11 is about document generation out of the model.

Chapter 12 discusses the integration with other engineering disciplines.

7.3 About SE^2 and APE

In the framework of INCOSE’s strategic initiative, the *Systems Engineering Vision 2020*, one of the main areas of focus is model-based systems engineering. In keeping with this emphasis, the European Southern Observatory (ESO; http://www.eso.org/) is collaborating with the German Chapter of INCOSE (http://www.gfse.de/) in the form of the “MBSE Challenge” team SE^2. The founders of the team were Robert Karban (ESO), Tim Weilkiens (oose) and Andreas Peukert (TUM). Afterwards Dr. Rudolf Hauber (HOOD Group), Michelle Zamparelli (ESO), Rainer Diekmann, and Andreas Hein (TUM) joined the team.

The team's task is to demonstrate solutions to challenging problems using MBSE. The Active Phasing Experiment (APE; see Gonte et al. 2004), a European Union Framework Program 6 project, was chosen as the subject of the SE^2 Challenge Team (http://mbse.gfse.de/). Many technical products in the telescope domain show an increasing integration of mechanics with electronics, information processing, and also optics, and can therefore be rightly considered as optomechatronic systems.

The SysML models were created by reverse engineering from existing documentation and from interviews with systems engineers. We will make use of Ingmar Ogren's concept of a common project model (Ogren 2000) to establish a common understanding of the system.

7.4 Abbreviations and acronmys

Table 1. Acronymns on VE

SysML	OMG Systems Modeling Language
IBD	Internal Block Diagram
BDD	Block Definition Diagram
PAR	Parametric Diagram

7.5 Glossary and definitions

Not applicable.

7.6 Document conventions

TBD

8 Project Description

9 MBSE Goals

10 How Do I Get Started?

11 Model Organization

If you start modeling a complex system considering many different engineering aspects you can end up very fast in a very confusing model because of a unclear organization of all its model elements. Therefore organizing the model in a proper way is crucial for the understanding and usability of the model.

11.1 Overall Model organization

SysML packages are the element to organize the model. How packages are used for organization is of fundamental importance in order to have a scalable, consistent, and navigable model structure which works also for large models. The model structure supports different levels of abstraction and deeply nested system hierarchies.

The SE2Profile defines an Ontology for the model’s organization where every package has a defined meaning, and certain diagrams must exist within those packages for navigation and definition of system elements.

The model is organized and decomposed along the product tree (i.e. the product breakdown structure) of the system being modeled. At each decomposition level a recurring package structure appears which corresponds to the different aspects of the system (e.g. structure, behavior, requirements, etc.).

11.2 Structuring the model using packages

To increase readability of and support navigation within the model, these guidelines for setting up model packages should be followed:

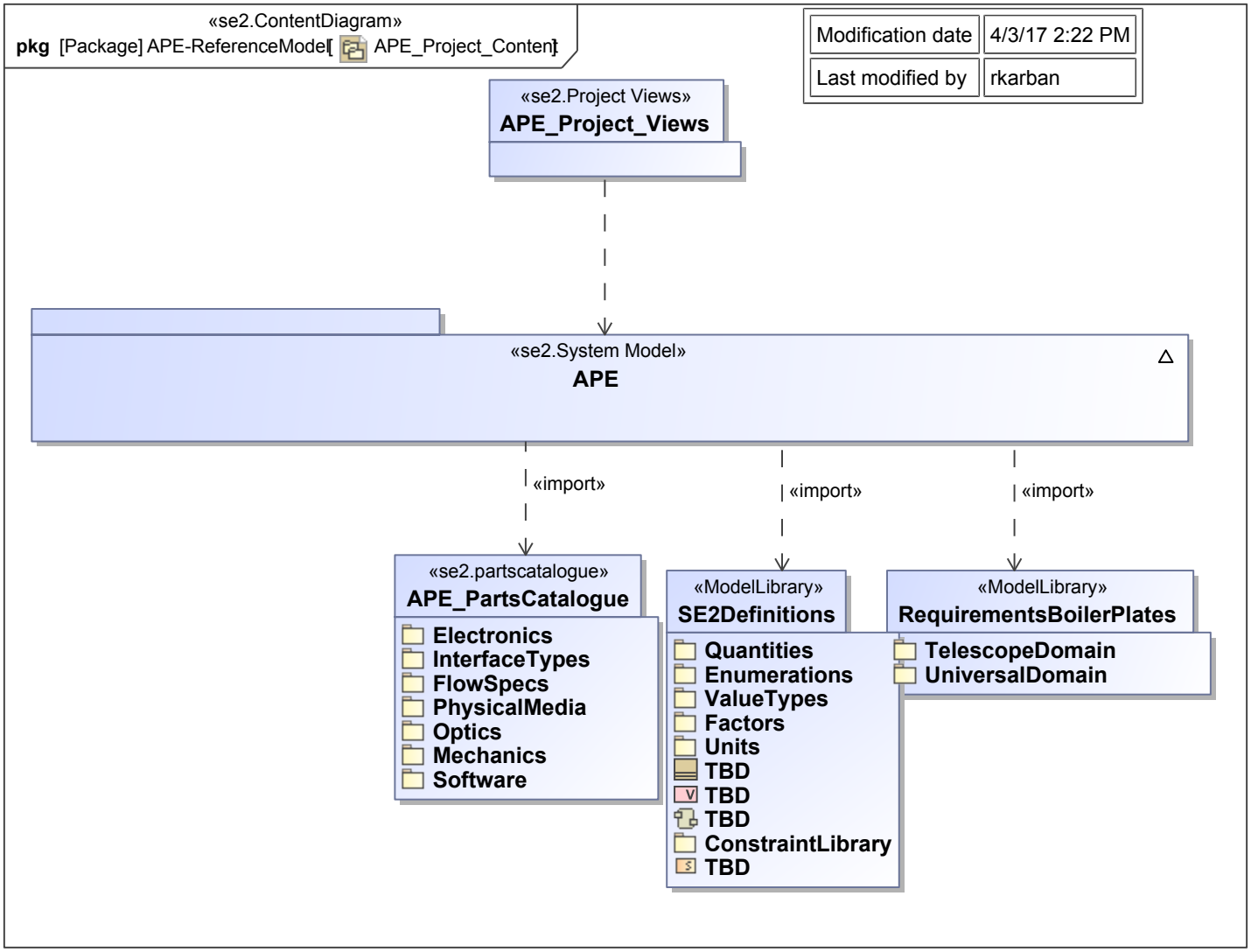


Figure 16. APE_Project_Content

- Packages are the basic tool for separating concurrent modeling tasks and assigning responsibility.
- Use recursive package structure patterns in the model to establish a good and easily understandable model structure: packages created at the highest level of the model shall be repeated for every lower level
- Structures (the design) are organized according to the system's product tree.
- Use packages to separate different aspects (each serving a distinct purpose) of the system. Having, conceptually, independent aspects allows:
    - Independent reorganization, if needed
    - Easier integration (import) of an external requirements model (like Doors)
    - Each aspect becomes simpler.
    - You can link in a more obvious way the model elements with <<trace>> and <<satisfy>>.
- Create separate (auxiliary) packages containing model information, references to external information (like drawings, existing requirements or design documents) and information about the model. They are prefixed with _ (underscore) to distinguish them from packages which represent a view of the system.
- The top level package is <projectname>_Project. It contains the System Model package, the used parts catalogs, applied profiles, and other model libraries.
- The BDD of each substructure defines its elements and therefore its product tree
- Group Comments, Errors and Issues in separate packages to find them easily.
- Recurring packages in every model level are:
    - Behavior
    - Context. The package Context exists typically only at the top level because the lower level context is implicitly defined by the IBDs of the higher level.
    - Data
    - Items

    - Performance
    - Requirements
    - Structure
    - Traceability
    - Variations
    - Verification
    - Views
    - _Comments
    - _Drawings
    - _Errors
    - _External
    - _Issues
    - _Problems
    - _Rationales
- At the top level the Requirements package contains Objectives, Stakeholder requirements and System requirements. Whereas on lower levels it contains only derived requirements of the sub structures. They are kept together with the design to have a consistent package, e.g. to give to a contractor. The same applies to behavior. Every decomposition level defines its behavior, which is part of the higher level behavior.
- Naming convention for model elements are another important point to improve understanding. For example, all APE packages related to a certain (sub) system have as a prefix the name of the (sub) system, i.e. its owning package to have a unique identification of the package for navigation. e.g. APE_Requirements, ASM_Requirements.
- All packages have stereotypes applied with a defined semantic according to the Model Structure Ontology.

Figure 17. The Containment tree of the Model Structure

The _Content diagram is a package diagram and is used to navigate with in a sub-system. It shows all its aspects as structure, performance, requirements, behavior, etc.

The _ProductTree is a block definition diagrams and is located in the structure package. It shows the product tree; i.e. how the sub-system is composed.

The purpose of the product tree diagram is merely to give a managerial perspective of which parts need to be built or procured, also, but not exclusively to get cost information for the project. The criteria to include elements by means of composition in this diagram is, that they have been built/procured explicitly for the system. The product tree **does not** in general show a snapshot of a system configuration. Multiplicity information is relevant for a product tree. Use the composition relationship in a whole-part relation when the whole and the part share the same life-cycle, use reference otherwise.

The need to provide uncluttered readable diagrams normally conflicts with the one to produce a high level overview of which parts are crucial / relevant to the system. It is recommended that a product tree diagram only contain at most two levels of decomposition. The need to have an all exploded comprehensive product tree may in theory be satisfied by automatically merging all existing product tree diagrams into a big one (remember, each sub-system contains recursively its product tree).

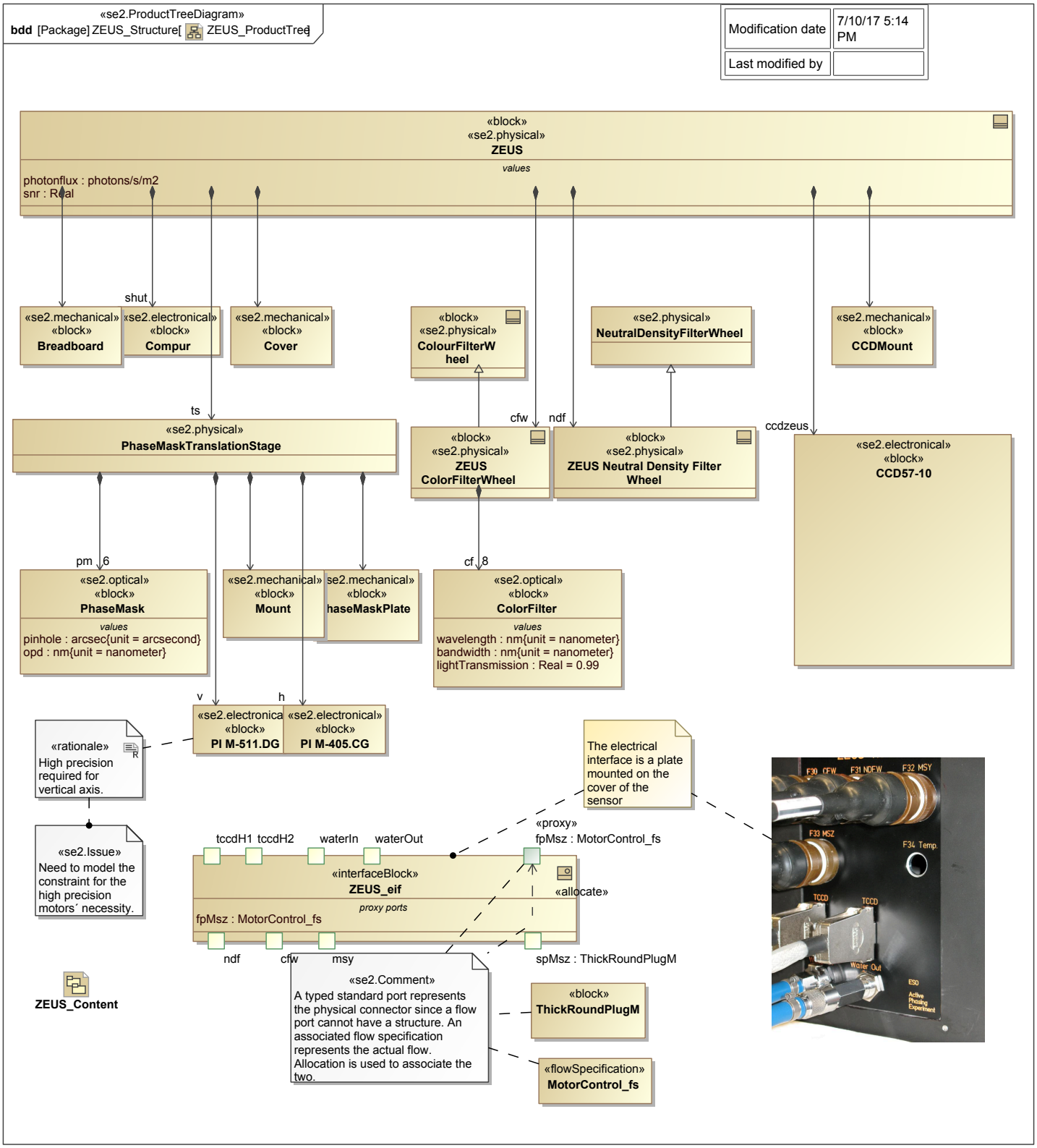


Figure 18. ZEUS_ProductTree

ZEUS is one of the evaluated phasing sensors (Figure 5) and is based on the modified Mach–Zehnder interferometer phasing sensor. It is mounted on a breadboard and consists of a shutter, a cover, a color filter wheel, a neutral density filter wheel, and a translation stage which carries a phase mask. Different phase masks can be moved to the focal position by means of a translation stage, able to move in the X and Y directions.

The two filter wheels located after the phase mask translation stage:

- A neutral Density Filter wheel: a set of 8 different neutral density filters are available

- An optical filter wheel: a set of 8 different optical filters centered on different wavelengths and with different bandwidths are available

11.3 Levels of Detail

The level of detail of a model is a very contentious issue. Systems can easily be “modeled to death” just for the sake of modeling.

Therefore the purposes of the model should be defined upfront, like:

- Which stakeholder shall use the model for what purpose?
- What does the modeler or systems engineer want to achieve with the model?
- What information shall be extracted from the model?
- Should the model give an overview or define in detail all flows?

We recommend to ask and answer these questions for each greater model element as well.

What is modeled and which model elements are used depends from the answers to the questions; e.g. a system engineer needs more sophisticated modeling elements to express different aspects, while a tester may need only test scenarios modeled simple in sequence diagrams.

11.4 Levels of abstraction

The levels of abstraction depend very much on the used methodology. The methodology defines if a logical (functional) model is built, if the logical model is only about behavior or also about structure, how the functional model relates to the physical model (the bill of materials), etc.

Since the intention of this cookbook is to keep it largely methodology independent, a simple split in functional and physical model is done.

- one modeling level is functional, describing system behavior (e.g. activities)
- one modeling level is physical, describing system structure (e.g. blocks)
- allocation is done by allocating behavior to structure

The physical model corresponds to a bill of material, i.e. a system which can be purchased off a catalogue.

Functional/Logical structure elements and their technological representations could be:

Table 2. Table1

Functional/Logical Element	**Technological Element**
Display	LCD, CRT, HMD, LED, etc
Receiver	Analogue radio receiver, DVB-T, etc
Electrical Generator	Solar Cells, Diesel-engine, Water-Turbine, etc
Cooling Unit	Fridge, Fan, Peltier, etc
Container	Shipping container, air container, etc
Data Storage Device	Disk, USB stick, DVD, etc
Computing Node	Desktop, Embedded, Blade-Server, etc
Motor	DC, stepper, etc

For each technological element there is in the end a concrete technology which is part of the bill of material and can be bought. For example, a particular Mercedes Diesel-engine, or a particular Siemens Water-turbine is used for the final system which complies with the performance constraints.

Typically, the concrete items are collected in a catalogue of pieces which can be re-used for every system.

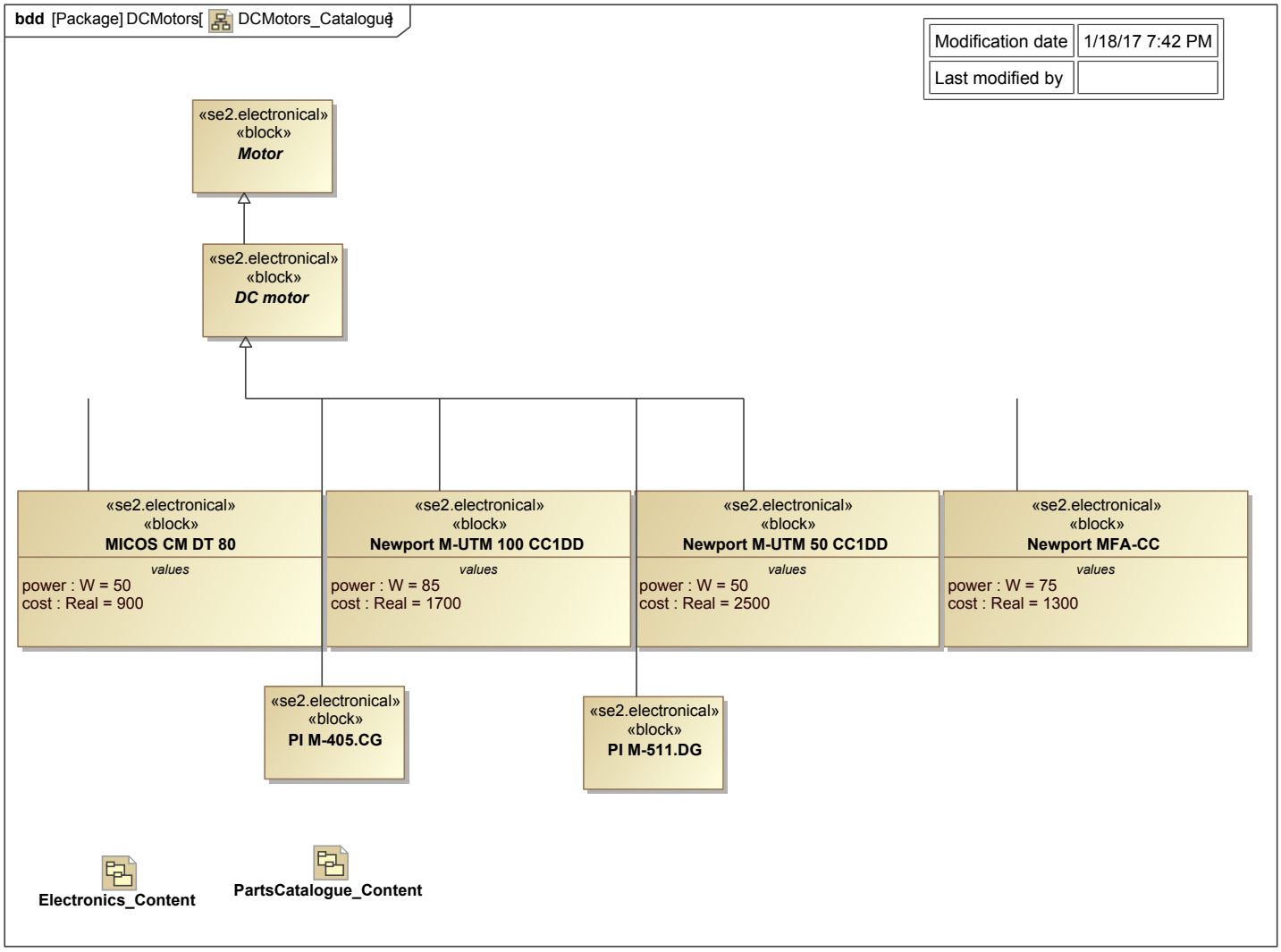


Figure 19. DCMotors_Catalogue

In the beginning of modeling it is often unclear which particular catalog item will be used in the end. It will dpend on different factors, like cost, performance, supplier warranty, etc. There might be even different variations of the system evaluated.

12 Styles, Layout, Naming

12.1 Formalizing the model with domain specific stereotypes

A major challenge is to keep the model organization, naming, and style consistent. One way is to define naming conventions and enforce them. However, since they are only strings it is more difficult to check them automatically. Having only naming conventions is also quite restrictive and a simple typo can change its meaning.

Therefore it is better to define one or more Ontologies (formal definition of terms and concepts and their relations), see chapter 12.

The Ontologies are implemented in the model by stereotypes, domain specific extensions of the SysML language.

This Cookbook defines a set of stereotypes which gives every model element, diagram, etc. a meaning. This meaning corresponds to the definition within the modeling Ontologies.

For example, a BDD which is used as product tree (product breakdown) diagram has the stereotype <<ProductTreeDiagram>>, an IBD which represents the electrical view of a block has the stereotype <<electrical>>, or a package which contains the behavioral aspect of the system has the stereotype <<Behavior Aspect>>.

Those stereotypes are introduced along the way.All stereotypes are defined in the <<profile>> SE2Profile.

12.2 Naming Conventions

12.2.1 Naming of Diagrams

Although each diagram has a fully qualified name within the model and is therefore unique, naming conventions make it easier for the modeler to navigate and understand the context of a diagram.

The header of a SysML diagram consists of four parts: **<diagramKind> [modelElementType] <modelElementName> [diagramName]**. The first 3 elements are set automatically by the modeling tool. Only the diagram name is set individually by the model builder.

- <diagramKind> is the type of the diagram
- [modelElementType] is the type of the model element that the diagram represents
- <modelElementName> refers to the element which is represented by the diagram
- [diagramName] describes what can be seen in the diagram; should be unique and say something about its context.
- If necessary, modelElementName is followed (after underscore) by more information, what the diagram contains:
    - postfix modelElementName of BDDs and IBDs with containing package name, e.g. bdd [Structure Aspect] APE_Structure [APE_ProductTree]
    - postfix modelElementName with specific view that are modeled e.g. ibd [Structure] **ibd** [System] APE [APE_Electrical], **ibd** [System] APE [APE_Mechanical]
- Use diagram info with timestamps and who modified last the diagram

The following postfix notations are used for the following diagram types:

Table 3. <>

Diagram Type	Meaning and Stereotype	Postfix naming convention
BDD	Product tree, <> Shows the product breakdown for a system element.	_ProductTree
PKG	Content, <> Shows the package structure within another package for easier navigation	_Content
IBD	Engineering Specific View <>, <>, <>, <>, <>	_Eelectrical, _Optical, _Mechanical, _Information, _Thermal

Diagram Type	Meaning and Stereotype	Postfix naming convention
PAR	Define system independent element, like quantities, units, stereotypes.	_Definition
BDD, CLASS	Define system independent element, like quantities, units, stereotypes.	_Definition

Table 4. Diagram Types

Diagram Type	Meaning and stereotype	Postfix naming convention
BDD	Product Tree, <<ProductTreeDiagram>> Shows the product breakdown for a system element.	_ProductTree
PKG	Content, <<ContentDiagram>> Shows the package structure within another package for easier navigation	_Content
IBD	Engineering Specific View <<electrical>>, <<optical>>, <<mechanical>>, <<information>>, <<thermal>>	_Electrical, _Optical, _Mechanical, _Information, _Thermal
PAR	<<element constraint>> Constraining a system element.	_Constraint
BDD, CLASS	Defining system independent element, like quantities, units, stereotypes.	_Definition

- _ProductTree … BDD
- _Content … PKG
- _Electrical, _Optical, _Mechanical, _Information … IBD
- _Constraint … PAR
- _Definition … BDD, CLASS

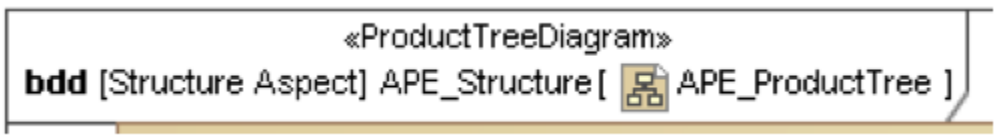


Figure 20. image1

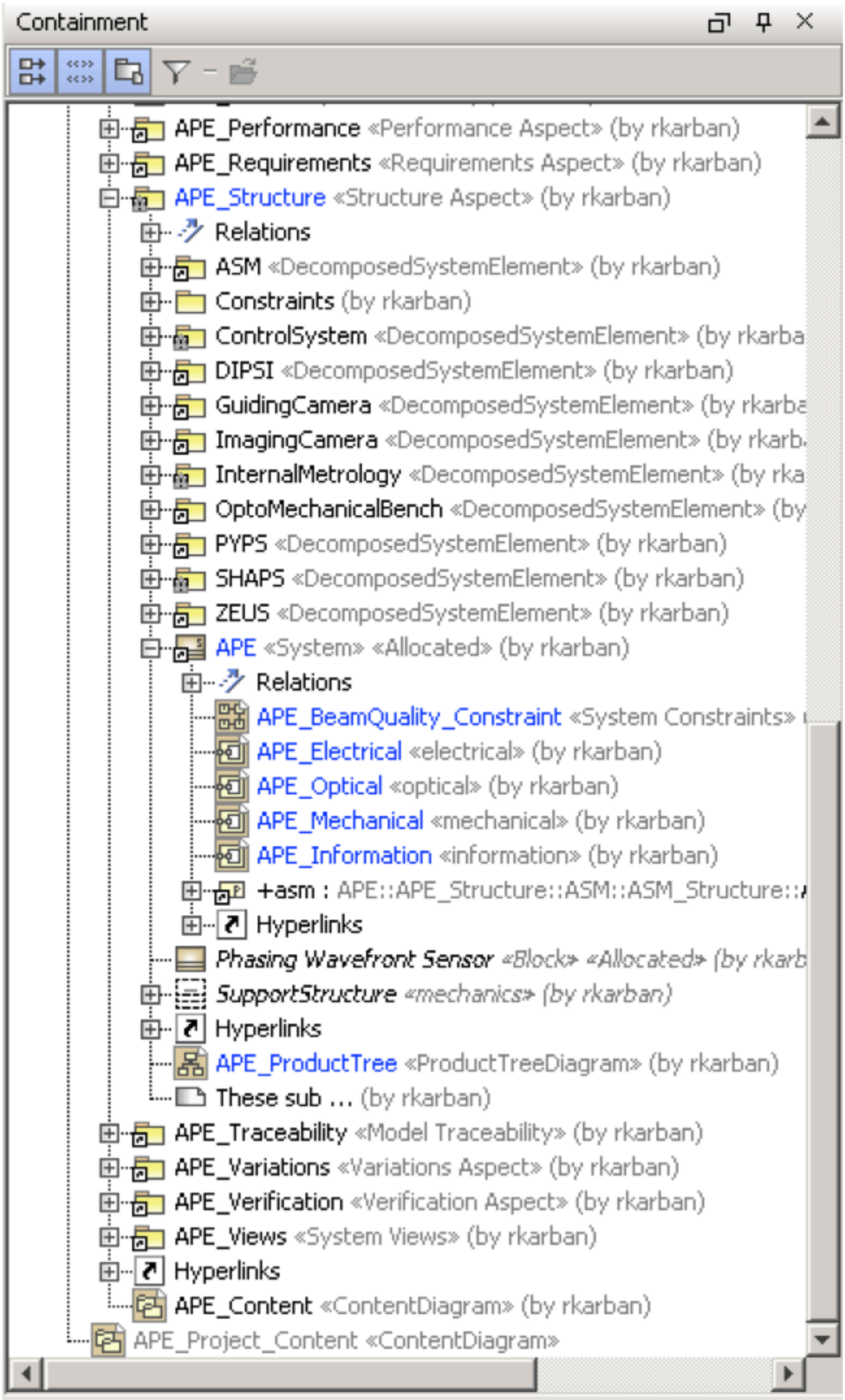

Containment
APE_Performance «Performance Aspect» (by rkarban)
APE_Requirements «Requirements Aspect» (by rkarban)
APE_Structure «Structure Aspect» (by rkarban)
Relations
ASM «DecomposedSystemElement» (by rkarban)
Constraints (by rkarban)
ControlSystem «DecomposedSystemElement» (by rkarba
DIPSI «DecomposedSystemElement» (by rkarban)
GuidingCamera «DecomposedSystemElement» (by rkarba
ImagingCamera «DecomposedSystemElement» (by rkarb
InternalMetrology «DecomposedSystemElement» (by rka
OptoMechanicalBench «DecomposedSystemElement» (by
PYPS «DecomposedSystemElement» (by rkarban)
SHAPS «DecomposedSystemElement» (by rkarban)
ZEUS «DecomposedSystemElement» (by rkarban)
APE «System» «Allocated» (by rkarban)
Relations
APE_BeamQuality_Constraint «System Constraints»
APE_Electrical «electrical» (by rkarban)
APE_Optical «optical» (by rkarban)
APE_Mechanical «mechanical» (by rkarban)
APE_Information «information» (by rkarban)
+asm : APE::APE_Structure::ASM::ASM_Structure::
Hyperlinks
Phasing Wavefront Sensor «Block» «Allocated» (by rkarb
SupportStructure «mechanics» (by rkarban)
Hyperlinks
APE_ProductTree «ProductTreeDiagram» (by rkarban)
These sub ... (by rkarban)
APE_Traceability «Model Traceability» (by rkarban)
APE_Variations «Variations Aspect» (by rkarban)
APE_Verification «Verification Aspect» (by rkarban)
APE_Views «System Views» (by rkarban)
Hyperlinks
APE_Content «ContentDiagram» (by rkarban)
APE_Project_Content «ContentDiagram»

Figure 21. image2

12.2.2 Naming of modeling elements

In general, do not use SysML keywords (like connector or interface) as names in the model to avoid confusion between SysML concepts and the system model.

12.2.2.1 Role Names

Use Role Names for Part Properties only when there is more than one part of the same type; e.g. if you have a primary and a backup server.

12.2.2.2 Names of classifiers (e.g. <<Block>>, <<valueType>>), Requirements, Activities, and Packages (Definition of something):

- use expressive short names
- do not write complete sentences, only string of words
- every word starts with a capital letter to distinguish Definition from Usage
- do not use underscores but use spaces to improve readability and allow word wrapping in the tools

12.2.2.3 Names of actions, pins, ports, parameters, attributes, operations and all properties (Usage of something):

- same as for classifiers but first word starts with lower case letter

12.2.2.4 Indicate type of model element in the name:

Table 5. table 1

Type	Suffix
flow specifications	<name>_fs
standard interfaces	<name>_if
flow ports	<name>_fp
standard ports	<name>_sp
electrical interface portgroup	<name>_eif
information interface portgroup	<name>_iif
mechanical interface portgroup	<name>_mif
optical interface portgroup	<name>_oif

A <<portgroup>> is specification for an interface of a system element. It is explained in chapter 9.

12.3 Style and Layout

12.3.1 DO NOT use grid in any diagrams

They really distract[1],.

---

[1] In MagicDraw You can switch off the default under Options->Project->Symbols->Default->Diagram

12.3.2 Instead of emphasizing the diagram, emphasize the elements that are hyperlinked to diagrams.

In systems engineering culture one expects to drill down from the 0-th level. In general, it is recommended to emphasize the relationship between system elements rather than the physical packaging, or the diagram types and icons. To make that work you MUST hyperlink every single element to something else (except for trivial leafs).

In general, navigation should happen through the diagrams, and not through the containment tree. Having that in mind, the diagram has a lot more importance, in particular hyper linking diagram elements to other diagrams.

12.3.3 Every time you place a SysML comment consider what editorial stereotype might bring it to life.

<<Rationale>>, <<Problem>>, <<Issue>>, <<ERROR>>, <<parsed>>. The above stereotypes use tagged values for metadata (there is an author:String tag). All mentioned stereotypes can be applied to SysML comments.

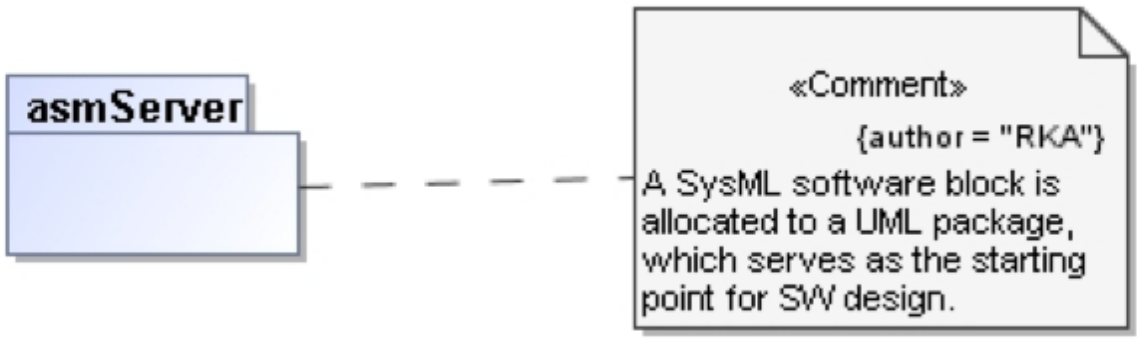


Figure 22. image1

12.3.4 "Definition" BDD diagrams for a context are overrated. Focus on IBDs.

- Just use a product tree for each Block showing its attributes, operations, and nearest related elements[1].
- Reflect the systems engineering process with your system sub packages. And please always hyperlink every single package to a package diagram.

---

[1] You can drag an IBD or BDD focus diagram for a Block onto parts typed by that Block in IBDs to "open the part up" into the Block that types it.

Place a matching BDD "product tree diagram" icon on the IBD of a Block

12.3.5 Span the Whole across its parts

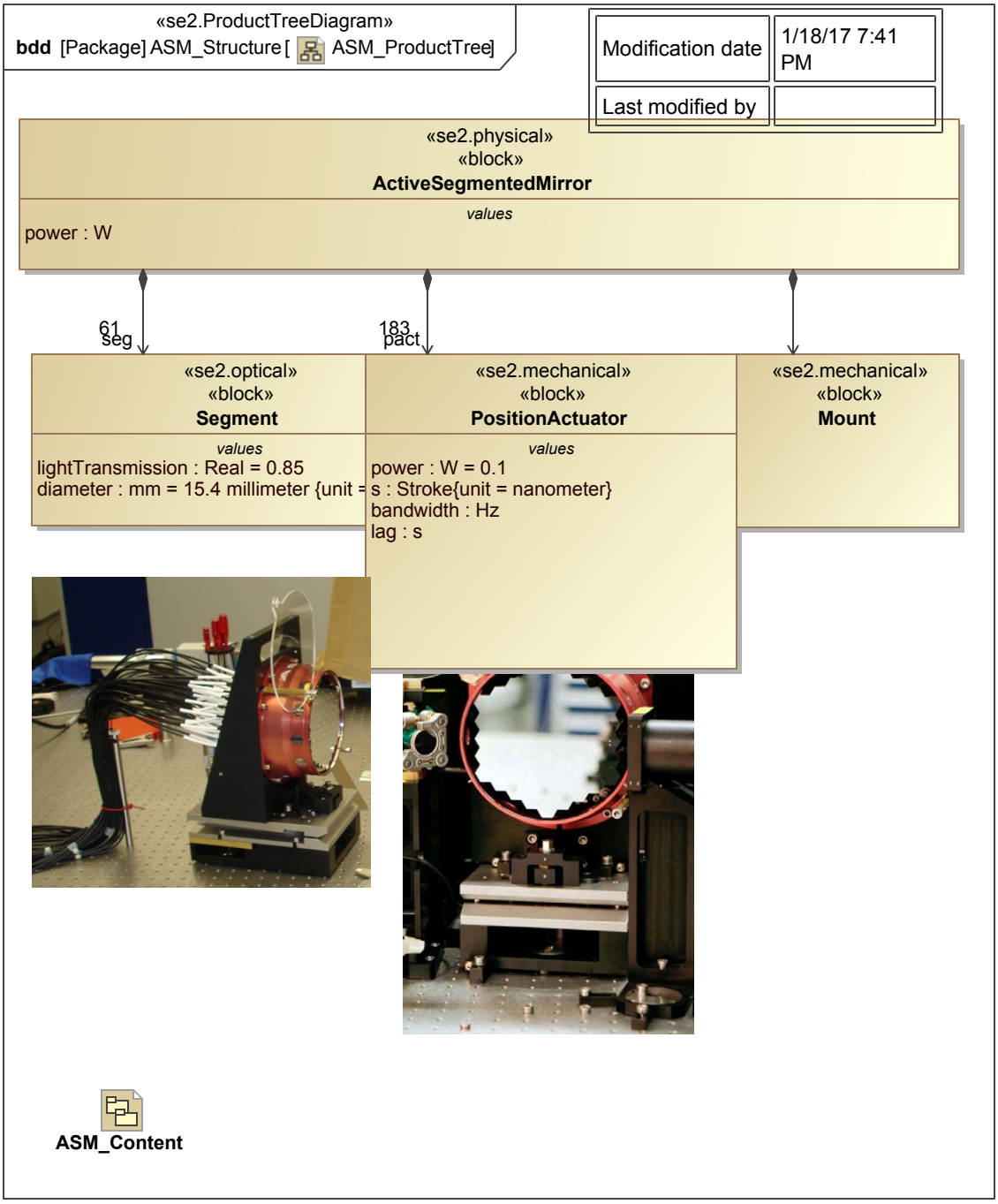


Figure 23. ASM_ProductTree

12.4 Model Documentation

12.4.1 General

- Each relevant model element has to be documented, e.g. package, block, port, interface, connector, data, and property.
- For each diagram describe what it shows. Think about why you draw the diagram and for whom. Describe what is important but not directly visible: design decisions, and so on

Two types of documentation exist, documentation about the model and documentation relevant for the system being modeled.

12.4.2 Documentation about the model

This documentation contains information why something has been modeled in a particular way, or if it unclear how to model something.

Annotations regarding the modeling approach or the meta-model:

- <<Comment>>:
  - for documenting system specific modeling approaches and decisions.
  - for documenting model elements which for example are extracted from a document, to document where they come from
- <<Issue>>: for marking questions on modeling methodology, e.g. what stereotypes should be used; if an <<issue>> is solved by decision of the team, its solution becomes part of this Modeling guidelines or its stereotype is changed to <<Comment>>
- <<ERROR>>: for marking SysML tool problems; use red borders with black text on white (not red background) for error comments \
- <<parsed>> : a model element has been automatically parsed from a text document

12.4.3 Documenting the System being modeled

Annotation can contain information about the system elements of the system that is being modeled:

- regular notes: for documenting the content of the system you model, to enhance understanding of the system;
- <<Problem>>: for marking a potential problem in the system development e.g. requirements cannot be met or interface conflicts
- <<Rationale>>: for justifying any decision during the development, e.g. <<derive>> of a requirement or decision on design alternatives

[1]

This allows better use of the documentation for generation of printed documents.

---

[1] IMPORTANT: In MagicDraw the specification of every model element has a Documentation category where plain text or HTML text can be entered. Use this category to maintain information about a system element instead of maintaining the text directly in a SysML Note. When a Note is attached to a symbol, the related documentation can be displayed in the Note automatically (select from the Note's context specific menu 'Text Display Mode' -> 'Show Documentation'). Other modeling tools provide similar concepts

13 Port and Flow Basics

Two types of Ports exist: • Standard Port (aka Service Port) • Flow Port

13.1 Standard Ports

A standard port is equivalent to a service port where different kinds of services are offered with defined operations and parameters, or signals. It originates from IT/SW usage. The interfaces are defined in a BDD. Standard ports can be perceived like a service port, with provided/required interfaces (services with operations), like defined in UML 2. They correspond to some programming model, like IDL or a class interface. They describe the logical level of communication. They are mainly used for command and control systems, and/or client/server situations. The interfaces (lollipops) specify the services that they provide/require. This clearly identifies the relationship between the parts/blocks as active and resulting in control, rather than simple data transfer. Typing directly an interface gives you only a limited form of service orientation. It is useful as long as: you only need to use a block/class as a contract with ports, not specify the implementation (you can however delegate from a port typed by an Interface to an internal part) Standard ports can be typed by a block, an interface, a data or value type. If a standard port realizes an interface, it must be typed by a block, which realizes the interfaces. This offers more flexibility than directly typing with an interface because you can transparently add another interface. An interface can have a State Machine Protocol associated with it to define the method call protocol. Having the port typed directly by an interface is only useful if you only need the port to be typed by one interface (you can't do multiple provide/require without a Block/Class type). To promote reuse of your blocks you should: • Define the ports and their required/provided interfaces in a BDD first, FOR EACH AND EVERY BLOCK, completely independent of any reuse context. That is a good place to document the block (alone) with comments, as well as document any interfaces specific to that block. Show operations/attributes etc. for that block. For a given higher level system using the defined blocks as parts with ports, "plan" your connections first in a matching BDD, ensuring that the ports to be connected do provide and require the correct interfaces. Use "display paths" to ensure that each interface is required and provided. DO NOT use associations yet to create the compositions for the top context block.

13.2 Flow Ports

A Flow Port specifies the input and output items that may flow between a block and its environment. Flow ports are interaction points through which data, material or energy (like current, oil, water, liquid) "CAN" enter or leave the owning block. Material, information, signals, even rats, can concurrently and bi-directionally flow between Flow Ports. Flow ports model a continuous flow of an item. It cannot only be of physical characteristic but also data, in case one wants to model a continuous data flow like in high performance computing. Input and output of data: a block performing a calculation may have an input and output flow port representing the input given to the calculation and the output results. Service port with getters and setters do not represent input and output - they represent provided and required services (operations). Examples are data processed with FPGAs or image processing pipelines where recipes to reduce images are used A flow port is NOT a property of a block; therefore it cannot represent a part of a block. It always needs to be related to a part property, which uses the things that flow over the port. Distinction is made between atomic flow port and non-atomic flow ports: Atomic flow ports relay a single usage of a block, value-type, data-type or signal. Atomic flows are one-way only where some item flows. Example: a pump may have an “in” flow port where it pumps the water into the pump and “out” flow port where the pump ejects the water out. A non-atomic flow port relays items of several types as specified by a flow specification. Flow ports have a direction: in, out and in-out. A flow specification consists of flow properties, which are typed. Flow properties also have a direction: in, out and in-out. The direction of the flow port and its specification must be consistent. Examples are serial lines, USB, CAN-bus, Ethernet. The distinction is made according to the flow port's type. A flow specification with a single flow property is equivalent to an atomic flow port. The atomic flow port is a simple short-hand. It does not mean that the type of the atomic flow port flows over the port. A non-atomic flow port can be conjugated. If it is, then all directions of the flow properties specified by the flow specification that types the flow port are relayed in the opposite direction (i.e., in flow property is treated as an out flow property by the flow port and vice-versa). If a structured flow port is connected to another port and the ports are on the same level, one port must be conjugated where the in become out and the out become in, like the famous 2 and 3 pins of a serial line - you need a null modem. Atomic flow ports can by definition never have a flow specification and therefore are never conjugated. If a structured flow port of a block is used in an IBD (delegation) and connected to a port of a block property the direction must be the same and not conjugated. The item flow gives the exact detail of WHAT is flowing between two ports. For example the port defines that liquid flows, whereas the item flow specifies that oil flows. Like that you can reuse the same block and ports and connect them with different connectors depending on the context. An item flow is itself again a normal block or value type with all its features like properties, operations and constraints.

13.3 Data Flows

Data structures are created on a BDD using value types to both create message hierarchies and to then type the flow port on the IBD. Usually, this takes the form of an inheritance hierarchy with the highest level Value Type in the hierarchy typing the port. Modeling in

this way means you can easily add new messages and you do not have to change any flow specification when a new message is added. This is particularly useful when modeling a system at the level of neediness and where the item flows model the information exchanges. These ports can later be allocated to the communications ports typed as Ethernet, RS-232, etc.

13.3.1 Sensor/actuator data

In monitor and control the system, we need to look at additional characteristics of the system, for example temperature, flow rate and so on. The systems engineer can then type this information as SI Value Types to communicate with both the software and hardware engineers, also specifying the level of precision that is needed for the telemetry used to measure the system. The telemetry can then be specified as analog, digital, pulse counters and so on, and the appropriate telemetry can be acquired that meets the job.

13.3.2 Input and output of events

An occurrence can be transmitted from an (out) flow port to another (in) flow port of another block. This is completely equivalent description to standard port with an interface having an event reception where one port has this interface as required and the other has this interface as provided. However using flow ports for this purpose instead of standard ports might be more intuitive since it is not relying on the complexity of standard ports, required\provided interfaces and event receptions. Ports can be connected directly with a Connector if you do not need to specify constraints, attributes, etc. You still can use a stereotype qualify the type of connector, like mechanical, optical, etc. If you need to describe more properties of the connection use another block to define them. For example a cluster of computers connected to the same Ethernet subnet, devices connected to a CAN-bus or a sewer where several sinks are connected require to model the medium as block where you connect. One also needs to be aware for example, of whether the different parts will be reused. Take the valve for example; typing the ports on the valve by residue or H2O means that only one type can flow through them. Creating an inheritance hierarchy of fluid, sub-typed by H2O and residue, and typing the ports by fluid means the either H2O or residue can flow through them.

13.4 Created related documents chapter in Recycle Bin

A port can realize different interfaces. It combines several interfaces; e.g. If the port is Ethernet it represents the sum of all its protocols. E.g. it realizes UDP, TCP, etc.

14 Combining ports and flows to represent interfaces

14.1 Modeling a structural interface like a socket, screw, hole etc.

Use a standard port typed by a block that specifies the structure. We do not need yet necessarily another part property "inside" the system. The structural information is contained in the interface description. The practical advantage is that we do not need to open the box to know the structural properties.

14.2 Modeling standards like RS232, CAN bus etc

You have several parts in the model: • The (mechanical) interface type, like DB-25, DB-9 • The electrical signals, like Receive(tr), Transmit(tx) and Ground(gr) • The hand-shaking, error detection protocol, etc. • The cable • The data which flows Depending on the required level of detail there exist different options:

14.2.1 Model "everything"

• Model the mechanical connector with a block, named DB-25, DB-9 • Model the electrical signals with a flow specification, where each signal is represented by a flow property. Where the meaning of the flow specification is in the sense that anything that supports this interface can flow through this port. • Create a flow port as part of your DB-25 block. This actually represents the assignment of the mechanics to signals, which may differ from one supplier to another. You need to create an application specific sub-type of your DB-25 to do the assignment. Or, you create another special interface block which owns a standard port typed by DB-25 and a flow port typed by RS-232. In any case, stereotype the interface block as <>. • The actual interface driver is a component within the receiving block. If it was a computer, for example, there would be an RS-232 port on the computer which would connect to an internal part that is an RS-232 interface driver with its respective ports on it. Its ports may include the individual pins or still be abstracted as a single port. You specify what flows by the physical interface specification. Its behavior, e.g. the handshaking, is modeled by a state machine attached to the part property which uses the flow port. • In the simplest version is using a connector. • The data (or any other flowing entity) is represented by item flows. The item flows are of type value.

14.2.2 Modeling ONLY A FLOW of entities.

• Represent the data with flow specifications (if you have more than one property) or a value type (semantically equivalent to a flow spec with one flow property) and type the flow port with the flow spec. For example, if you want to define the packages which flow over a CANbus See Figure 49 Catalog Definition of TCCD heads, Figure 24 Electrical IBD of the Observatory Context

14.3 How do I model a cable?

The cable can be modeled in various ways, depending on the detail you need: • a simple connector • a block if you are interested in the properties of the cable and you want to re-use it. N.B. That the cable can also have ports to represent plugs and sockets. • an association block in this case you can type the connector with the association block and you get a very compact representation in the IBD. NOTE: it must be an association between types, e.g. the types of the ports you want to connect. The re-usable element consists of the two types and their association - NOT the block itself. See Figure 51 Models of Cables.

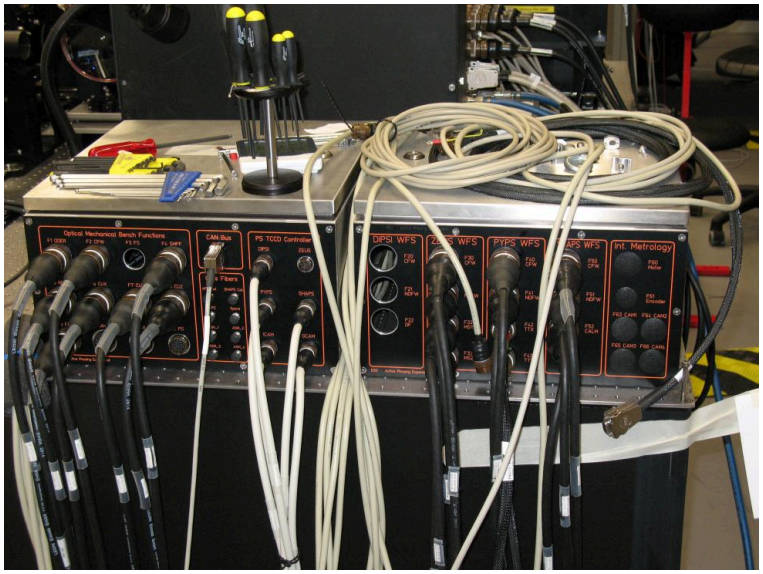
14.4 Combining physical connector type and flow

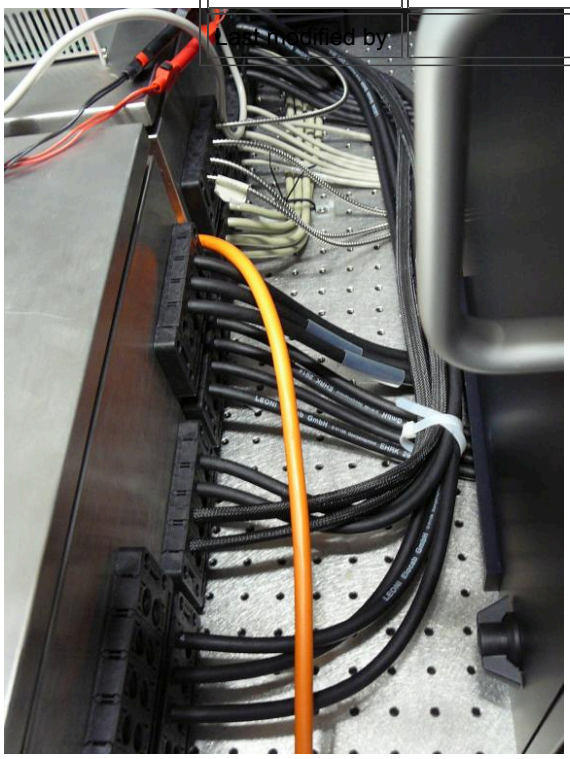
In APE there are a so-called Junction Boxes (JB1 and JB2) which connects the Control System Electronics to the Field Electronics. The Junction box has two interface plates. One where the cables coming from the control system are connected (the “external” interface plate) and one where the cables to the actuators on the opto-mechanical bench are connected (the internal interface plate). Each interface plate has a group of plugs for each subsystem located on the bench, e.g. ZEUS, DIPSI, IM. The two interface plates are modeled as nested ports and each group belonging to a subsystem is again modeled as nested port. In this case we have a double nested port as shown in Figure 54. The Interfaces are represented by standard ports, typed by a block which is stereotyped <>. It serves as the specification of the port and groups together several interfaces. The Block can itself have ports and therefore several levels of nesting can be achieved. <> itself is generic. There are different stereotypes to be specific about the kind of interface: • <> … electrical interfaces • <> … optical interfaces • <>.,. mechanical interfaces • <> … protocol interfaces • <> … information interfaces The aim of grouping the interfaces is to define physical connector and flow at border of part and hide internals of a block. The main benefit is that a single type is needed to define physical connector and flow and that the interface type can be re-used. Traditionally, several unrelated ports are needed. Different flows over a connector (e.g. pin assignment) can only be assigned via specialization of the block which types the port The Junction Box has an internal wiring which connects the interface plates plugs, which is shown in

Figure 55. The connection between Control System, Junction Box, and Subsystem are shown in Figure 40. APE uses special technical CCD heads which provide and require different interfaces. The head needs to be cooled; therefore there are interfaces to integrate it into a cooling circuit (coolingSupply, coolingReturn). The type TCCDCoolingConnectorSupply is a block, and owns a flow port lc, typed by an <> Coolant. The type TCCDCoolingConnectorSupply inherits from block “SelfSealingFluidFemale” which is a standard catalog connector for fluids. By specializing, the specific flow port for the Coolant can be added and allows combined modeling of physical structure and flowing items (Figure 56, Figure 57). An alternative is shown for the coolingReturn interface, where a <> is used. The <> owns a flow port and a standard port for the Coolant and the physical connector. <> is used to indicate over which physical connector the Coolant flows.

«se2.ProductTreeDiagram»
bdd [Package] OptoMechanicalBench_Interfaces[OptoMechnicalbench_Interfaces_ProductTree]

Modification date	1/18/17 7:41 PM
Last modified by	





«se2.physical»
«block»
JunctionBox1

«block»
«se2.physical»
JunctionBox2
: JunctionBox2External_eif
: JunctionBox2Internal_eif

«interfaceBlock»
JunctionBox2External_eif
: JB2Zeus_eif
: JB2IM_eif
cfw ndfw msy msz : Souriau851

«interfaceBlock»
JunctionBox2Internal_eif

«interfaceBlock»
JB2Zeus_eif
cfw ndfw msy msz : Souriau851

motor enc
«interfaceBlock»
JB2IM_eif
proxy ports
cam1
cam2
cam3
cam4 : IM_Cam4_fs
cam1 «proxy» cam2 «proxy» cam3 «proxy» cam4 : IM_Cam4_fs «proxy»

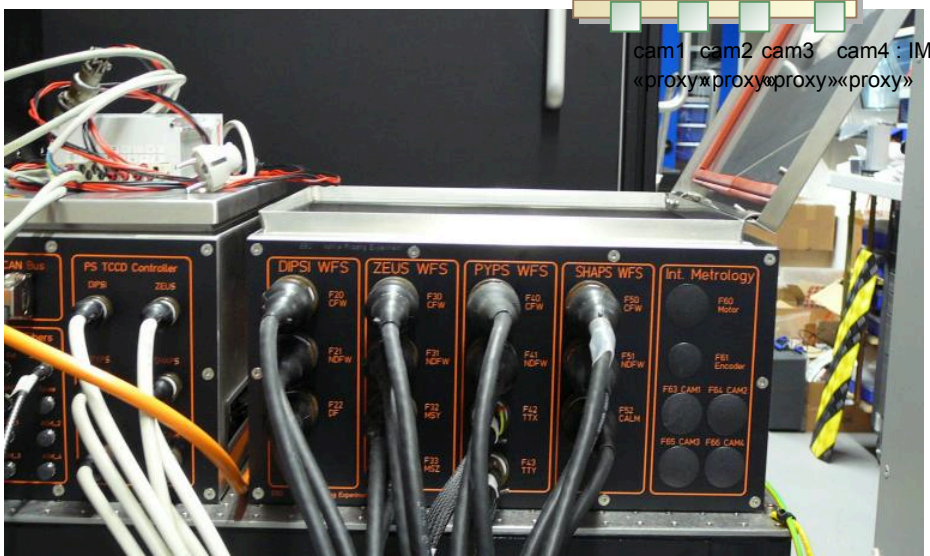


Figure 24. OptoMechnicalbench_Interfaces_ProductTree

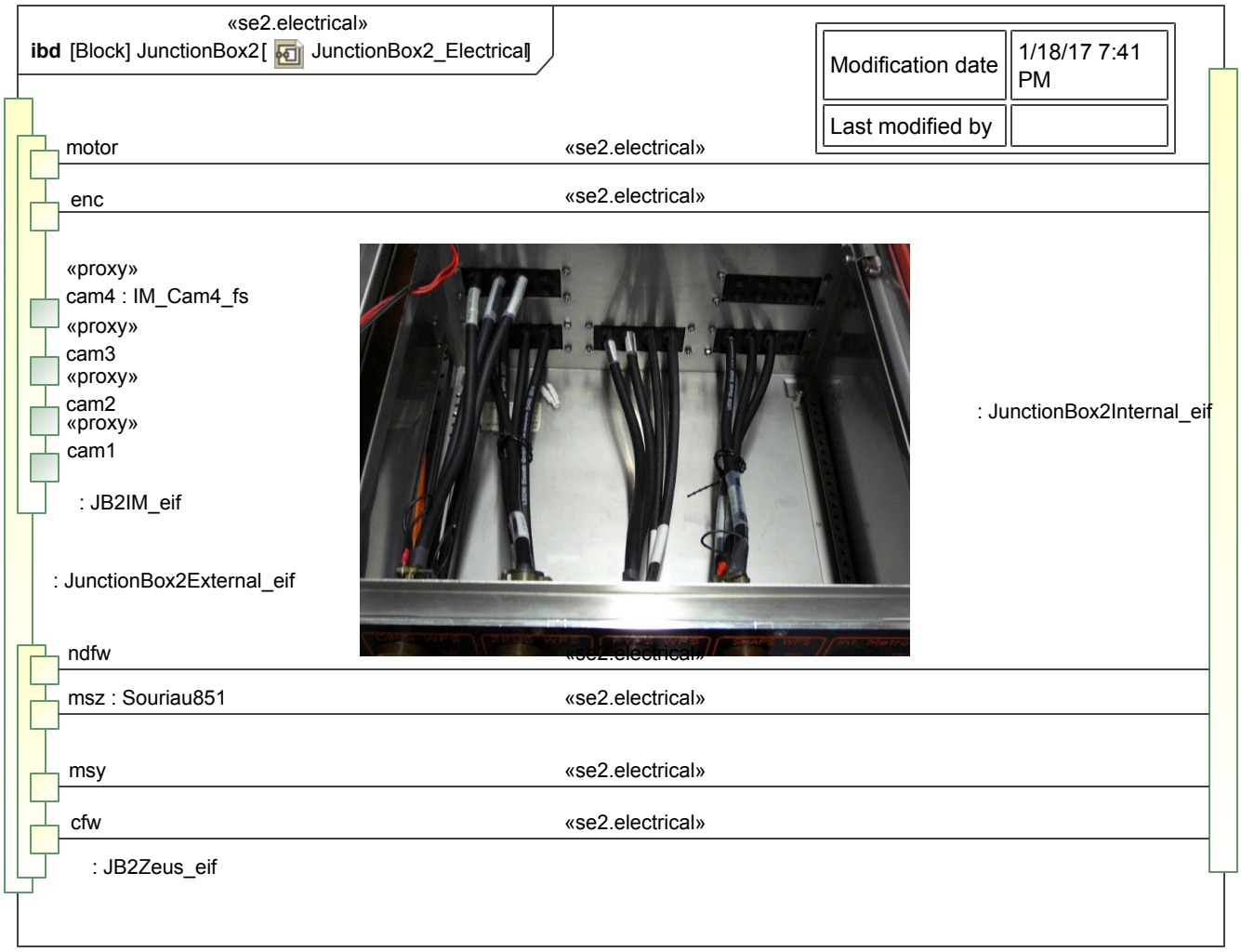


Figure 25. JunctionBox2_Electrical

Figure 26. Interfaces of TCCD head

Figure 27. Definition of TCCD Interface

15 Layered Command and Data Interfaces

15.1 Example

The Local Control System of the Active Segmented Mirror provides and requires command and data interfaces which run over different LAN interfaces, and use different protocols.

15.2 Context

Different information interfaces can use different physical interfaces, e.g. data on different LAN port, using different protocols, e.g. CORBA. Allocation of structural elements across multiple abstraction layers is needed.

15.3 Problem

Traditional SysML modeling way would use several unrelated flow and standard ports which make re-use more difficult and creates cluttered structures.

15.4 Solution

Define Command and Data interfaces in one Block which enables grouping of interfaces which have a high coupling, re-use, extendibility, and consistency. • Aggregated information, electrical, and protocol specifications for ports • <> information ports to physical ports and protocol ports • Profile with stereotypes for interfaces types • Special port types for better readability (cluttered diagram by stereotypes) Figure 58 shows the data and command interface. It is specified by an <> <> which <> and <> standard UML interfaces. The Data interface is modeled with a flow port which is typed by a <>, representing the data which flows. The required command interface and the data interface run over a TCP/IP protocol and a Ethernet 100 base T network. The provided command interface runs over a proprietary protocol which is based on TCP/IP, and another LAN port. Add ports for protocols or use properties. Defining a property of the logical interface defines the protocol for the whole interface. This is not what we intend here. The proper way is to add protocol ports for each protocol (a DDS port, a CORBA port). The logical ports are then allocated to both, the protocol ports AND the physical ports. Multiple allocation is possible. Which protocol is used on which port, can be derived from the two allocations. The implementer of the control system and/or control software has to take care of the proper allocation. Since the diagram gets cluttered, not all allocations should appear in the diagram but a dependency matrix should be created. Why not define the protocol stack by tags, like in WSDL? Before a value can be assigned to a tag the port would have to stereotyped. The properties of the stereotype become the tags. There is a stereotype <>. However: • you need more clicks • you cannot create a dependency matrix which shows the allocation • you could not show it in the diagram because there would be too many stereotypes. Would be useful only in a document generator. • which port should get the tag? the logical? the physical? We would lose a clear separation. Allocation constraints can be defined to constraint the supplier and client of the allocation relationship The Protocol Stack is modeled according to the ISO/OSI stack model. The allocation matrix shows how the command and data interfaces are allocated to protocols and LAN ports. The <> block AsmServer provides/requires the information interface and the IBD shows how it is connected. It could be different software blocks for each information interface.

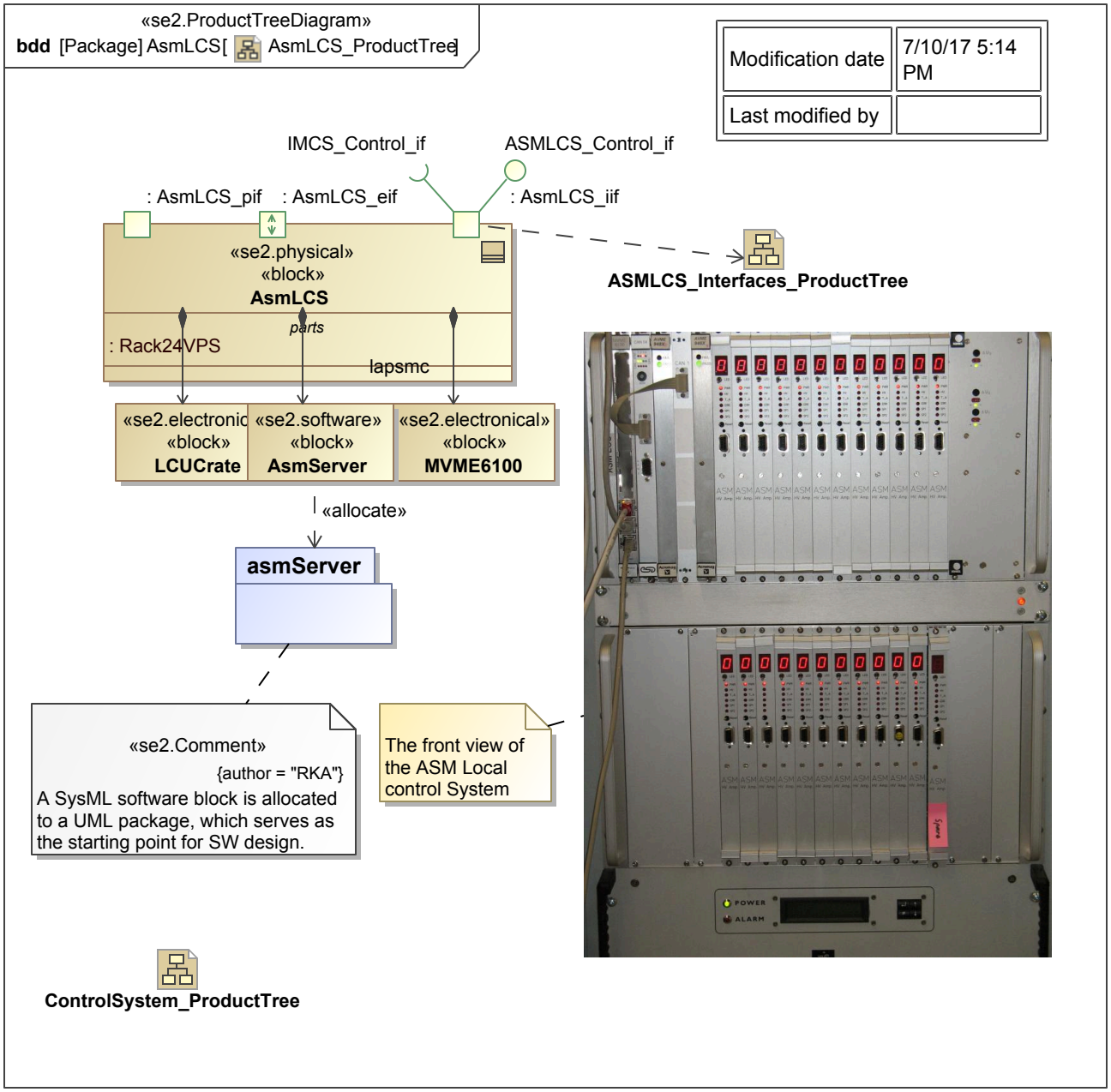


Figure 28. AsmLCS_ProductTree

Table 6. AsmLCSPortAllocationMatrix

		ccs	fpCtrlLan	fpImcsLan	imData	tcp/ip	ASMLCS_Control_if
ccs							✔
fpCtrlLan							✔
fpImcsLan					✔		
imData				✔		✔	
tcp/ip					✔		
ASMLCS_Control_if		✔	✔				

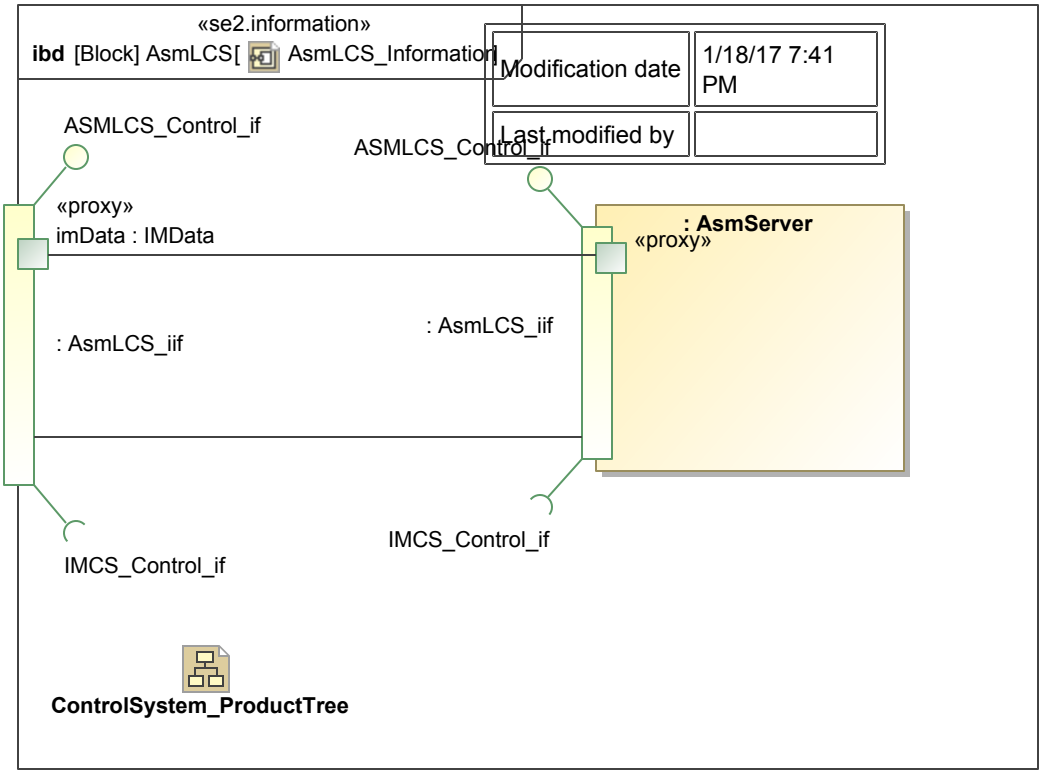


Figure 29. AsmLCS_Information

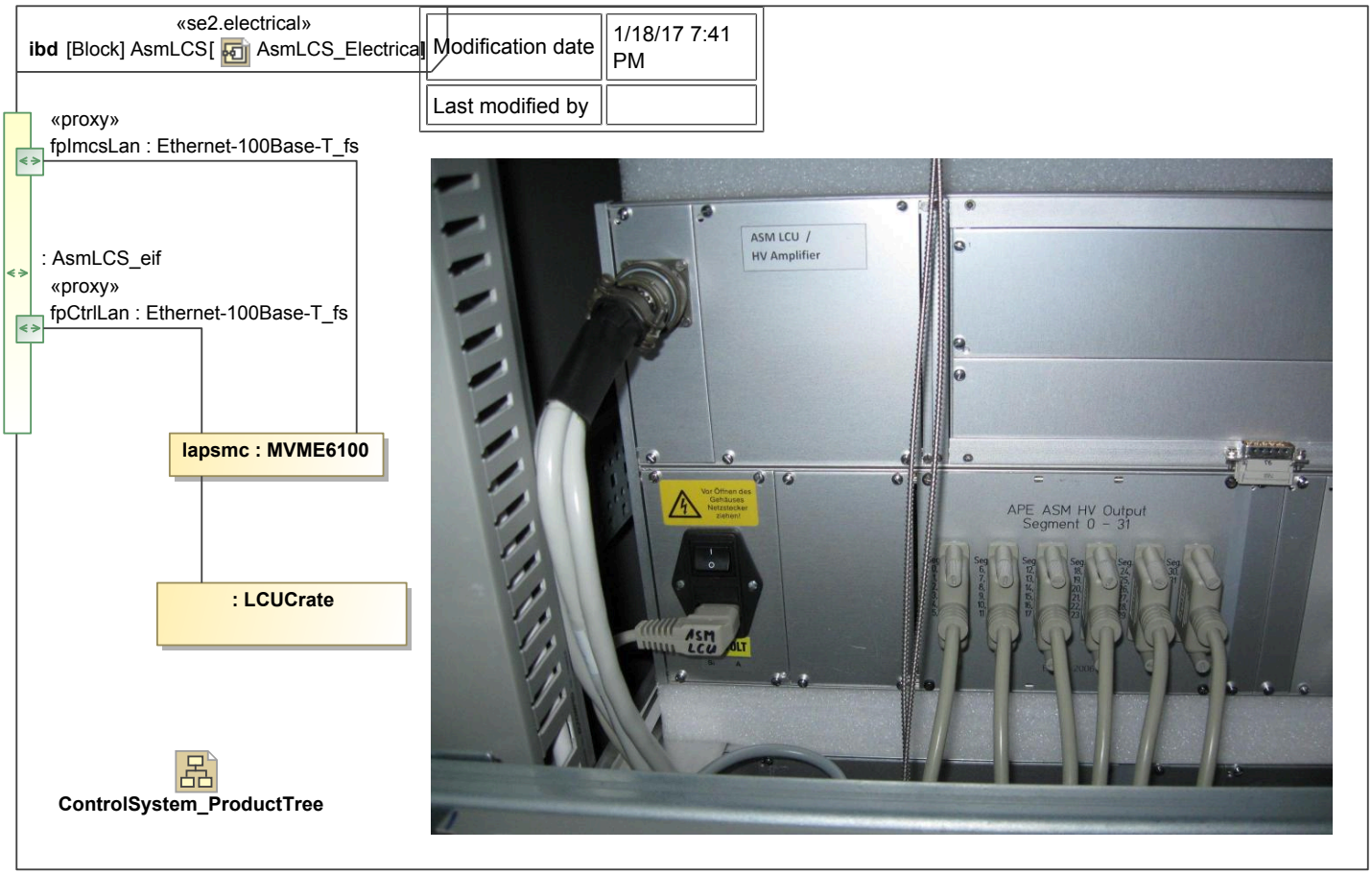


Figure 30. AsmLCS_Electrical

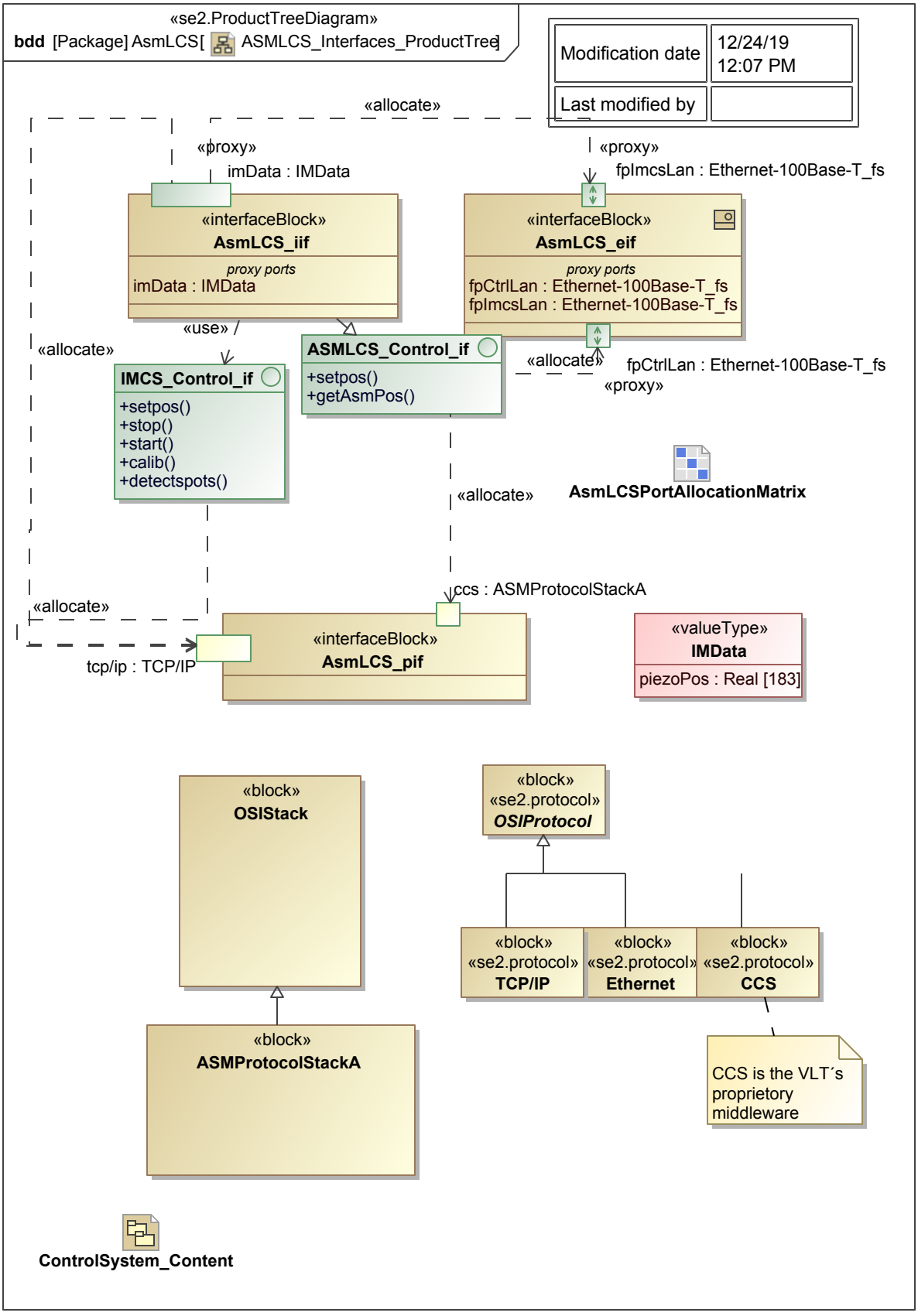


Figure 31. ASMLCS_Interfaces_ProductTree

15.5 SysML Status

There are no plans to support discipline specific interfaces types. That would be contradictory to the unified approach of SysML. It is a task for the stereotypes mechanism. Allocation is a stereotype of UML abstraction and the semantics (i.e. the exact mapping) of allocate are not defined in SysML. The mapping is to be defined. For practical reasons use a Note.

15.6 Allocations of a nested block

Allocation is a stereotype of UML abstraction and the semantics (i.e. the exact mapping) of allocate are not defined in SysML. Therefore allocation of the top most port does not necessarily mean that all nested ports are also allocated. This makes it possible to allocate d1, d2 to lan1, lan2 and the port which realizes the control commands to yet another physical port. The interface itself cannot be allocated because it might be realized by different ports.

16 Flow Properties vs. Flow Ports

16.1 Example

A video camera provides analog and digital video streams which are models as flow ports.

16.2 Context

The flows which are the visible interface of a block (the camera) are delegated to nested flows, internal to block.

16.3 Problem

When connecting the ports to the internal ports the flow port typed by the flow specification has some disadvantages. Its flow properties are not visible, therefore it is unknown how the flow properties digVideoA and digVideoB are connected to the internal ports a and b. Flow specs the diagram doesn't tell you what's inside the port on the diagram.

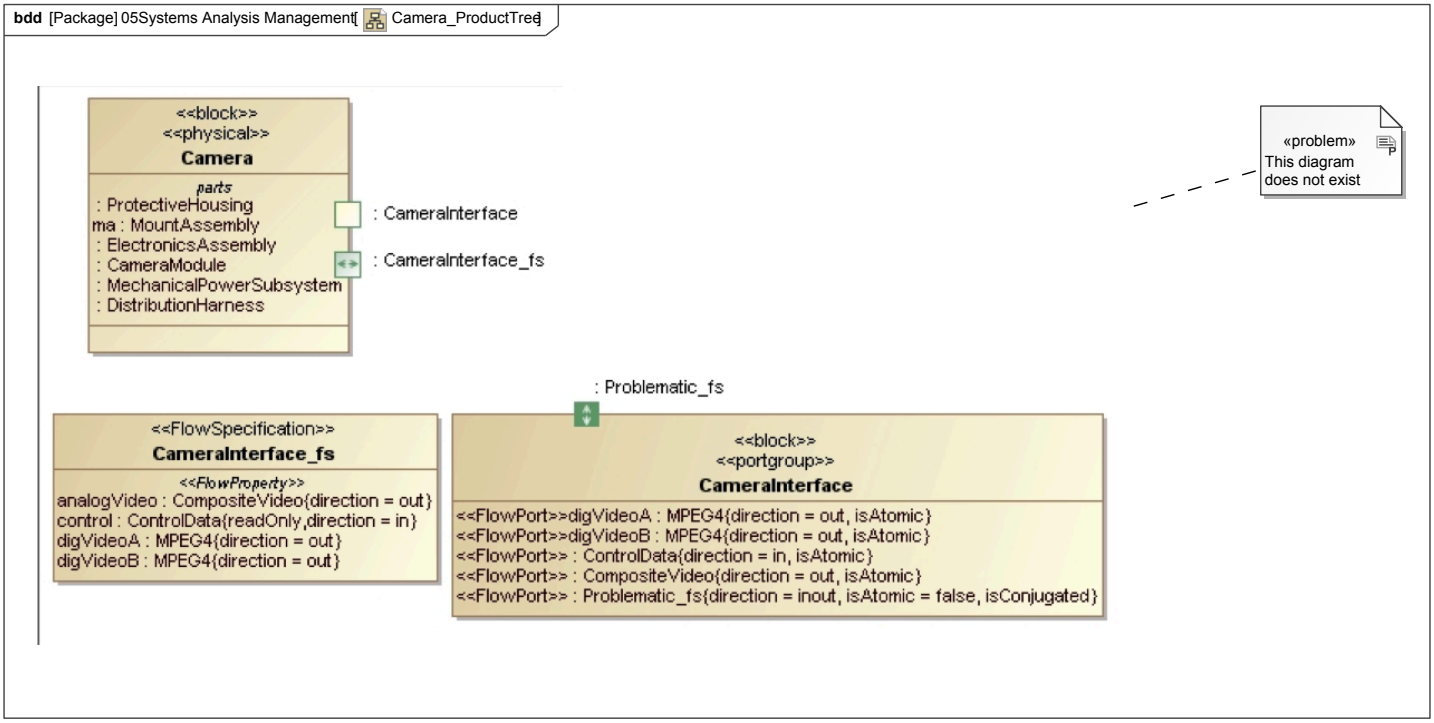


Figure 32. Camera_ProductTree

16.4 Solution

Using a nested port with nested flow ports (which correspond one to one to the flow properties of the flow spec), the individual ports can be connected. The example describes a camera block with two interfaces. One modeled with a flow port which is typed by the flow specification. Another is modeled with a <>, using nested ports.

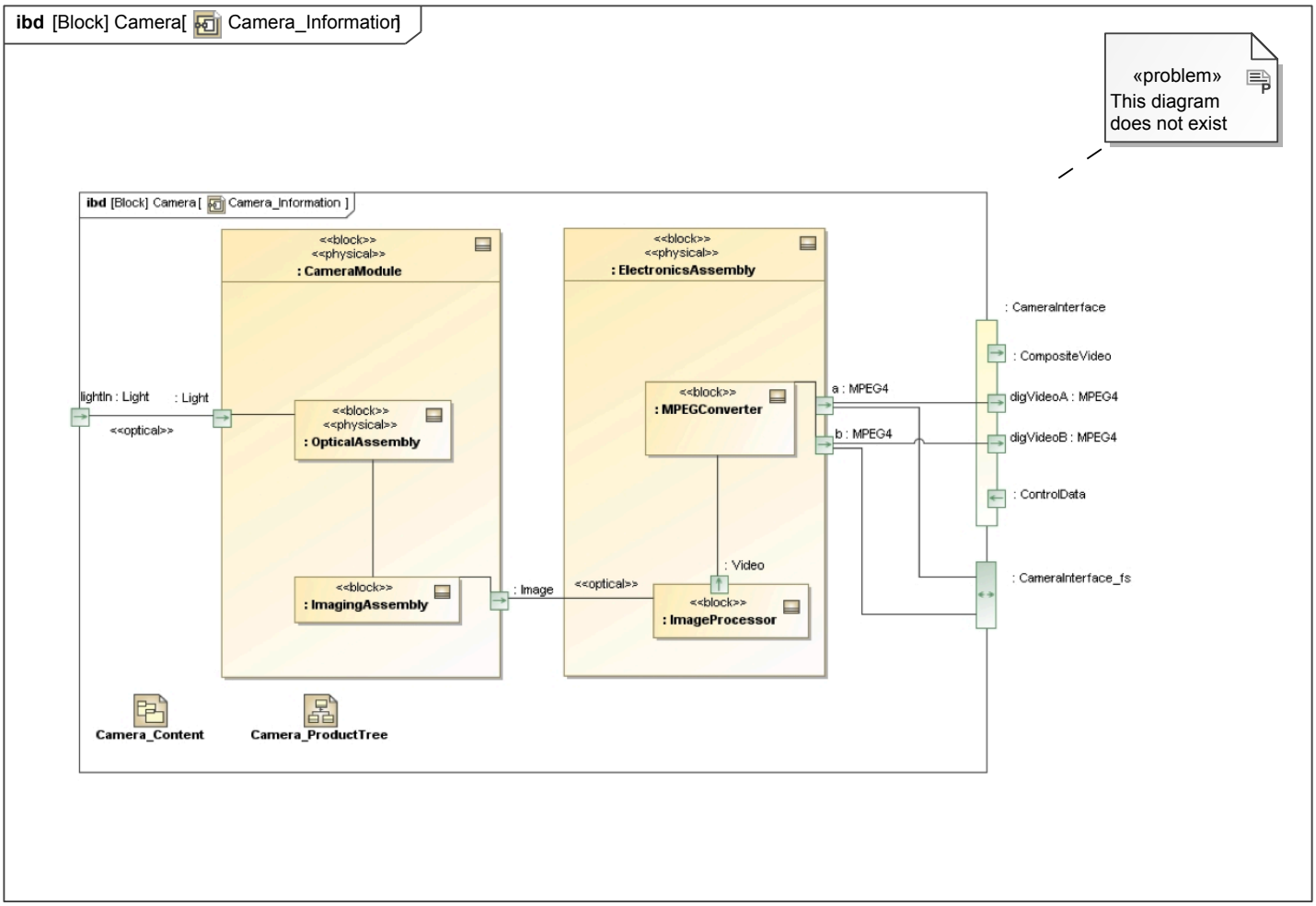


Figure 33. Camera_Information

17 Modeling Interfaces which are represented by a document (e.g. ICD - Interface Control Document)

If an interface is already specified in an ICD document it makes no sense to complete model it again, but reference it.. The documents are represented by value types, stereotyped <>. They are value types because they represent a uniform type of information. Its name is the document number or title. The connector between them is stereotyped <>. Depending on the level of detail you require you can simply use ICDs or model parts of it using ports and flows.

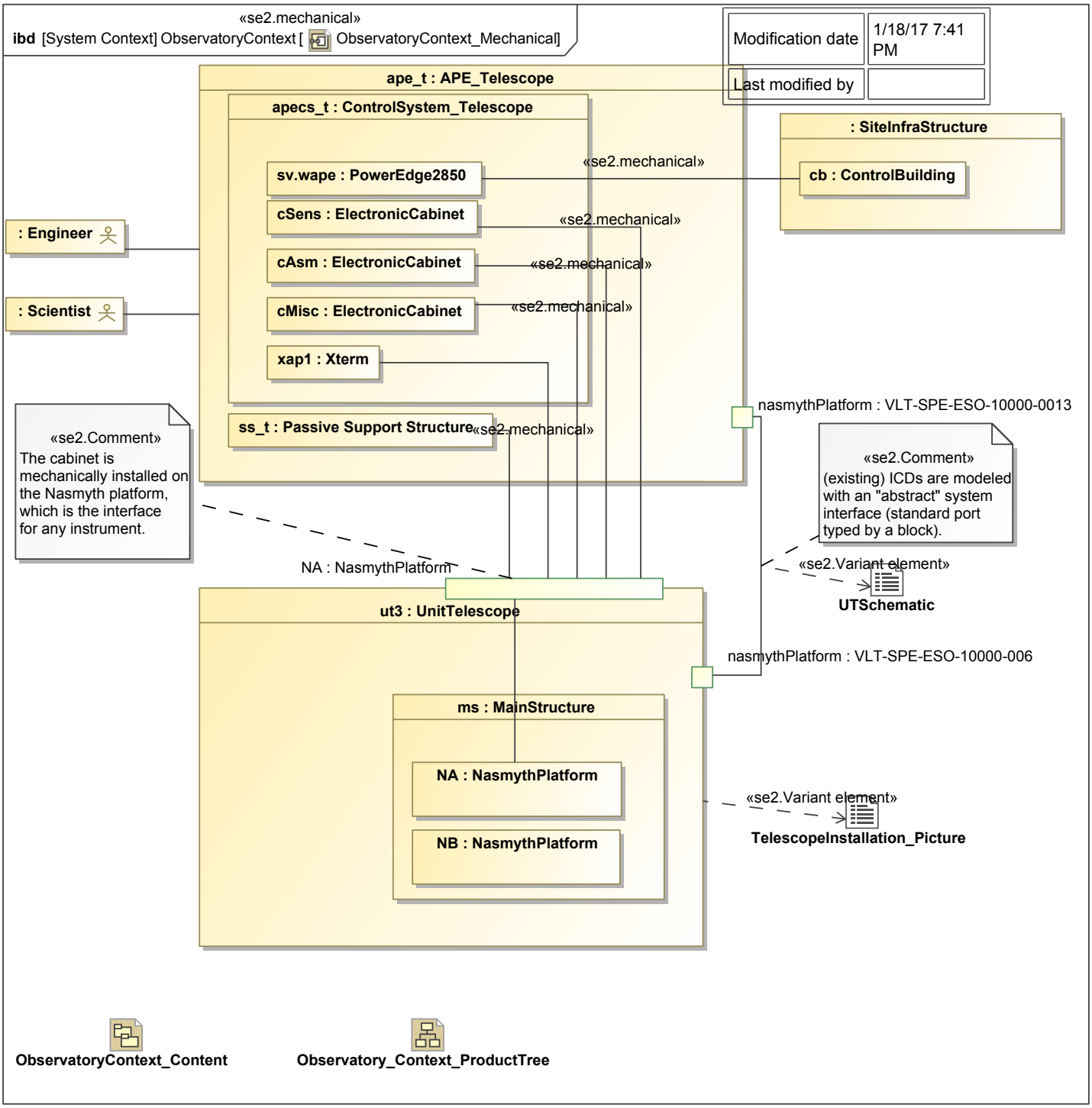


Figure 34. ObservatoryContext_Mechanical

18 Relations between Interfaces

18.1 How can I type a connector between ports?

Connectors are typed by associations, although they are often left untyped. It should be an association between PORT TYPES. Stereotypes and types are different things. A stereotype introduces a new model element by specializing an existing model element, e.g. physical connectors.

18.2 How/When do I use realize with Ports?

Realize is only used with standard interfaces, i.e. when something provides a service. As a consequence only standard ports can be used with a realize relationship. Only blocks can realize and/or use more than one interface. Therefore a port must be typed with a block before it can realize/use an interface. You can either type the port directly with the interface or go through realization. The latter is more flexible because you can later on easily add more interfaces and realize them without changing the type of the port. N.B.: Realize relationships exist also between Use Cases and Test Cases or other Use Case realizations.

19 Flows

19.1 Model that something flows in or out

Use a flow port (atomic, conjugated, simple, etc.). The flow port is typed by a flow spec or a Block/Signal/ValueType (atomic) to specify what flows. • To model that the block provides or requires some services, e.g. a software API use standard port with interfaces. • To model a combination like structure plus flow specification use a complex port: standard port typed by a block that specifies the structure and owns a flow port. The flow port is typed by a flow spec or a block/signal/valuetype (atomic) to specify what flows. See 9.2, Combining ports and flows to represent interfaces • Telescope SCP: the CEE plugs or the coolant supply/return connectors are described by the standard port typed by a block. The description of what flows is described in the flow spec. The coolant is defined by a block and the properties of the coolant (temp, pressure, glycol, etc.) as value types. This block is used either as a type of the flow property (in case we have one flow port with two flow properties) or as the type of the flow port (in case we have a flow port for each connector and therefore an atomic flow port).

19.2 Should I use direction on flow ports?

"Trust the port direction" rather than indicate like thingIn and thingOut: also DO type ports. Although throughout nearly every previous systems engineering model you will see the direction (in/out/inout_ kind of flow ports indicated in the (contrived) flow port name, this is no longer necessary. For atomic flow ports DO instead trust the flow port direction indicator.(And for complex port DO trust the indication of conjugation.) The directional information CAN be gleaned from the SysML model. You will avoid errors, can change port directions WITHOUT changing names, and your diagrams will be easier to read.

19.3 Is the flow specification describing the physical layout of a medium or the items which flow?

The flow specification defines the kinds of things that can flow through this interaction point on a part or block. If a more logical layer is allocated to a physical layer, appropriate flow specifications on both layers are valid, e.g. describing data structures on the logical layer and current on the physical layer.

19.4 Do I put the specification for an image flow on the port or as item on the connector?

The type of the port can define the image format such as Digital Video, where as the thing that flows may define the content of the image, such as ”target image“. This would indicate that any Digital Video can flow through this port, but this connector conveys a Target Image. This is only one way to characterize the port and the item flow, but there are others.

20 Overview of Interfaces modeled with Flows and Ports

A ItemFlow offers the possibility to specify concrete rates or flow properties by itemProperty. The following table (Table 2) list a number of interface elements for different interaction media, connectors, isolators, and converters and the interacting elements.

Table 7. <>

Type	Electrical	Mechanical, Hydraulic, Optical, Thermal	Human-Machine	Date
Interaction Medium	<> Item Flow = <> Current Flow defined by <> and <>, <> for cables (in connection with their connectors)	<> Item Flow = << Item >> Force, Fluid, Photons, Heat Flow defined by <> and <> <> for pipes, fibers,	<> Item Flow = << Item >>information (e.g. audio, visual, finger print, iris), mechanical force Flow defined by <> and <>	<> Item Flow = <>
Connector	<> typed by a block. e.g. DB25, RJ45	<> typed by a block e.g. Joint coupling, flange, Valve, fiber optic connectors, e.g. metallic foil bundle	<> typed by a block e.g. Display or a separate <>	<> typed by a block, realizing interfaces or <> for data flows. e.g. data I/O items
Isolator	<>, <> typed by a block e.g. RF shield insulator	<>, <> typed by a block e.g. Shock mount bearing, Seal, Shutter, air	<>,<> typed by a block e.g. cover window	n/a
Converter	<> e.g. Antenna A/D converter	<> e.g. Gear Train Piston, Reducing valve Pump, Lens group, Peltier	<> e.g. Keyboard, lever, loudspeaker, steering wheel, touch screen	<> e.g. FPGA
Protocol	<> and <> E.g. RS232, USB, CAN, Ethernet	n/a	n/a	<>, typed by a block e.g. CORBA, DDS, TCP/IP, OSIStack

21 Integration with other disciplines

This chapter describes the integration of system engineering modeling with other disciplines.

21.1 Transition to UML for software

How can a system model be used to seamlessly start software engineering? Notion • Seamless transitions from SysML <> and <> blocks to UML classes, mapping also ports and interfaces How to • <> block to package • Alternative I: <> SysML ports to UML ports and <> the same interfaces. Use interfaces for information access to map flow ports. • Alternative II: create a UML „part class“ representing the SysML block and create connectors for SysML ports to UML ports in IBD and class diagram SysML status • <> implies dependency of System to SW or vice versa. • Classes are excluded from SysML The main problem is how to trace SysML ports/interfaces to UML ports/interfaces. Interfaces are easy, they are simply realized by some classes. We present two options to do the transition: Define a "part-class" of the SW block which represents the SW in the SysML model The UML class is then referenced in the UML SW package and has associations with the other classes of the SW ¥ Pro o The ports and interfaces can be seamlessly connected from the top (control system) to the class o strong coupling of SysML and UML ¥ Con o Classes are excluded from SysML (add to SysML status) o strong coupling of SysML and UML o Flow ports do not exist in UML o depends on development process o depends if tool supports having UML and SysML at the same time available Allocate class ports to ports ¥ Pro ¥ Only the SW block appears in the SysML model. The cut between the models is done at SW block level. ¥ System service ports are connected to the SW block ports (there can be more than one software block) -> strong coupling ¥ The SW developer defines how UML elements are allocated to the SysML elements by allocating them. ¥ Con ¥ usage of allocation requires anyway a mixed language approach like in option one because allocate has to be used. ¥ If you want to show how SysML interfaces are implemented in SW you need allocate. Only those flow ports are interesting to SW, which are not physical; i.e. information flow ports (DDS like). Define one SW interface for information access. All flow items of the flow spec of the flow port can be mapped to subclasses of the abstract data class; i.e. each concrete data class has a dependency to the flow item. In general a software block shall be allocated to ONE package and there shall be no detailed SW design, like creating different blocks for interfaces, control or entity as suggested by other authors. We think the detailed design should be left to the SW developer. The developer gets a block with defined logical data and command interfaces, available (allocated) physical interfaces (like LAN ports), possibly defined protocols, and allocated functions. This allocation is the minimum step necessary. If further detail is needed a mixed language approach is needed. The logical interface at a higher abstraction level, like control system or system, shall be replicated on SW block level, to have a proper tracing. In the IBD of the next higher abstraction level the ports of the higher level are connected to the ports of the SW block(s). Here starts the cut between System and Software (Figure 77). The SysML model does not include the discipline specific models like UML. The UML model uses information from SysML, i.e. the SysML SW block with it's interfaces is the system class in the UML model. The correct way would be to transform the SysML block to the UML class. The pragmatic way is to use the same model element, i.e. the SysML SW block is the same as the UML system class. But it is not necessary a one-to-one mapping. A SysML software block could be mapped to one or more classes. Also several SysML <> blocks could be mapped to one single UML class. Important is the direction: the SysML model doesn't include the UML model, but the UML model uses information from the SysML model. There should be no UML elements in the SysML model. In theory a generator/mapping creates elements in the UML model from the SysML model. In practice we have no big gap between both models, since we could stay in the same model.

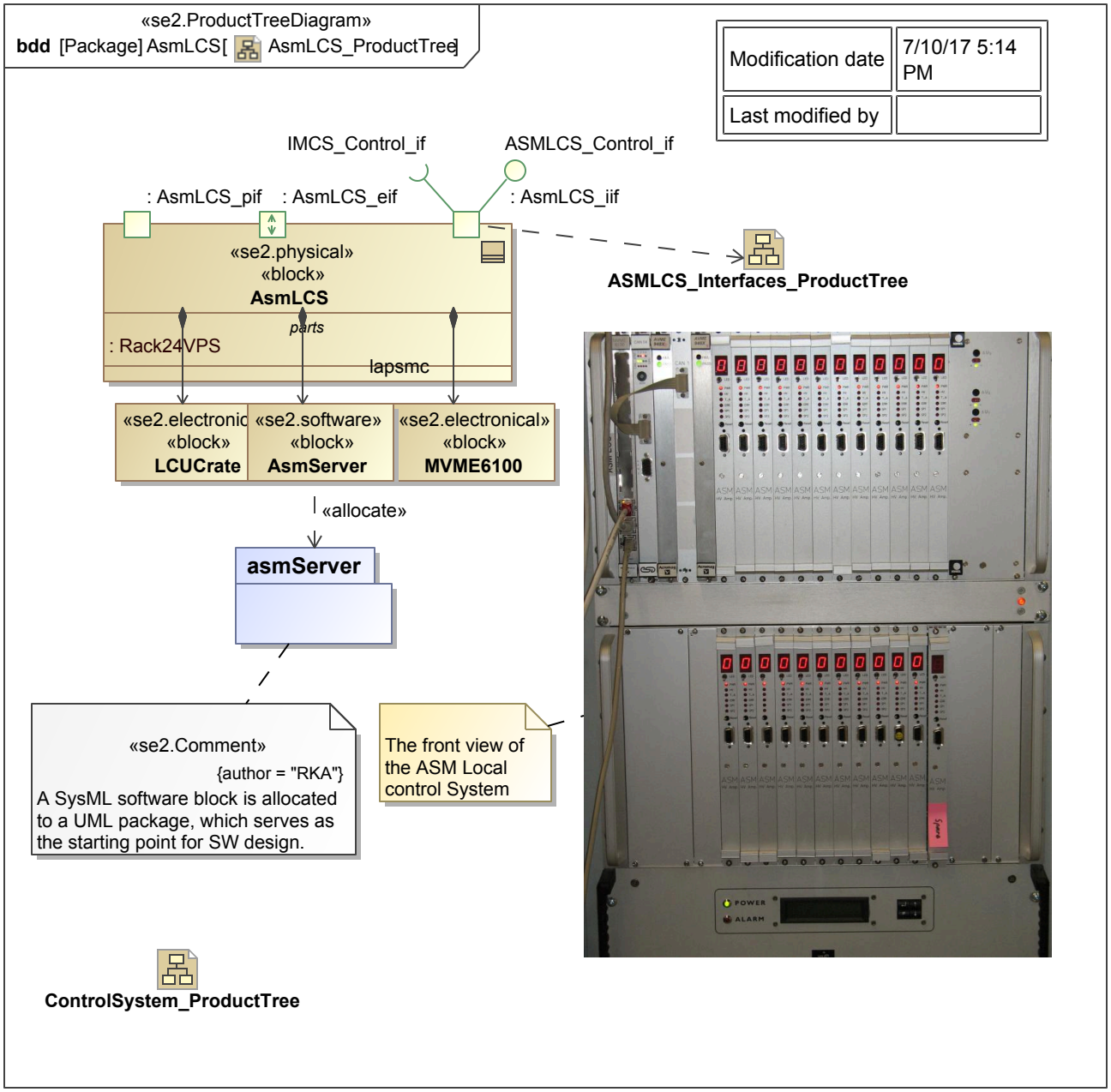


Figure 35. AsmLCS_ProductTree

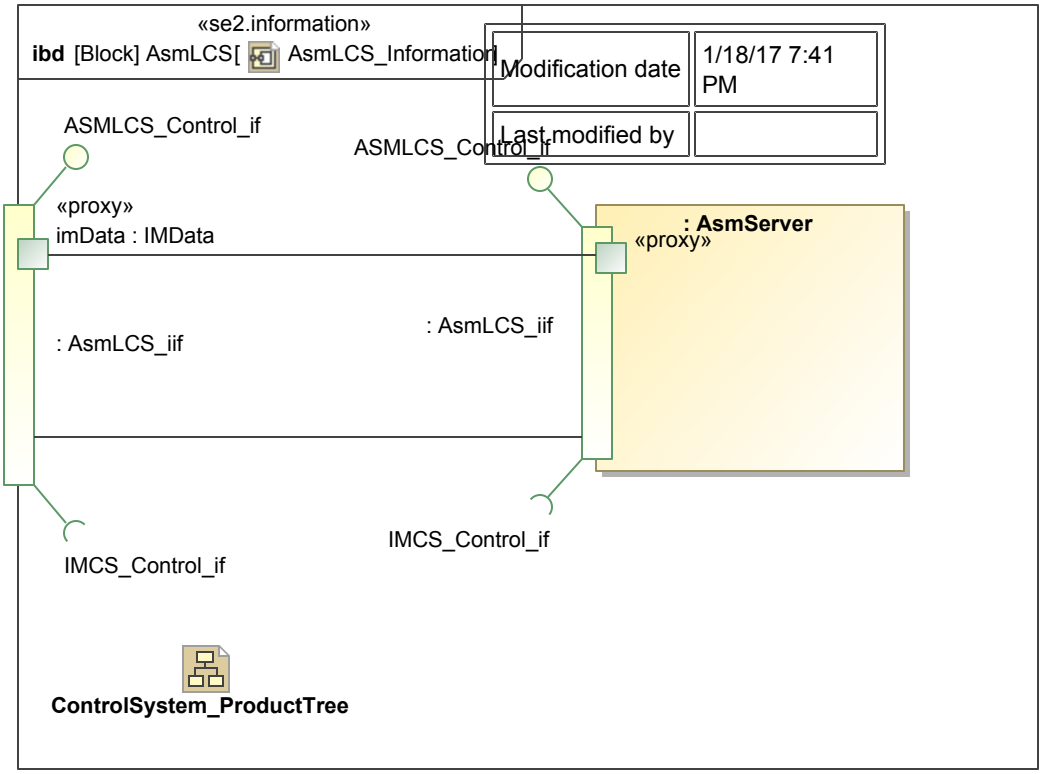


Figure 36. AsmLCS_Information

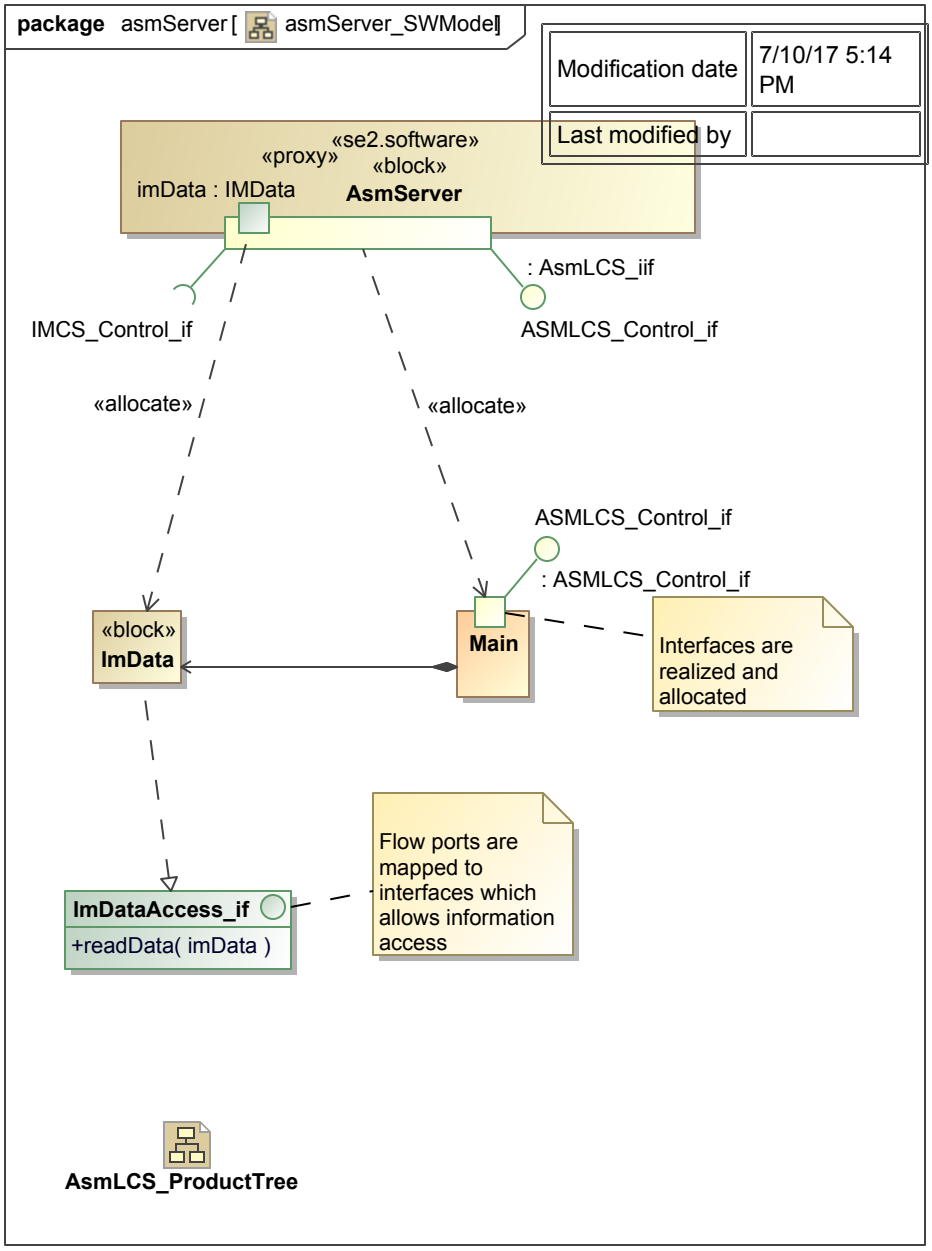


Figure 37. asmServer_SWModel

21.2 Interdisciplinary analyses and trade off

It is useful to note that while a parametric expression can be evaluated, it is not a behavior. A behavior describes a computational algorithm that evaluates in a specific sequence. A parametric equation is different in that binding any n-1 parameters of a parametric relation allows use to define the remaining one. Thus, parametric relations are not behaviors; they are statements of the relationship of quantifiable properties of a set of items. A parametric relationship states how the value of one property impacts the value of other properties • Used for Engineering analysis • Mechanism to integrate engineering analysis (performance, reliability models) with SysML assemblies • Let user create network of constraints among properties of a system, built using basic mathematical operators. • Can be used to support tradeoff analysis by representing evaluation function, can be used jointly to probability modeling available from the Auxiliary Chapter • Not defined to be executable/simulatable Time The following example shows a parametric diagram, describing relations between environment properties, optical system properties, and electrical system properties. The environmental property is the flux of the star in photons/s/m2. The optical properties are light transmissions, and the electrical properties are quantum efficiency. Each element in an optical system will not transmit 100% of the light passing through it, e.g. a fraction of the light will be absorbed by the mirrors, lenses, filters, etc. It is important to quantify the amount of light lost in an optical system and, in general, to minimize the loss. If there is too little light, one can then evaluate the CCD characteristics: integration time, binning, etc. and also determine the specifications of the CCD like the readout noise, the dark noise, etc. It can also happen that a star is too bright, and it is then important to know the amount of light on the detector in order to not saturate it. The original photon flux is influenced by the

optical elements in the light path, whereas the final flux arriving at the detector (ZEUS), is influencing together with the detector's quantum efficiency the signal to noise ratio.

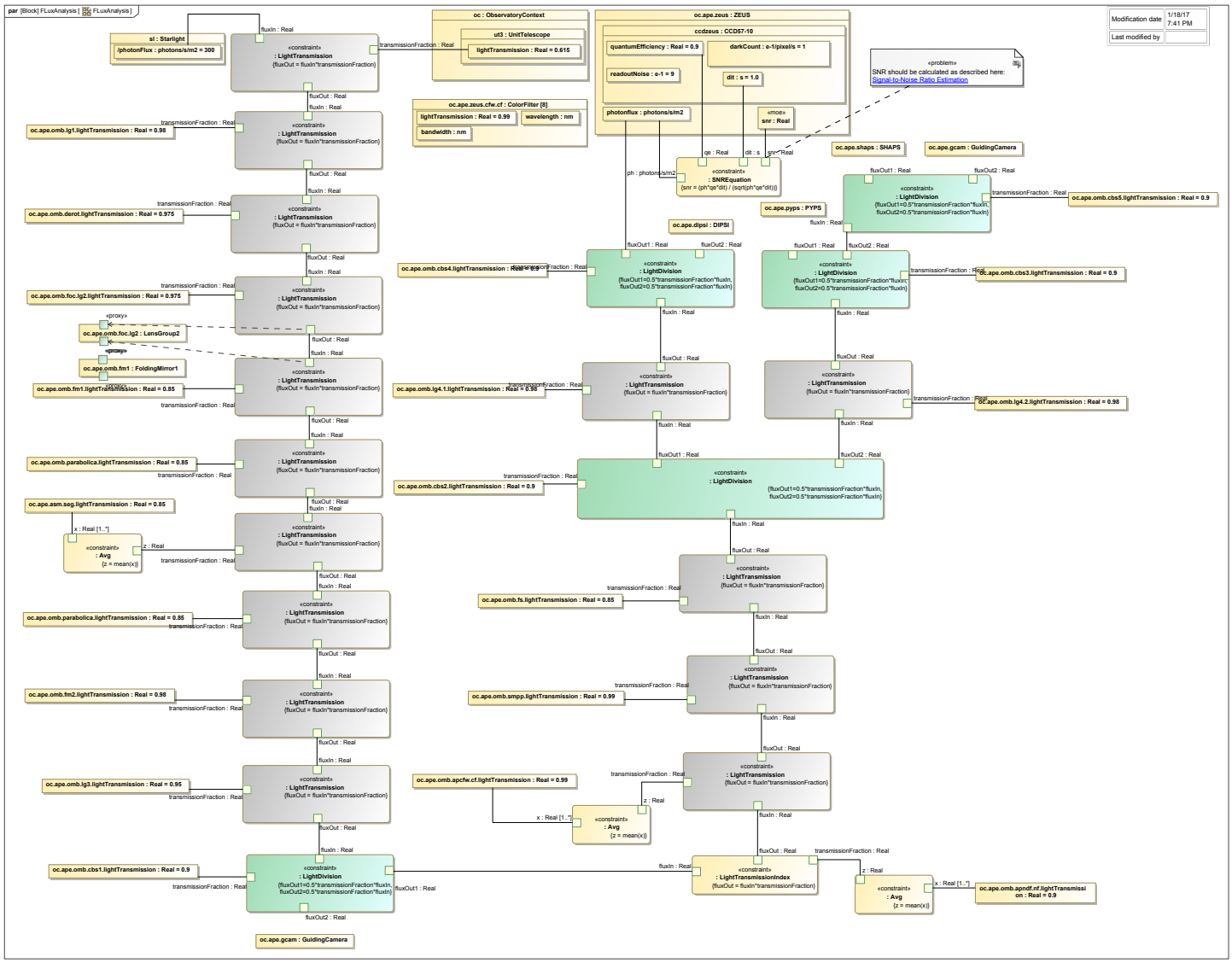


Figure 38. FLuxAnalysis

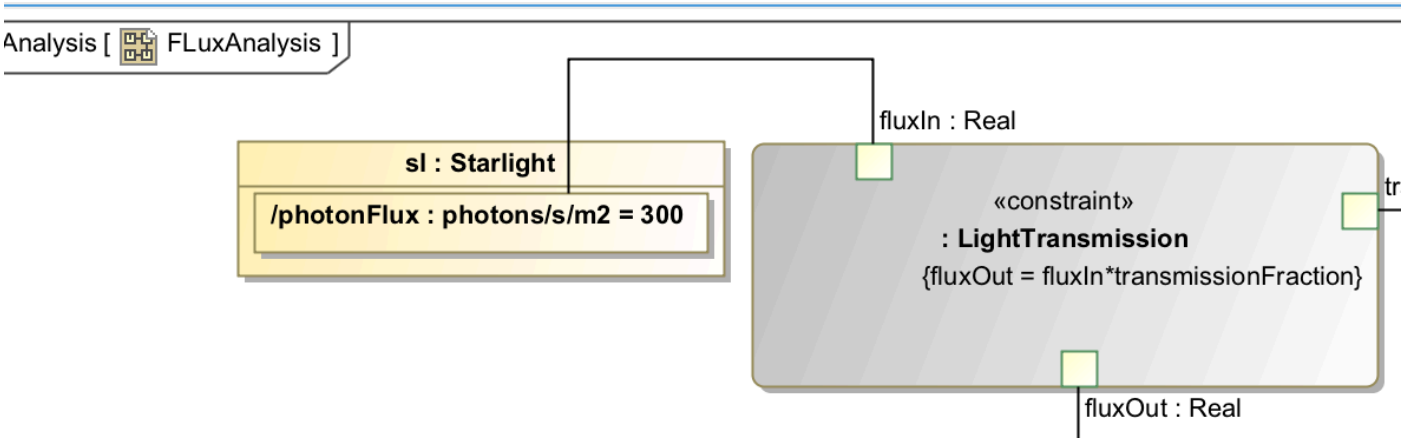


Figure 39. Flux received from Environment - the star itself

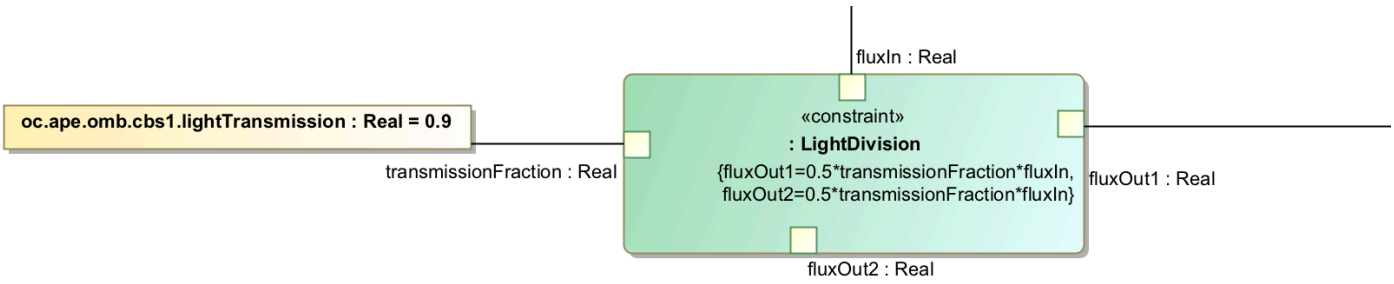


Figure 40. Flux is reduced by beam splitters and transmission factor of optical elements

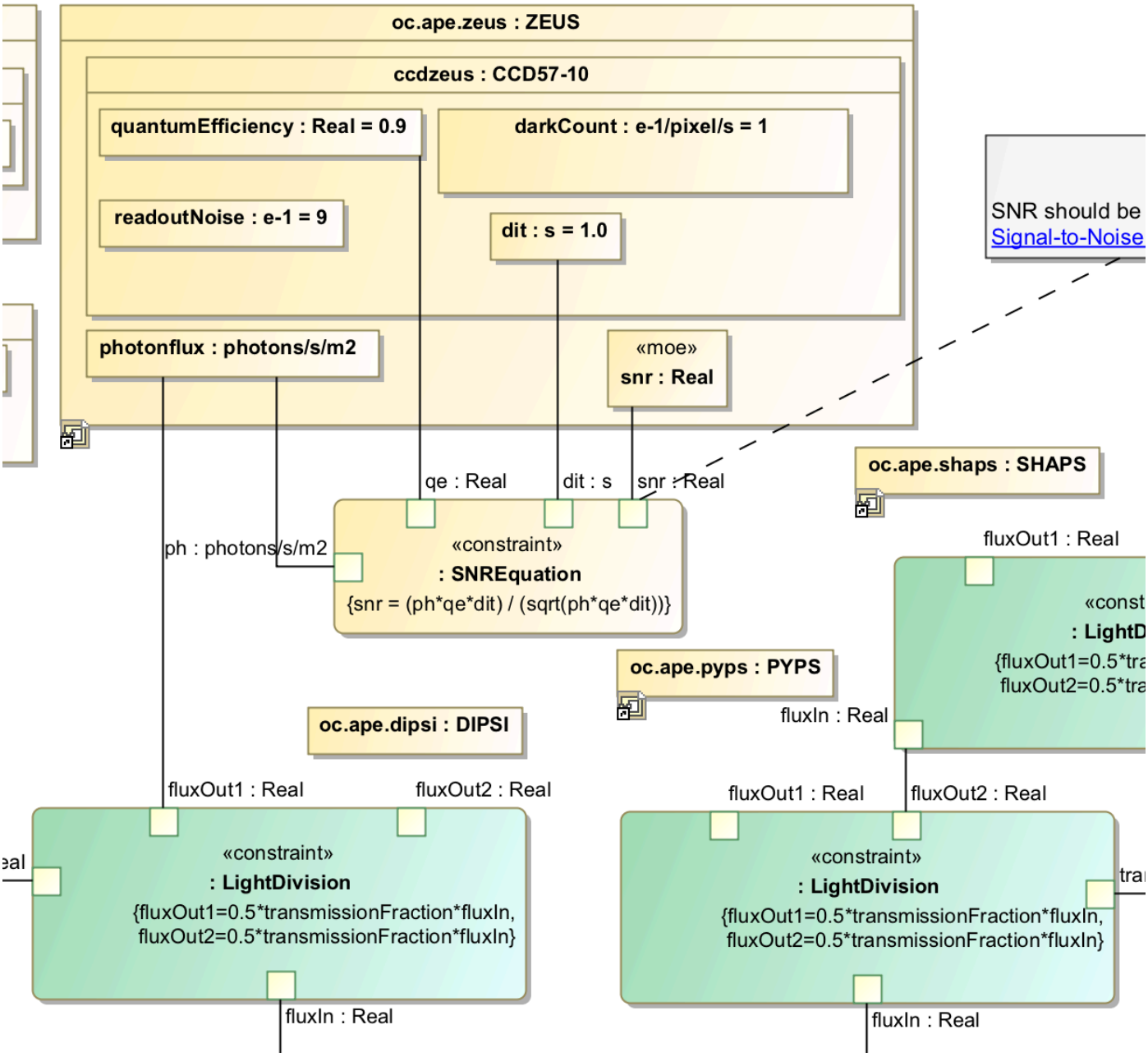


Figure 41. Signal to noise ratio at ZEUS detector

22 Cross-Cutting the Model & Traceability

22.1 Guidelines for allocations

22.1.1 Can the same element be allocated to different blocks?

In principle yes. The spec says: a single «allocate» dependency shall have only one supplier (from), but may have one or many clients (to). (15.3.2.1 Allocate (from Allocations) – constraints). Allocations of activities are a bit different. There an allocation is always a 100% allocation. If you have to allocate an action to more than one component you miss some information and you need more granularity in the activity diagram. Multiple allocation of activities is effectively a copy to different blocks, like executing the same action in parallel.

22.1.2 Should I allocate to part properties or to blocks?

Consistent allocation at same abstraction level is often not possible. Allocate at same level and then refine model by allocating functions to next lower level of structure. If a part is owned by one block and referenced by another <> has to be used to indicate that they are effectively the same part in the system.

22.1.3 How do I map an information port/connector to a physical one?

For a control system you need (at least) 2 perspectives: • the information one, where I define standard ports realizing interfaces, resp. flow ports defining data flows. They are connected by <> connectors • the physical one, meaning the transport layer. i.e. the information (commands, data, etc) is transported over CANbus, ethernet, rs232, etc. • Start with a information perspective where you simply specify components and their ports. At that point you do not know yet how they are physically connected. You might only know that one port runs over a low speed connection, another over a very high speed one, etc. • Later on, they are allocated to some hardware. • The information view defines what information is flowing; the electrical view defines how it is physically flowing. <> the information port to the physical and/or information connector to physical. This scheme can be used to model data which flow in one IBD and allocate those information flow ports to a CANbus port in another IBD.

22.2 WBS, PBS and Architecture

According to Wikipedia "a work breakdown structure (WBS) in project management and systems engineering, is a tool used to define and group a project's discrete work elements in a way that helps organize and define the total work scope of the project. A work breakdown structure element may be a product, data, a service, or any combination. A WBS also provides the necessary framework for detailed cost estimating and control along with providing guidance for schedule development and control. Additionally the WBS is a dynamic tool and can be revised and updated as needed by the project manager.http://en.wikipedia.org/wiki/Work_breakdown_structure - cite_note-BAH-0 The Work Breakdown Structure is a tree structure, which shows a subdivision of effort required to achieve an objective; for example a program, project, and contract. In a project or contract, the WBS is developed by starting with the end objective and successively subdividing it into manageable components in terms of size, duration, and responsibility (e.g., systems, subsystems, components, tasks, subtasks, and work packages) which include all steps necessary to achieve the objective. The Work Breakdown Structure provides a common framework for the natural development of the overall planning and control of a contract and is the basis for dividing work into definable increments from which the statement of work can be developed and technical, schedule, cost, and labor hour reporting can be established." To reflect the tree structure of the WBS it can be modelled as hierarchical package structure. Important information as work package budget, work package owner, etc. can be added as tagged values. This information can be synchronized with a project planning tool like MS Project for detailed task and resource planning.

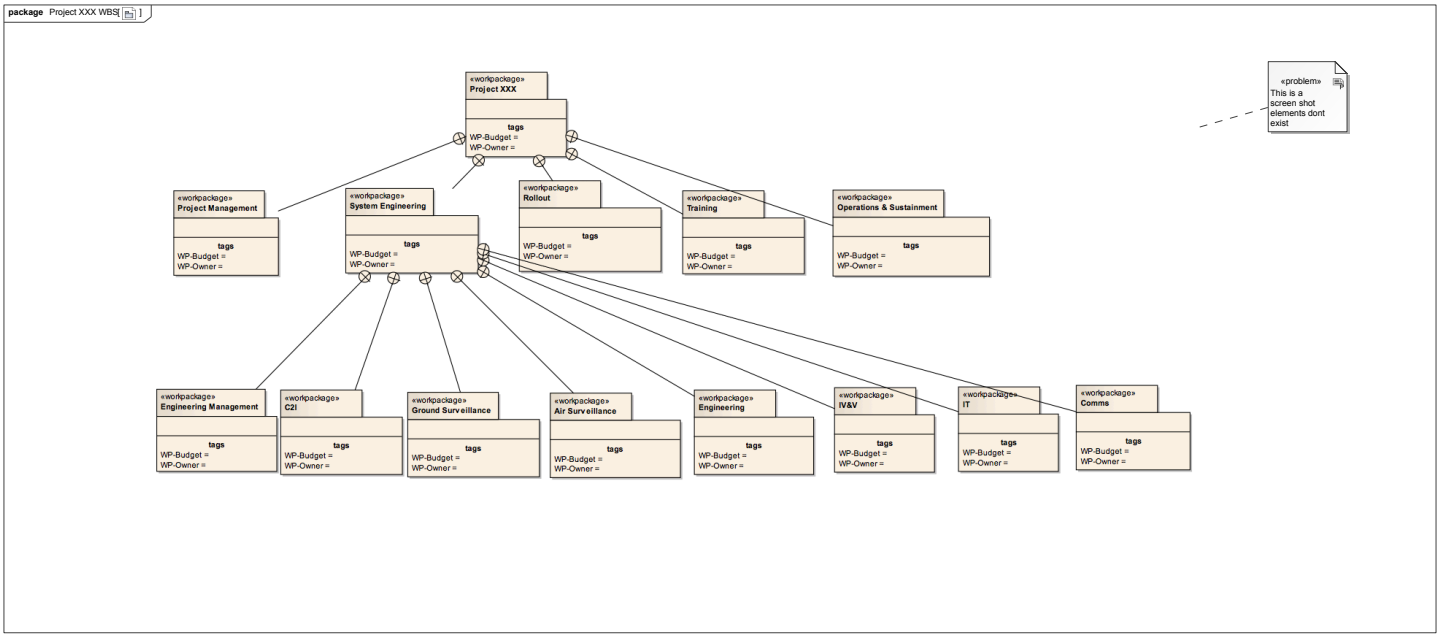


Figure 42. Project XXX WBS

Based on the WBS a Product Breakdown Structure (PBS) can be defined. According to Wikipedia "in project management, a product breakdown structure (PBS) is a tool for analysing, documenting and communicating the outcomes of a project, and forms part of the product based planning technique. The PBS provides an exhaustive, hierarchical tree structure of deliverables (physical, functional or conceptual) that make up the project, arranged in whole-part relationship. This diagrammatic representation of project outputs provides a clear and unambiguous statement of what the project is to deliver. The PBS is identical in format to the work breakdown structure (WBS), but includes only the physical architecture of a product. The WBS includes the data and service elements necessary to complete the system, as well as the all product elements described in the PBS." To reflect the tree structure of the PBS it can be modelled as hierarchical package structure. Important information as product accountable, etc. can be added as tagged values. Again, this information can be synchronized with a project planning tool like MS Project for detailed task and resource planning.

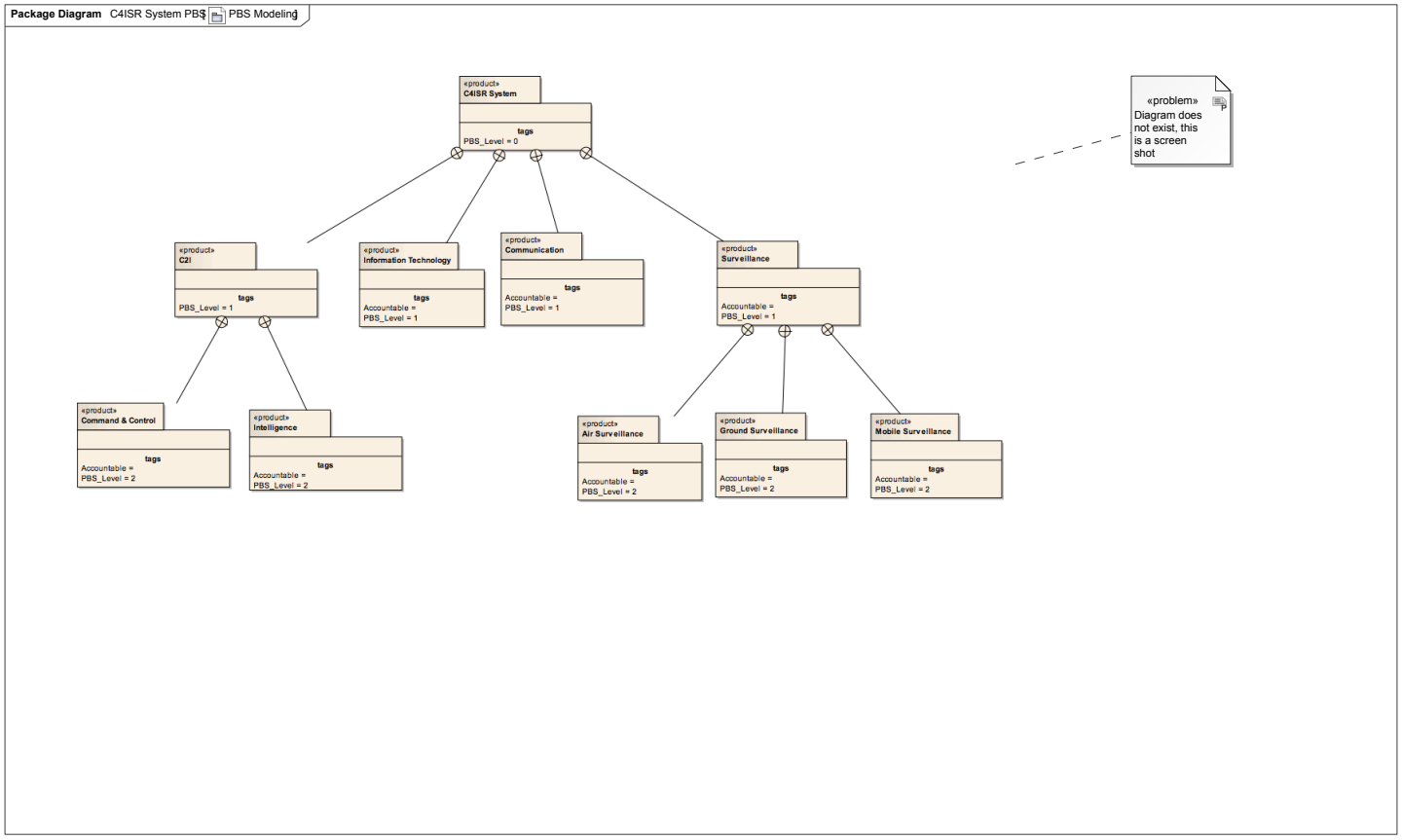


Figure 43. PBS Modeling

The allocation of WBS to PBS can be modelled as stereotyped dependency :

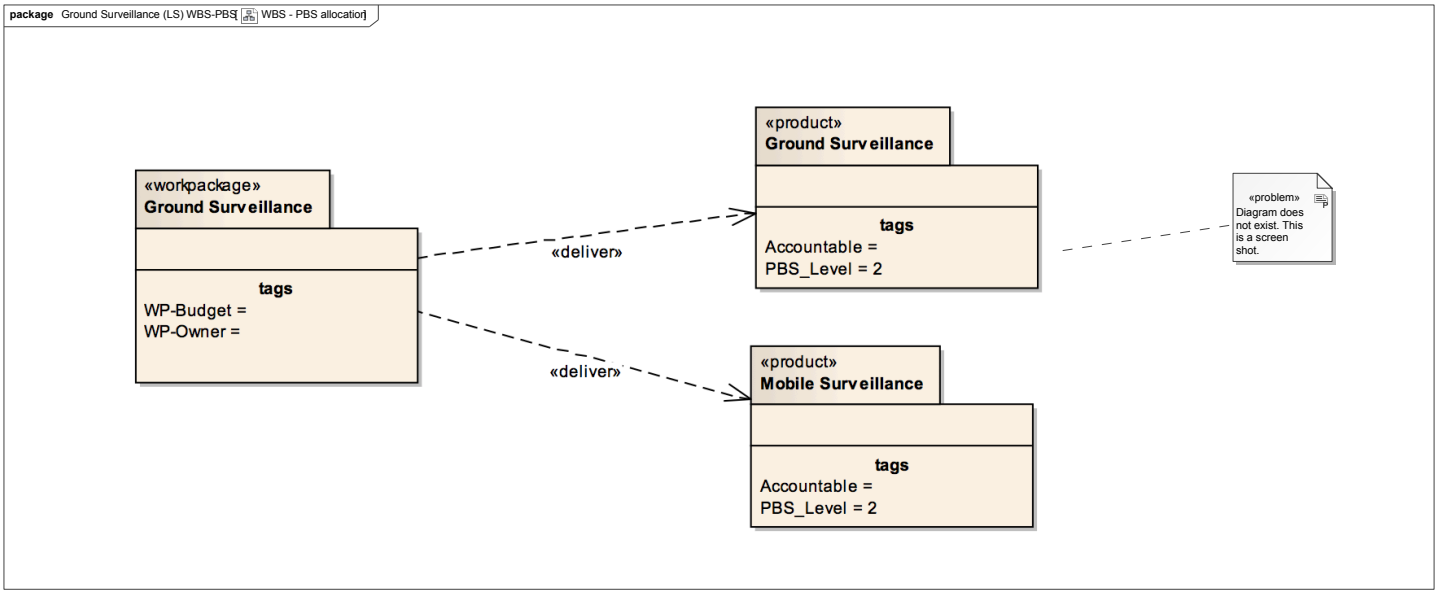


Figure 44. WBS - PBS allocation

The PBS elements can be allocated to the logical architecture elements and the physical architecture elements. Stereotypes can be used to express the different semantics of the allocation:

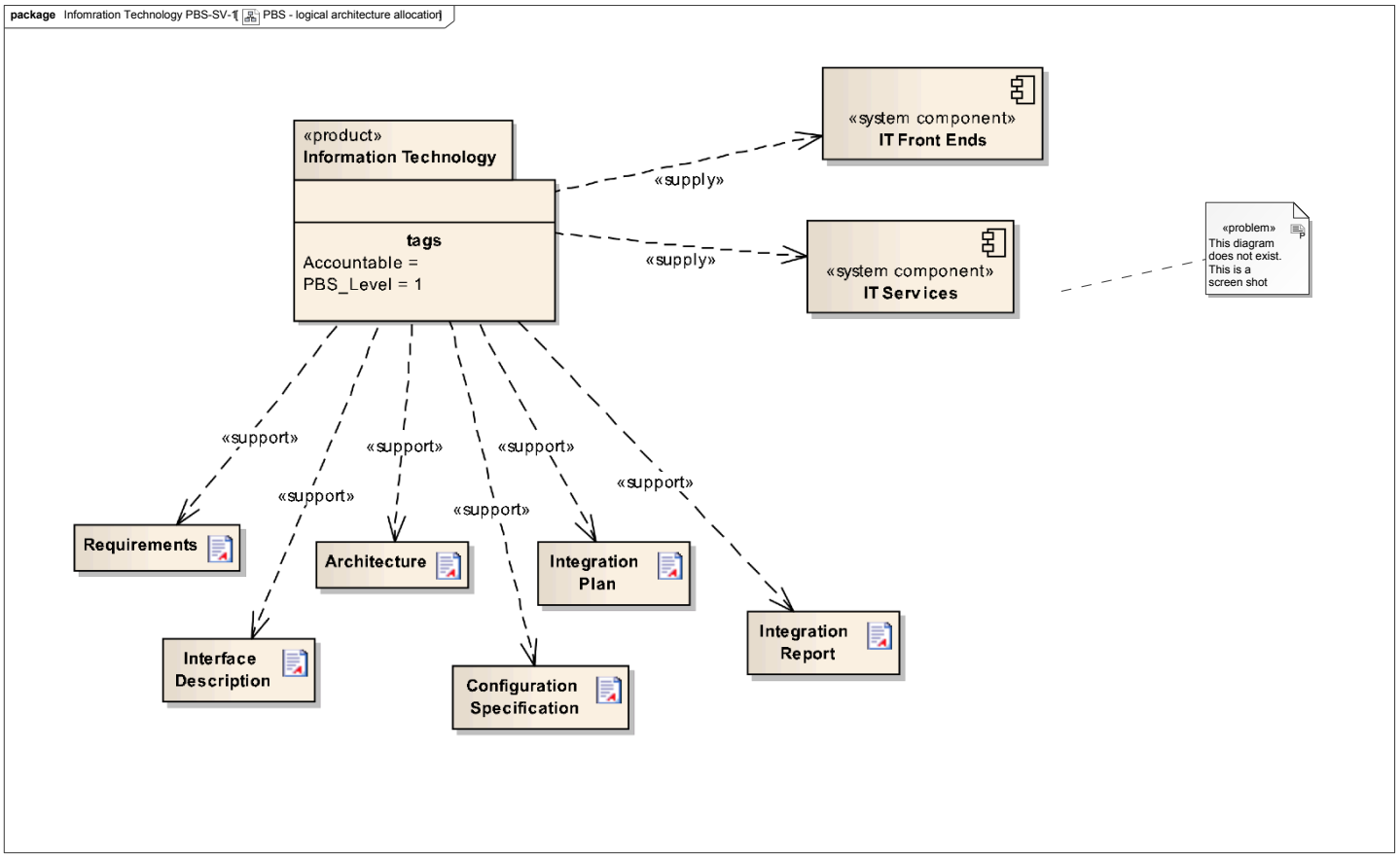


Figure 45. PBS - logical architecture allocation

• <> means that the work package supports the generation of an work product. For example if a work package contributes to an system requirements or architecture document. • <> means that a work package provides a logical architecture element or component. • <> means that a work package designs a logical architecture element or component but does not provide it.

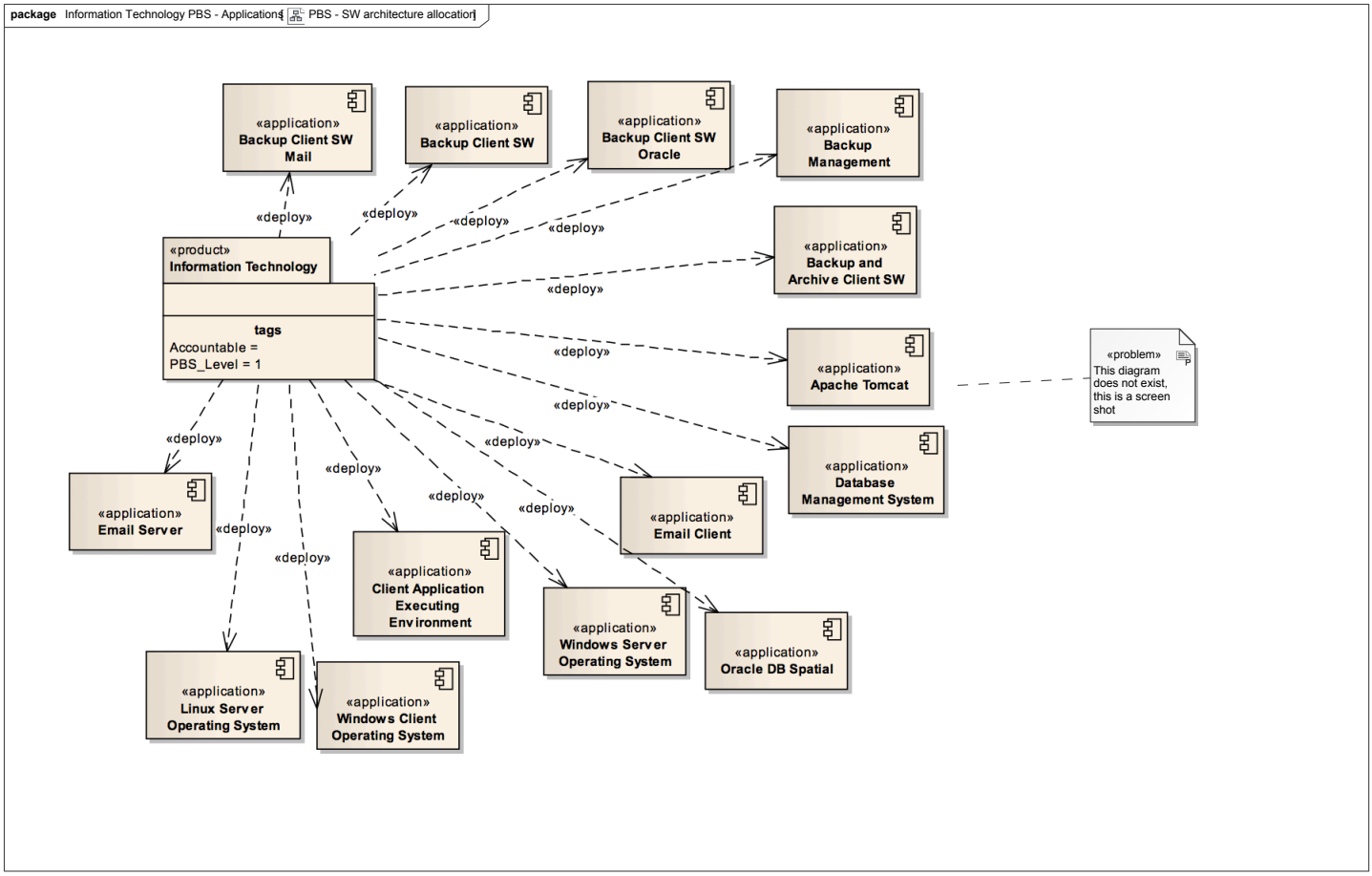


Figure 46. PBS - SW architecture allocation

<> means that the work package deploys a SW application or HW device

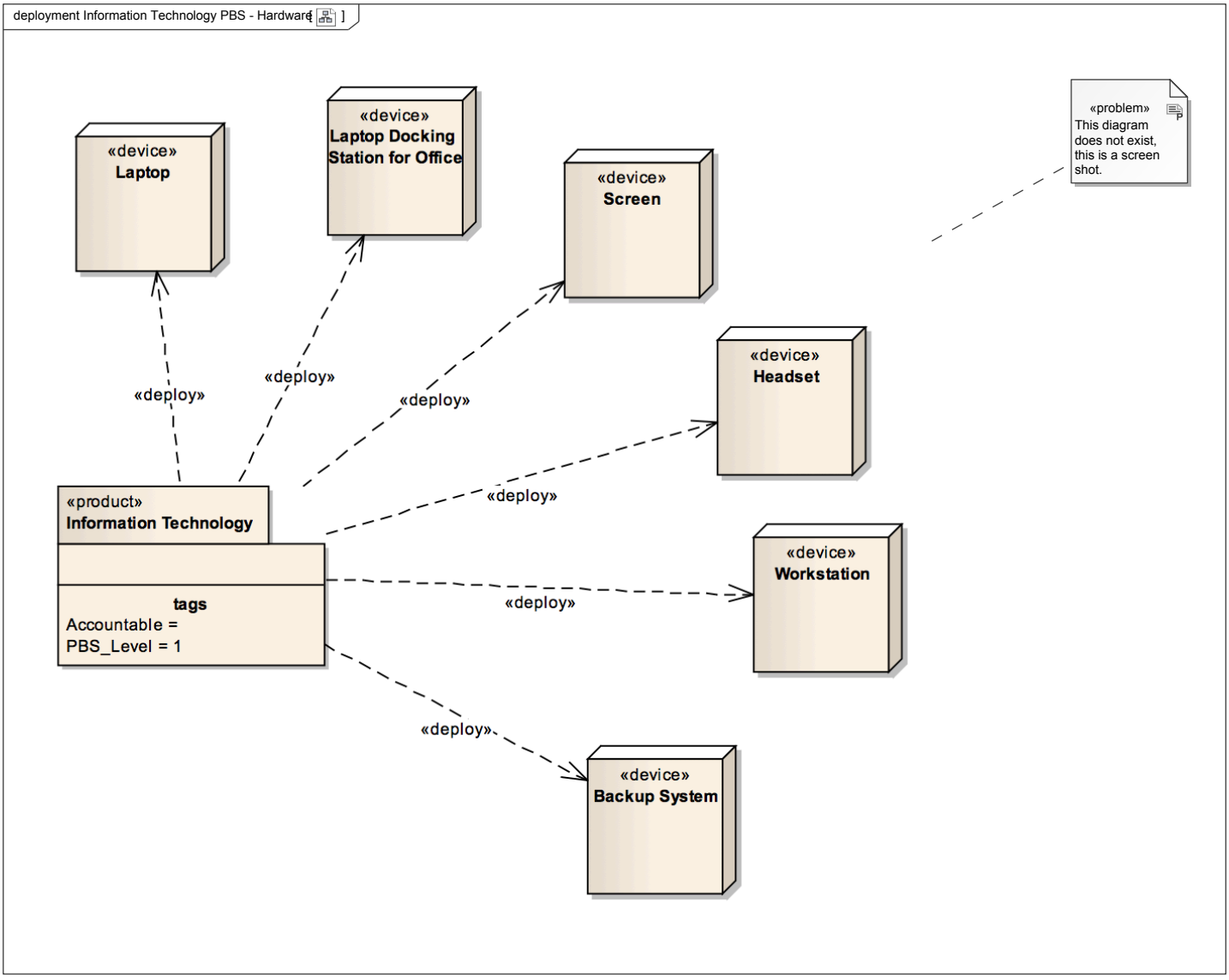


Figure 47. PBS - HW architecture allocation

This is the metamodel of the WBS-PBW-modelling:

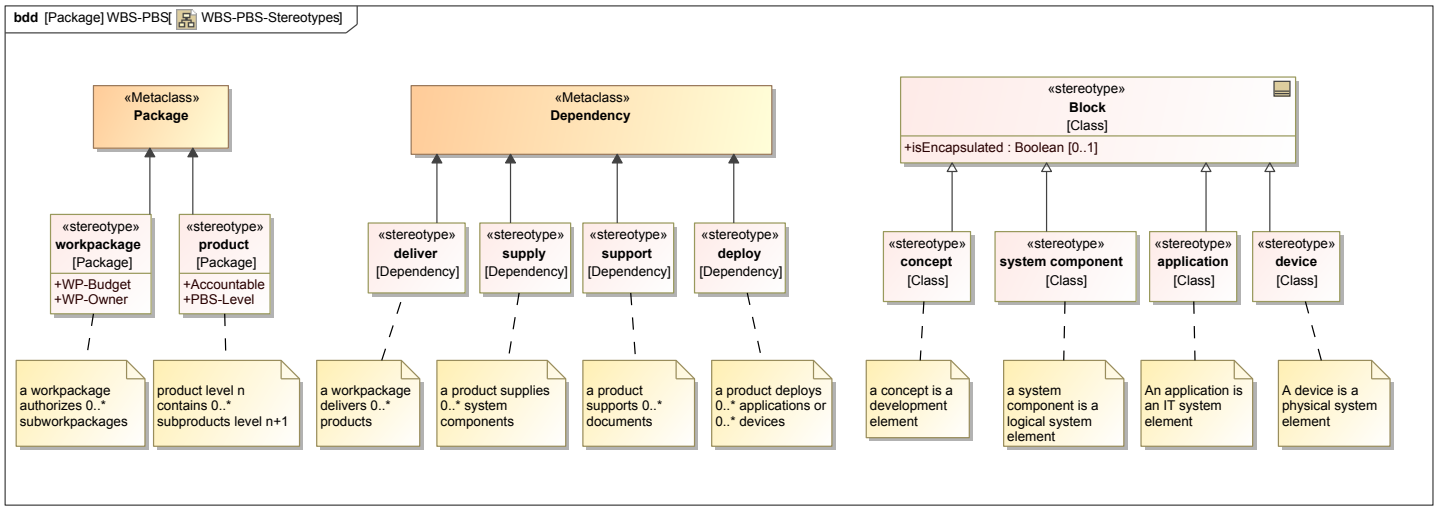


Figure 48. WBS-PBS-Stereotypes

23 Variant Modeling

The modeling of system variants is a core technique for model based systems engineering. You need to model variants • for analysing design alternatives, • for evaluating variants via trade-offs, • for modeling of product families, and for the separation of a logical and a physical architecture.The challenge is to separate the variant from the common part and to manage the dependencies.

23.1 Definitions

A variation contains a set of variants that have a common discriminator. A variant is a complete set of variant elements that varies the system according to the variation discriminator. A variation point marks a core element as a docking point for a variant element. A variant element is an element in a variant package. A core element is an element that is valid for all variants.

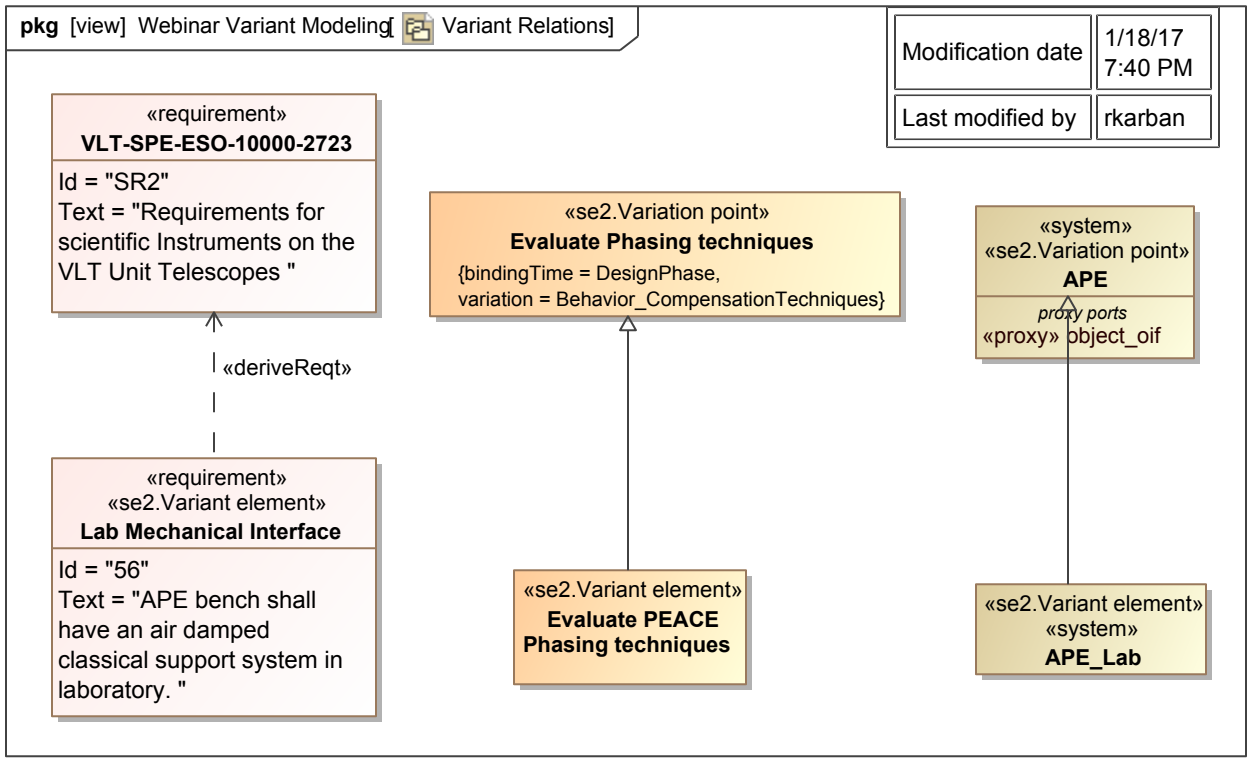


Figure 49. Variant Relations

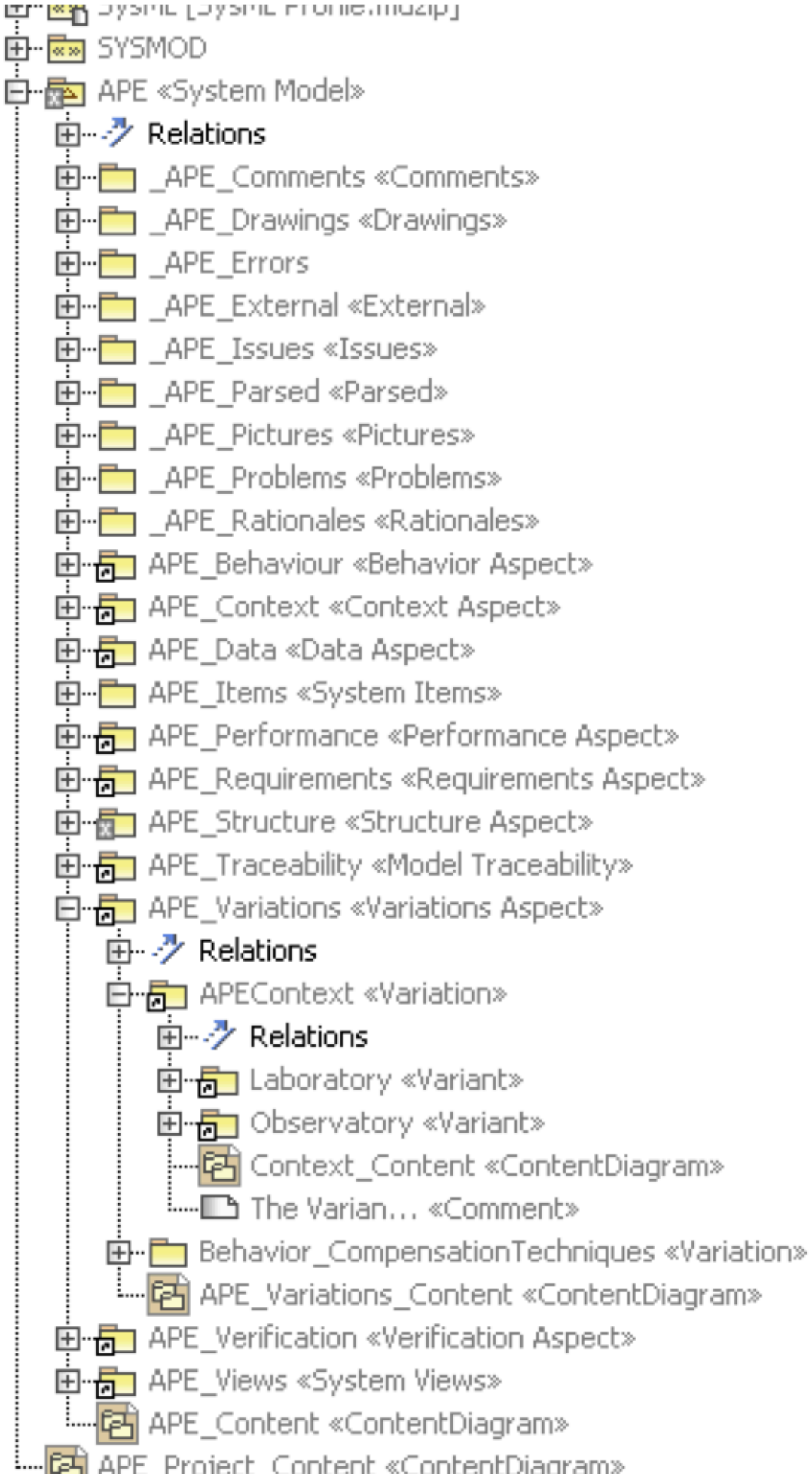

SYSMOD
APE «System Model»
Relations
_APE_Comments «Comments»
_APE_Drawings «Drawings»
_APE_Errors
_APE_External «External»
_APE_Issues «Issues»
_APE_Parsed «Parsed»
_APE_Pictures «Pictures»
_APE_Problems «Problems»
_APE_Rationales «Rationales»
APE_Behaviour «Behavior Aspect»
APE_Context «Context Aspect»
APE_Data «Data Aspect»
APE_Items «System Items»
APE_Performance «Performance Aspect»
APE_Requirements «Requirements Aspect»
APE_Structure «Structure Aspect»
APE_Traceability «Model Traceability»
APE_Variations «Variations Aspect»
Relations
APEContext «Variation»
Relations
Laboratory «Variant»
Observatory «Variant»
Context_Content «ContentDiagram»
The Varian... «Comment»
Behavior_CompensationTechniques «Variation»
APE_Variations_Content «ContentDiagram»
APE_Verification «Verification Aspect»
APE_Views «System Views»
APE_Content «ContentDiagram»

Figure 50. Separation of concerns: core and variations

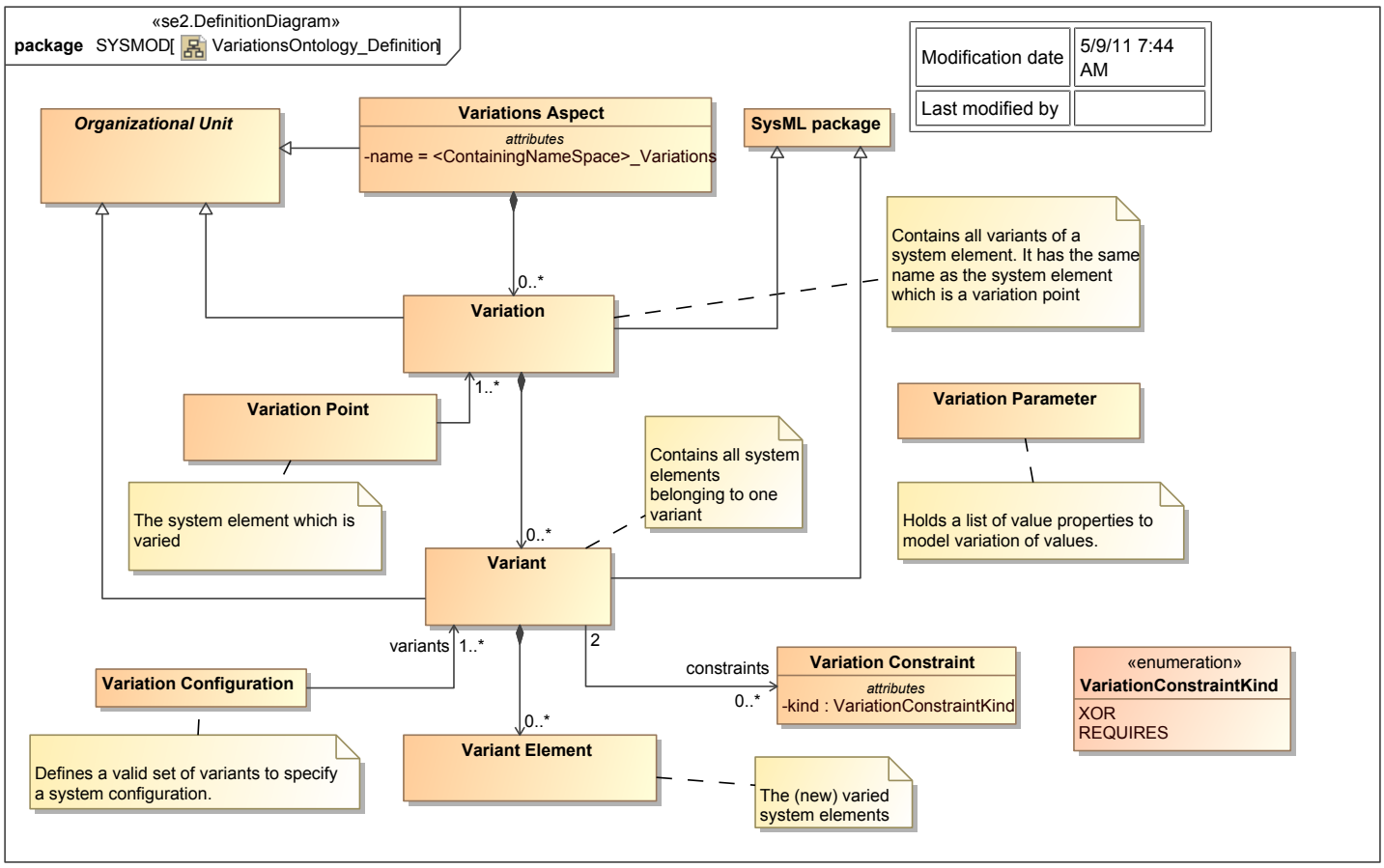


Figure 51. VariationsOntology_Definition

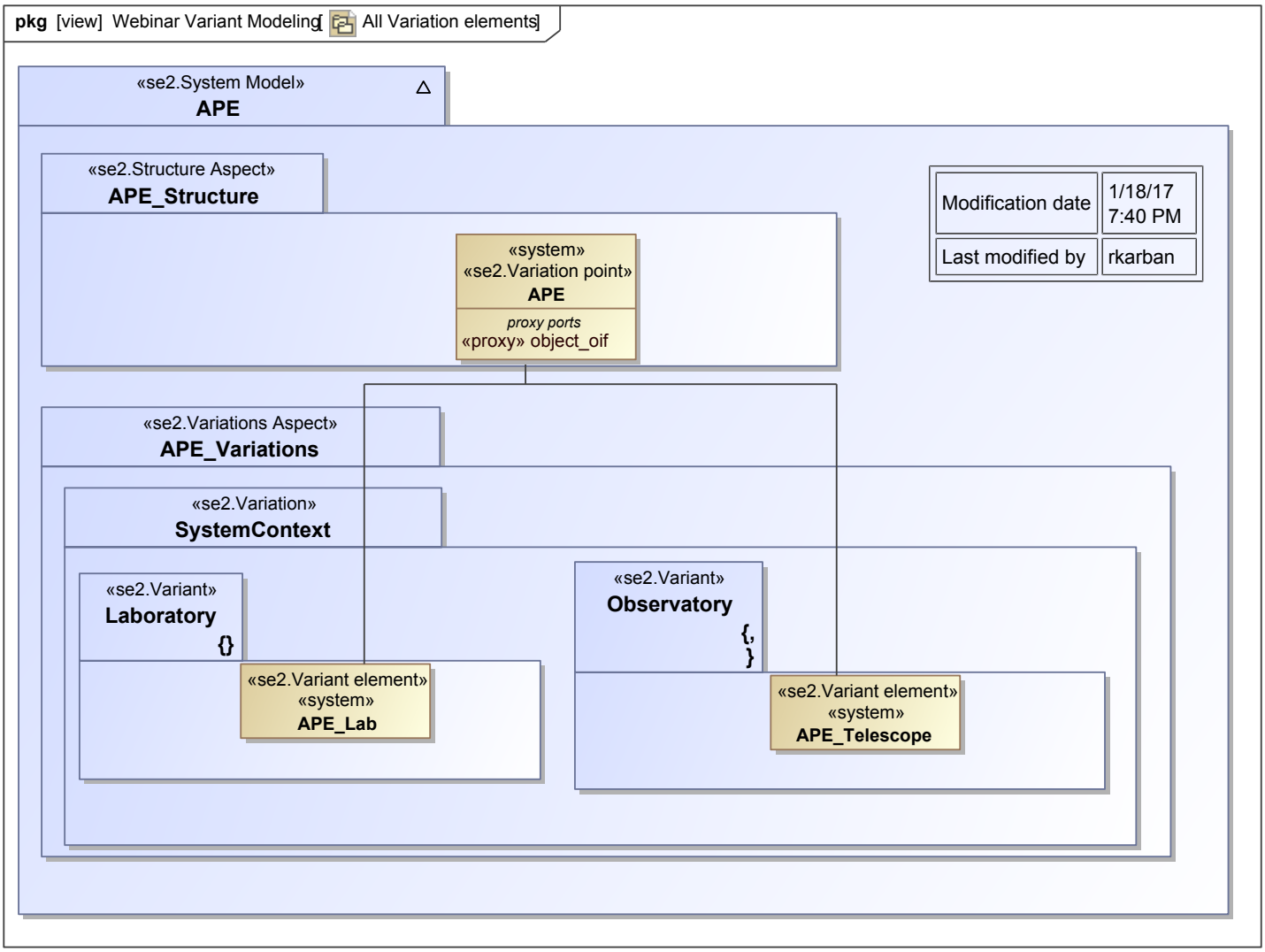


Figure 52. All Variation elements

23.2 SYSMOD Variant Profile

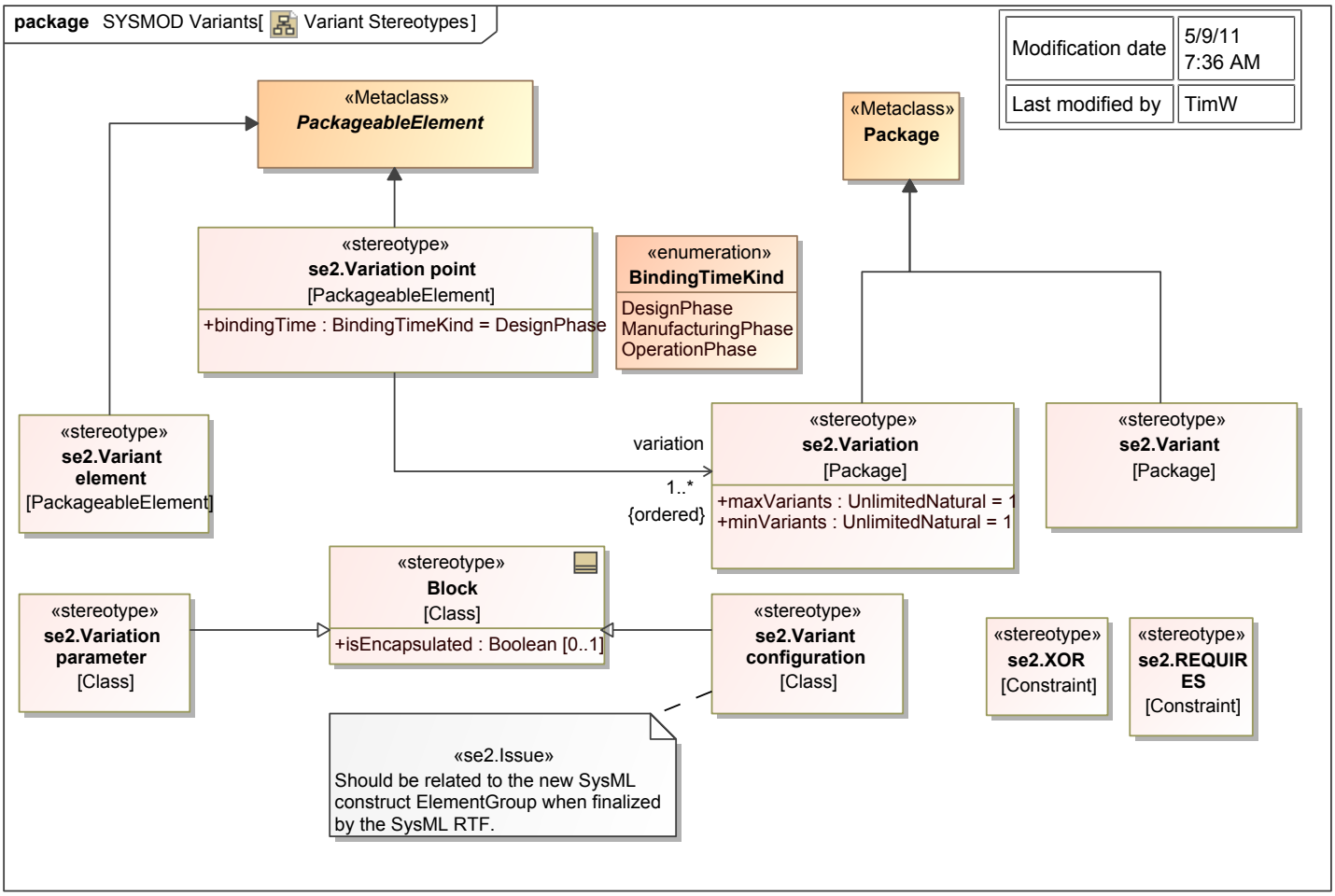


Figure 53. Variant Stereotypes

23.3 Variant configurations

The Featured Oriented Domain Analysis (FODA) is a modeling method, which allows to describe features (variants) of a specific product. In Figure 88 an FODA example feature tree of a car is provided. The above example could also expressed with SysML and the different variant stereotypes: Rules formulated in a textual DSL and embedded in UML constraints are defining the variation possibilities. In Figure 89 Car variants, the “Comfort” option of a car is defined by the rule, that the car has automatic transmission and air conditioning on board. The TransmissionType variation is AirConditioningOption is optional, so the stereotype tag minVariant is zero.

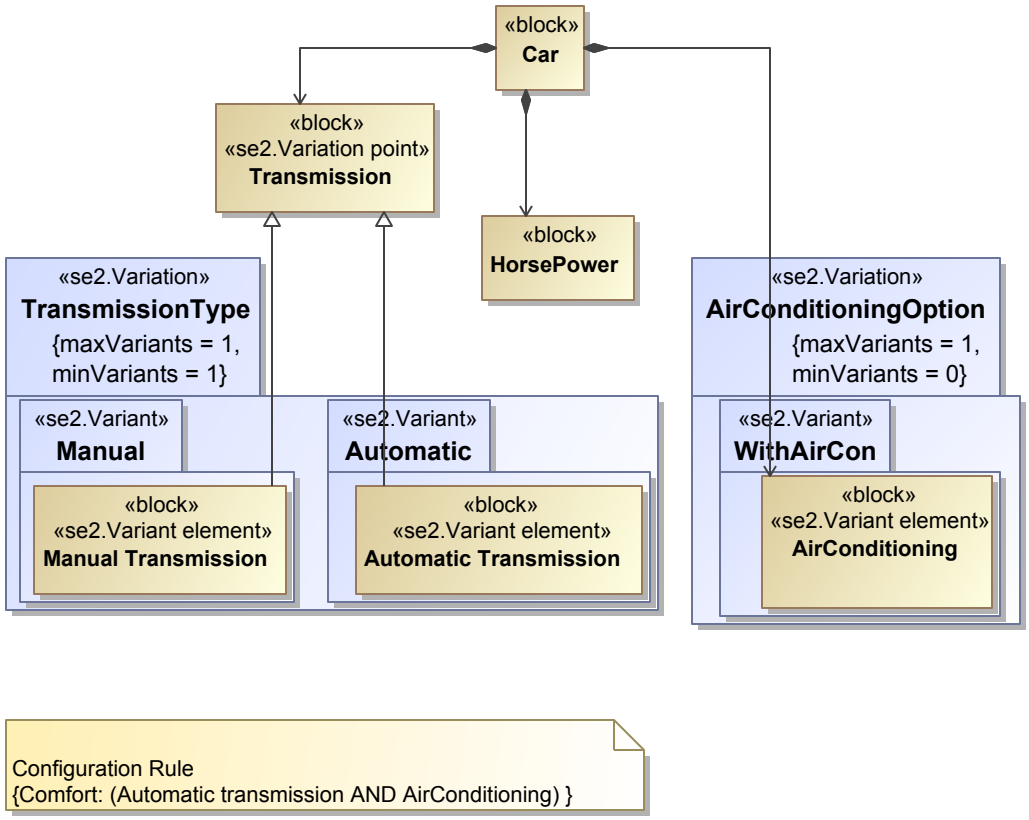


Figure 54. CarExample BDD

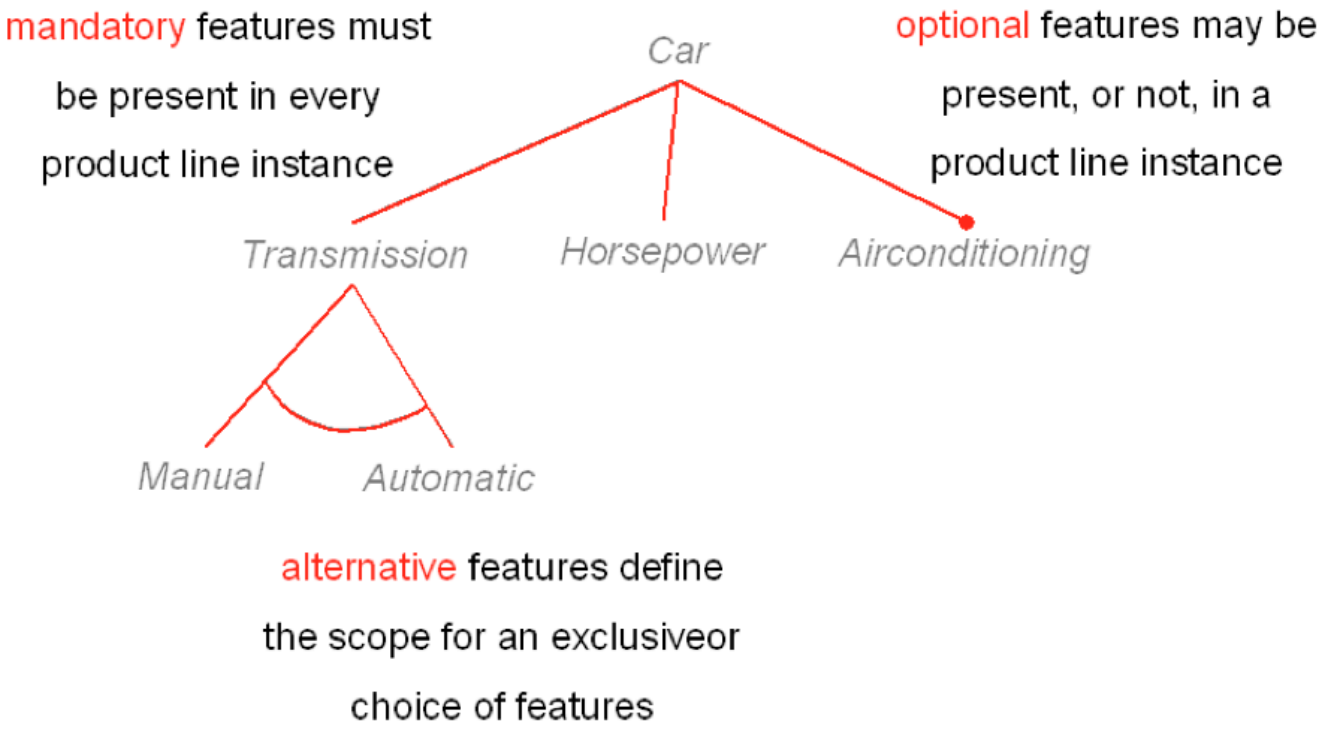


Figure 55. FODA example(Source: Myra Cohen, Matthew Dwyer: Software Product Line Testing Part II : Variability Modeling)

23.4 Model transformations

Even simple variations are resulting into complex configuration spaces. It is necessary to have a simple view for a selected configuration. This view could be produced by a M2M-Transformation (M2M=Model to Model). Three variations are spanning a three-dimensional configuration space (see Figure 90 3D configuration space) and eventually many more possible configurations. A configuration is one point in the configuration space.

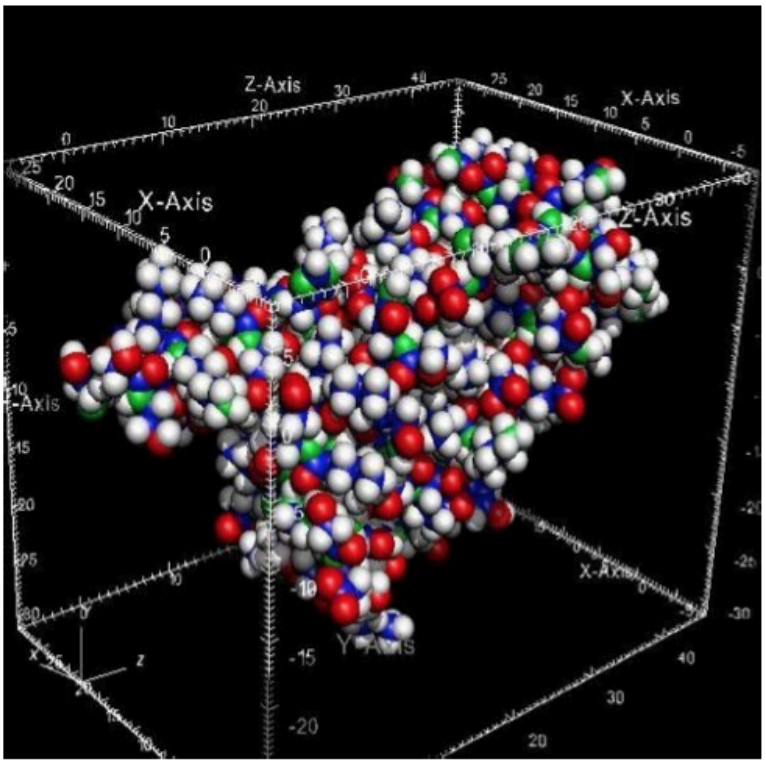


Figure 56. 3D configuration space

The aims of the model transformation are: • Face-out of irrelevant details. • Creation of a product model out of a product family model. • Elimination of non-existing variants and closure of variants because of superfluous abstractions. The following categories of M2M transformation exist: • View- vs. Copy-Strategy: o View: The transformation creates a view in the source model. -> Advantage: Seperation of Product Line Engineering and Product Engineering o Copy: The transformation creates a new model -> Effect: Discard of variants during the development phase. • Filter vs. Refactoring-Strategy: o Filter: No more required model elements will be deleted (from the view or the copy)) by the transformation -> Easy to apply, but some „ballast“ remains o Model-Refactoring: There exists not „the one and only“ transformation, but a set of adequate refactorings, with a corresponding non-trivial transformation -> Efect: best possible reduction of complexity, but hard to implement An example of the simple filtering M2M approach is shown in Figure 91Filter approach:

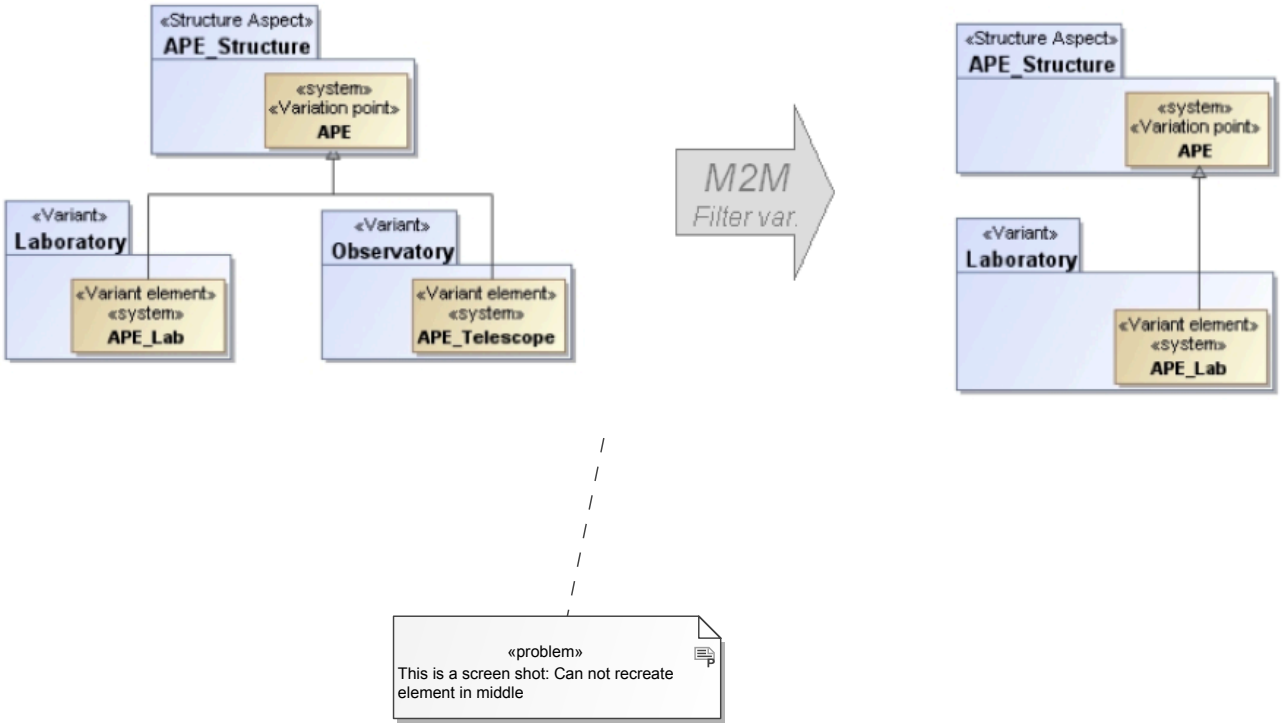


Figure 57. Filter approach

The elements of the laboratory variant are presented. All other elements of the observatory variant are filtered out. But inheritance superclasses, used to derive variations, are still available in the model. An example of the refactoring M2M approach is shown in Figure 92Refactoring approach:

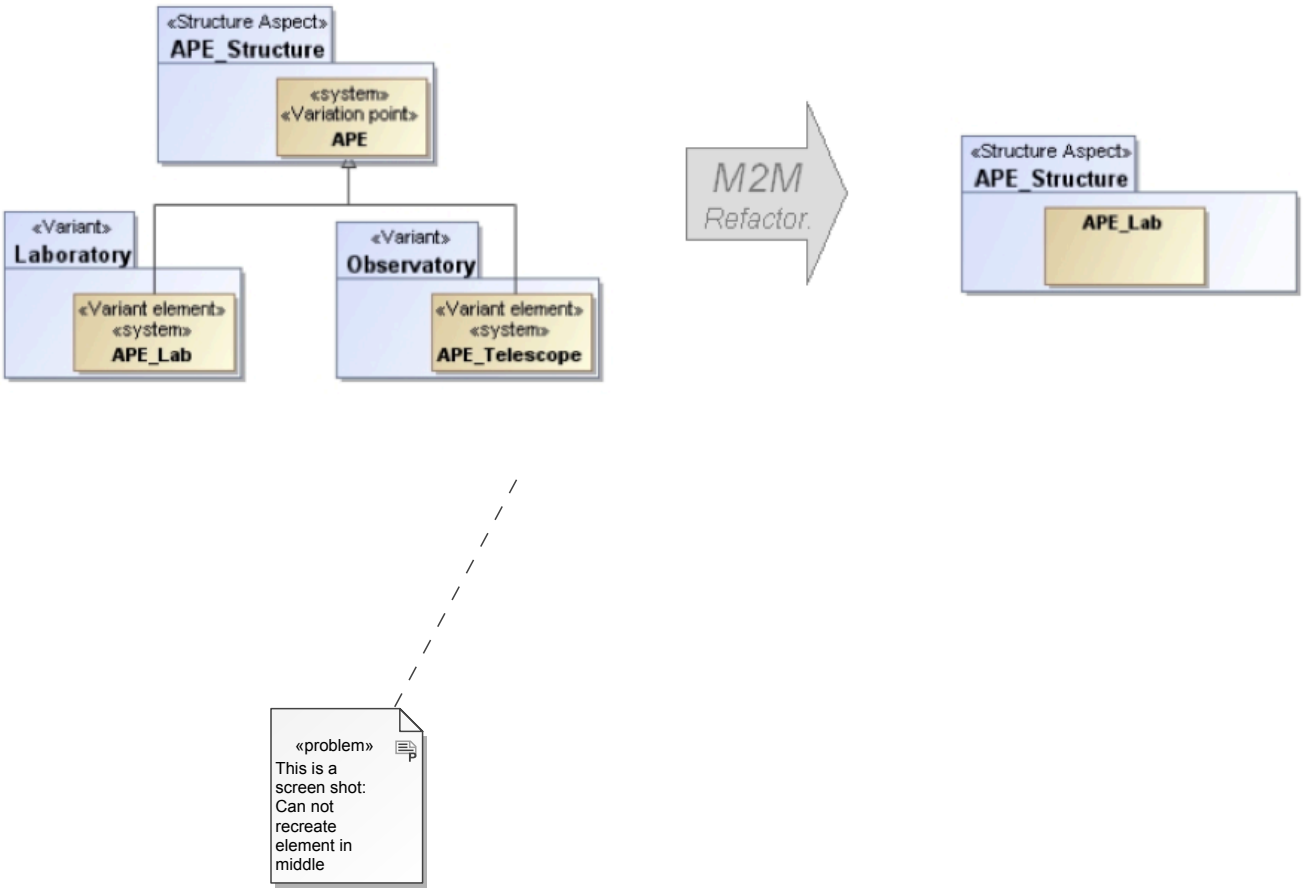


Figure 58. Refactoring approach

Here the resulting model after the transformation, only contains the elements of the laboratory variant, the inheritance superclasses for deriving the variants, are also refactored out of the model.

23.4.1 Open issues

There are open issues concerning M2M transformations: • Until now model transformations are only manually applied in the telescope model. • Simple approach could be easily implemented: E.g. MagicDraw offers the Module concept. This could be used to hide all other variants and present only the elements belonging to that module/variant. • Automatic model transformation shall be evaluated by using transformation frameworks like OpenArchitectureWare, which is now part of the Eclipse Modeling Project http://www.eclipse.org/modeling/

23.5 Trade-Off analysis

Different alternatives/variants could be evaluated/weighted by applying trade-off studies. In Figure 93 Trade-Off Analysis shows the trade-off analysis process for three different variants. The performance indicators cost, reliability and performance are calculated and weighted by a effectiveness function defined as a parametric constraint. The “most effective” variant is the result of the trade-off.

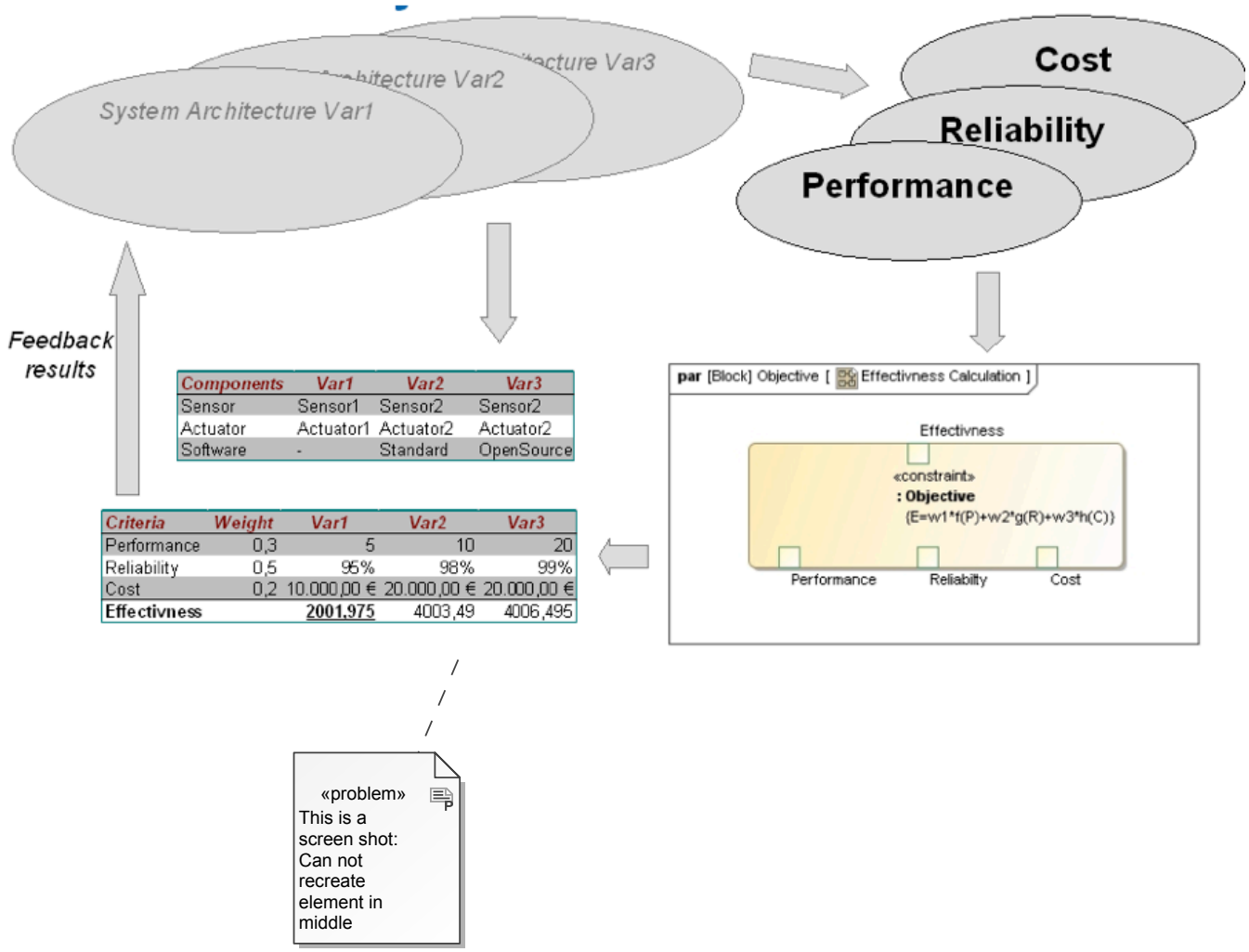


Figure 59. Trade-Off Analysis (Source: Sanford Friedenthal:, Advancing Systems Engineering Practice Using Model Based Systems Development)

24 Trade Study Pattern - DRAFT-

24.1 Intent

24.2 Motivation

To use this pattern, you have to define your own trade study analysis context block, which is inherited from the Trade Study Pattern block, and then make the classifier or block, which is the type of alternatives to be inherited from Alternative.

24.3 Concept

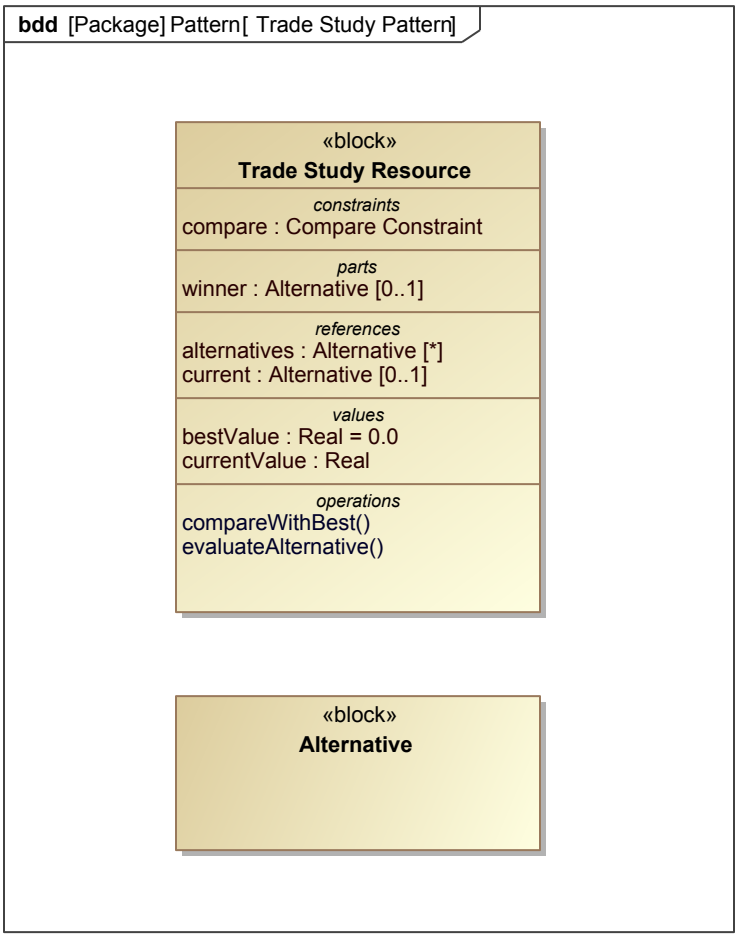


Figure 60. Trade Study Pattern

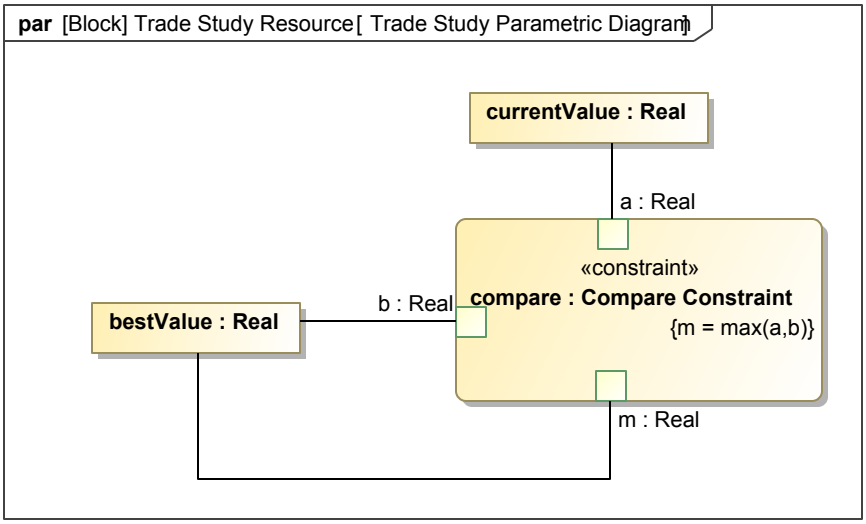


Figure 61. Trade Study Parametric Diagram

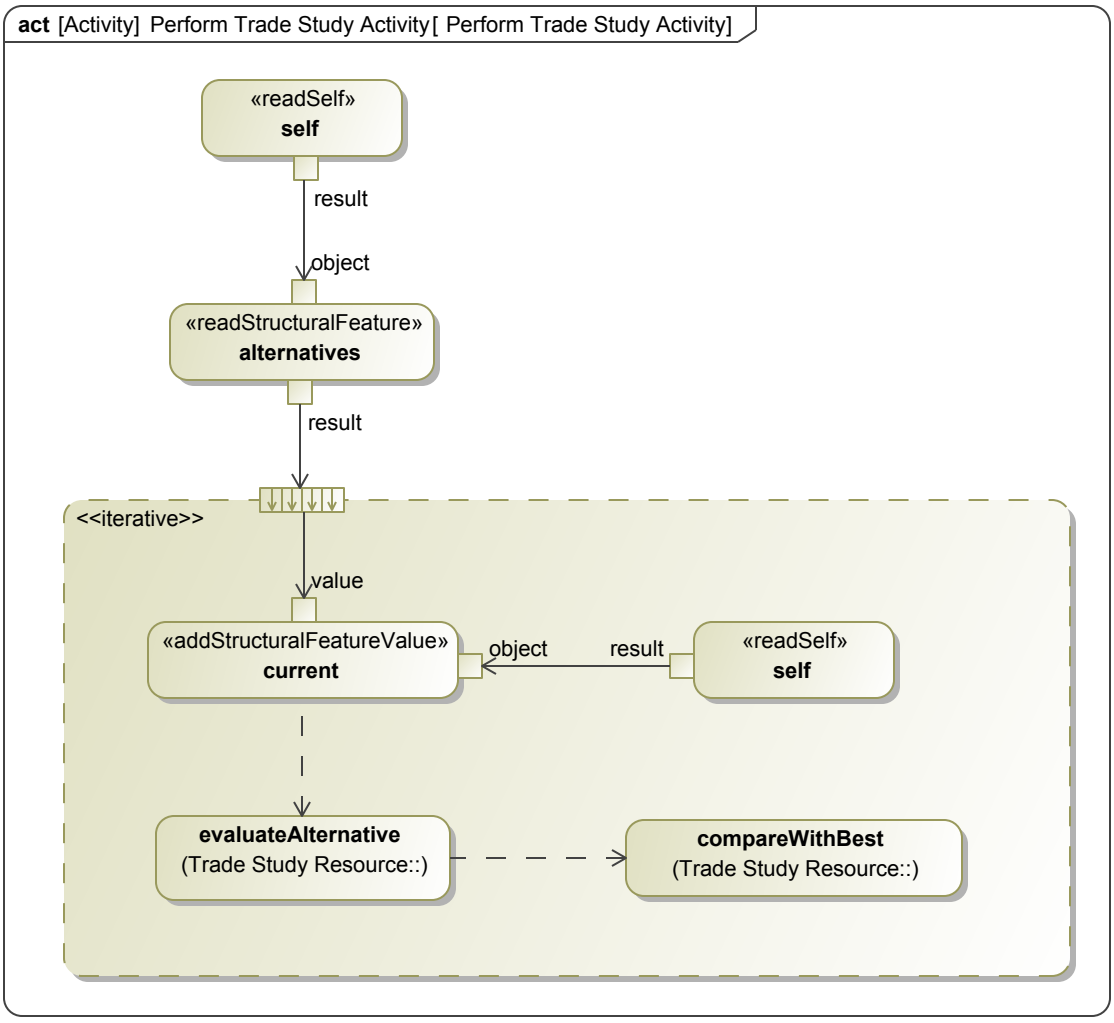


Figure 62. Perform Trade Study Activity

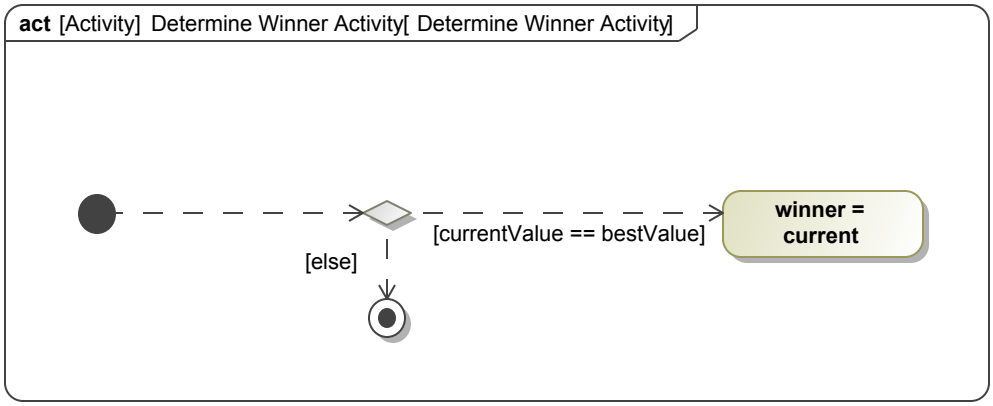


Figure 63. Determine Winner Activity

Table 8. <>

Model Element	
Alternative	
Compare Constraint	
Trade Study Resource	
Trade Study Parametric Diagram	
Perform Trade Study Activity	
Evaluate Alternative	
Determine Winner Activity	
Trade Study Pattern	

24.4 Consequences

24.5 Implementation

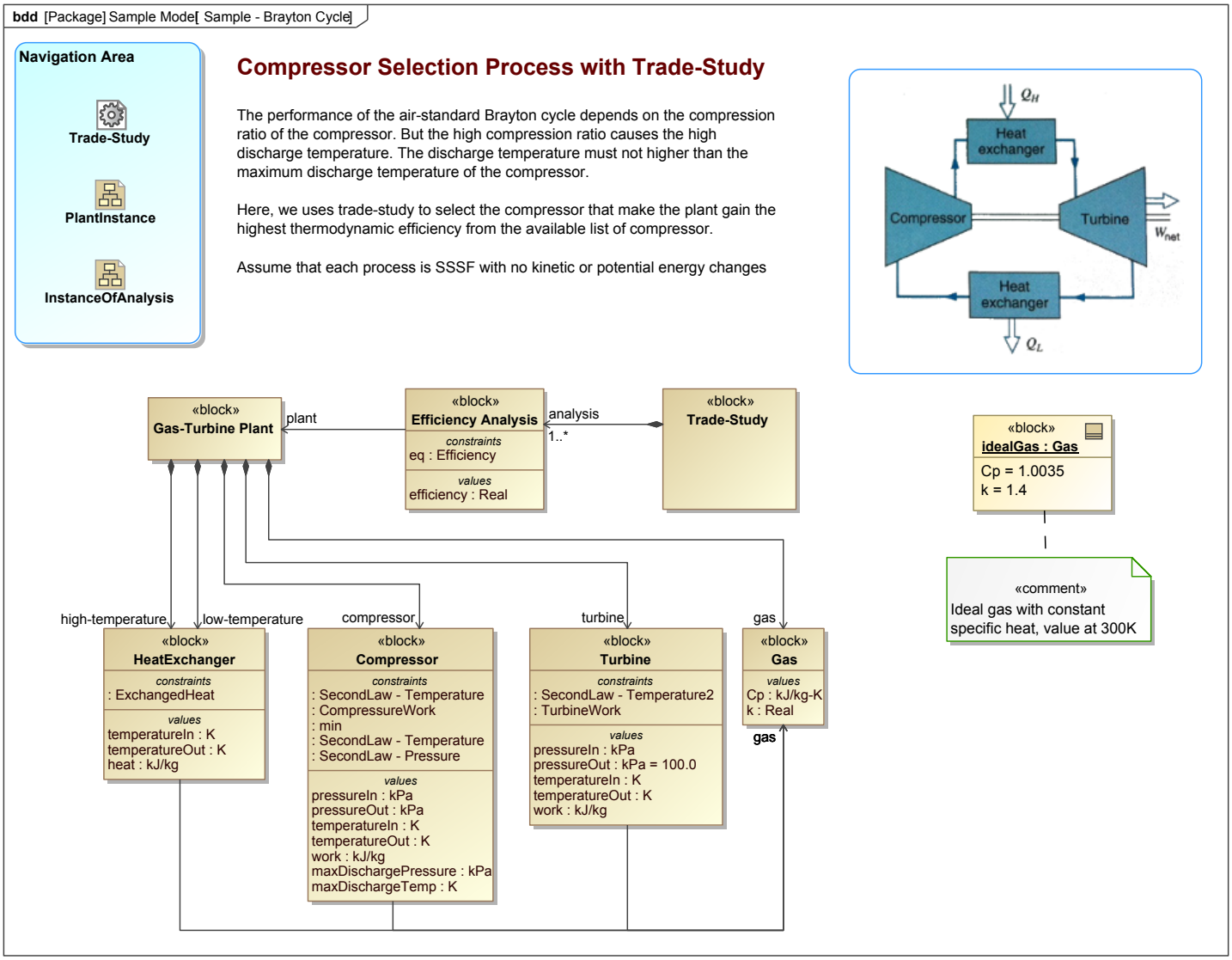


Figure 64. Sample - Brayton Cycle

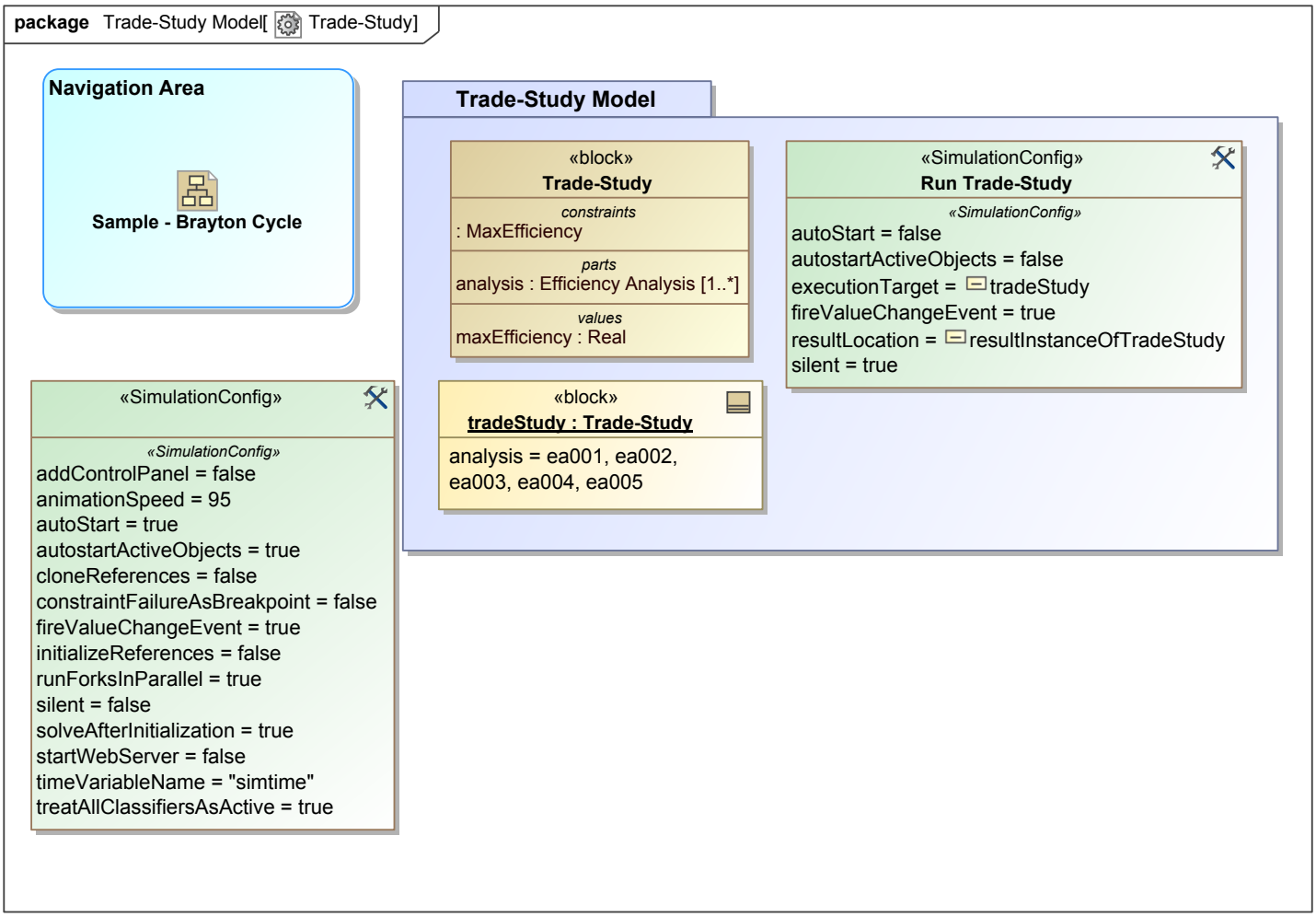


Figure 65. Trade-Study

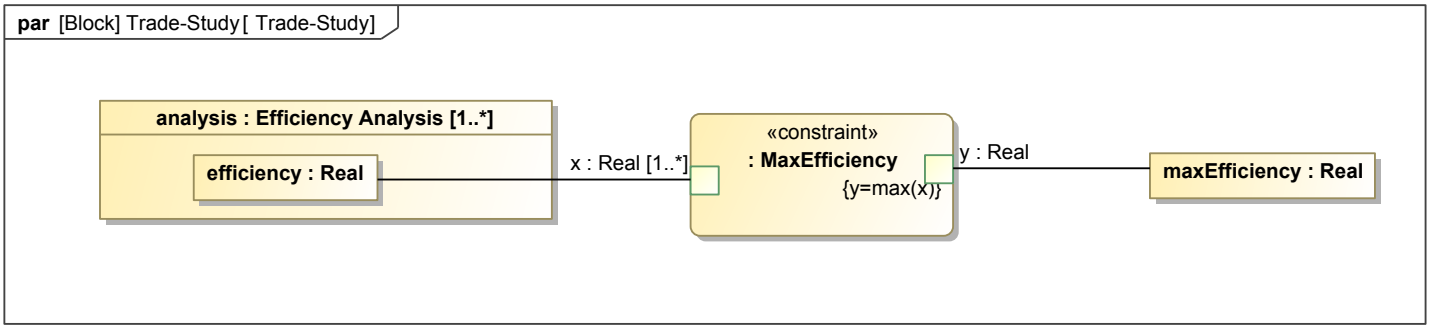


Figure 66. Trade-Study

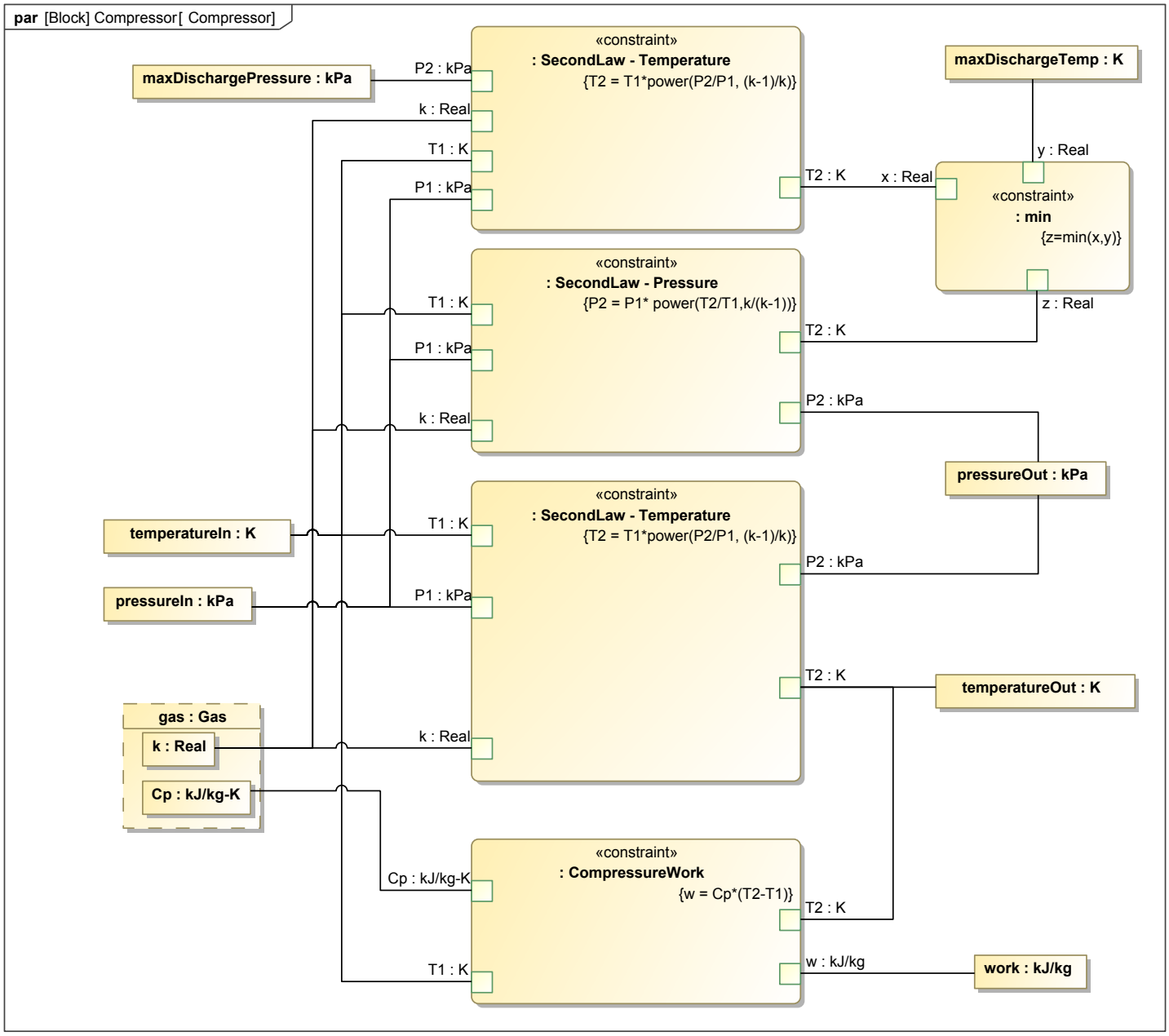


Figure 67. Compressor

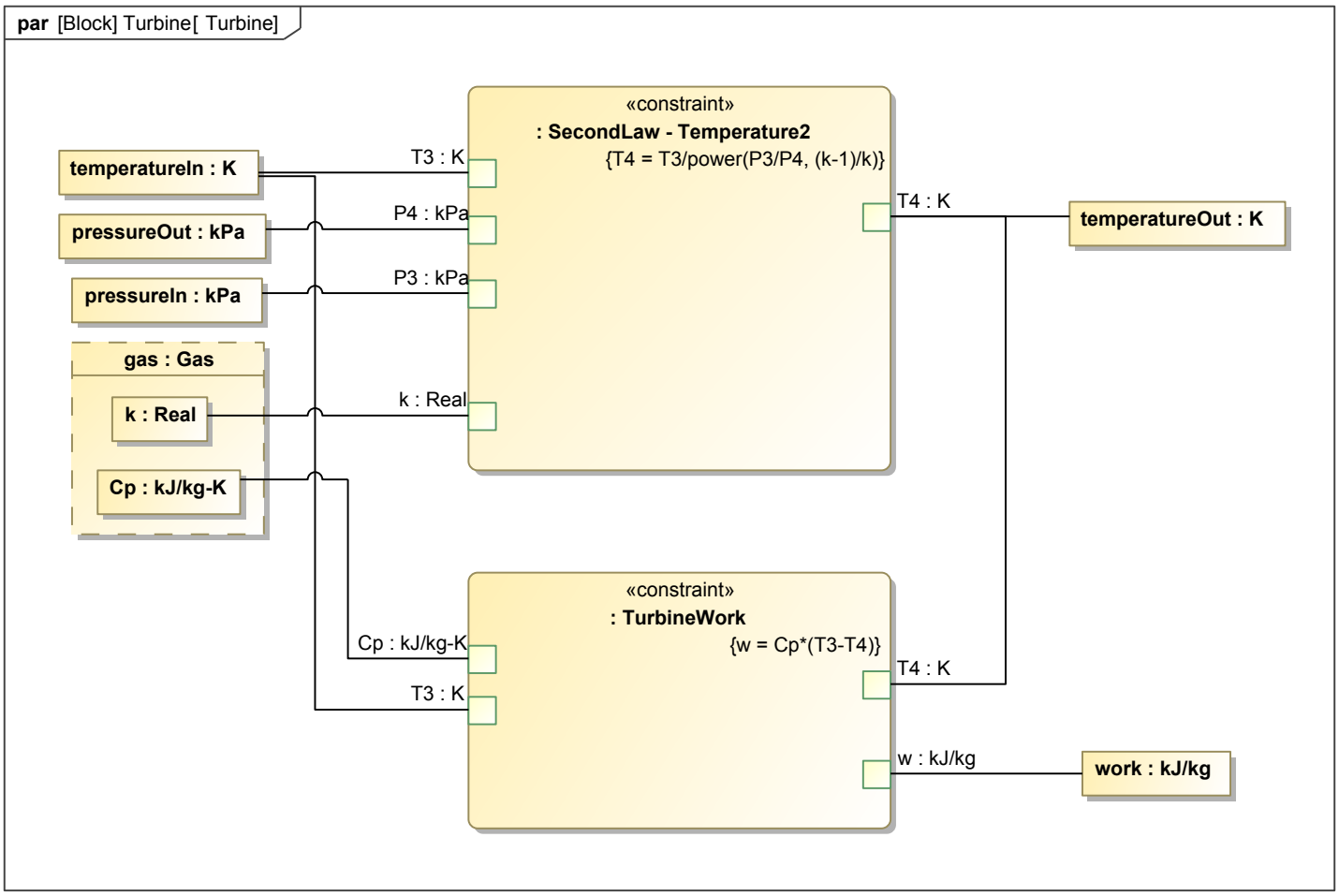


Figure 68. Turbine

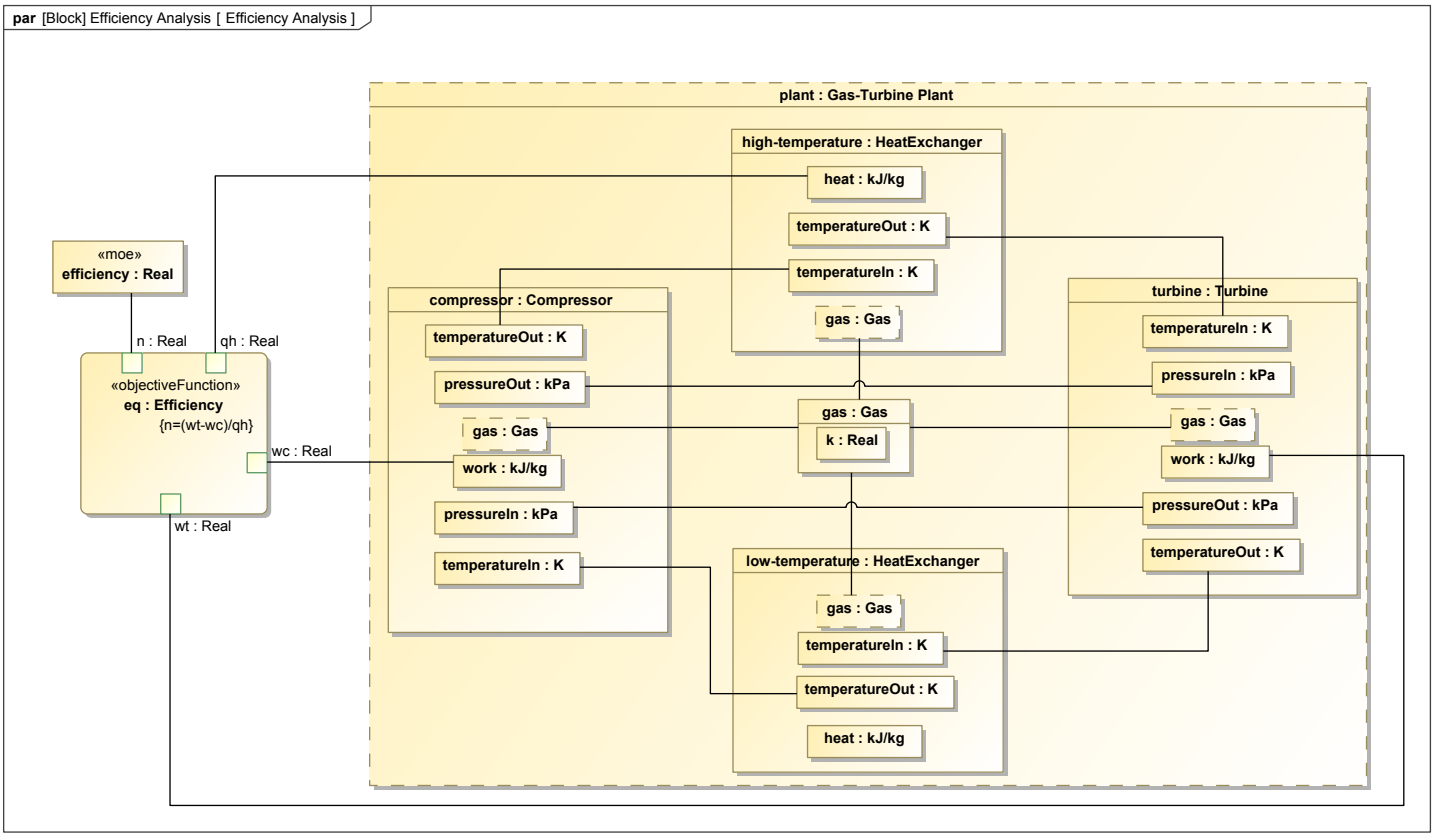


Figure 69. Efficiency Analysis

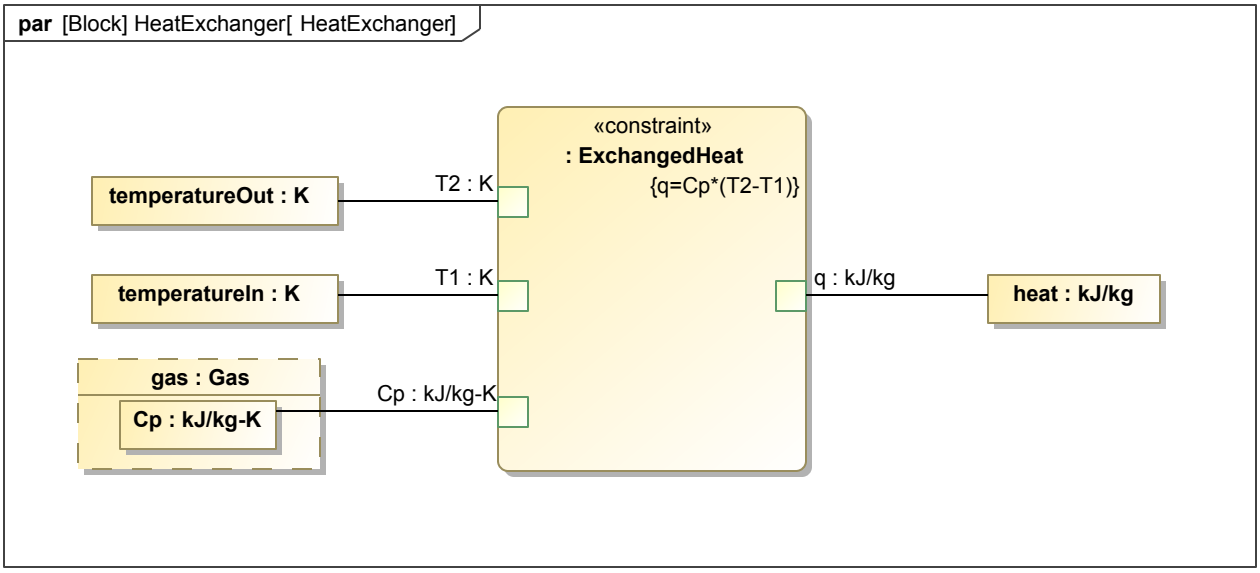


Figure 70. HeatExchanger

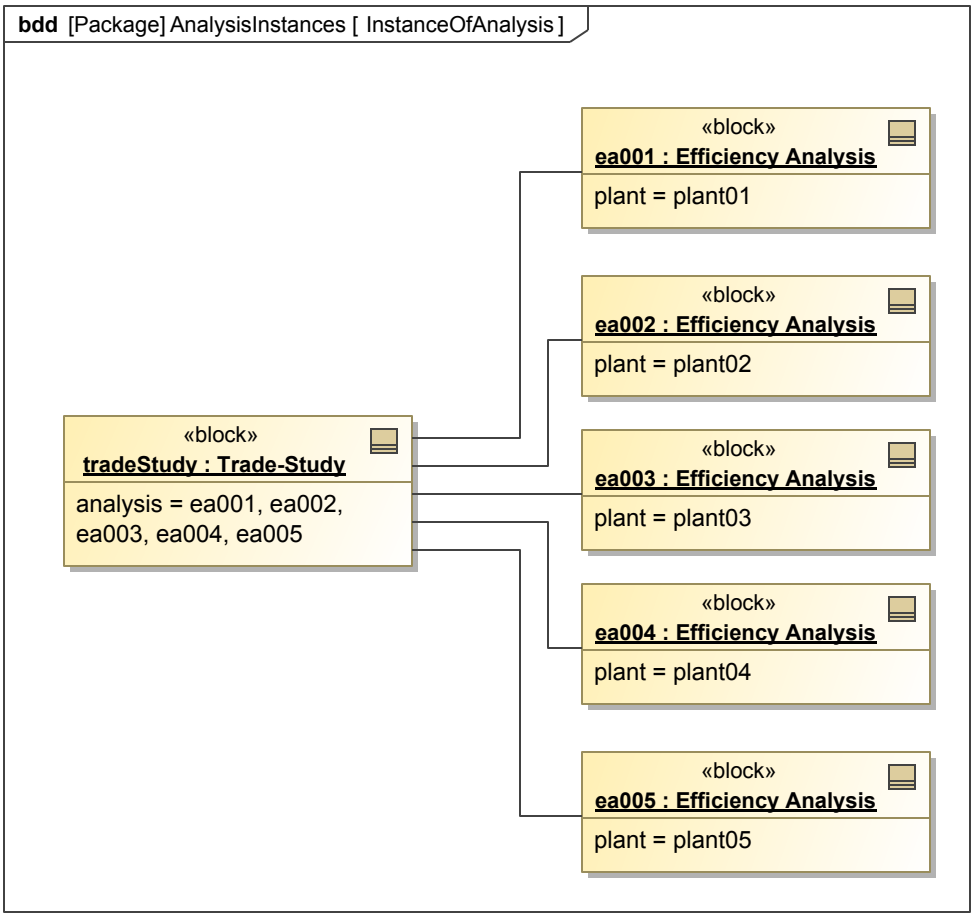


Figure 71. InstanceOfAnalysis

24.6 Known Uses

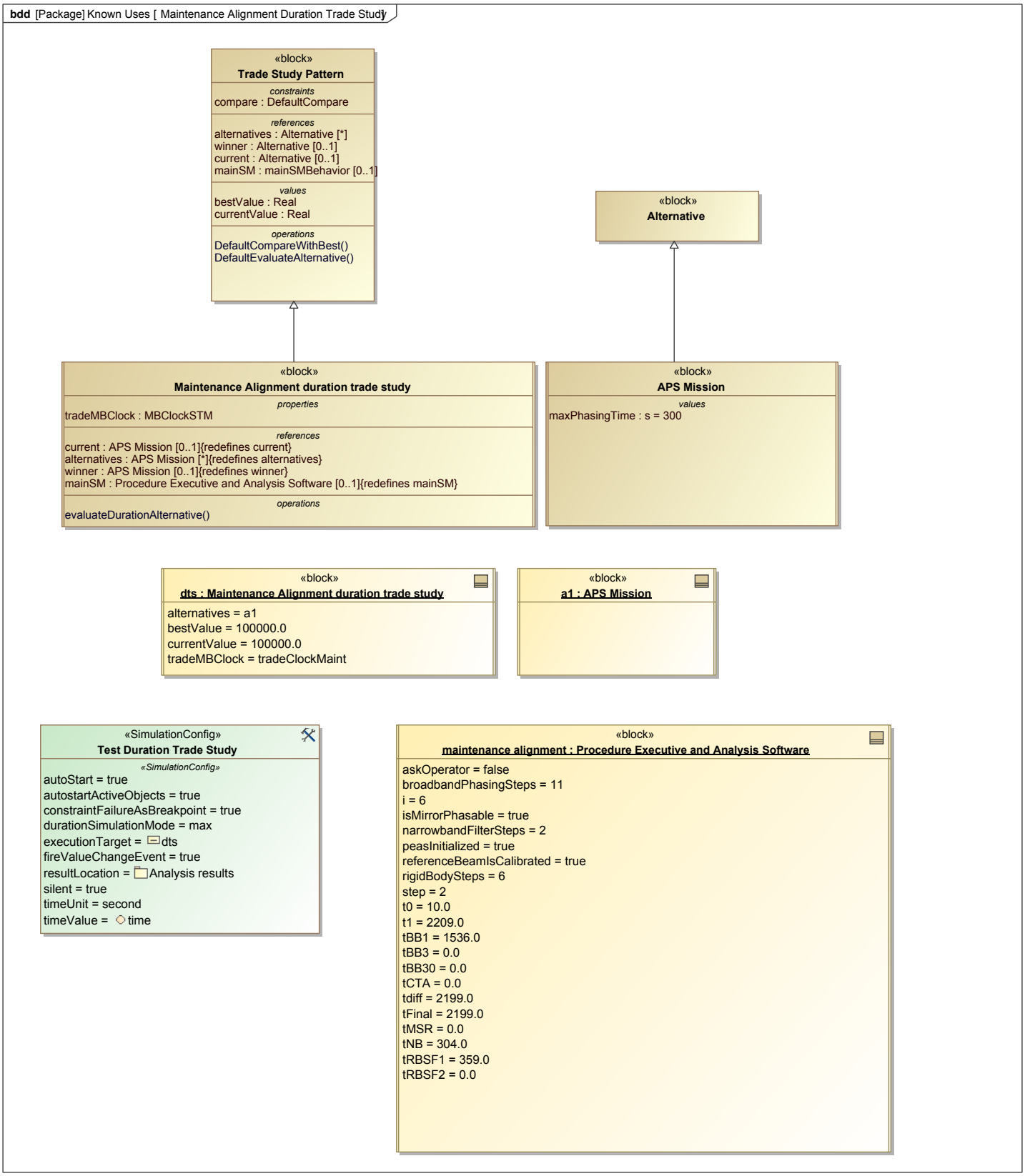


Figure 72. Maintenance Alignment Duration Trade Study

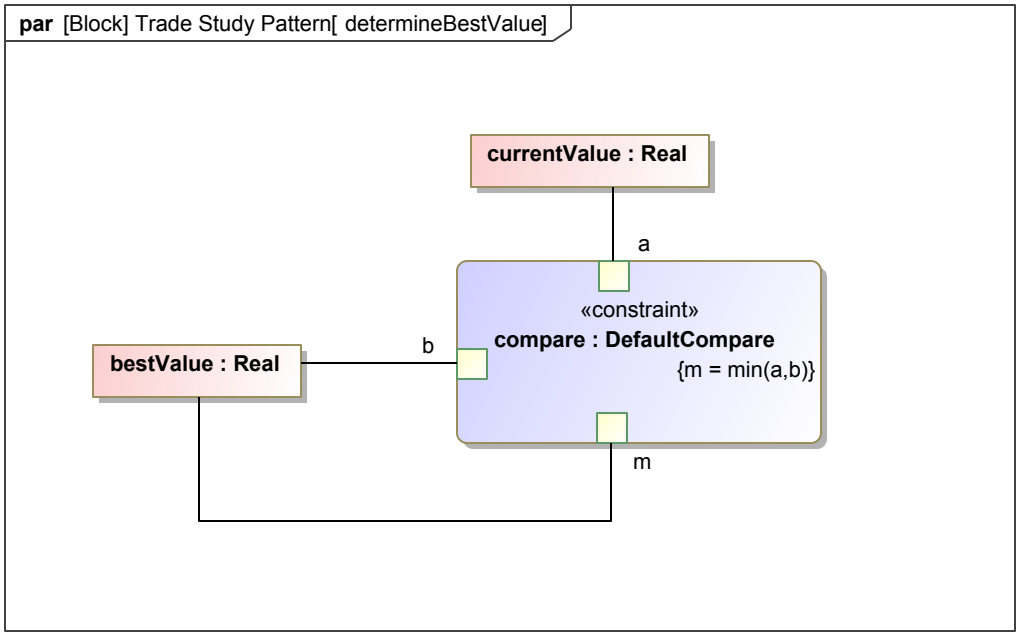


Figure 73. determineBestValue

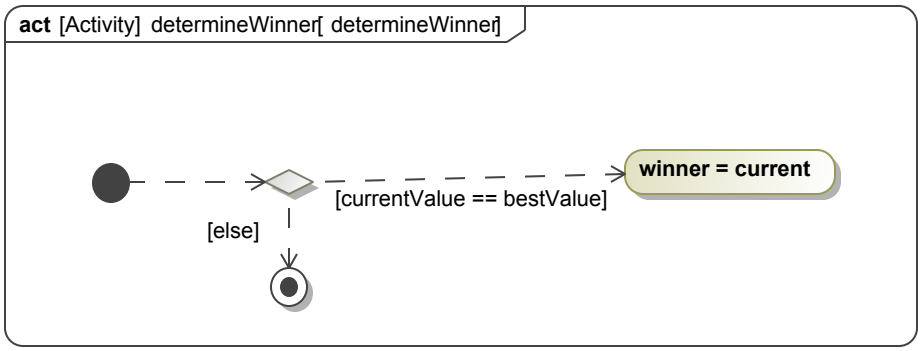


Figure 74. determineWinner

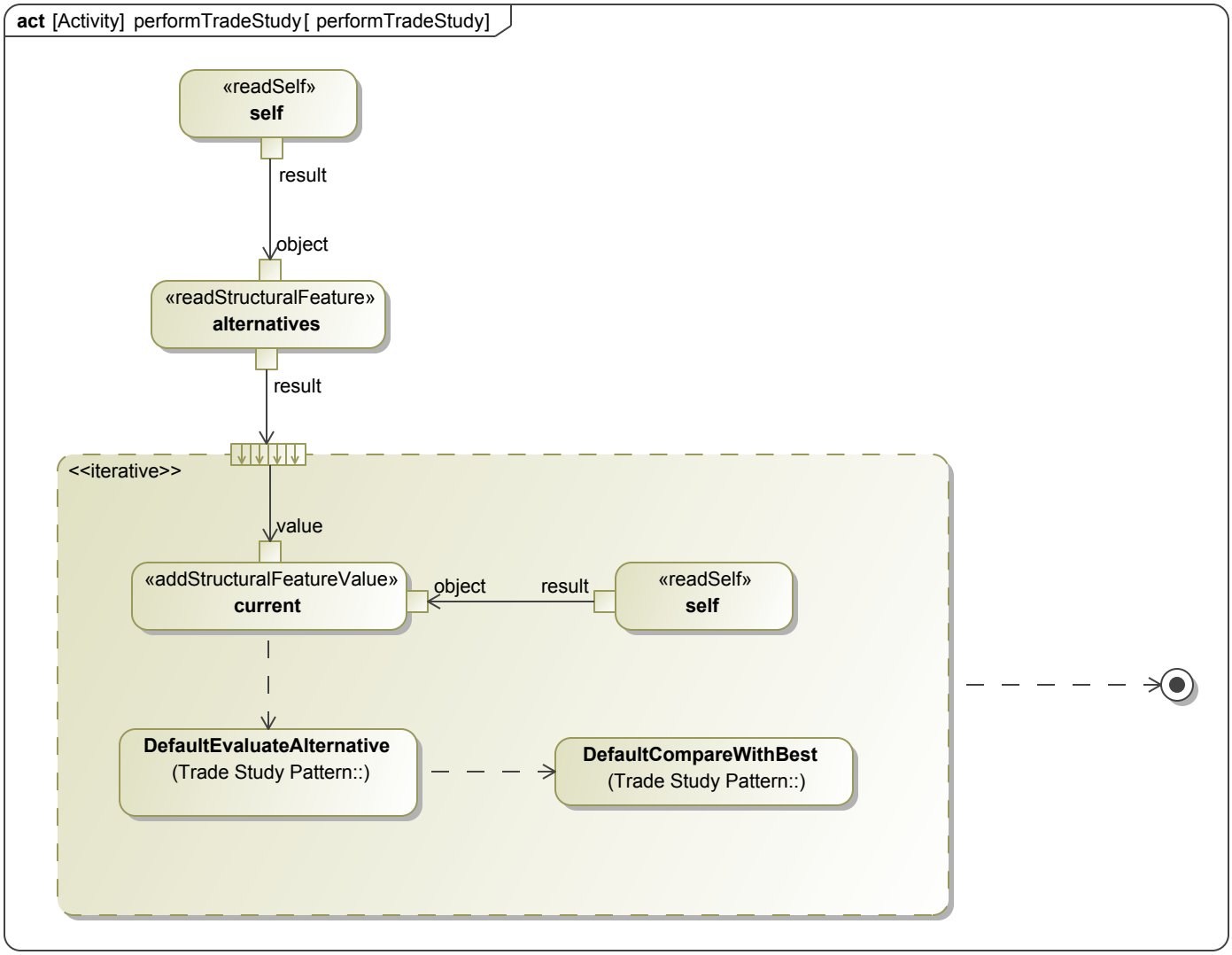


Figure 75. performTradeStudy

Table 9. Maintenance Alignment Duration Trade Study Result

Name	tFinal : Real	tRBSF1 : Real	tBB1 : Real	tNB : Real
maintenance alignment	2199.0	359.0	1536.0	304.0

Table 10. Maintenance Alignment Duration Best Trades

Name	bestValue : Real	currentValue : Real
Maintenance Alignment duration trade study at 2015.08.14 17.24	2199.0	2199.0

24.7 Analysis

24.8 Related Patterns

24.9 Tooling

24.9.1 Cameo Simulation Toolkit

25 Protocol Pattern -DRAFT-

25.1 Intent

The intent of the Protocol Pattern -DRAFT- is to provide SysML constructs to model protocols and to ensure communication about a system's features and structures. In information technology, a protocol is the special set of rules that end points in a telecommunication connection use when they communicate. Protocols specify interactions between the communicating entities, and exist at several levels in a telecommunication connection. Protocols are described in an industry/international standard (i.e. Open Systems Interconnection (OSI)). A Protocol Stack is a set of layers that transforms items to enable their exchange, such as for purposes of communication.

25.2 Motivation

An interconnection is a connection between elements, such as a connection between a spacecraft and ground system, or a connection between a power switch and an electrical load. Use this pattern to describe connections between elements, in a context, through which energy or material or information flows. This pattern describes what flows through an interface and allows capture of those attributes in a particular context.

To fully specify the interfaces on a component, the protocol elements that make up the “stack” must be defined. A protocol can be defined as a set of rules and formats (semantic and syntactic) which determines the communication behavior of entities in the performance of certain functions.

Transmission Control Protocol (TCP)/ Internet Protocol (IP) is often referred to as a stack which includes the layers through which all data passes at both client and server ends of a data exchange.

The Protocol Stack of a component performs an orthogonal part of the functions needed to transfer data.

An interface view is another abstraction approach to deal with interface complexity.

The design decisions must be considered from the perspective of different stakeholder viewpoints that may include different engineering disciplines such as electrical, mechanical, and software perspectives. An interface view presents the interface information that addresses a particular stakeholder viewpoint.

25.3 Concept

The Protocol Pattern -DRAFT- documents the definition of protocols and stacks in SysML, and interrelationships between protocols and stacks. The Protocol Pattern diagram displays the relations between hierarchical protocol interfaces. The Stack Pattern diagram displays the relations that could exist between protocol layers and stacks. The Interconnected Protocols diagram displays the connection between protocol interfaces.

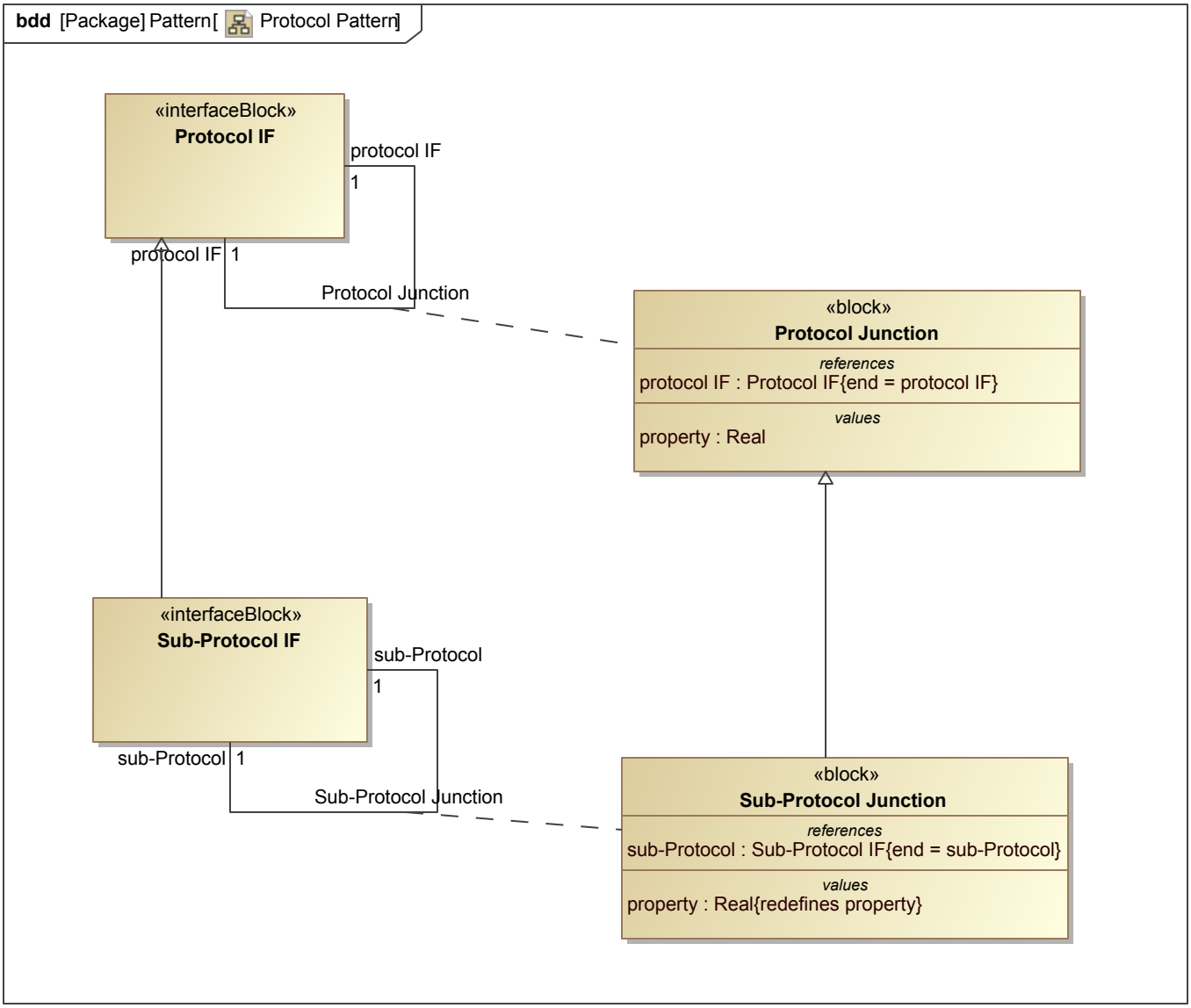


Figure 76. Protocol Pattern

A protocol in Sys

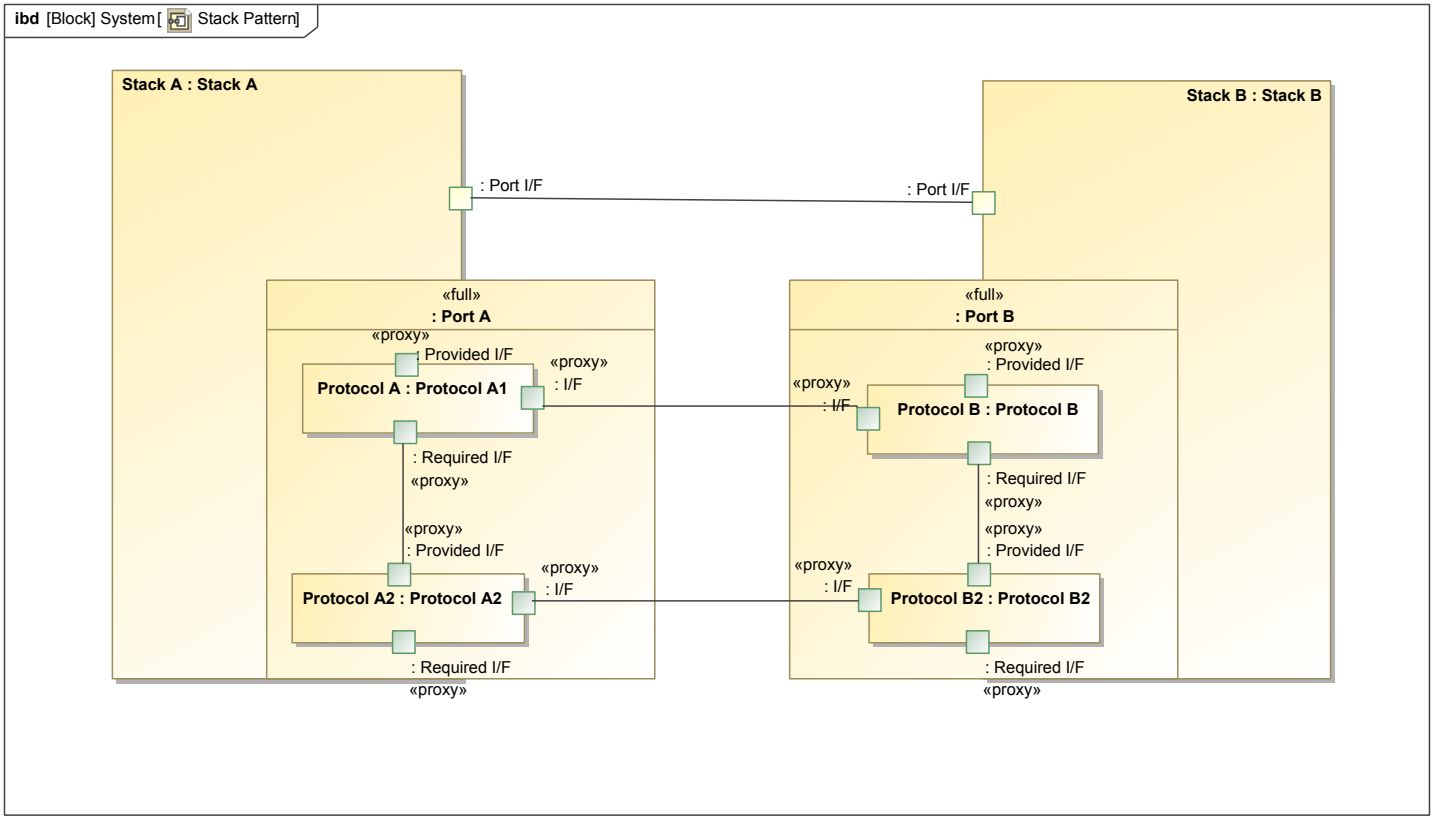


Figure 77. Stack Pattern

A protocol is a special set of rules which specifies interactions between communicating entities and at various level of communication. A protocol stack is a group of protocols that run concurrently and are utilized for the implementation of a network protocol suite.The protocols specified in the stack determine the interconnectivity rules for a layered network model. A defining feature of a stack is that the protocols must be interoperable and allow for vertical and horizontal connection. The protocol can connect vertically between the layers of the network, and horizontally between the end points of each transmission segment. In total, the protocol stack allows for the usage of various protocols that sets the boundaries of network activities.

A protocol can be represented in SysML as blocks and ports. A protocol stack is represented by a block (i.e. Stack A,Stack B). A stack owns a group of protocols, and is represented through a port. The full port which is typed by a block (i.e. Port A,Port B) possesses part properties. Protocols can be modeled as part properties of the stack. The end points of each transmission segment is defined by proxy ports (i.e. Protocol A, Protocol A2, Protocol B, Protocol B2).

Communication can exist among the protocol stacks through the usage of either full or proxy ports depending on the flow of information.

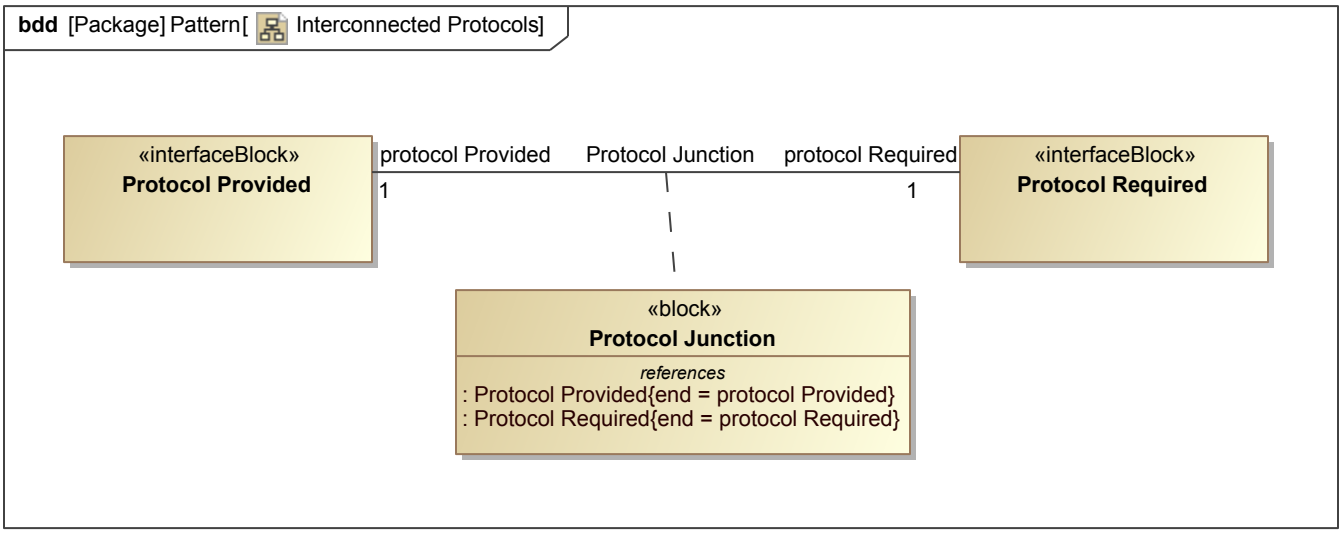


Figure 78. Interconnected Protocols

25.4 Consequences

There are several trade-offs to consider when configuring the Protocol Pattern -DRAFT- pattern to the modeler's system.

Action	Consequence

25.5 Implementation

25.6 Known Uses

Examples of real usages of the pattern.

25.7 Analysis

How this pattern helps to analyze a system model

25.8 Related Patterns

Table 11. Related Patterns Table

Pattern Name	Similarities	Differences

26 Guidelines for Modeling Non-Functional Aspects

26.1 Quality of Service

26.1.1 How do I define Quality of Service?

SysML activity diagrams offer only a rate to define details of a pin. Often more QoS are needed, like latency, jitter, clocked. The solution is to define a stereotype Qos with the properties clocked, jitter, latency, which can have different values for every Pin. If the QoS is valid for both ends of the edge, the edge itself is stereotyped, as suggested already by C.Bock. The main point of discussion is, if the pin of the action or the parameter of the activity shall be stereotyped. The correct approach seems to stereotype the parameter and the tool shall propagate it to the associated pin -. SysML status • SysML only provides only <> stereotype which extends Activity Edge and Parameter. • Allocation of Ports to Pins not addressed in SysML standard 1.1 • Synchronization of Parameter and Pin is tool-dependent. • UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms

26.1.2 How does it relate to Parts and Ports?

Activities are as usual allocated to parts or blocks. Each pin can be allocated to a flow port. In case pins are bundled on a port they are allocated to the same port. The allocation of pins to ports is optional. If there is a one to one mapping the data type of the flow port and the object node have to be the same. Allocation ObjectFlow to ItemFlow The ObjectFlow (Edge) describes that in the context of an Activity the output of one Action is bound to the input of another action. In the context of a block a item flow describes the flow of an object from one part or port to the connected part or port. The allocation of the ObjectFlow to an ItemFlow defines which ObjectFlow corresponds to which ItemFlow in a given context. Supplier and producer and context need always be defined. Allocation Pin to Port (not addressed in SysML standard 1.1) The pin defines which objects flows in/out of an action from a functional point of view. The port defines which object flows in/out of a block from a structural point of view. The allocation of pin to port defines the mapping of functional to structural view, independent of a context. The supplier and producer need to be known, e.g. when certain data flows over an Ethernet port but it is irrelevant who is connected to it. • Activities are allocated to blocks if the allocation is true for all parts of this block • Actions are allocated to parts if the activity is only relevant for a particular part. • Block operations to parts This is particularly the case when sequence diagrams are used to describe behavior rather than activity diagrams. Operations of a block (the whole) in a sequence diagram can be allocated to its parts, it is composed of.

27 Domain Specific Model Extensions

27.1 Additional stereotypes

• Blocks which are used as a context for parametrics are stereotypes analysis context to distinguish them properly from normal blocks. otherwise there is always <> in the name of the block • blocks which are used to type standard ports in order to realize interfaces and group ports, and represent a collection of interfaces are stereotyped <> • blocks which act as a grouping mechanism of a system which is distributed within another system (e.g. an entertainment system in a car) but has its own lifecycle and product tree, yet is not necessarily co-located is stereotyped <>.

27.1.1 Where do I put (new) domain specific model elements, like stereotypes?

Create a Profile package.

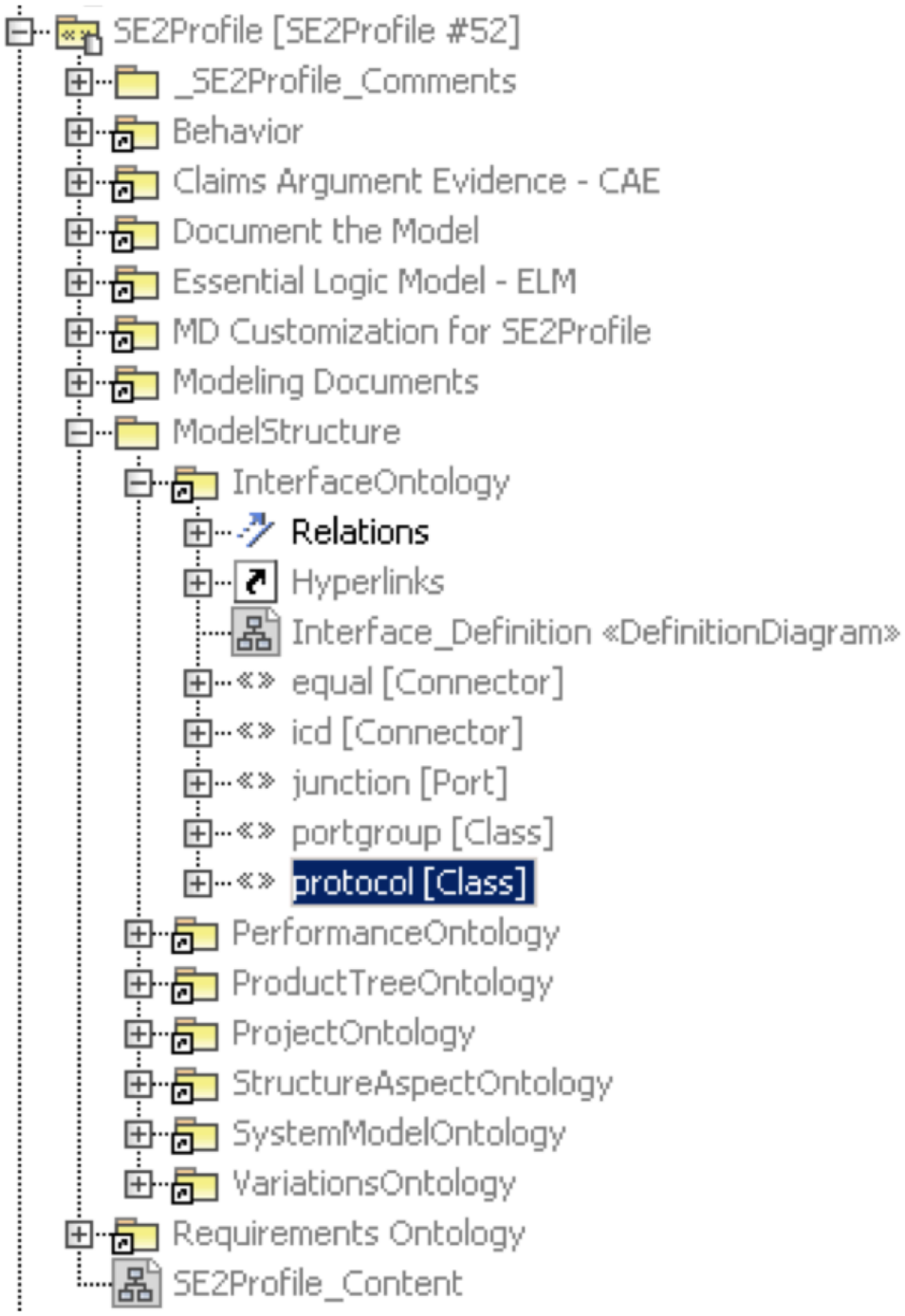


Figure 79. Containment tree of SE2 profile

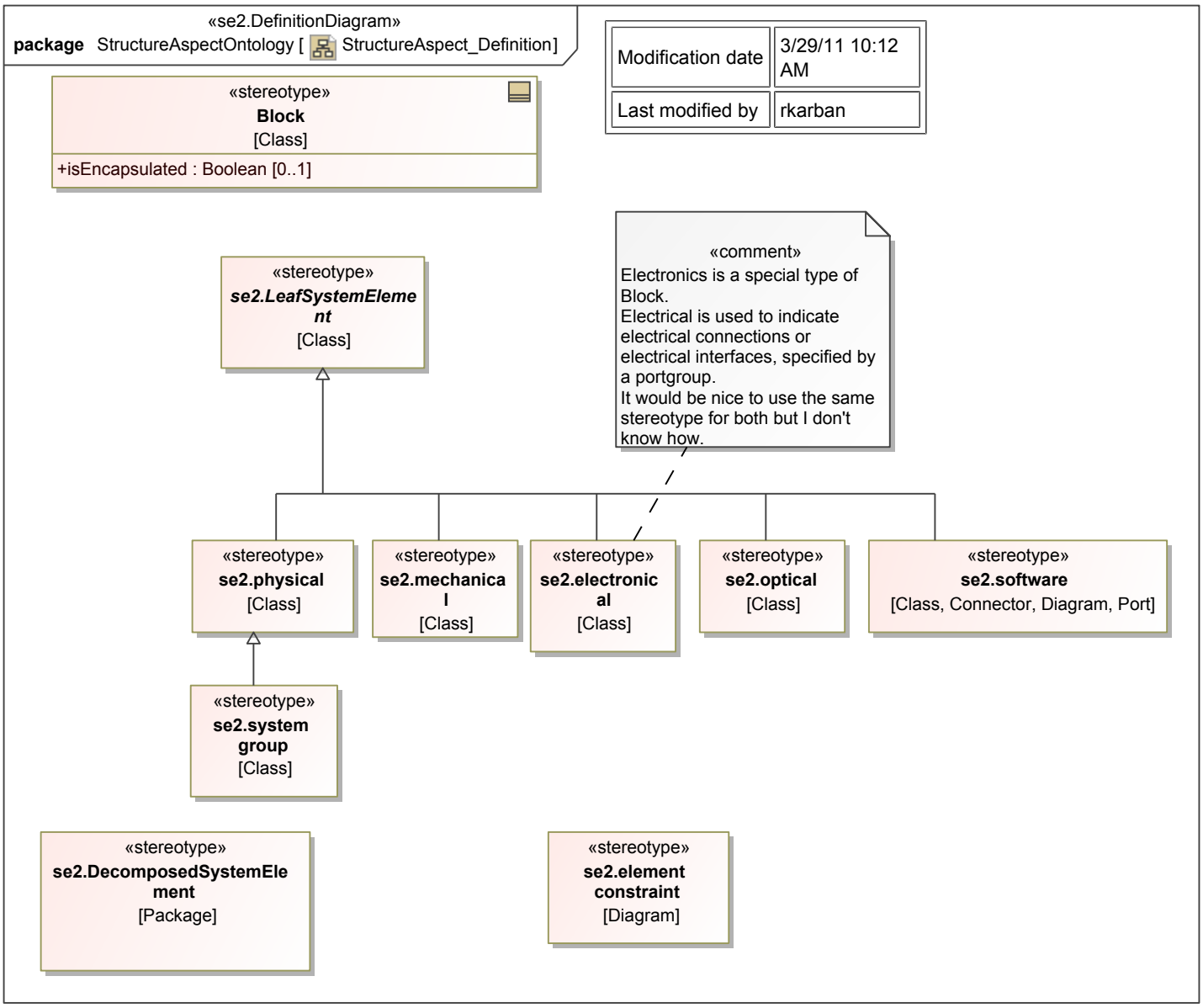


Figure 80. StructureAspect_Definition

27.2 Modeling domain specific Quantities, Units, and Values Types

Create a package, stereotyped <> for domain specific value types. If you define your domain specific units based on the QUDV and SI Value Type ontology and profile, e.g. Jansky. You have to use the QUDV, SI Value Type libraries. In the package Factors you find four factors for the quantities and units. In the package Quantities you find w derived quantities. In the package Units you find Jansky as DerivedUnit.

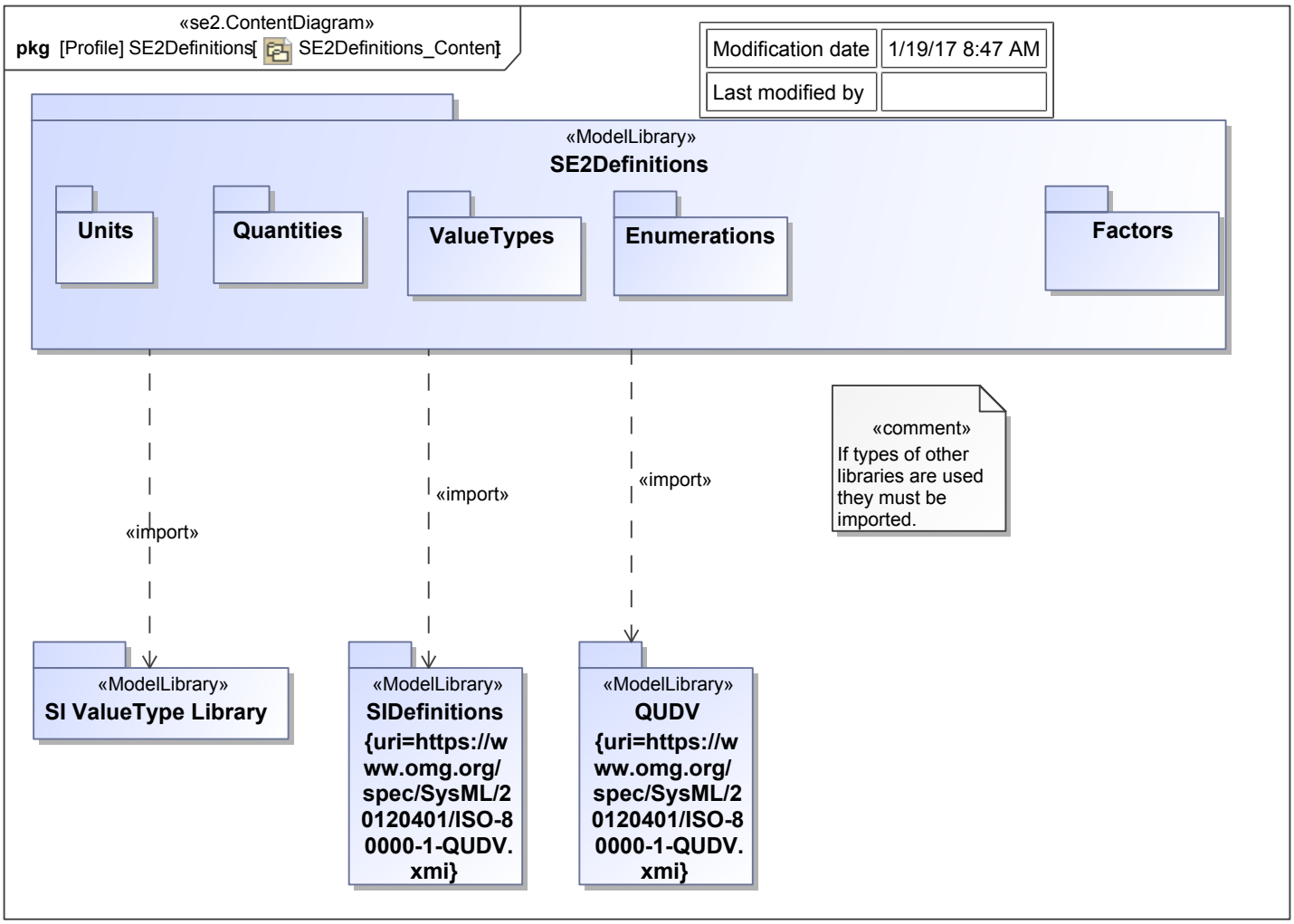


Figure 81. SE2Definitions_Content

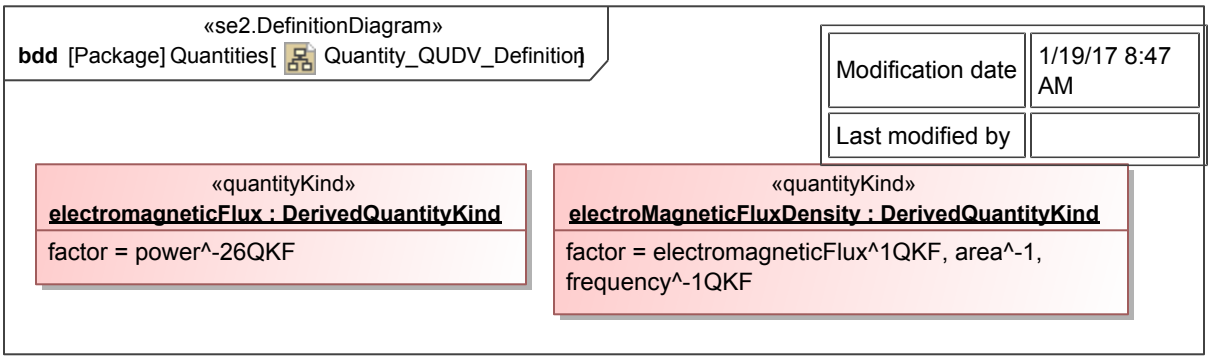


Figure 82. Quantity_QUDV_Definition

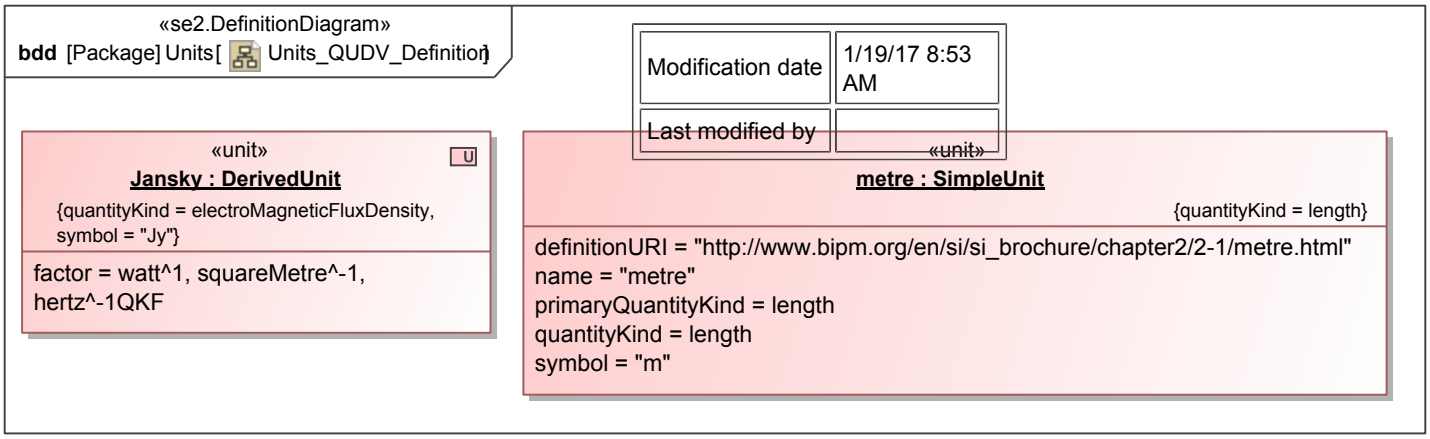


Figure 83. Units_QUDV_Definition

28 Aggregation of Convenience Pattern

28.1 Intent

28.2 Motivation

28.3 Concept

28.4 Participants

28.5 Simulation

28.6 Consequences

28.7 Implementation

28.8 Known Uses

28.9 Analysis

28.10 Related Patterns

29 Guidelines for modeling requirements

The SysML specification defines the <> model element, its properties (ID, name, text), relations among requirements (e.g. <>), and relations to other model elements (e.g. <>. However, the semantic of those relations are not defined in a formal sense and are subject to interpretation. Therefore it is necessary to define some kind of heuristics, guidelines, and practices how those relationships should be used in order to have a consistent model.

29.1 Requirements Engineering Best Practices

According to Requirements Engineering best practices there are multiple levels of requirements: 1. Stakeholder Requirements are the top level of requirements. They capture the needs of users, the customer and other sources of requirements like legal regulations and internal company high level requirements. Stakeholder Requirements making the stakeholder needs "smart"; i.e. using requirements quality criteria to generate a precise and understandable set of feasible and verifiable requirements, which is complete and consistent. If we visualize such a stakeholder requirements in SysML, we use the SysML Requirement Element stereotyped by <>. 2. The next level is system requirements. The aim of system requirements is to set precise technical requirements for the system development. System requirements are derived from stakeholder requirements by considering existing technology, components and so on. If we visualize such a system requirements in SysML, we use the SysML Requirement Element stereotyped by <>. 3. The next level(s) are subsystem and component requirements. The aim of subsystem and component requirements is to set precise technical requirements for the development of a subsystem/component. Subsystem/component requirements are derived from system requirements by considering existing technology, components, and interfaces and so on. If we visualize such a subsystem or component requirements in SysML, we use the SysML Requirement Element stereotyped by <>. If it seems that you can get a complete and consistent set of stakeholder requirements or that the customer is very solution-driven it makes sense to capture also the actual objectives of the customer, why he wants the system; e.g. business goals or desired capabilities. In this case, the objective can help the engineering to find an optimal solution for achieving the objective not hidden by solution-constraint requirements. If we visualize such objectives in SysML, we use the SysML Requirement Element stereotyped by <>.

29.2 SysML for Requirements Development

Since a long time there are a lot of specialized Requirements Management tools like DOORS, Caliber, IRQA and so on. These tools are built to efficiently manage requirements information; i.e. handling attributes filtering information and establishing and analyzing requirements traceability. So, why using SysML for requirements? Well, SysML has its strength in visualizing system engineering information and emphasizing specific aspects and relations of system engineering elements. Therefore why not using the best of both world and synergizing SysML and traditional Requirements Management? Modeling and especially SysML are perfect tools for requirements development, because by abstraction and focusing on specific aspects model are a very good communication tool to achieve a common understand between the customer, user and supplier. E.g. use case diagrams and activities can be used to gain a common understand of the underlying business process. Block (and class) diagrams can be used to capture business objects and their relation. IBDs can be used to find/set the scope of the system and identify interfaces. All these models can be used as basis for requirements specification and the resulting requirements should be traced to these underlying models.

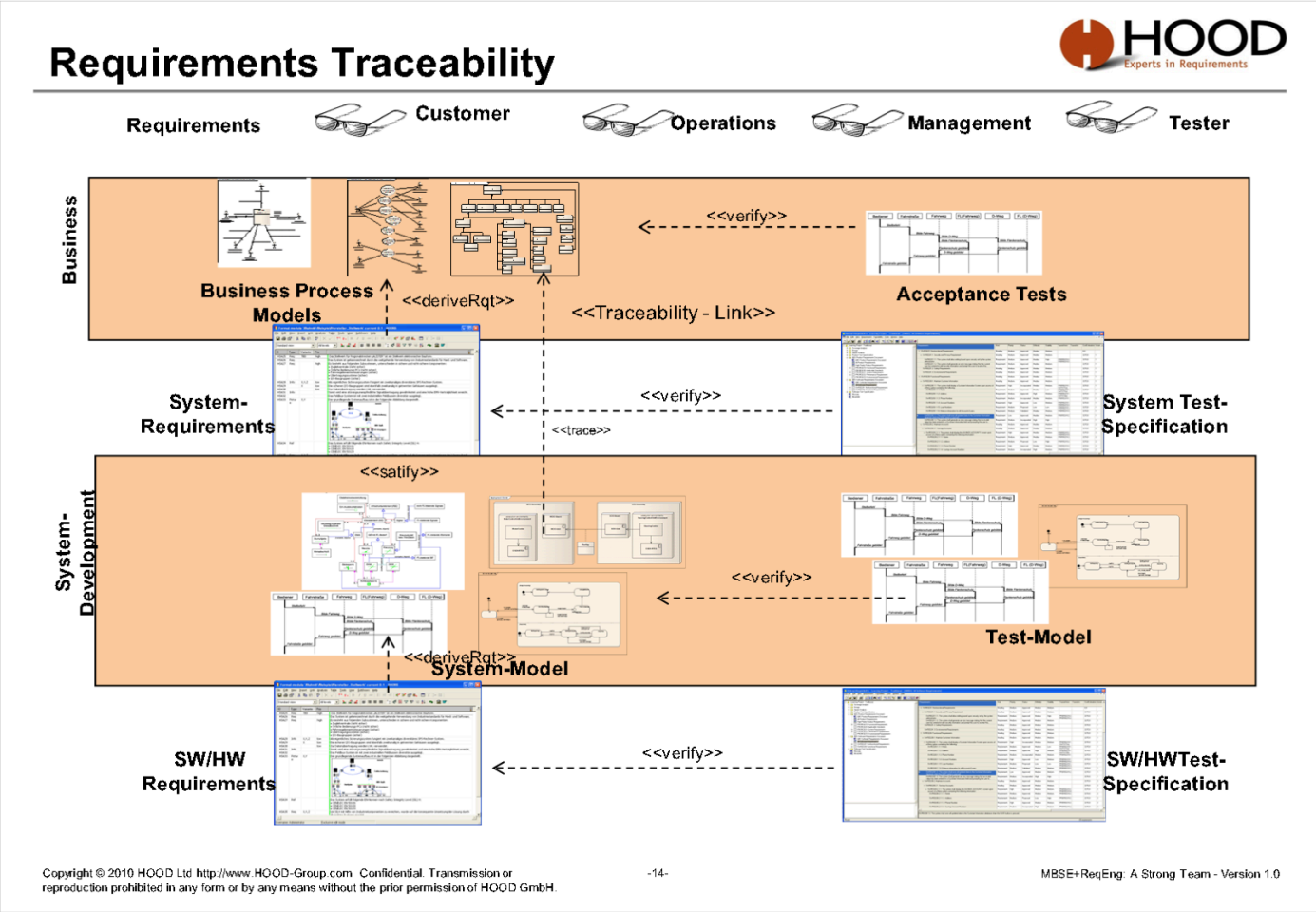


Figure 84. Requirement Tracability

Modeling and especially SysML are perfect tools for requirements development, because by abstraction and focusing on specific aspects model are a very good communication tool to achieve a common understand between the customer, user and supplier. E.g. use case diagrams and activities can be used to gain a common understand of the underlying business process. Block (and class) diagrams can be used to capture business objects and their relation. IBDs can be used to find/set the scope of the system and identify interfaces. All these models can be used as basis for requirements specification and the resulting requirements should be traced to these underlying models.

29.3 Modeling for Requirements Specification

Depending on the constraint of the project, modeling can be also used for requirements specification: • Use Cases can specify the intended usage of a system or application • Activity diagrams can specify the intended workflow of a business process • SysML Block diagrams can specify the handled business objects/information and their relations • SysML Internal block diagrams can specify the context and interfaces of a system • SysML ports can specify technical details of a system or SW interface • Sequence diagrams can specify the message flow and timing of a specific scenario • Activity diagram can specify the workflow of a use case or an service • State diagram can specify the system states and modes of the state of a particular business object • Block constraints can specify non-functional properties of a system/subsystem; e.g. system.weight <= 1400kg; system.color = anodized black • Feature properties can specify quality of service of feature/service; processOrder().availability = 99.95%

29.4 From Requirements to SysML Architecture Models

Based on existing input requirements, the system architects identify their solution “space” and develop multiple possible solutions. Next these “solution candidates” have to be evaluated and the global maximum has to be selected. SysML provides brilliant means for developing and documenting • structural aspects of the solution candidates in terms of IBDs • and behavioral aspects in terms of activities and sequence diagrams. However, it should be documented how these solution candidates implements the requirements. That’s were SysML requirements drop in: SysML requirements are perfectly suited to visualize the impact of requirements on the architecture. Since graphical SysML requirements are not suited to perform typical Requirements Management activities like filtering

requirements by complex attribute filters and analyzing requirements links graphs, our suggestion is to use SysML <> to visualize requirements managed in a traditional Requirements Management tool, which provides a tabular form. However, in the future the capabilities of the modeling might change and make a separate RM system superfluous.

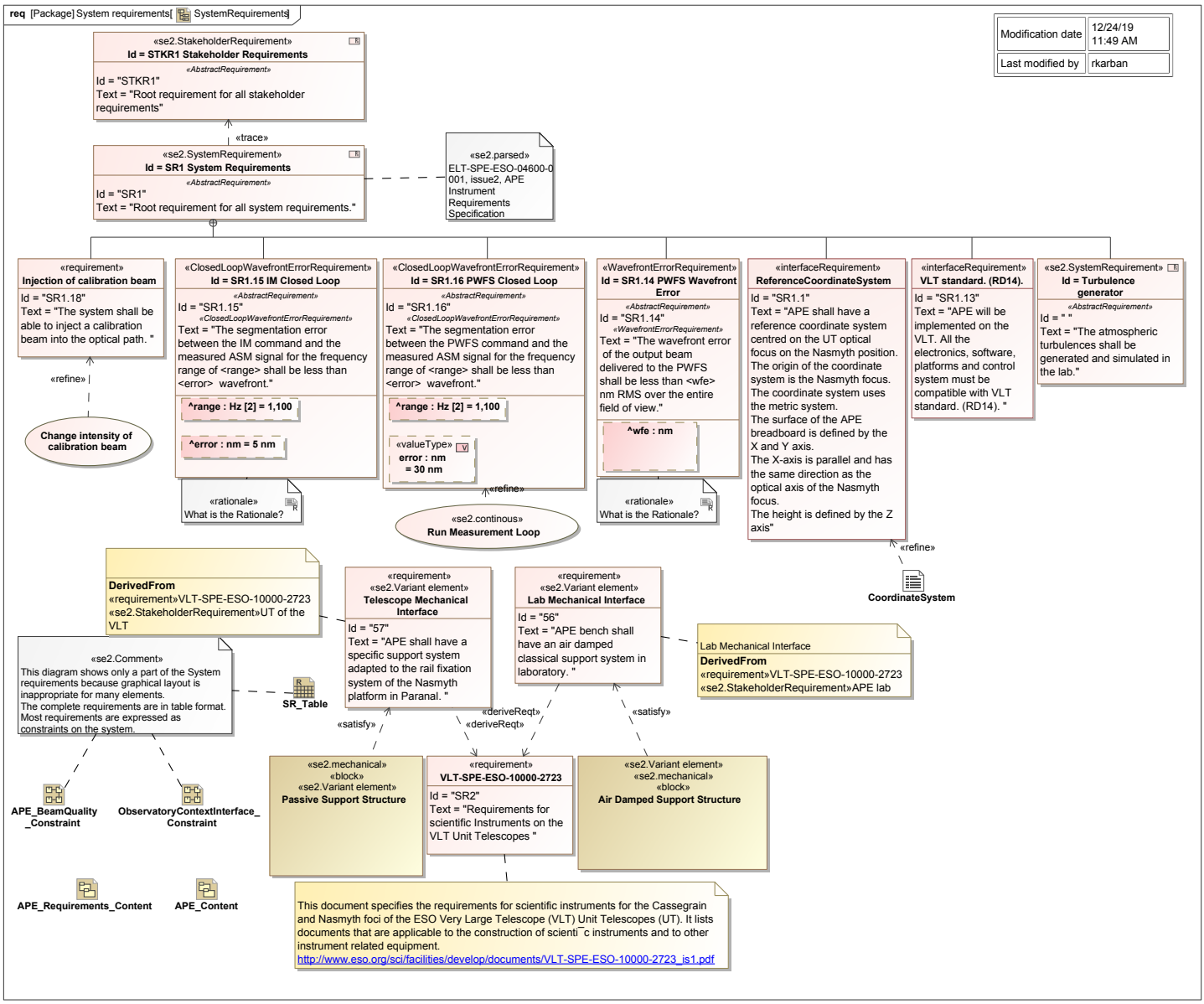


Figure 85. SystemRequirements

29.5 Guidelines for modeling the system requirements

- SysML requirements are not a replacement of Requirements management (RM) tools but a visualization aid for architectural important requirements.
- Distinguish Objectives, Stakeholder Requirements, System Requirements and Analysis elements (e.g. Use Cases) • The objectives describe the major goals of the project. They are modeled with requirements elements, stereotyped objective.
- The stakeholder requirements describe the system from the perspective of the stakeholder, mostly technology independent. They are modeled with requirement elements, stereotyped as stakeholderRequirements. Stakeholder Requirements are <> to objectives.
- Requirements shall be decomposed with <> (i.e. nesting) only on the same abstraction level as a purely organizational mean until they are verifiable
- Use <> among Stakeholder and System requirements because they are tightly coupled and cannot change without impact on each other. System requirements restate User Requirements from a System (Technical) perspective. Usually they assume a design decision which shall be justified by a <>

- Use <> to elaborate a requirement with another model element, like a use case refines a requirement, a state machine, an activity, an interaction or a table, text document and so on represented by a block (in UML you represent such elements by the artifact which is not part of SysML). Strive to use <> within the same level of abstraction.
- Do NOT use a user requirement package if the system is part of another system (system of systems). • Use <> for (system) requirements that are discovered by some design or analysis effort, which do not restate a user requirement but rather apply to the next lower system level or next level of abstraction. They shall be justified by a Rationale (e.g. Technical report) or <> to an existing design
- Use <> to integrate existing requirements, like interface requirements, international standards, policies etc. Model them with a requirement element where its name is the document number. The elements belong to the context of the system.
- Use <> to show explicitly the dependency of a requirement to a design (a block), justified by a Rationale • Use <> to model the relationship between functional and non-functional requirements.
- Use <> between top-level containers of each level to indicate that there is some kind of relationship among their nested elements.
- If it is difficult to decide if to use refine or derive - choose either. the important thing is that there is a relationship between them. The semantic difference between refine and deriveReqt is secondary.
- Use a RM tool like DOORS (if available) to manage the requirements and trace them to model elements. If some requirements are very important (e.g. security, reliability, performance, ..) they could be shown using the SysML <> elements and the <> dependency.

29.6 Background derived requirements

- The derived requirement depends on the value of the source requirement that is generally based on a form of analysis.
- The derived requirement is not a member of a 'set' of the supplier requirement in the same context as the contained relationship. The supplier requirement can be complete and contain no additional requirements with or without the derived requirement. Derived requirements are derived from different other requirements and have their own lifeline.
- When all derived requirements are met it does not mean that the others, where they are derived from, are met. In general, a derived requirement is more specific and directed toward some sub-element of the project.
- Derived requirements often occur through the process of analysis. They are the technical choices for each function. Each stage of derivation will involve some assumptions about the design of the system. A derivation often involves a specific analysis.
- A derived requirement often applies to the next level of the system. For the above example "the vehicle shall achieve a speed of 30 km/h under the specified conditions in table 1", a derived requirement may specify the weight, power, coefficient of drag, etc for the components of the vehicle in order to achieve this value.
- Derived requirements can change as a result of changes in the design, usually without reference to the customer, so it is very important to keep track of what's derived and what's not. You don't want some old internal decision constraining your future ability to meet your real customer requirements.

29.7 Stakeholder vs. System requirements

Stakeholder requirements are often part of the contract and define what the stakeholder is expecting from their (non-technical) point of view. System requirements are requirements that the real system shall meet in order to fulfill the stakeholder requirements. The system requirements refine the Stakeholder requirement from a technical point of view. You can attach a rationale to the relationship to show how you arrived at the refinement.

29.8 How do I model relationships between requirement and design element?

- If a design element fully satisfies a requirement, use the <> dependency
- If a design element exists due to the requirement (but not fully satisfy the requirement, use the <> dependency

29.9 How should I structure a requirement hierarchy?

The top-level requirements are ON THE ENTIRE SYSTEM and not packaged near a specific design solution. Requirements are put in a separate, top-level package, and recursively for each decomposition level in a separate package, which is stereotyped <>. Putting them in a separate packages is also for practical reasons to define the scope of dependency matrices. In a fully MBSE based approach there is no explicit requirements modeling necessary for each level. The requirements exist implicitly in the model as operations, constraints, functions, attributes, etc. because the design information of one level become the requirements of the next deeper level. However, it might be necessary to create explicit requirements anyhow. They are collected in the <> package. At least, the Stakeholder Requirements, System Requirements, and possibly Objectives shall be modeled, also in a full MBSE approach. Those

three types and their relations are shown for a part of the hierarchy in Figure 28. Typically, there should be a specification for each major component that is being developed either internally or subcontracted. Each specification will have its own requirements diagram and be contained in its own package. Use a trace to indicate an abstract relationship between the specifications. The requirements derivation is directly supported by some type of analysis as indicated in analysis xyx. Use the callout notation to reduce the clutter. Most relations between Objectives and Stakeholder Requirements are hidden in Figure 28 to avoid clutter. It is much more convenient to show them in an (automatically) created matrix as in Figure 29. Above a certain number of requirements, they become difficult to visualize graphically. It is better to use the tabular format as in Figure 30. APE has to satisfy certain interface requirements in order to be allowed to be installed on the telescope. SysML constraints can also be used to define quantifiable system interfaces. Figure 31 shows a parametric description of the physical interface between a VLT Unit Telescope and an Instrument placed on one of the telescope's Nasmyth platforms. The system properties of the telescope (e.g. volume, mass) constrain the properties of the attached instrument; in the example the instrument is APE. The constraints of the «constraint» NasmythInstrumentSpecification (1) evaluate the properties of APE against the properties of the telescope. On either side, those properties become in turn requirements of the lower level system hierarchy and determine the properties of its parts. In the end, a mass roll-up of the complete APE system can be done to verify that it complies with the constraints given by the specification. On the telescope side, the allowed quantities are propagated down the system hierarchy to properly reflect the design decisions taken at the highest level. For example, the telescope has an allowedMass property with a value of 8000kg. The Constraint specifies that the realMass shall be less or equal than the allowed mass. The real mass of APE is set to 5000kg. This mass can be used to establish requirements at the lower levels of APE itself, using constraints again. Another example of modeling requirements formally with constraints is the following. The original APE requirements document contains a requirement: "The quality of the input beam shall be the same as the beam fed into the wavefront sensor". The APE optical view defines the item properties, called beamIn and beamToSHAPS which represent the incoming beam and the one delivered to one sensor as seen in Figure 32. The properties of those two beams can be constrained with an appropriate constraint which defines their relationship as shown in Figure 33.

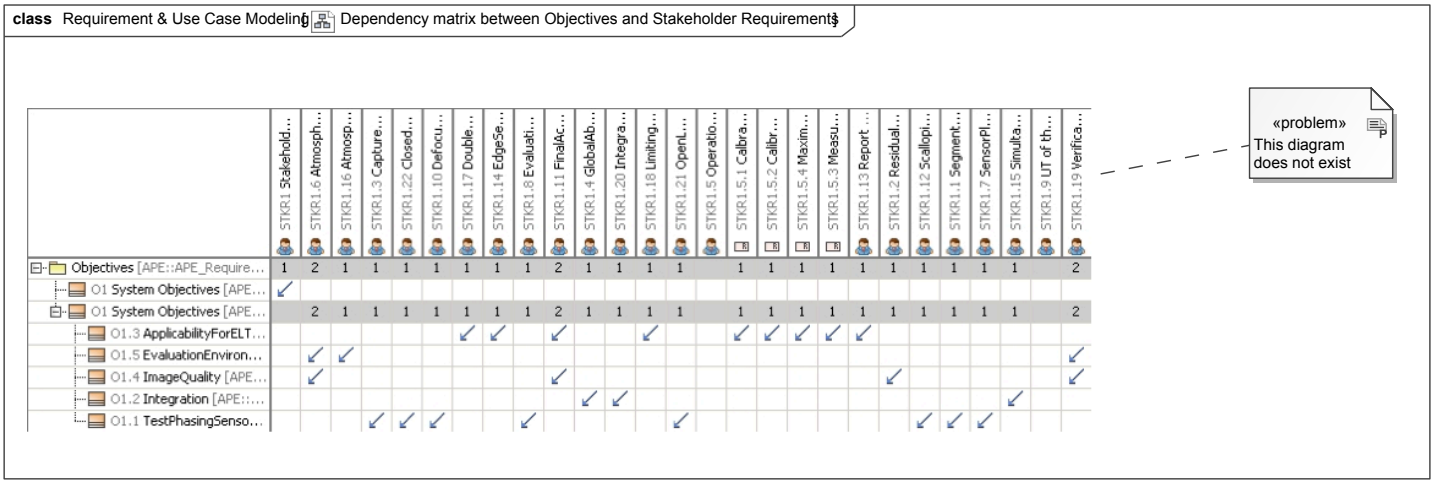


Figure 86. Dependency matrix between Objectives and Stakeholder Requirements

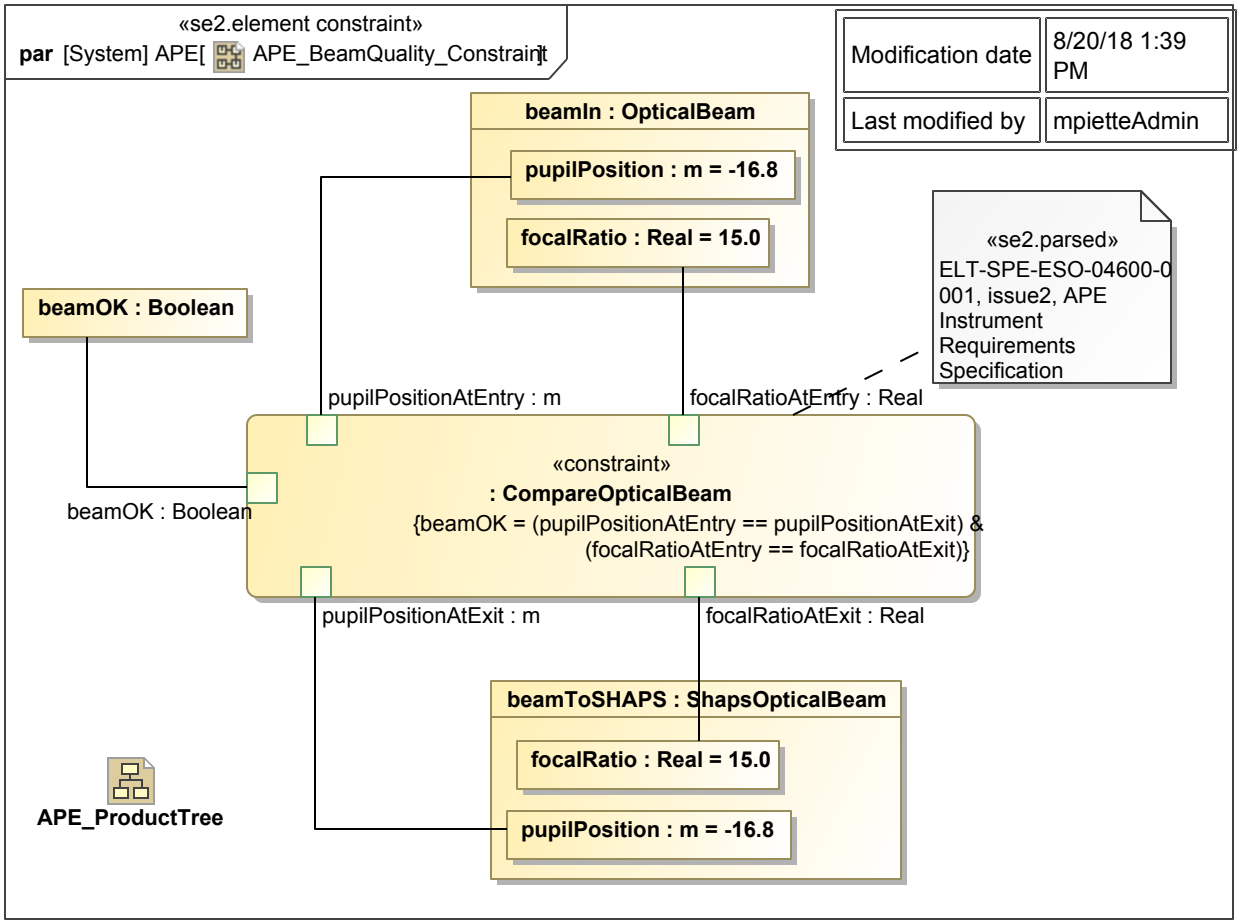


Figure 87. APE_BeamQuality_Constraint

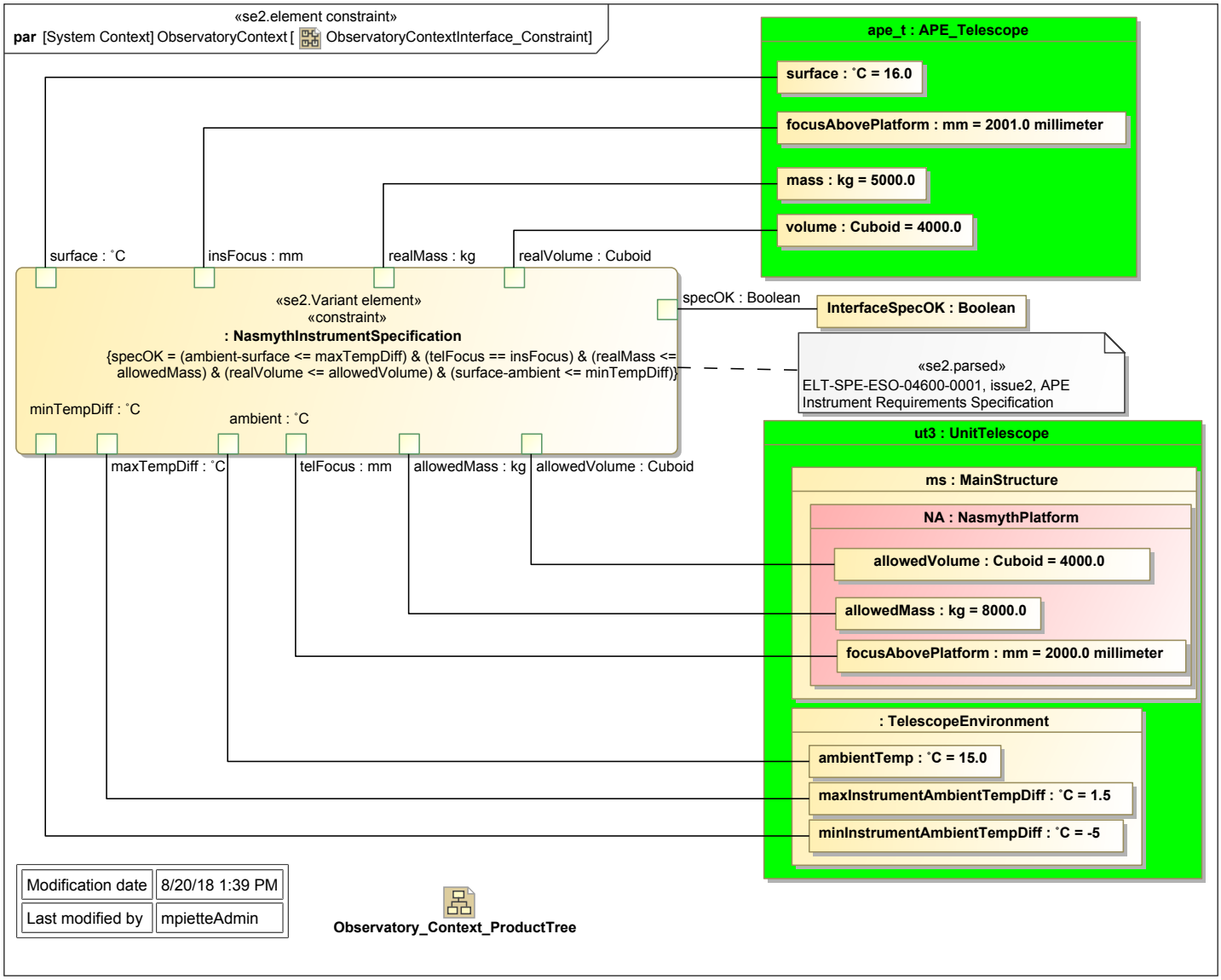


Figure 88. ObservatoryContextInterface_Constraint

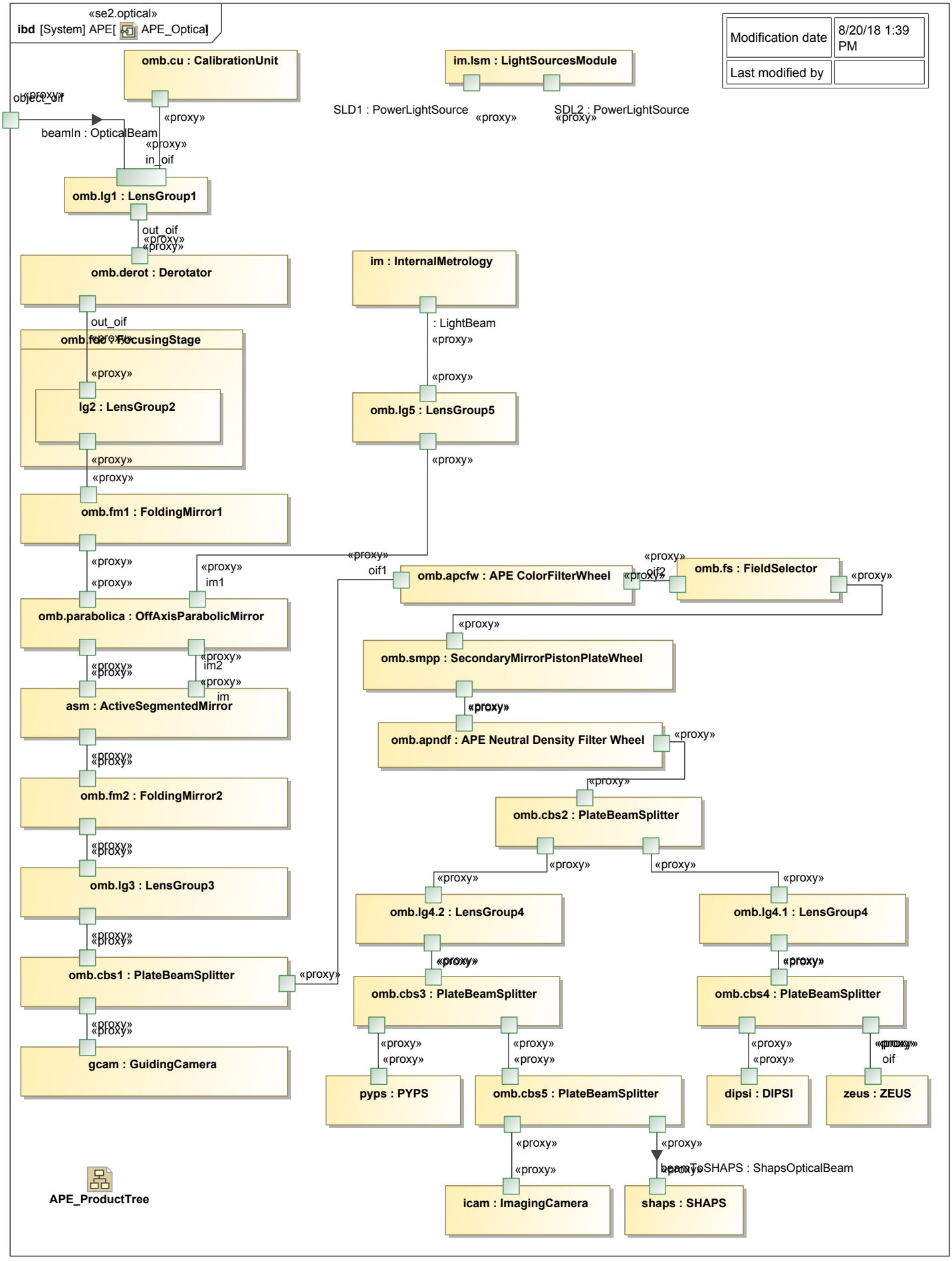

«se2.optical»
ibd [System] APE[APE_Optical]
Modification date
8/20/18 1:39 PM
Last modified by
omb.cu : CalibrationUnit
im.lsm : LightSourcesModule
object_oif
«proxy»
beamIn : OpticalBeam
in_oif
SLD1 : PowerLightSource
SDL2 : PowerLightSource
omb.lg1 : LensGroup1
out_oif
omb.derot : Derotator
im : InternalMetrology
: LightBeam
omb.fcc : FocusingStage
lg2 : LensGroup2
omb.lg5 : LensGroup5
omb.fm1 : FoldingMirror1
oif1
im1
omb.apcfw : APE ColorFilterWheel
oif2
omb.fs : FieldSelector
omb.parabolica : OffAxisParabolicMirror
im2
im
omb.smpp : SecondaryMirrorPistonPlateWheel
asm : ActiveSegmentedMirror
omb.apndf : APE Neutral Density Filter Wheel
omb.fm2 : FoldingMirror2
omb.cbs2 : PlateBeamSplitter
omb.lg3 : LensGroup3
omb.lg4.2 : LensGroup4
omb.lg4.1 : LensGroup4
omb.cbs1 : PlateBeamSplitter
omb.cbs3 : PlateBeamSplitter
omb.cbs4 : PlateBeamSplitter
gcam : GuidingCamera
pyps : PYPS
omb.cbs5 : PlateBeamSplitter
dipsi : DIPSI
zeus : ZEUS
beamToSHAPS : ShapsOptical Beam
icam : ImagingCamera
shaps : SHAPS
APE_ProductTree

Figure 89. APE_Optical

Table 12. SR_Table

Id	Name	Text
SR1	System Requirements	Root requirement for all system requirements.
SR1.1	ReferenceCoordinateSystem	APE shall have a reference coordinate system centred on the UT optical focus on the Nasmyth position. The origin of the coordinate system is the Nasmyth focus. The coordinate system uses the metric system. The surface of the APE breadboard is defined by the X and Y axis. The X-axis is parallel and has the same direction as the optical axis of the Nasmyth focus. The height is defined by the Z axis
SR1.13	VLT standard. (RD14).	APE will be implemented on the VLT. All the electronics, software, platforms and control system must be compatible with VLT standard. (RD14).
SR1.14	PWFS Wavefront Error	The wavefront error of the output beam delivered to the PWFS shall be less than nm RMS over the entire field of view.
SR1.15	IM Closed Loop	The segmentation error between the IM command and the measured ASM signal for the frequency range of shall be less than wavefront.
SR1.16	PWFS Closed Loop	The segmentation error between the PWFS command and the measured ASM signal for the frequency range of shall be less than wavefront.
SR1.17	PWFS Reference Source	The reference source shall have magnitude v, seeing s, at wavelength lambda

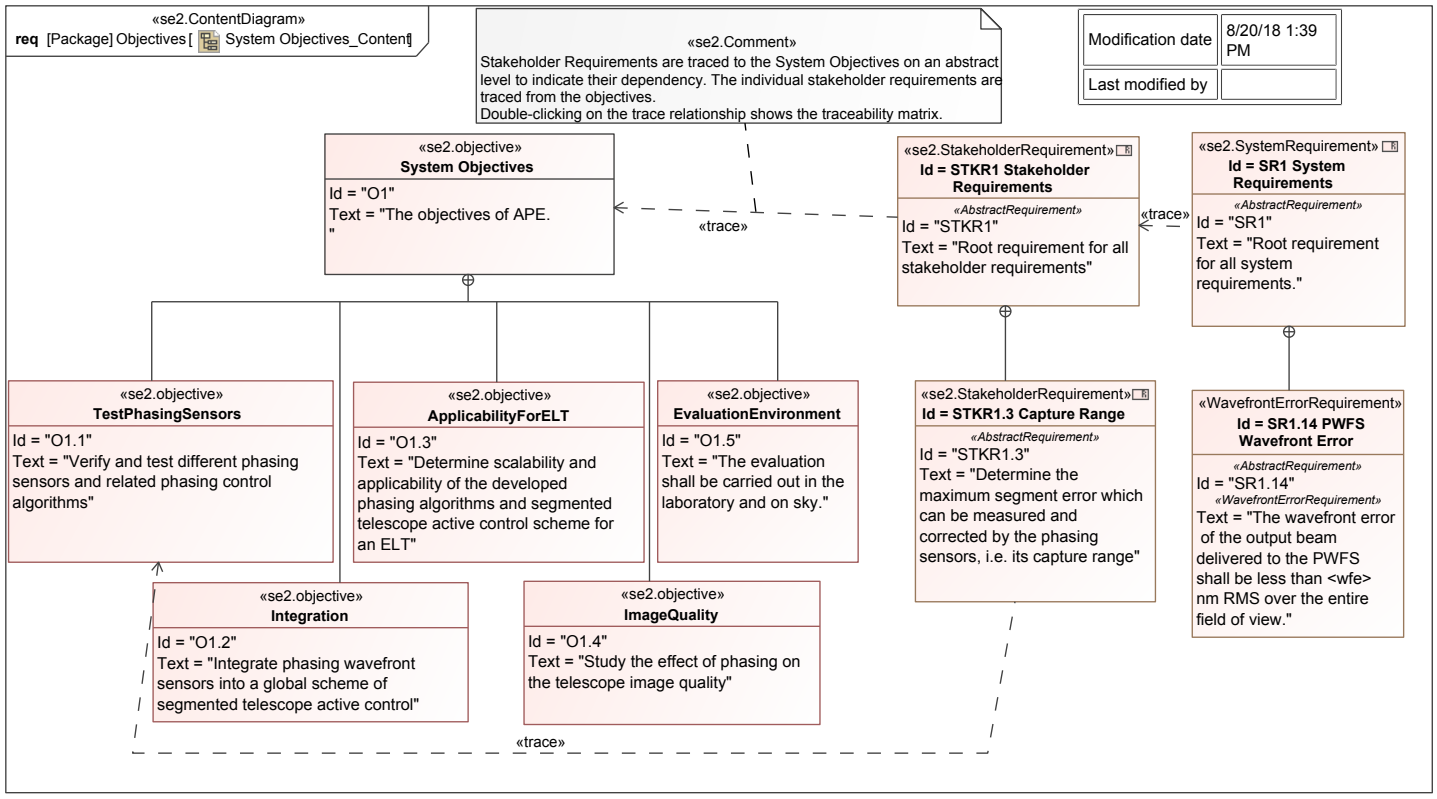


Figure 90. System Objectives_Content

29.10 Requirements quality criteria

There are quality criteria for requirements:

- A requirement must be understandable
- A requirement must be verifiable.
- A requirement should be independent of the technical realization; i.e. do NOT contain any technical details.
- A requirement must be feasible; i.e. it should be possible to model how the requirement is satisfied by other model elements.

30 Context Modeling

MOve to design management

30.1 Purpose of context diagrams of a system

The purpose of the context diagram is to determine the system scope. It identifies the parts which belong to the system and which do not – the system boundaries. The system context shows the system as a black box and all interacting external elements. All external elements are shown as actors; they can be systems, persons, etc. In order to model the context of the system, a special <> block is created which acts as a container of the system and all external actors. In Figure 19 it is called APEContext. The system being modeled is called APE with the stereotype <>. External non-human entities are blocks stereotyped as <>. Human entities are represented by stick-men <>. The actors are involved in use cases requesting a service from the system. When breaking down the system each decomposed system element has a context at decomposition level. Taking the example of a control system, the actors are the ones which are controlled but still inside the system's boundaries. The context diagram should show the concerned system and its relationship to external entities. That does not mean that one needs a context diagram for each and every subsystem! At top level of the model, the real system is shown in its context. The connections of the system with the outside world are shown in the IBD of the special context block (Figure 20 shows the optical context). For each engineering discipline a different context IBD can exist. Note that APE can exist in different contexts. Here it is shown in the observatory context, where LensGroup1 of APE (mounted on the OptoMechnicalBench) is optically connected to the NasmythAdapter of the telescope. The NasmythAdapter provides the opto-mechanicl element which "hands" over the light collected by the telescope to the instrument, APE in this case. How different contexts are modeled is shown later. To describe for example all internal connections of APE create an IBD of APE, which has the properties Control System, ZEUS, Bench, etc. Mostly each subsystem is simply used by another system or subsystem at a higher level. Therefore you do not need repeat the same thing. i.e. create a context and describe its structure. Rather, create another IBD in a block for each subsystem as a context because the structure of this System defines also the context of each of its subsystems. However, mostly it suffices to simply reuse each block or subsystem in a higher context. Typically <> means something that is NOT modeled in the entire system. You need to address the so-called 0th level, which is the (entire) <>. It provides the top-level context for your SINGLE <>. The project thus handles ONE <>. That ONE <>may be subject to <> influences THAT ARE NOT MODELLED precisely. The detailed modeling starts at the 1st level, the <>. The mechanical IBD of the APE system (Figure 21) shows the internal, mechanical, connections of APE but at the same time is the context diagram of its parts, like ZEUS. The context diagram of ZEUS is then simply the IBD of its containing higher level, APE. All the opto-mechanical elements are mounted on an optical bench, therefore there are <> connectors between the elements and the bench. The connections are modeled at varying levels of detail. Sometimes a simple connector indicates a mechanical connection, sometimes a port represents a mechanical interface of an element.Typically, one enters a project and finds a 0th level <> that provides a context for the <> and <>.

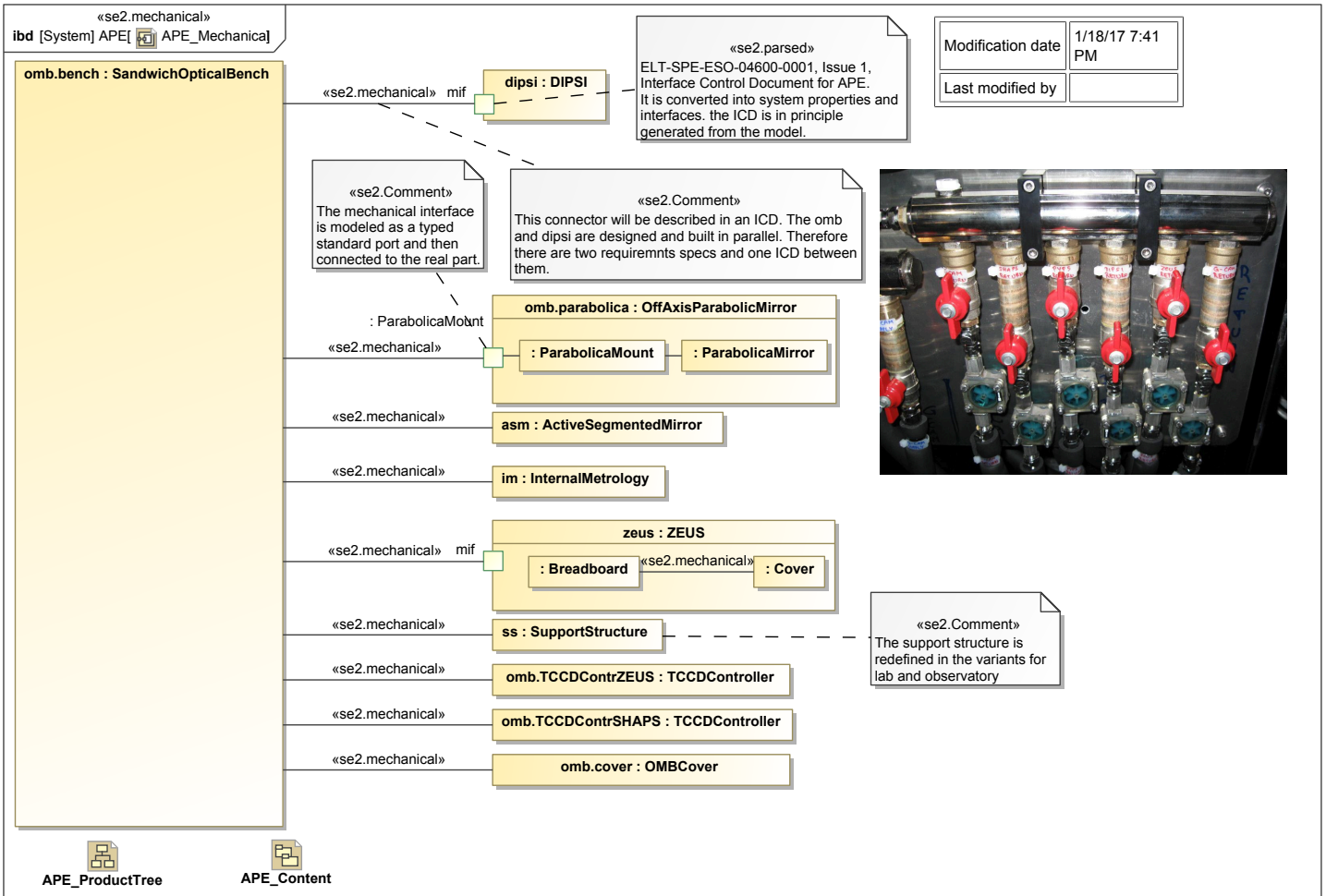


Figure 91. APE_Mechanical

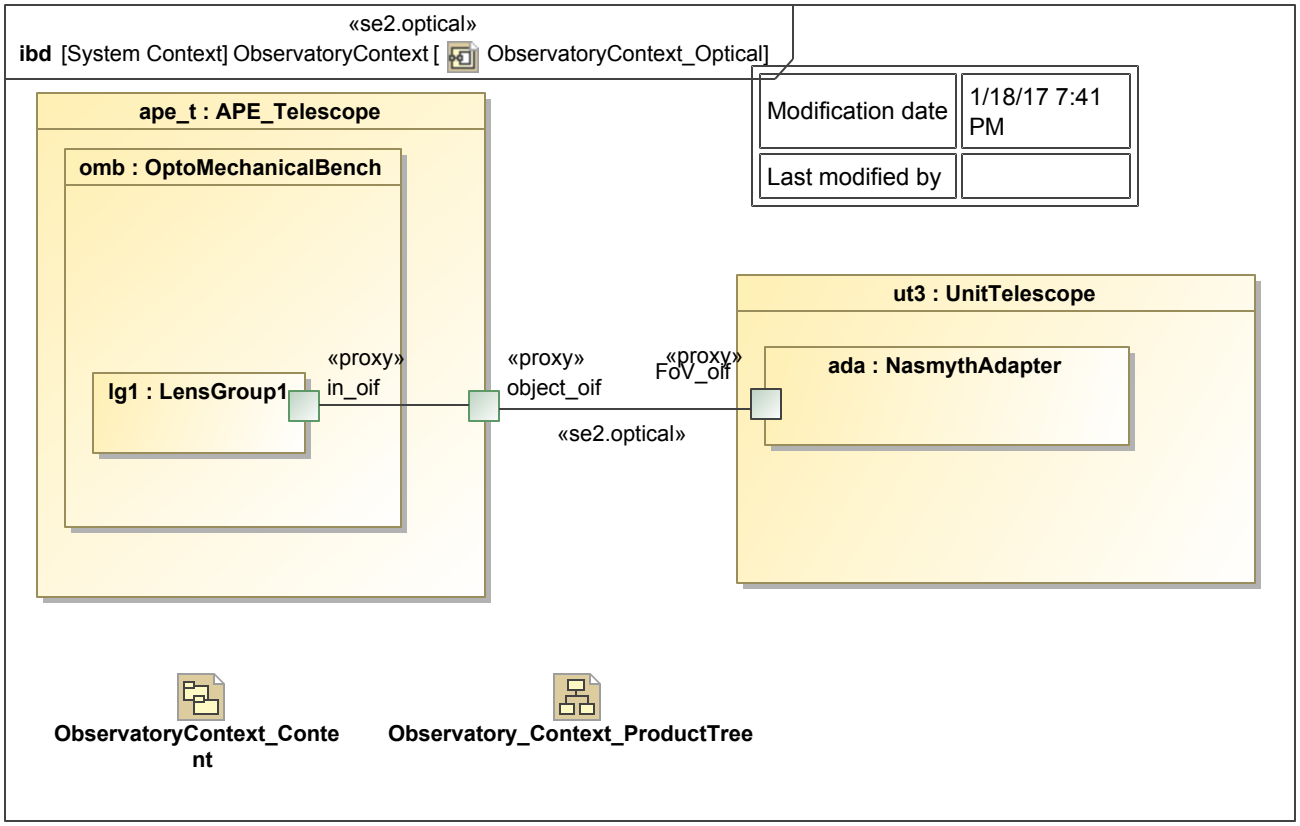


Figure 92. ObservatoryContext_Optical

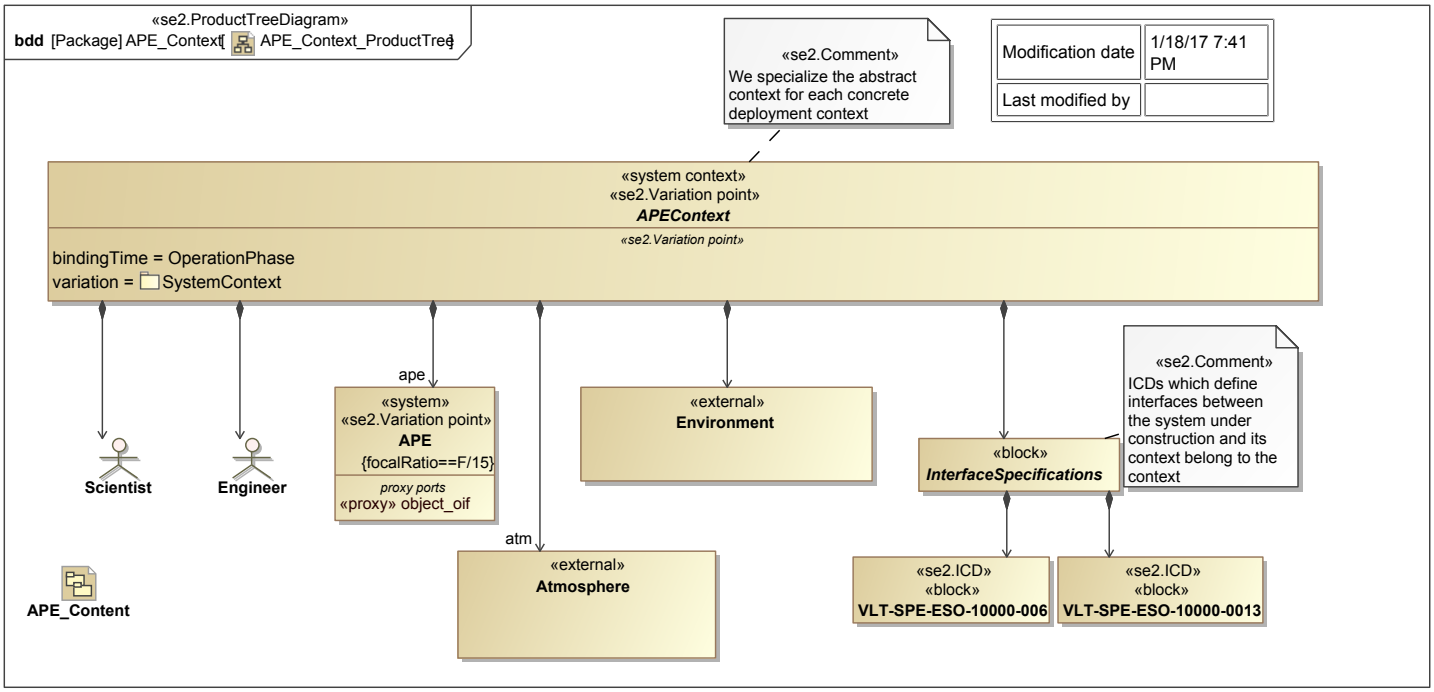


Figure 93. APE_Context_ProductTree

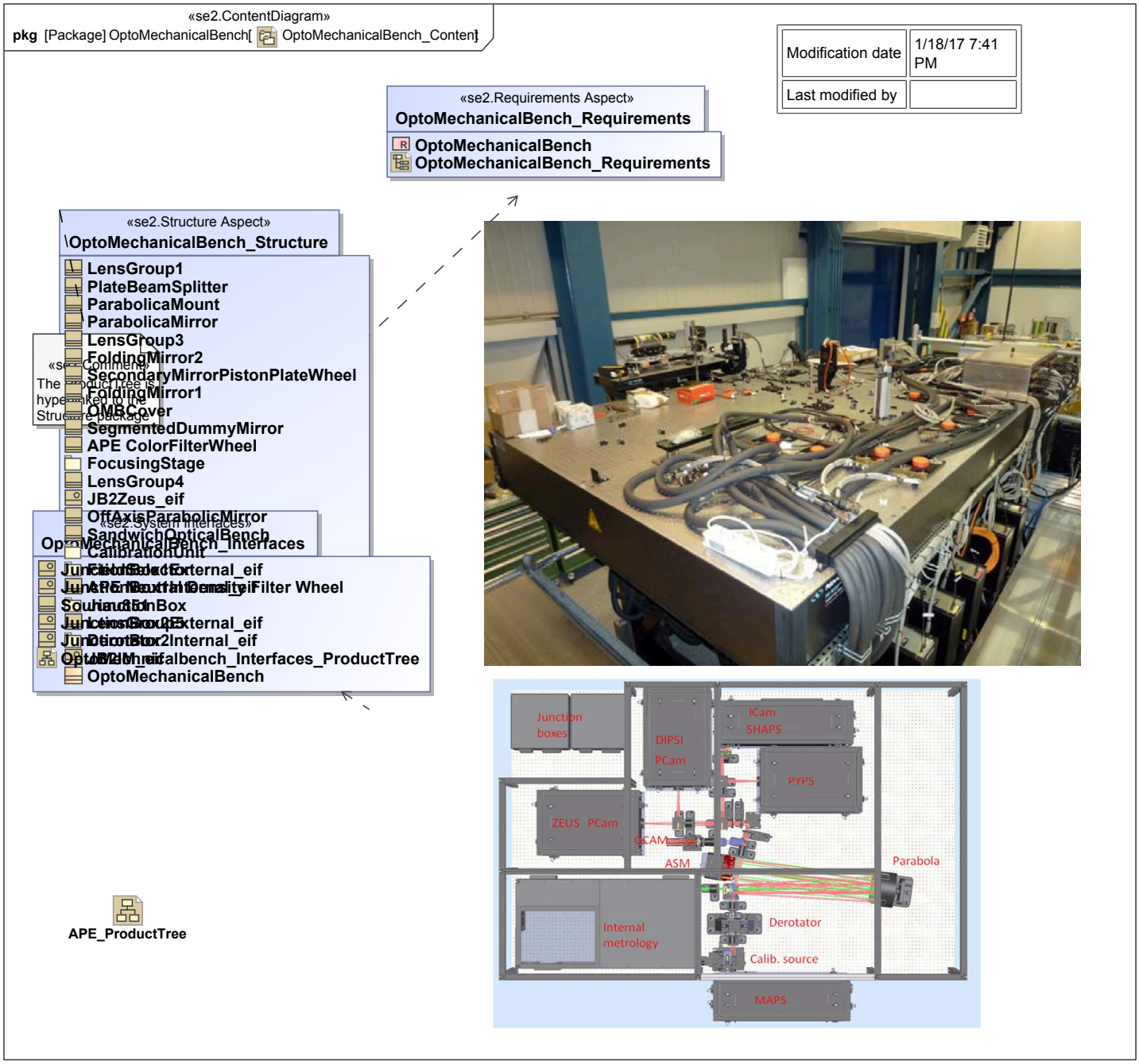


Figure 94. OptoMechanicalBench_Content

30.2 Modeling the information flow in the context diagram

If information flow to external is modeled to improve understand of the context relations of the system, model the information flow as flow items on the connections. Pay attention to avoid diagram cluttering, and achieve completeness. If necessary create a separate diagram for the information flow to each external system.

30.3 Modeling different contexts

The System context is modeled using IBDs. Our main focus is on system interfaces. Three different possibilities are shown to model an interface • Combination of mechanical and flow interface at block level (Model physical and logical properties at border of block without opening it.) • Mechanical and flow interface at part level • Mechanical and flow interface at block and part level. • Abstract interface representing and ICD (using standard ports). A problem is ensuring consistency between ICD document and the model which is used to create the CD. APE is deployed in the lab and at the observatory. Each environment has slightly different interfaces or external actors, i.e. the context is different. The different contexts are modeled as variants, which are explained in detail in chapter 14. For each relevant aspect (mechanical, optical, electrical) the context is modeled. To avoid cluttering by IBDs, for each context a

separate block is created which inherits from a common, <>, context block. The common block contains the elements which are shared among the other contexts. In the APE example the abstract context block is called APEContext, and can be identified by the slanted characters in the name. For each context, this block is In the case of APE, the SupportStructure, the ControlSystem, and other items differ slightly for the ObservatoryContext, which can be seen as specializations in Figure 23. Again, there exist IBDs for this particular context, as shown in Figure 24.

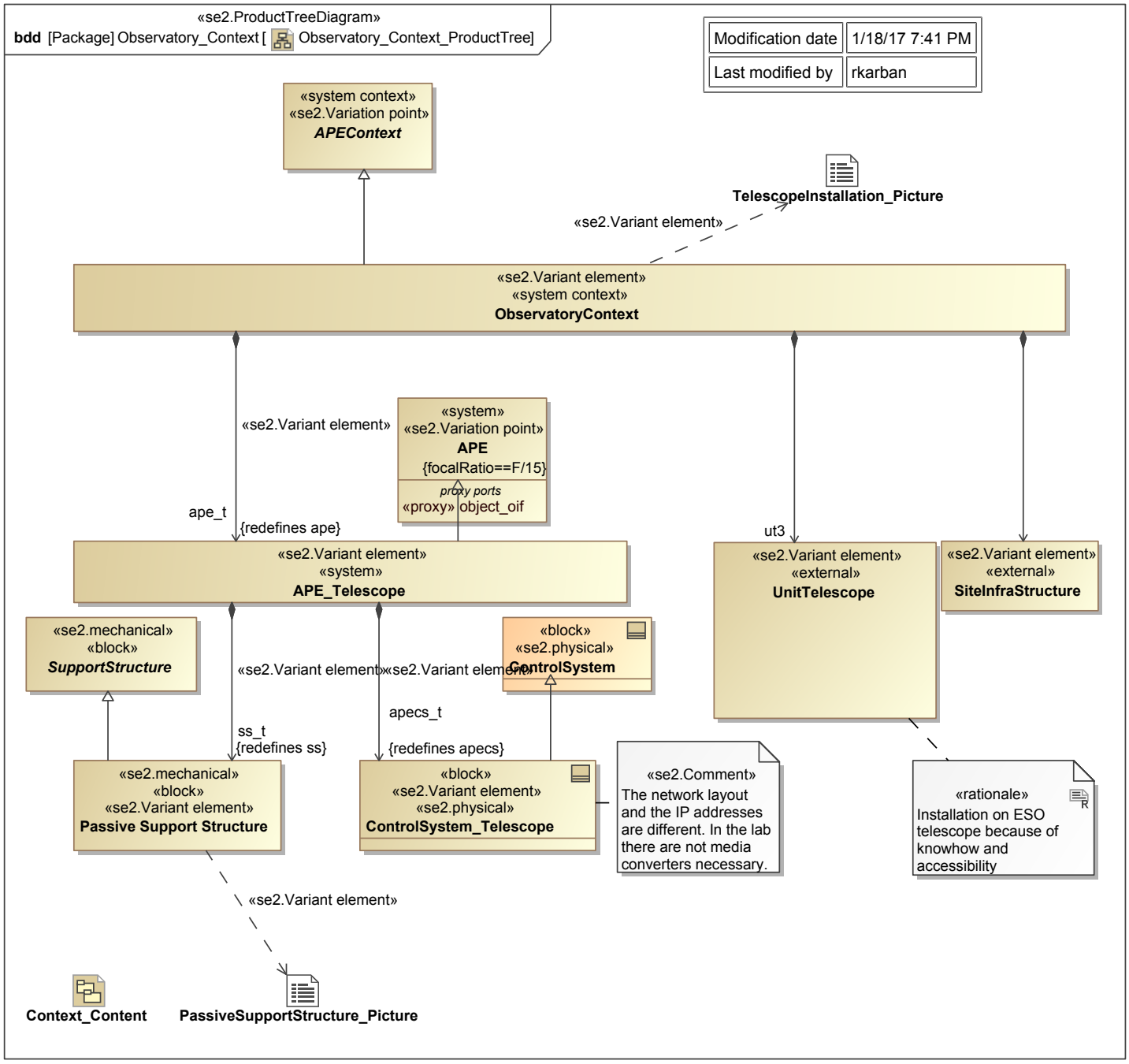


Figure 95. Observatory_Context_ProductTree

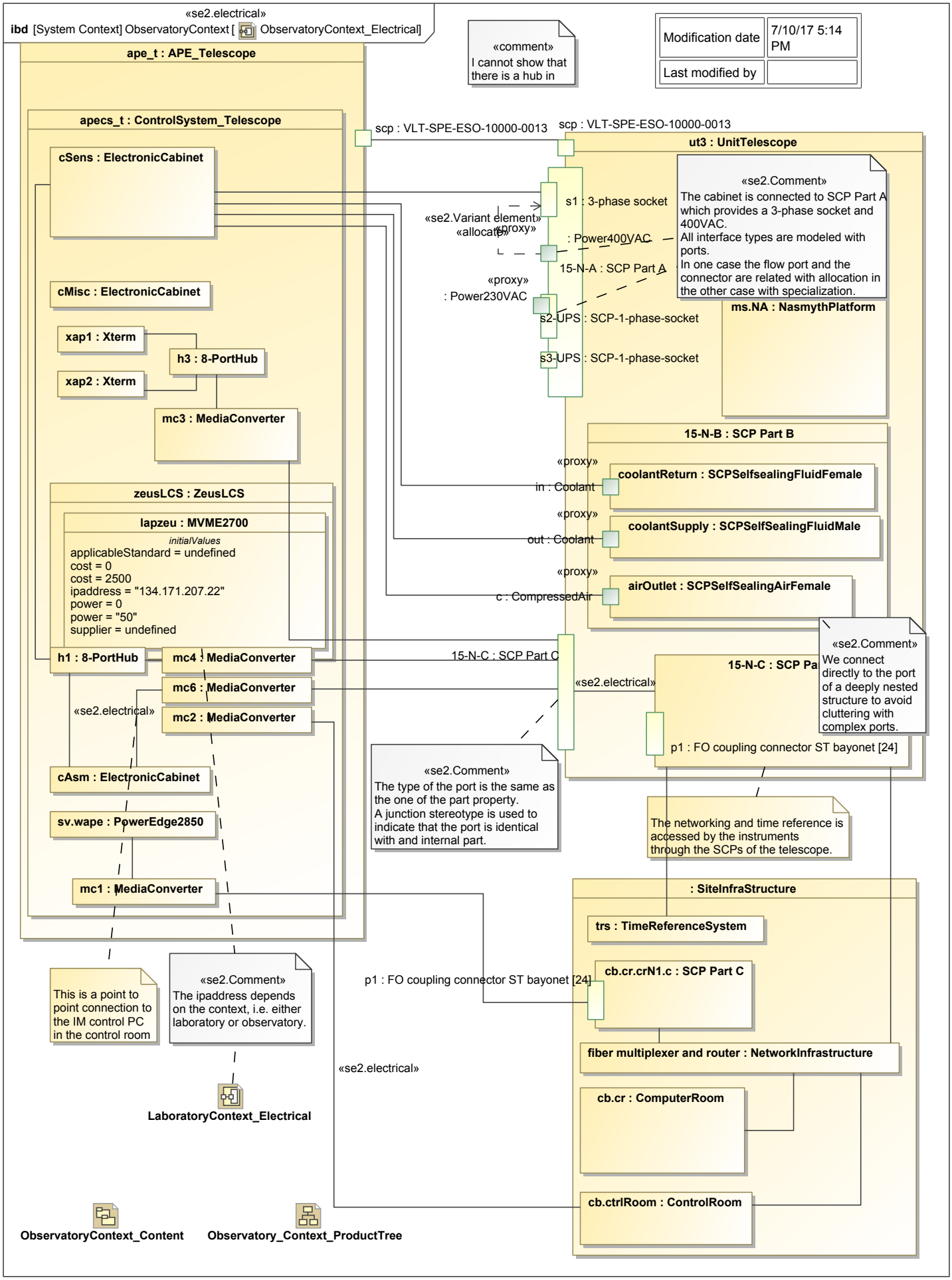

«se2.electrical»
ibd [System Context] ObservatoryContext [ObservatoryContext_Electrical]
«comment»
I cannot show that there is a hub in
Modification date
7/10/17 5:14 PM
Last modified by
ape_t : APE_Telescope
apecs_t : ControlSystem_Telescope
scp : VLT-SPE-ESO-10000-0013
scp : VLT-SPE-ESO-10000-0013
ut3 : UnitTelescope
cSens : ElectronicCabinet
s1 : 3-phase socket
«se2.Comment»
The cabinet is connected to SCP Part A which provides a 3-phase socket and 400VAC.
All interface types are modeled with ports.
In one case the flow port and the connector are related with allocation in the other case with specialization.
«se2.Variant element»
«allocate»
«proxy»
: Power400VAC
15-N-A : SCP Part A
«proxy»
: Power230VAC
s2-UPS : SCP-1-phase-socket
s3-UPS : SCP-1-phase-socket
ms.NA : NasmythPlatform
cMisc : ElectronicCabinet
xap1 : Xterm
h3 : 8-PortHub
xap2 : Xterm
mc3 : MediaConverter
15-N-B : SCP Part B
«proxy»
coolantReturn : SCPSelfsealingFluidFemale
in : Coolant
«proxy»
coolantSupply : SCPSelfSealingFluidMale
out : Coolant
«proxy»
airOutlet : SCPSelfSealingAirFemale
c : CompressedAir
zeusLCS : ZeusLCS
lapzeu : MVME2700
initialValues
applicableStandard = undefined
cost = 0
cost = 2500
ipaddress = "134.171.207.22"
power = 0
power = "50"
supplier = undefined
h1 : 8-PortHub
mc4 : MediaConverter
mc6 : MediaConverter
mc2 : MediaConverter
15-N-C : SCP Part C
«se2.electrical»
15-N-C : SCP Pa
«se2.Comment»
We connect directly to the port of a deeply nested structure to avoid cluttering with complex ports.
p1 : FO coupling connector ST bayonet [24]
«se2.electrical»
«se2.Comment»
The type of the port is the same as the one of the part property.
A junction stereotype is used to indicate that the port is identical with and internal part.
The networking and time reference is accessed by the instruments through the SCPs of the telescope.
cAsm : ElectronicCabinet
sv.wape : PowerEdge2850
mc1 : MediaConverter
: SiteInfraStructure
trs : TimeReferenceSystem
cb.cr.crN1.c : SCP Part C
p1 : FO coupling connector ST bayonet [24]
fiber multiplexer and router : NetworkInfrastructure
cb.cr : ComputerRoom
cb.ctrlRoom : ControlRoom
This is a point to point connection to the IM control PC in the control room
«se2.Comment»
The ipaddress depends on the context, i.e. either laboratory or observatory.
«se2.electrical»
LaboratoryContext_Electrical
ObservatoryContext_Content
Observatory_Context_ProductTree

Figure 96. ObservatoryContext_Electrical

31 Use Case Modeling

31.1 Purpose of a Use Case

A use case is a sequences of interactions between the system and the actors to achieve a specific purpose for the triggering actor. We used use cases for refining high level requirement. Refinement means elaboration in this context, i.e. add more information about the interaction and provide a better understanding of the requirement. However, Use cases can also used as basis for elaboration of lower level requirements. A use case describes how a functionality/service is offered by the system as description of the interaction of the system and its actors. The system is like a "service provider". A use case has at least one actor, is started by a domain specific trigger and ends with a domain specific outcome. The sequence between trigger and outcome is coherent in time, i.e. there is no domain specific interruption. The use case shall be described with an essential sequence, which is the standard success scenario. It should not contain any decisions about technology or design. Non-standard sequences (exceptions, etc.) are described with additional activity diagrams. Describe use cases only with essential steps (i.e. raise them to a higher level of abstraction and describe them technology independent). A use case should usually have between 2 and 8 steps. A use case can be encompassed by preconditions and post conditions for a more precise description, but there may not be a 1:1 mapping of pre- and post condition to later state machine implementations. A trigger is generated OUTSIDE the system and starts the execution of a use case. The result of the use case is an EXTERNALLY observable consequence. Pre- and post conditions are the state of the system. They are INTERNAL. A use case can be restricted by a business rule. They should go to a separate section and the use case should only refer to them because they might be applicable to several use cases. A use case can be associated with non-functional requirements (performance, safety, etc.). Since the performance requirements might apply across use cases or across the system, it is better to reference them only to avoid duplication and allow better traceability. There are two types of use cases: • Primary - Primary use cases are directly connected to actors, they have a trigger, an outcome and post-conditions. They represent the core value for the actors. • Secondary - Secondary use cases have no actors, no trigger, no post-conditions and are a mean to factorize common paths in primary use cases. Use them only when you have more than three instances of the same path. A use case should have a name, description, incoming/outgoing data, trigger, result/ benefit, pre-/post-condition. The description elaborates the most important activities (aka scenarios). The steps of the activities should be described in its essential form, i.e. rather technology independent. For example, instead of “Enter PIN” the step should be “Authenticate User”. The number of essential steps serves as a measure for the effort to implement it. Some of the APE system use cases and the corresponding GUI is shown in Figure 25.

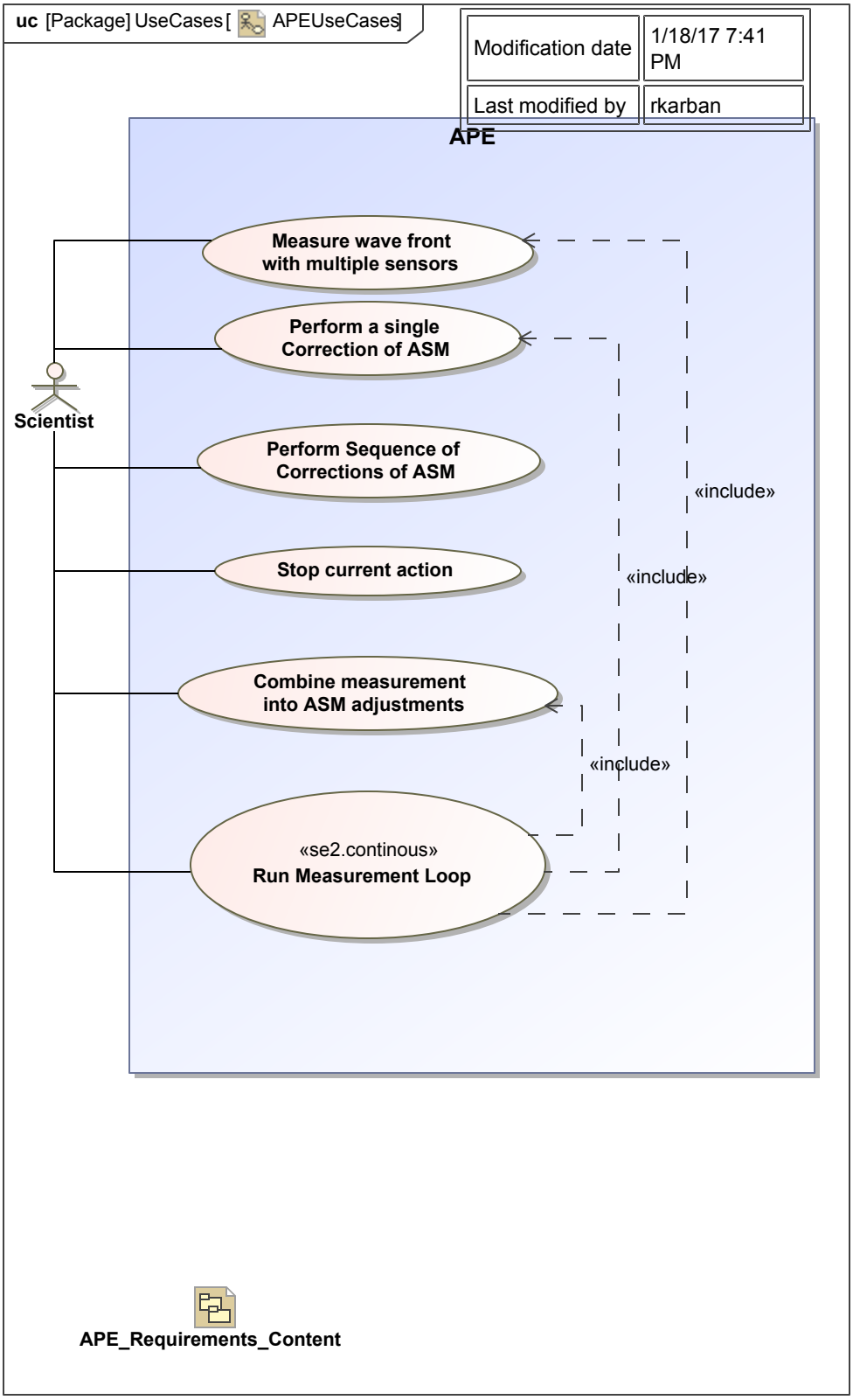


Figure 97. APEUseCases

31.2 Modeling monitoring and control activities

A monitoring or control activity can be started, then it runs, and in the end it is stopped. Here is only ONE use case required. It would be triggered by the start and run until a message arrives to stop it. It can be modeled as a <> use case.

31.3 Modeling operations related to subsystems with use cases

For example, a lamp is switched on, then it is on and in the end it is switched off. The use case would be rather simple and therefore does not add any additional information. You don't need a Use Case. Model it with a normal requirement that you want to switch on, off the lamp. However, to model the change of intensity you might refine this requirement with a use case.

Figure 98. Refinement of System Requirements

31.4 External element types

The following type of actors can be distinguished: • interacting actors - involved in use cases • mechanical systems - interfaces defined but not involved in use cases • environmental influences - like vibration forces There should be different diagrams for those categories. Every external element MUST have a defined interface (interface in sense of System Engineering).

31.5 Modeling a system of systems with use cases

Use cases should have only ONE level for a certain system, the <> use case. Use cases shall not be deeply nested because it bears the risk of doing structured analysis with use cases. That’s not the intention of use cases, because each use case shall have a measurable benefit for the actor. Note: use cases can be used at different level. However, we do not recommend using different use case levels for Telescope domain. <> use cases are a pure organizational aid to avoid duplication of information. Between <> and <> use cases exists the <> relationship. However, use <> only if more use cases share the same functionality. Avoid building structured analysis like <> tree structure!

31.6 Use Cases vs. Standard UML Interfaces

• Use interfaces to elaborate Use Cases at lower levels • The panel (with knobs and gauges) of a subsystem is represented by a set of Use Cases • From the perspective of another subsystem the same functions are modeled by interfaces, try to avoid Use Cases among systems • Always let ports realize an interface. More flexibility • The connector from a port at system/assembly border acts as delegation

31.7 Tracing test cases to use cases and requirements

A specification for a subsystem often contains requirements on the necessary tests to be executed, the test requirements. A client, for example, would specify test requirements if the subsystem is contracted to a supplier. The supplier has to meet those test requirements. Those requirements are not the detailed procedures but only requirements for them. The details of the <test case=""> are described by activities, sequence diagrams or state machines. Furthermore, the model should describe who executes the test. Test cases ccan be directly derived from use cases, because each scenario of a use case is a test case. An example verification model is shown in Figure 27 which shows the relations among all the participating entities. The TestRequirement is derived from a SystemRequirement, and refined a Test Use Case. The set of <use case="">s corresponds to a traditional test plan. Each Use Case is realized by one or more <testcase>s, which in turn <verify> the SystemRequirement. The <testcase> can be described by activity diagrams. In fact, a single use case can verify multiple system requirements that are realized by multiple test cases. The actors represent the roles involved in the test. Refer to [RD1] for realization relationship. Actually, the test case stereotype is applied to an activity diagram. The use case is realized directly by the activity diagram with the test case stereotype applied. One way in which this approach would add value is if a single use case can refine multiple test requirements in a similar way that a typical operational use case refines multiple system requirements. If so, you can synthesize a test system with fewer number/type of tests to verify the system requirements and reduce your testing cost. This needs to be assessed.

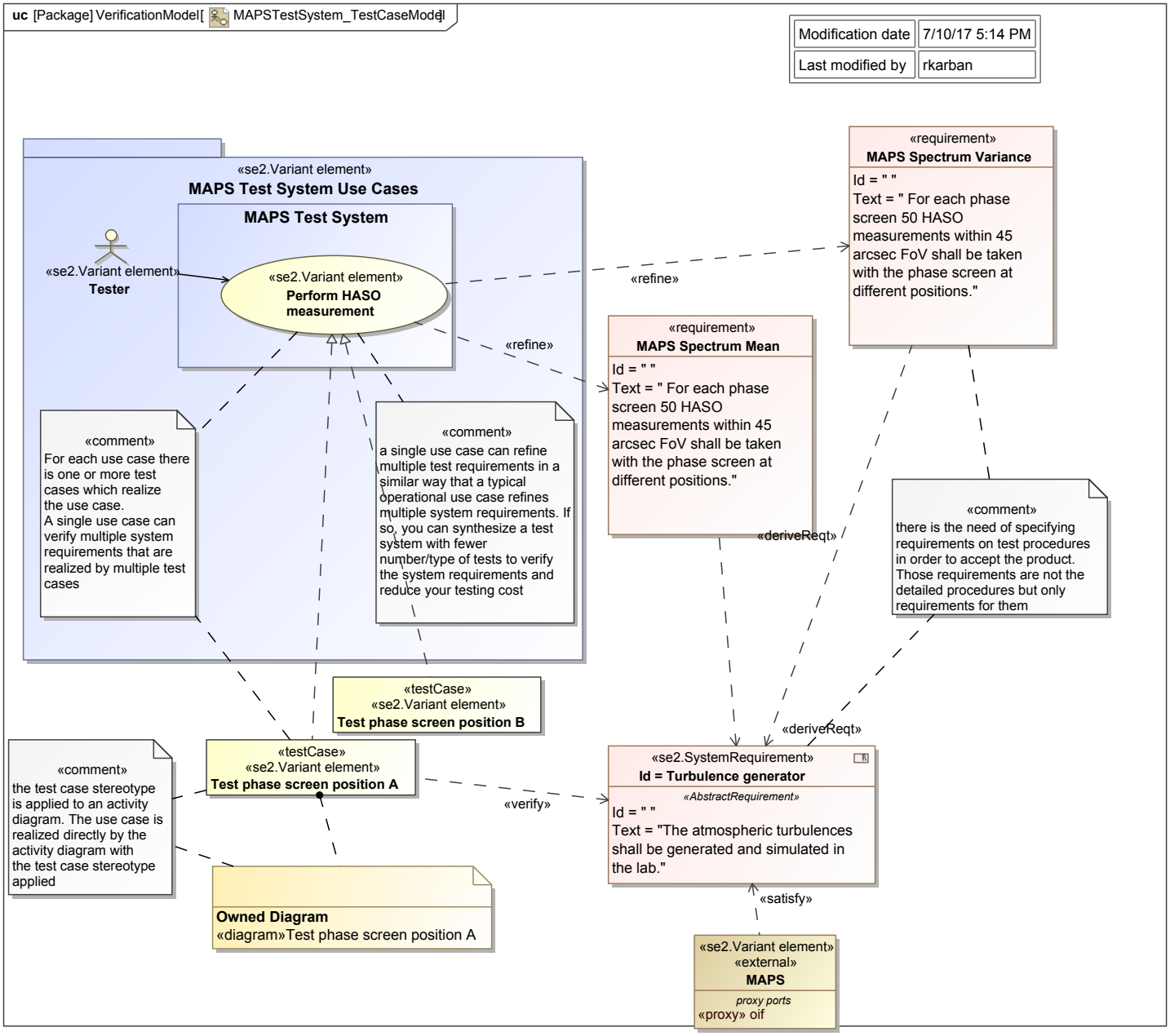


Figure 99. MAPSTestSystem_TestCaseModel

31.8 Naming of Use Cases

- Create a first set of use case with only their name in the form: "verb + [qualified] object". You can do that by creating actors and use case bubbles.
- Use active (not passive) verbs. Avoid vague verbs like "Do" or "Process". Avoid low-level verbs such as create, read, update, delete.
- Each object name in the use case name should be defined in the glossary.
- Here are some verbs for informative Use Cases: Analyze, Discover, Find, Identify, Inform., Monitor, Notify, Query, Request, Search, Select, State, View
- And for performative: Achieve, Allow, Change, Arrange, Classify, Define, Deliver, Design, Ensure, Establish, Evaluate, Issue, Make, Perform, Provide, Replenish, Request, Setup, Specify
- DO NOT put any non-functional requirements into use case. DO NOT put any business rules in Use Case - only reference them.

31.9 Do I need to refine every requirement with a Use Case?

No. You only focus on the main services in the use cases and not on trivial things.

31.10 Developing TMT Use Cases

31.10.1 Purpose

The purpose of developing use cases is to capture high level procedures the Active Phasing System (APS) uses to perform various telescope alignments and calibrations. The use cases verify requirements on the performance of the telescope such as timing, electrical power, and pointing errors. The identified use cases cover a spectrum of interactions with other subsystems of TMT, and to accurately predict the performance of APS.The use cases are developed according to the principles of the Executable Systems Engineering Method (ESEM) which prescribes the functional and physical decomposition of the system into a nested tree of components, and the specification of the behavior of each. The primary model elements that are used to develop TMT use cases are state machines, activities, sequence diagrams, simulation configurations, instance tables, and views. The process for developing use cases consists of extending the behavior of the Procedure Executive and Analysis Software through state machines and activities, specifying a context for automated analysis, specifying the behavior for a particular use case through sequence diagrams, configuring simulation configurations for system execution, exposing instance results to a view, and evaluating simulation results. The TMT use case, Post Segment Exchange Alignment, will provide context on each of these steps below.

31.10.2 Model Organization

The TMT model structure supports different levels of abstraction and deeply nested system hierarchies. However, the model organization that contains all the elements required to develop a Use Case will be discussed.

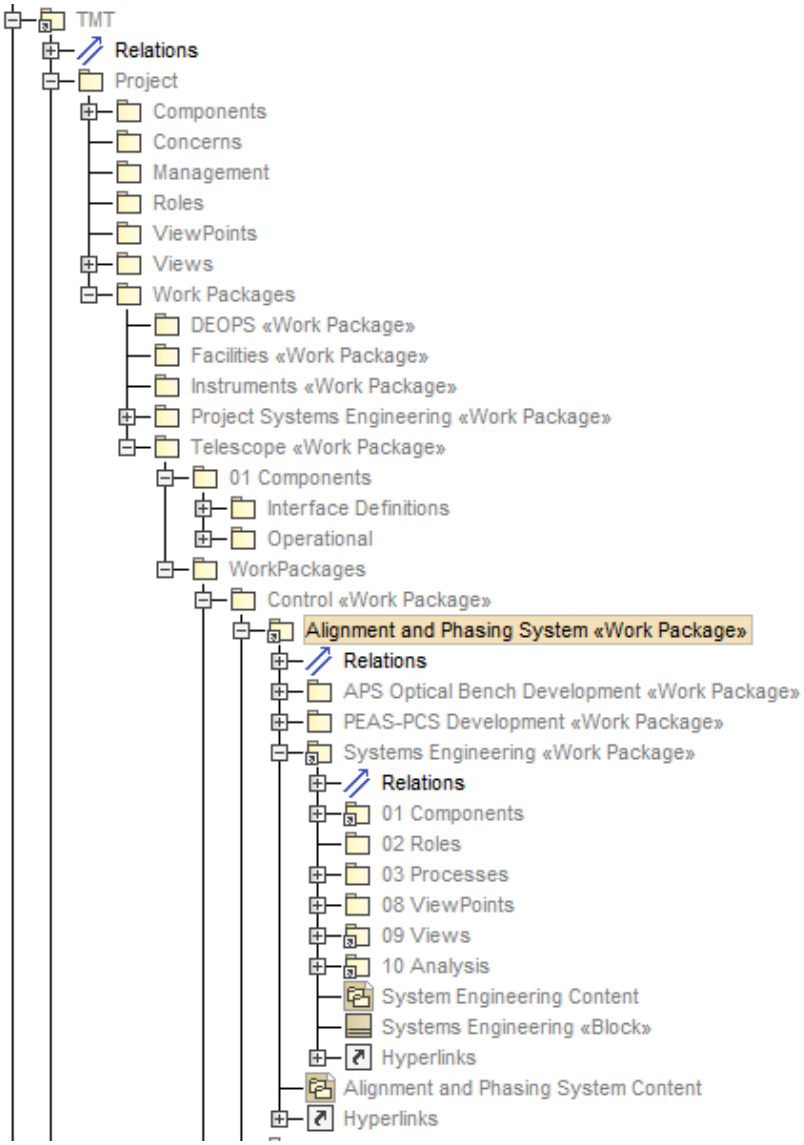


Figure 100. TMT Model Organization

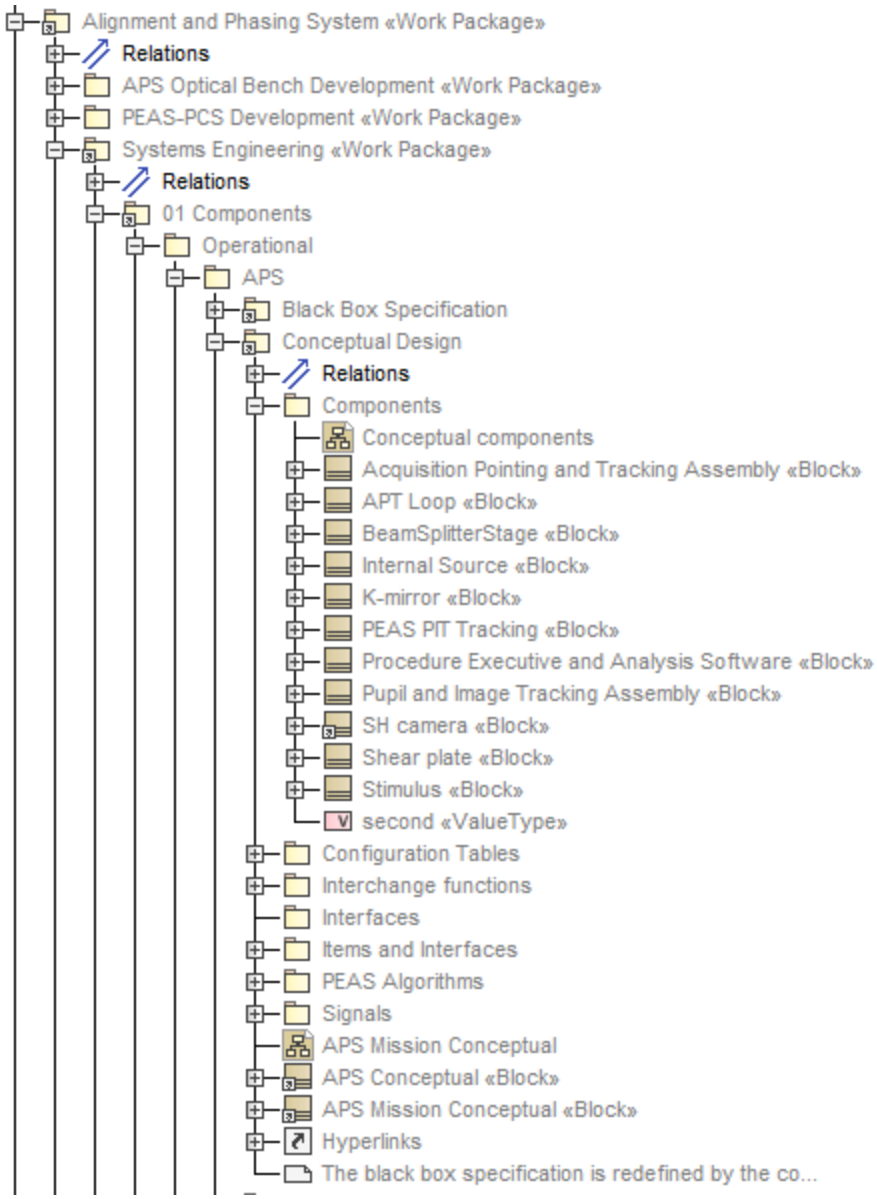


Figure 101. Conceptual Design Components

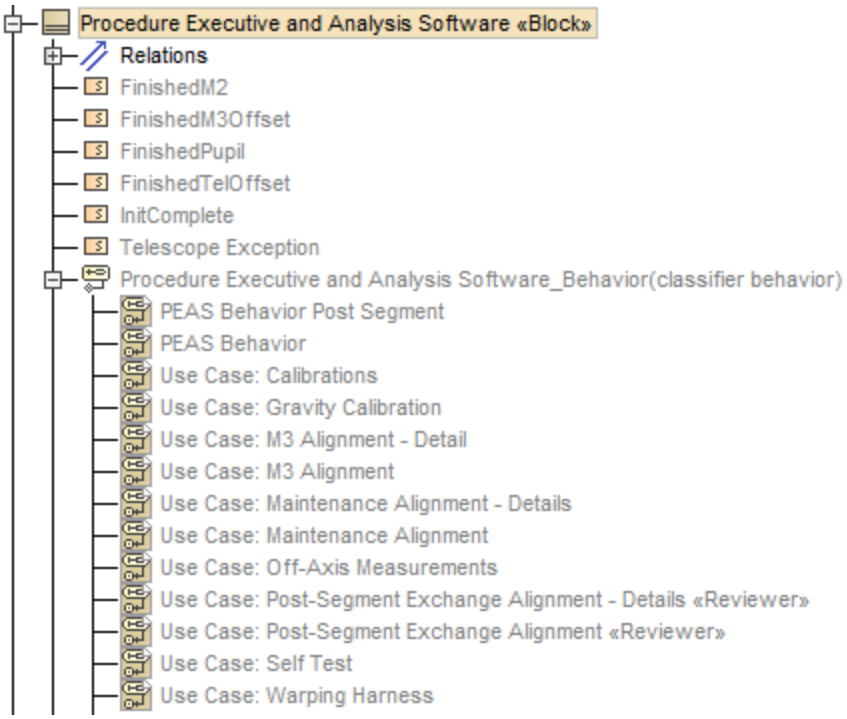


Figure 102. PEAS Use Case Diagrams

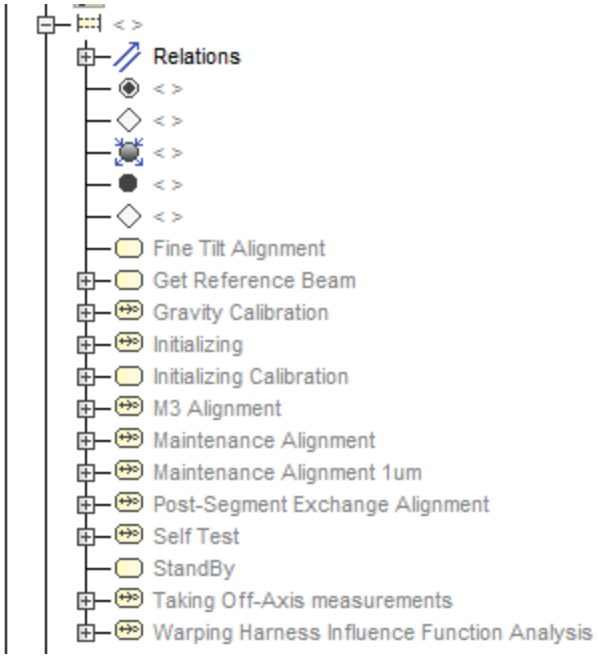


Figure 103. PEAS Use Case States

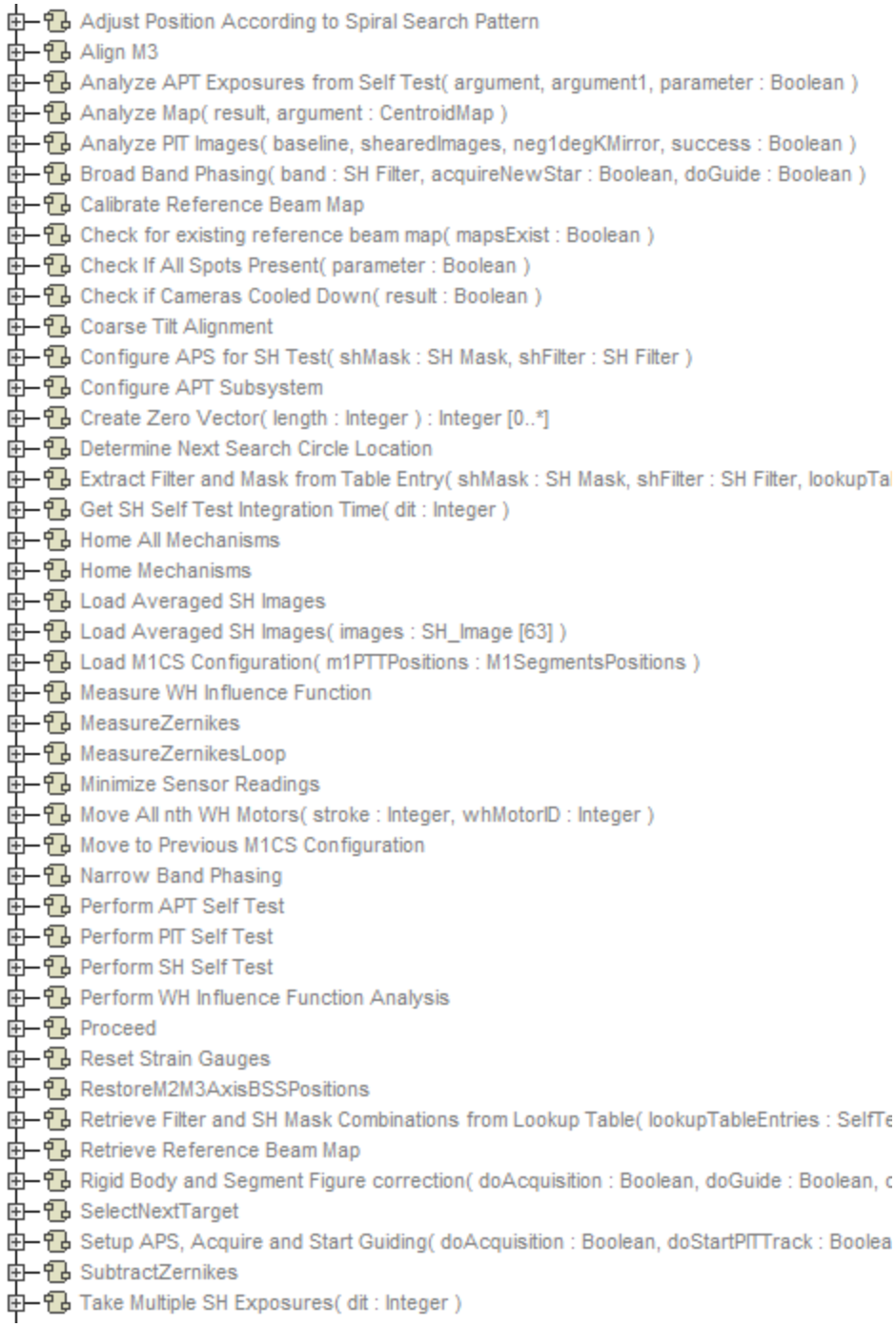


Figure 104. PEAS Activities

31.10.3 Defining Behavior in State Machines and Activities

The behavior of components are distinguished between life-cycle behavior and distinct isolated activities of a component. State machines are used to specify the life-cycle behavior of components. The life-cycle behavior describes the behavior of a component from beginning to end-of-life, and describes the state of a component at any instant in its operational lifetime. State machines o?er a method to specify the various available modes of a system using states, and the conditions for transitioning between them. Activities capture specific tasks that represent only a part of the overall behavior of a component. The flow charts allow for the specification of sequential and parallel behavior, and decision points and synchronization points for an intuitive description of intended behavior.

A use case state machine begins cf name([cf:Initializing.name]) does not exist will the do-activity cf name([cf:InitializePEAS.name]) does not exist . When transitioning to state cf name([cf:StandBy.name]) does not exist , a signalEvent of cf name([cf:InitComplete.name]) does not exist will occur. An external signal that is specific to the use case (which may be provided by a user) causes a trigger to transition from cf name([cf:StandBy.name]) does not exist to the use case specific state. Within this use case

specific state, the primary states of the use case are specified. The “do” behavior of each state/phase of the use case specifies the behavior of PEAS when in that particular state. This part of the overall flow of events is a call to another activity, where the setup procedures are specified in more detail. This method of calling other activities allows for the reuse of isolated, distinct tasks within multiple contexts without having to redefine the flow of events. PEAS is specified to only be able to leave the primary use case state if an cf name([cf:Abort.name]) does not exist signal is provided (internally or externally), and will return to the state cf name([cf:StandBy.name]) does not exist .

The use cases of PEAS can be found in the model at the following location: TMT::Project::Work Packages::Telescope::WorkPackages::Control::Alignment and Phasing System::Systems Engineering::01 Components::Operational::APS::Conceptual Design::Components::Procedure Executive and Analysis Software.

31.10.4 Defining Behavior in Sequence Diagrams

For each corresponding use case of the PEAS, a duration operational scenario is represented by a SysML block. The classifier behavior of the duration scenario block is a sequence diagram. To simulate an operator initiating a particular operational scenario, signals must be injected that trigger appropriate behavior of the various components .Signals are sent from an Analysis Driver (the operator) to the Procedure Executive and Analysis Software.The asynch signals that are sent from the Analysis Driver are Abort, Calibrate, and Abort. The dependent variable of each sequence diagram is the value of each duration constraint between the signals. The purpose of the operational scenario is to determine the final time and overall duration of a particular use case. Therefore, each duration scenario block owns a parametric diagram. Depending on the use case, the tFinal value property of the Procedure Executive and Analysis Software could be constrained to a constraint of the APS Operational Blackbox for requirement verification purposes.

31.10.5 Simulation Components

To automate the simulation of each use case, a CST simulation configuration is used. The execution target of the simulation configuration is the duration scenario block. The instance results for each use case simulation is stored in a corresponding package at can be found at: TMT::Project::Work Packages::Telescope::WorkPackages::Control::Alignment and Phasing System::Systems Engineering::10 Analysis::Duration Analyses::Automatic Duration Analysis::Results. An instance table is used to display the results of the simulations, and the classifiers of the instance tables are the Procedure Executive and Analysis Software and Executive Software components.

The operational scenario blocks, sequence diagrams, parametric diagrams, and simulation configurations can be found in the model at the following location: TMT::Project::Work Packages::Telescope::WorkPackages::Control::Alignment and Phasing System::Systems Engineering::10 Analysis::Duration Analyses::Automatic Duration Analysis .

31.10.6 Evaluating Results

Through simulation, the conceptual design is automatically evaluated, and verifies whether the system conforms to a requirement. The definition of executable behavior and a formalization of requirements using mathematical constraints and parametric diagrams, allows for a particular performance requirement to be met. The exposed instance tables display whether the instances have passed or failed a requirement. The values of the resulting instances should be identical or have minimal differences. Therefore, the simulation is run more than once to confirm that the values can be reproduced. In addition, a manual baseline estimation should be available for comparison to ensure there are no model, simulation, or tooling errors.

31.10.7 Document Generation

The model development kit (MDK) is utilized to sync the TMT model with the model management system (MMS) and implement the DocGen language, which allows for document generation using the view and viewpoint concept. The standard view structure for a TMT use case is composed of the following views: Purpose of use case, Typical observing parameters, Entrance requirements and conditions, Use Case activity, Optical Performance Requirements, and Time to execute. In MagicDraw, views are created using a template, and diagrams expose simulation results and key activities. In ViewEditor, documentation is added to the views of the use case. A template of the view structure is reused for each use case developed to reduce time. The use case view template can be found in the model at the following location: TMT::Project::Work Packages::Telescope::WorkPackages::Control::Alignment and Phasing System::Systems Engineering::09 Views::TMT-APS-SE::TMT-APS Use Cases::Templates::Use Case Template.

31.10.8 Sample TMT Use Case

It has been estimated that 8 segments will need to be exchanged in a single night every two weeks. The purpose of the Post-Segment Exchange Alignment use case is to re-align the telescope after new segments have been installed or exchanged.

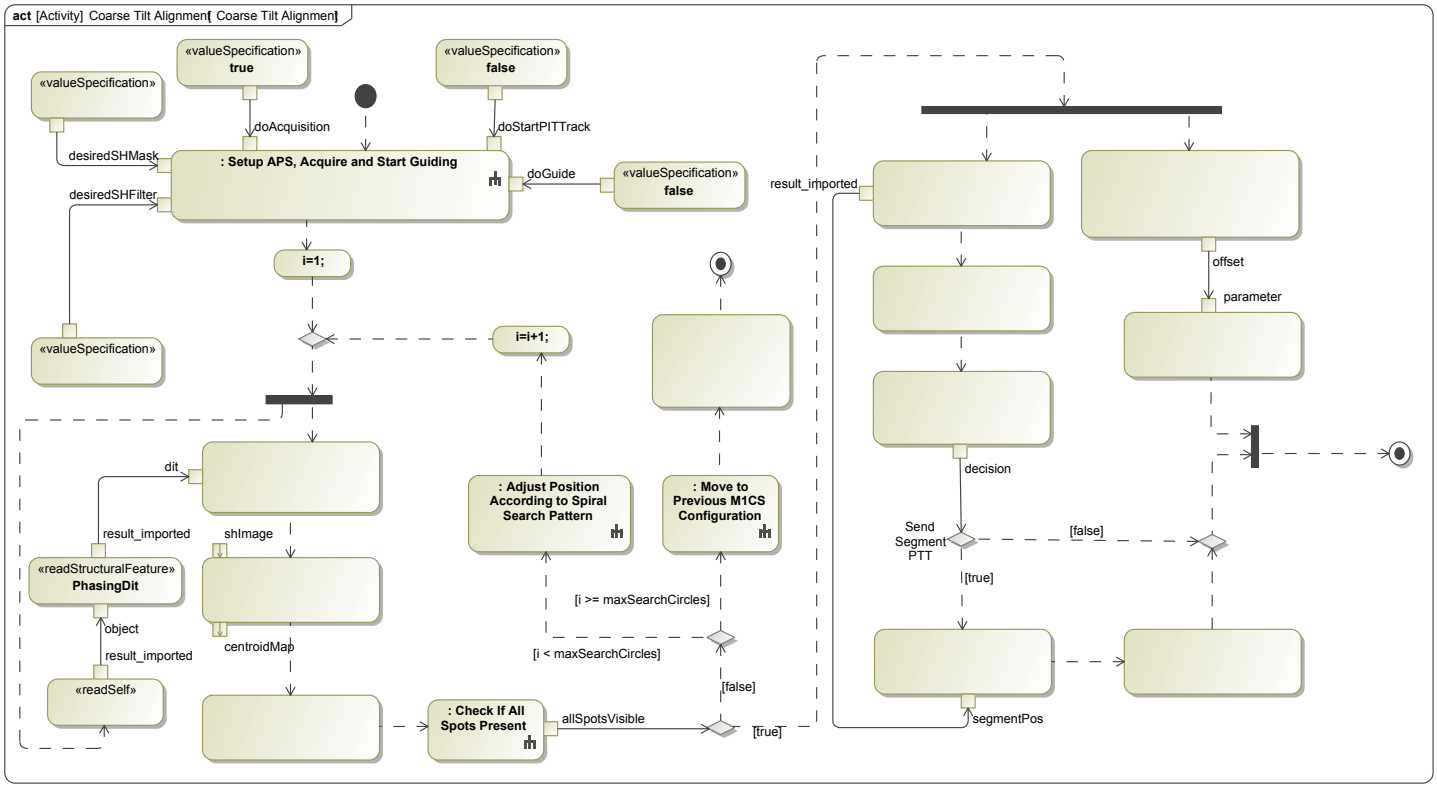


Figure 105. Coarse Tilt Alignment

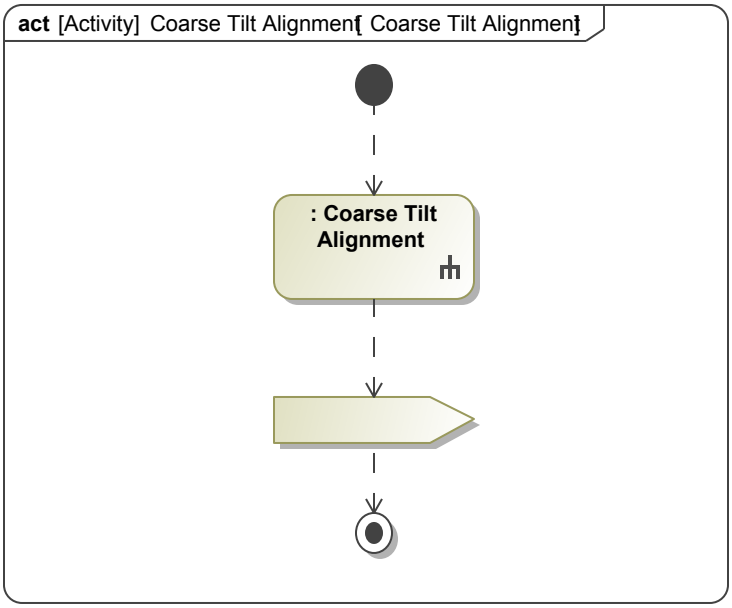


Figure 106. Coarse Tilt Alignment

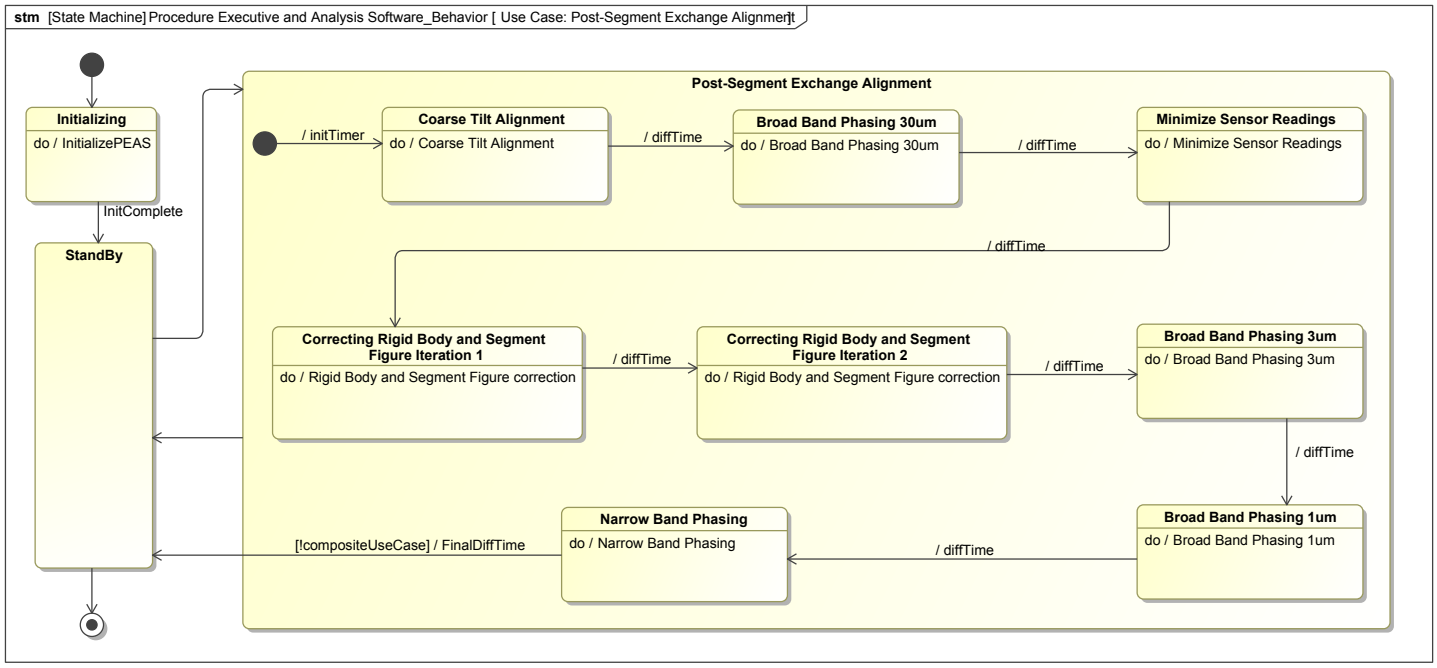


Figure 107. Use Case: Post-Segment Exchange Alignment

The Post-Segment Exchange Alignment activity will be executed after new segments have been added to the primary mirror.

32 MEV-CBE Power Roll Up Example

This pattern shows how to separate concerns of a system model and an analysis model. This example evaluates a car's power for Current Best Estimate (CBE) and Max Estimated Value (MEV) and the difference between the two. Using Systems Reasoner the system model and the analysis model can be kept consistent when the system structure changes. Starting with a general structure for your system (Car), use Systems Reasoner to specialize structure recursively and independently twice, once for MEV and once for CBE.

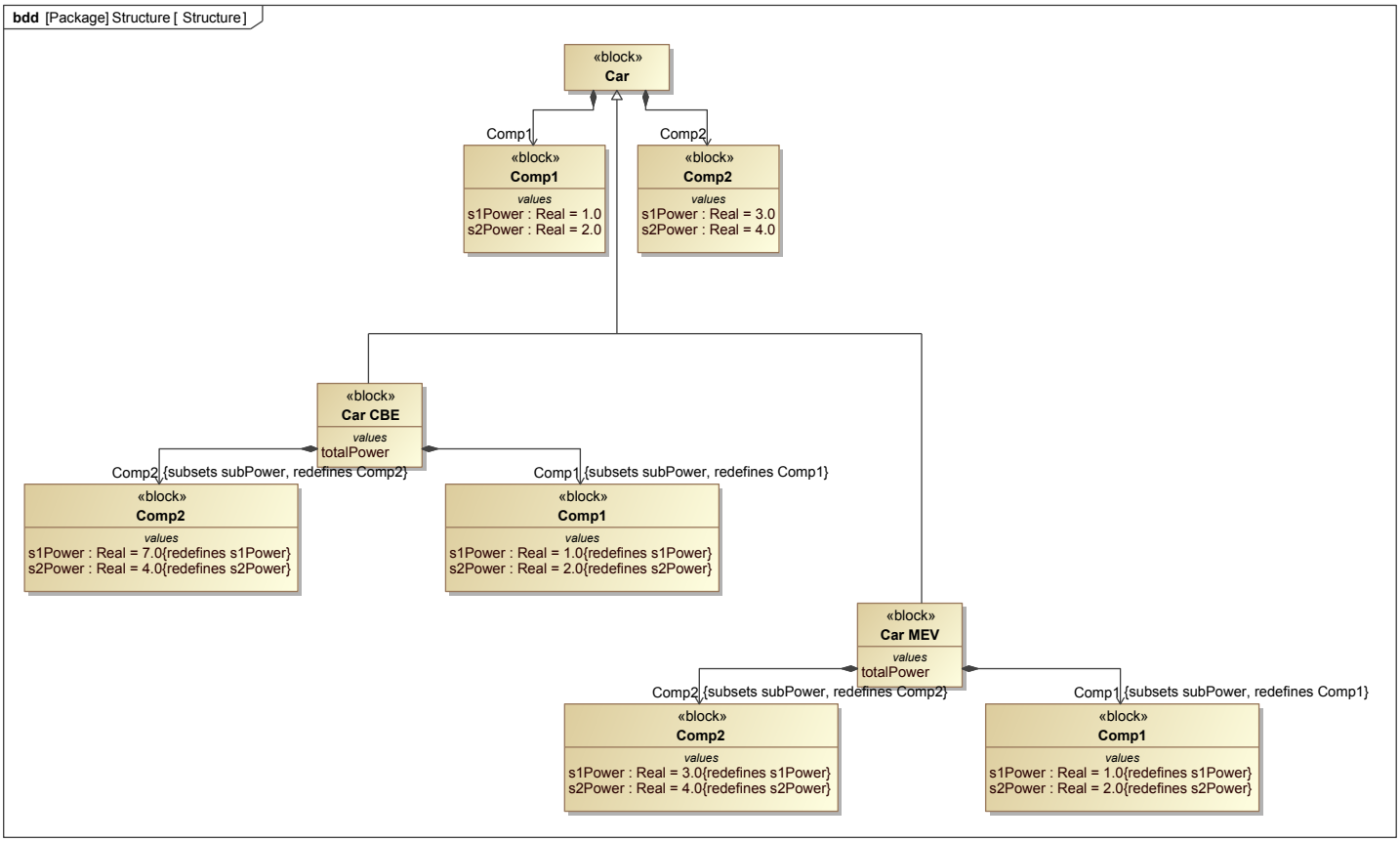


Figure 108. Structure

Apply general power rollup pattern to both MEV and CBE recursively. To calculate the difference of the MEV and CBE values, create a constraint called Difference and set it as the constraint of a block called Analysis.

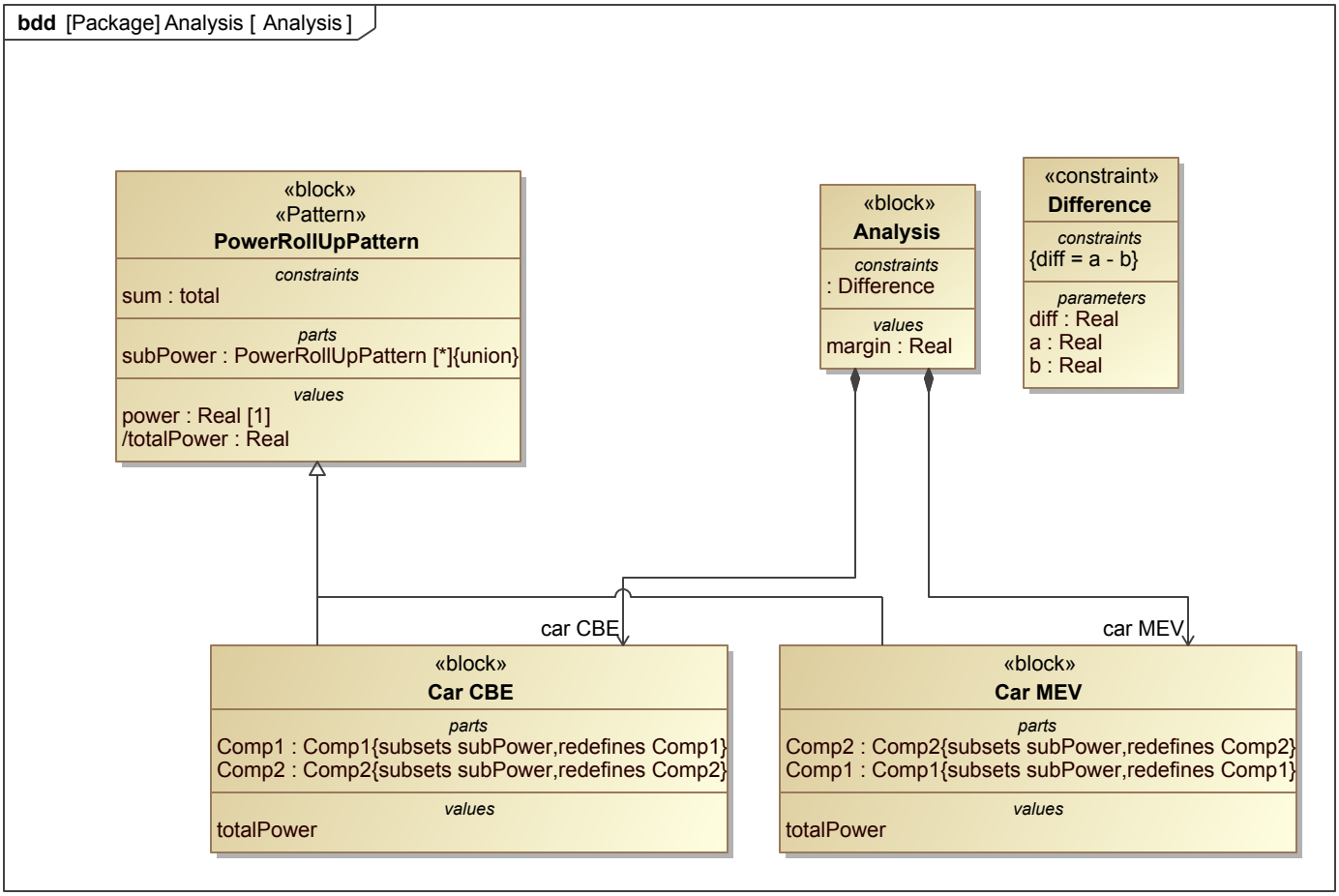


Figure 109. Analysis

Shown below is the parametric diagram of Analysis that defines the Difference constraint.

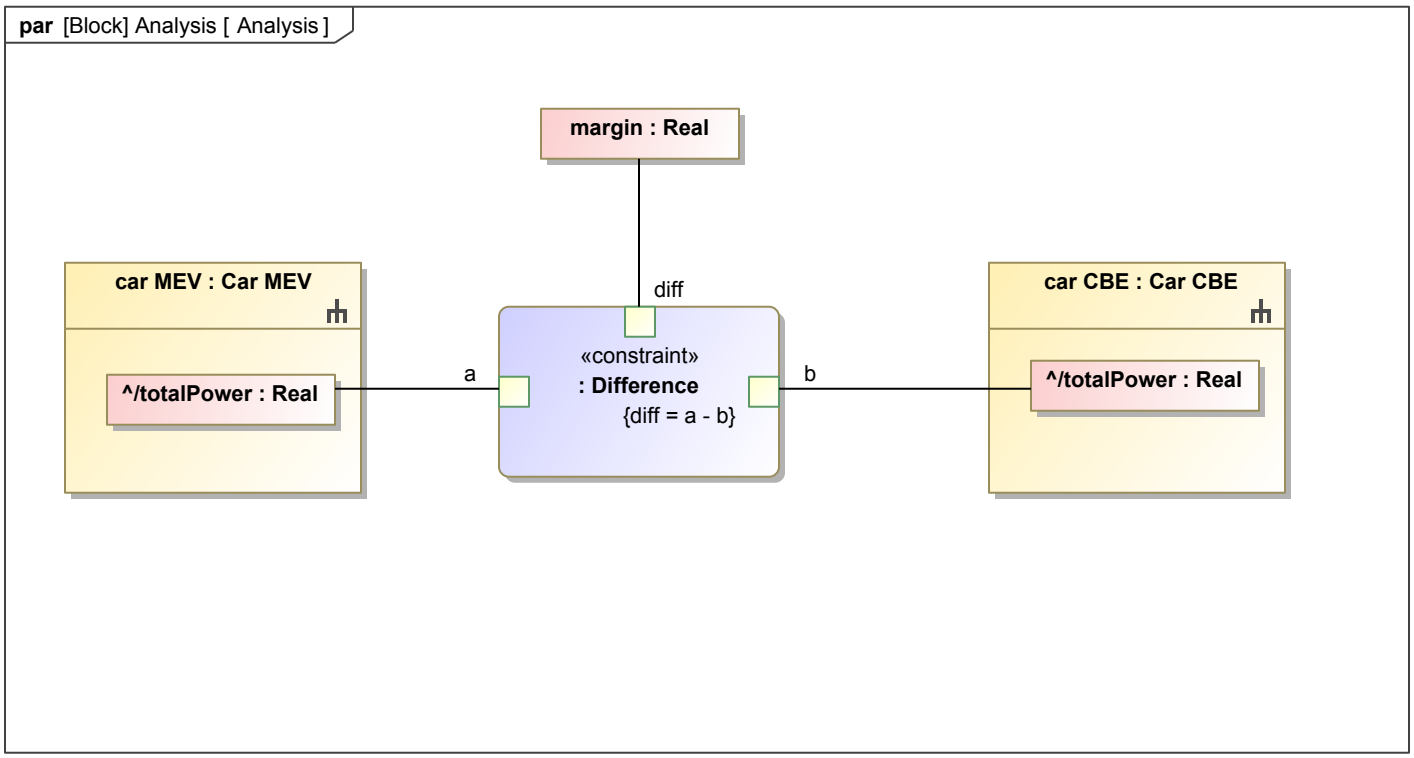


Figure 110. Analysis

Below is an instance table that contains the results of multiple runs of this analysis. The name of each instance contains a timestamp so that the table tracks the changes of the values over the course of system development. The totalPower is the value of the MEV and CBE total power, and the .comp notation shows the initial values of the components. Margin is the difference between the two, calculated by the constraint defined above.

Table 13. Instance Table2

Name	totalPower : Real	margin : Real
analysis at 2020.03.12 12.14		-1.0
analysis.car MEV3	5.0	
analysis.car MEV3.comp2	3.0	
analysis.car MEV3.comp1	2.0	
analysis.car CBE3	6.0	
analysis.car CBE3.comp2	4.0	
analysis.car CBE3.comp1	2.0	
analysis at 2020.03.12 12.08		-4.0
analysis.car MEV2	4.0	
analysis.car MEV2.comp2	3.0	
analysis.car MEV2.comp1	1.0	
analysis.car CBE2	8.0	
analysis.car CBE2.comp2	7.0	
analysis.car CBE2.comp1	1.0	
analysis at 2020.03.12 12.03		-4.0
analysis.car MEV1	4.0	
analysis.car MEV1.comp2	3.0	
analysis.car MEV1.comp1	1.0	
analysis.car CBE1	8.0	
analysis.car CBE1.comp2	7.0	
analysis.car CBE1.comp1	1.0	