03 Systems Design Management

Table of Contents

List of Tables

List of Figures

1 Introduction

The Systems Design Management Case Study addresses how to define and specify the structure, function, and behavior of a system.

2 Behavior Modeling

Many projects model only the system structure and interfaces. However, missing to specify the system behavior, using behavior modeling, typically results in serious integration problems. Therefore we highly recommend capturing the system behavior in the system model using SysML activities, state charts and sequence diagrams.

2.1 Modeling Activities

Activities define the workflow of actions to perform, the input and output of the actions and the decisions/condition-dependent sequence of the actions. The model shows at the same time the physical effect of a system (like distortion of wave front) as well as sensing, actuating actions and control flows. We need to describe the wave front control scheme of APE, the relationships among internal/external disturbances, opto-mechanical effects, and control decisions, to understand the context of those decisions and to refine the control use cases. The wave front control schemes are derived by analyzing the system requirements and the opto-mechanical system architecture. From this fairly static view of dependencies, a wealth of information can be derived, like interfaces among components, communication network dimensions, synchronization requirements and required sensors and actuators. Figure 69 shows the top level behavior of APE; i.e. the evaluation of the phasing techniques. The model of the Wave front control becomes the central piece for further activities. The modeling element of choice is an activity diagram, using the SysML specific add-ons for systems modeling (rate, continuous flows, etc.). This prescriptive model maps the relevant information to activity elements (e.g. disturbances to object nodes, opto-mechanical effects to actions) and all the relevant information is kept in one diagram, thus enabling proper effects and relationship analysis.

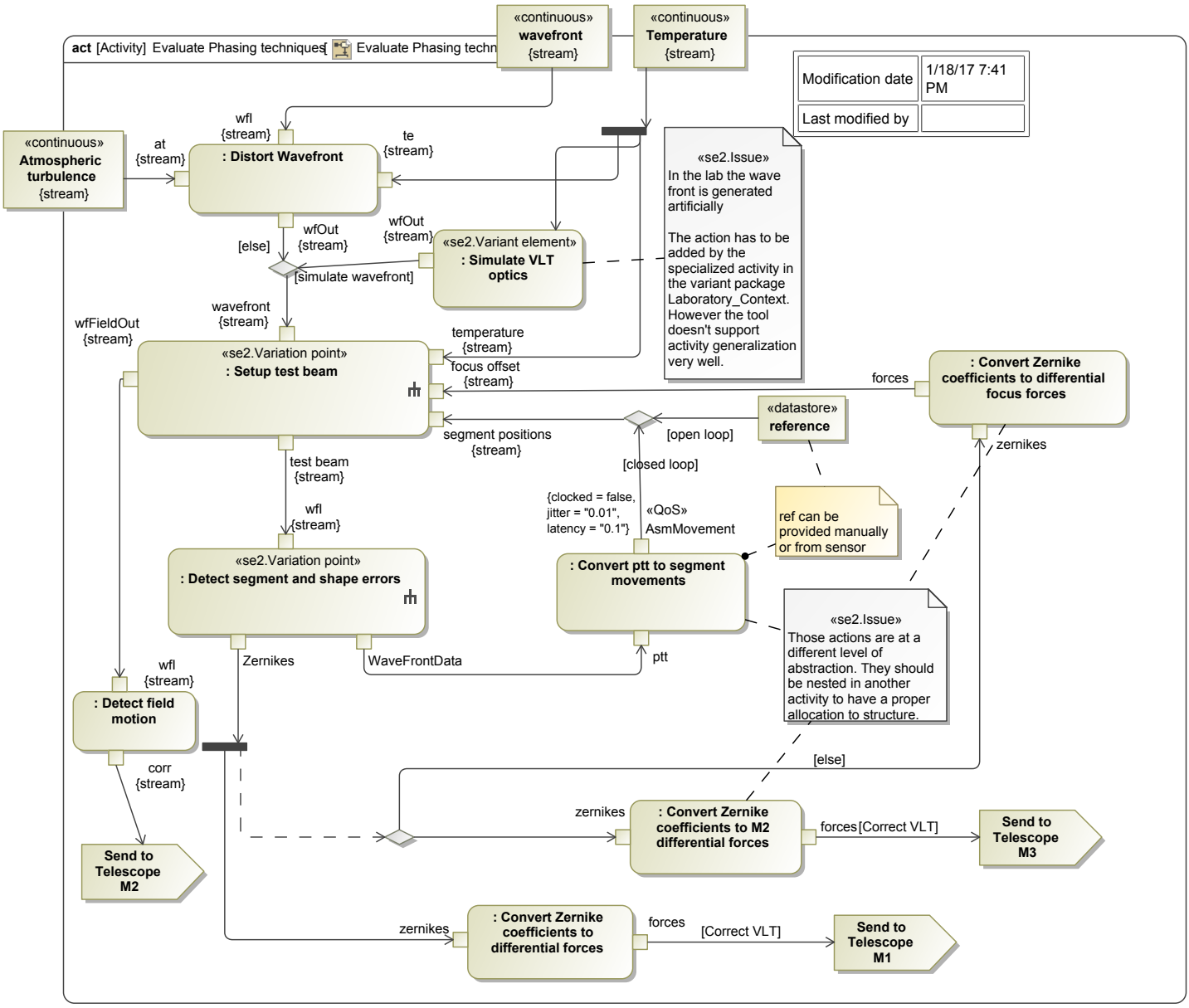


Figure 1. Evaluate Phasing techniques

2.1.1 What is the relationship of Activity, Activity diagram, Action, CallBehaviorAction?

An action is never actually defined through an activity diagram. If an action is a usage of an Activity (within another activity) then the Activity that types the action is defined through an activity diagram.

2.1.2 How can I model a decision in an activity which is taken asynchronously (like an operator decision)?

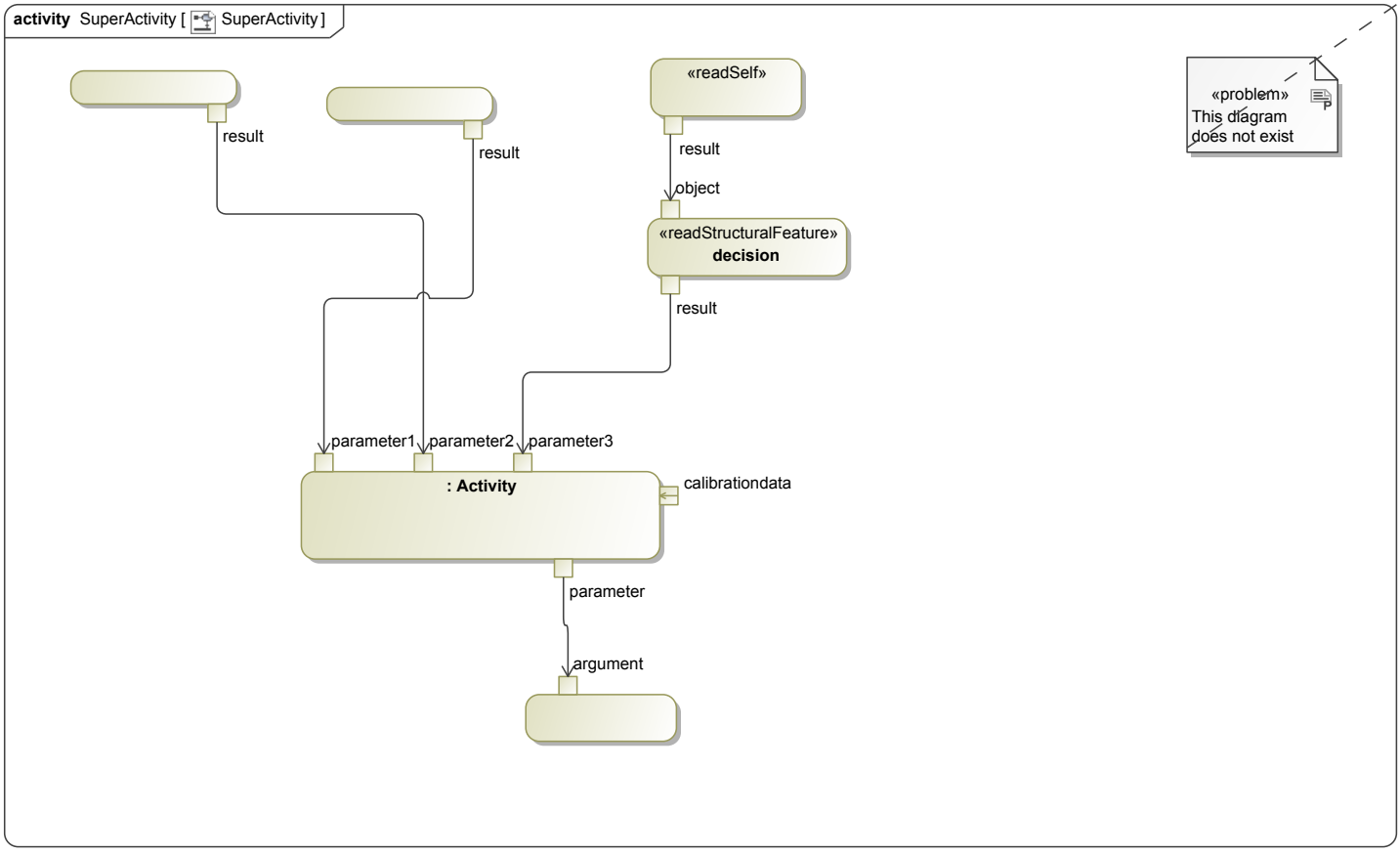


Figure 2. SuperActivity

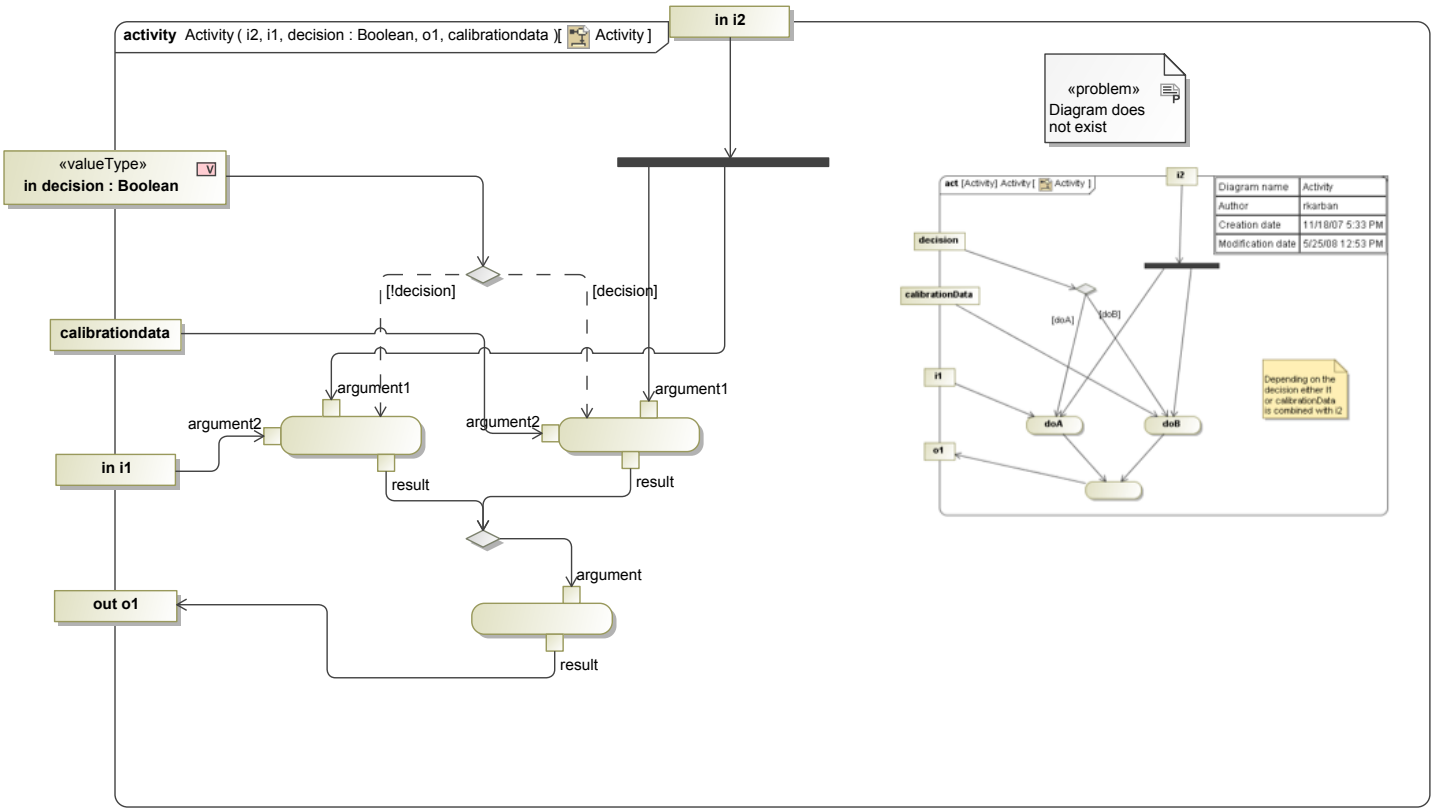


Figure 3. Activity

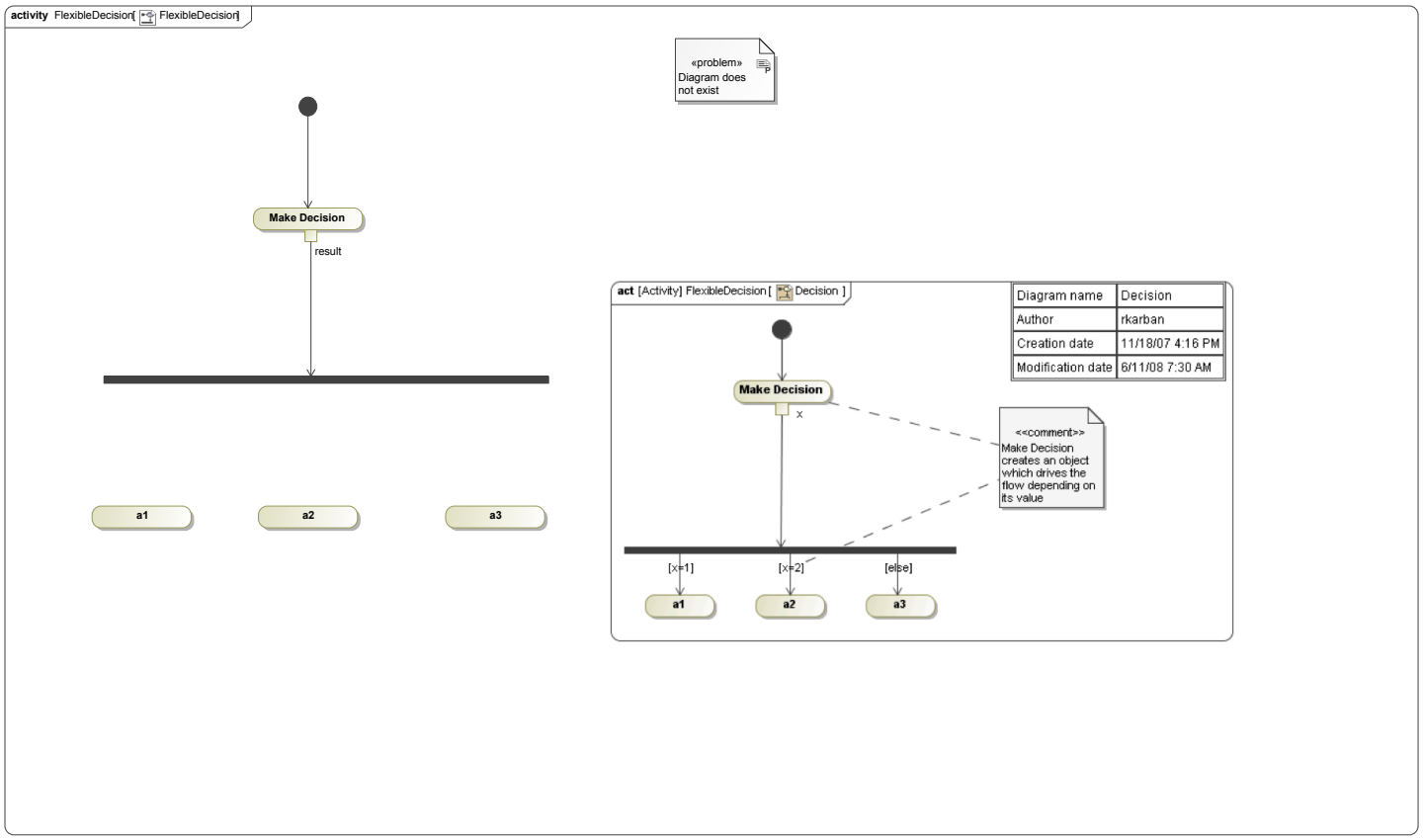


Figure 4. FlexibleDecision

2.1.3 When should I use <<discrete>> or <<continuous>> in activity diagrams?

On a philosophical basis, we would expect things that are defined by their <> to be continuous, while items which have discrete properties have sudden steps in rate. For example, when modeling some infra-red detectors one needs to consider (count) photons. They arrive randomly and have a distribution (typically Poisson). However light, as mentioned below is often thought of as continuous. The choice between <> / <> is a critical modeling assumption and is an important contextual decision. To get it 'wrong' constrains the modeling. Additionally this include the decision about <> or <> time, that is, can we resolve the (potential) step changes in rate. You should treat air, light, liquid as <> object flows, where the object is e.g. photons, etc. Information is a normal object flow. You can define an interruptible region with a timeout to create ”timed“ flows. In this region you could define an action which creates a <> flow you process later on. Or, you use a control operator with the control value enable/disable which enables/disables the action which produces light. In Model-based Systems Engineering, CRC Press, 1995, A. Wayne Wymore described quite rigorously the transformation from discrete to continuous and back.

2.1.4 How do I represent control loops?

Use <> activity diagrams with streaming and non streaming activities. The following shows an integrator.

2.2 Guidelines for Modeling Non-Functional Aspects

2.2.1 Quality of Service

2.2.1.1 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

3 Structure Modeling

To model the structure of a system, SysML offers two kinds of diagrams that works together: Block Definition Diagrams (BDD) show the architecture building blocks of a (sub-)system in form of a product tree. Furthermore, they define the properties and features of the building blocks. The Internal Block Diagram (IBD) shows how the building blocks are connected to realize the (sub-)system.

3.1 Starting to Build a Design Model

For easy navigation you should create always pairs of IBDs and a BDD. From the IBD you should be able to navigate to the BDD which defines the Block which is the context of the IBD. From the BDD you should be able to navigate to the corresponding IBDs of a Block. Design your building blocks without any reference to a connection context then connect them up into higher and higher assemblies. You are indeed encouraged to build the (sub-)system by “dragging” blocks from the BDD to become parts in the IBD like an engineer building assemblies. Define all the blocks in a BDD (Figure 37) and then use the blocks. This BDD defines the product tree of APE. APE consists of an ActiveSegmentedMirror, a GuidingCamera, etc. You can then GO BACK to the BDD to constitute a composition from the (sub-)system to the building block.

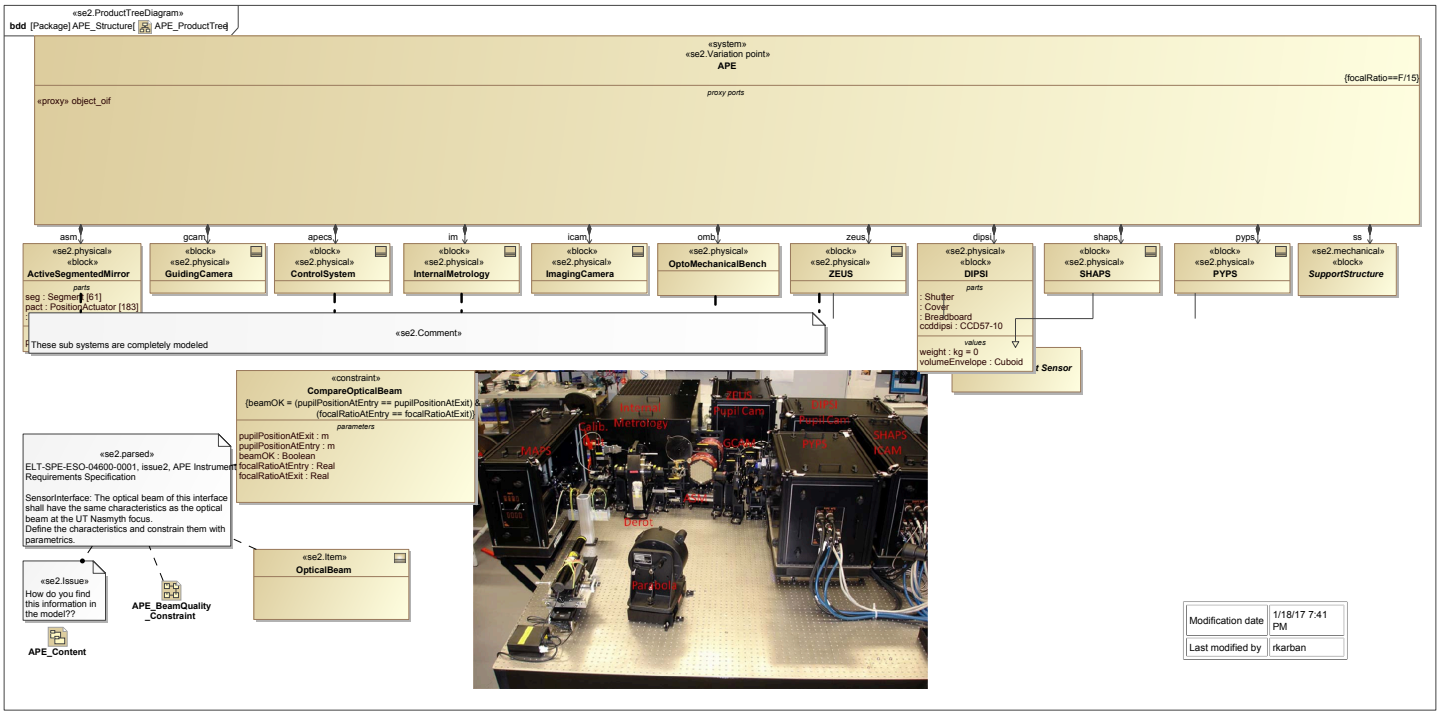


Figure 5. APE_ProductTree

3.2 Structure Breakdown

3.2.1 Definition of System Hierarchies

Hierarchic breakdown of a system into smaller units is a well-known and always-used principle in systems engineering. Unfortunately the analysis of a system is not an unambiguous process. Several models can represent the same system in reality. It is therefore necessary to define some guidelines for the modeling of hierarchies in the SysML model. In the Product Tree of the ZEUS Local Control System (ZeusLCS, Figure 38) The ZEUS opto-mechanical system is referenced. The ZeusLCS is assembled from some electronic boards, like a motor controller (MAC4INC) and an amplifier (VMESA01). The Information View (IBD) shows how information flows between the parts of the control system and the sensors and actuators of ZEUS (Figure 39). The electrical connections can only be shown in the IBD of the APE system, at the next higher level of the structure because several substructures of APE are involved, like ZEUS, ZeusLCS, and the JunctionBox (Figure 40). Alternatively, the connections could also be shown in the IBDs of the substructure, using references, using shared part properties like the dashed ZEUS part in Figure 39 but this maybe end up in a maze of references and potentially creates confusion. The modeling of hierarchies in the SysML model is very closely related with the question, at what system level interfaces (represented by SysML ports) shall be attached to a block. Examples for these issues are:
• Differentiation between functionally grouped (sub) systems with abstract interfaces vs concrete components with real interfaces: EXAMPLE: The entertainment system in a car consists of many components: speaker, radio, TV, cables, etc. These components are

modeled as parts of the entertainment system, which is modeled as a subsystem of the complete car. The components are distributed all over the car (geographically), e.g. the loudspeaker is built into the door. It is very difficult, or even impossible, to model the entertainment system as black box without modeling the components of it, because the entertainment system is just an artificial grouping of "real" blocks. If one wants to model the interfaces of the entertainment system, this is not possible at the subsystem level directly, because only (concrete) components have real interfaces. Therefore the components must be modeled and the ports then attached to these blocks. Use the stereotype <> for those types of systems. The other parts of the car can reference (shared property) the components of the entertainment system to indicate their physical location. • Modeling of connectors crossing several levels of a system hierarchy: EXAMPLE: If one wants to model the connection between one component of the entertainment system (e.g. radio) with one component of the power supply system (e.g. battery), this can only be done with a connector directly from the radio port to the battery port. This looks simple at the moment. However, the connector between both components crosses two levels of hierarchy on its way: If one would just take a look at the subsystems, entertainment system and power supply system, you would also see the connection, because radio and battery are parts of these subsystems. Junction ports can be used in this situation. The Local Control System (which is responsible for a substructure, like the ZEUS wave front sensor) of the APE Control System can be considered as a <> because it interfaces with sensors and actuators of the opto-mechanical system. The Local Control System “creeps” into the opto-mechanical system and has interfaces with deeply nested parts of the system under control. Nevertheless, the Local Control System can be tested standalone (using simulation) and can therefore be considered as a subsystem of its own right. From the point of view of the supervisory part of the Control System it appears like a black box with a well defined interface that hides the internals and delegates the requests coming from the supervisory part.

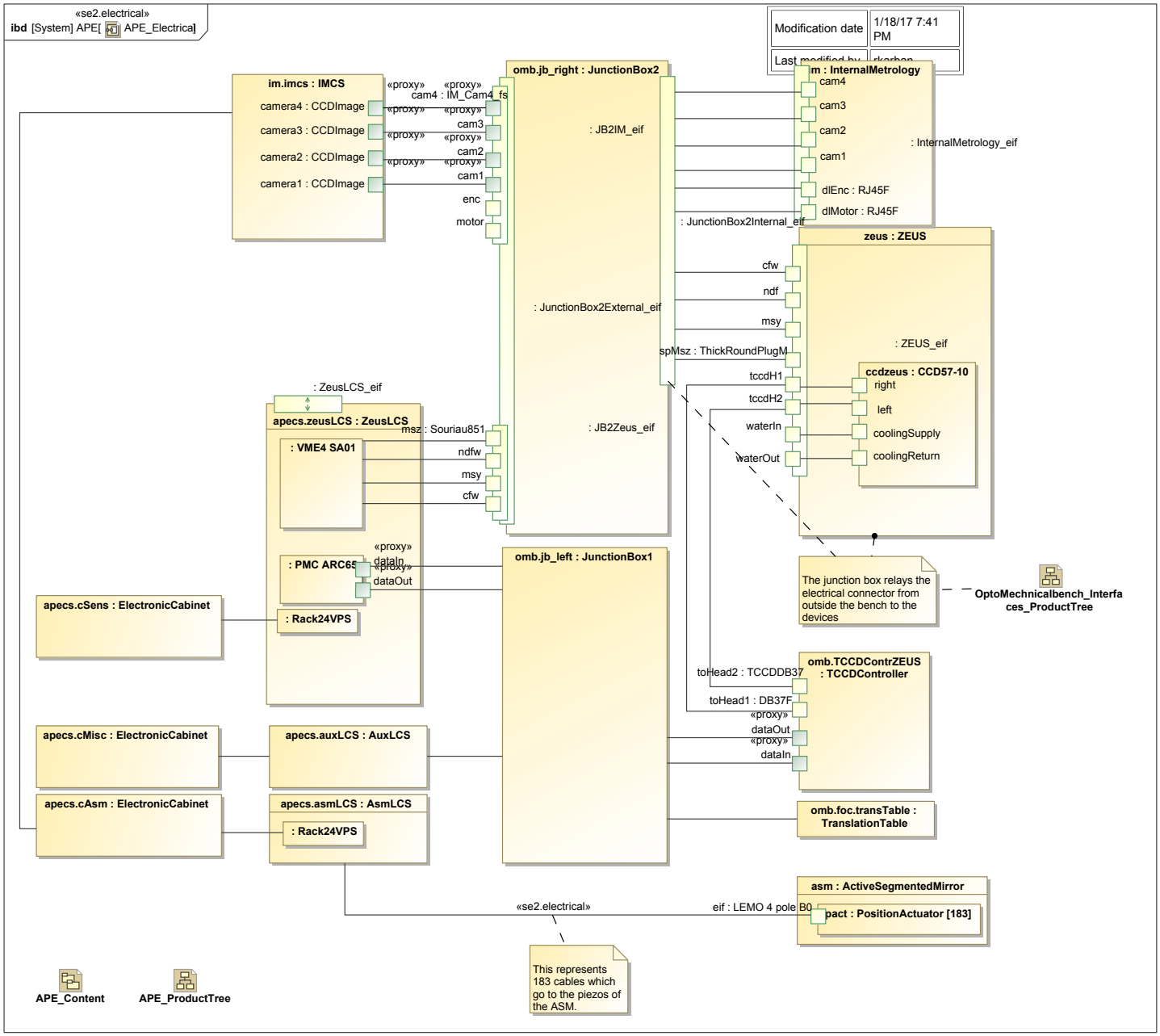


Figure 6. APE_Electrical

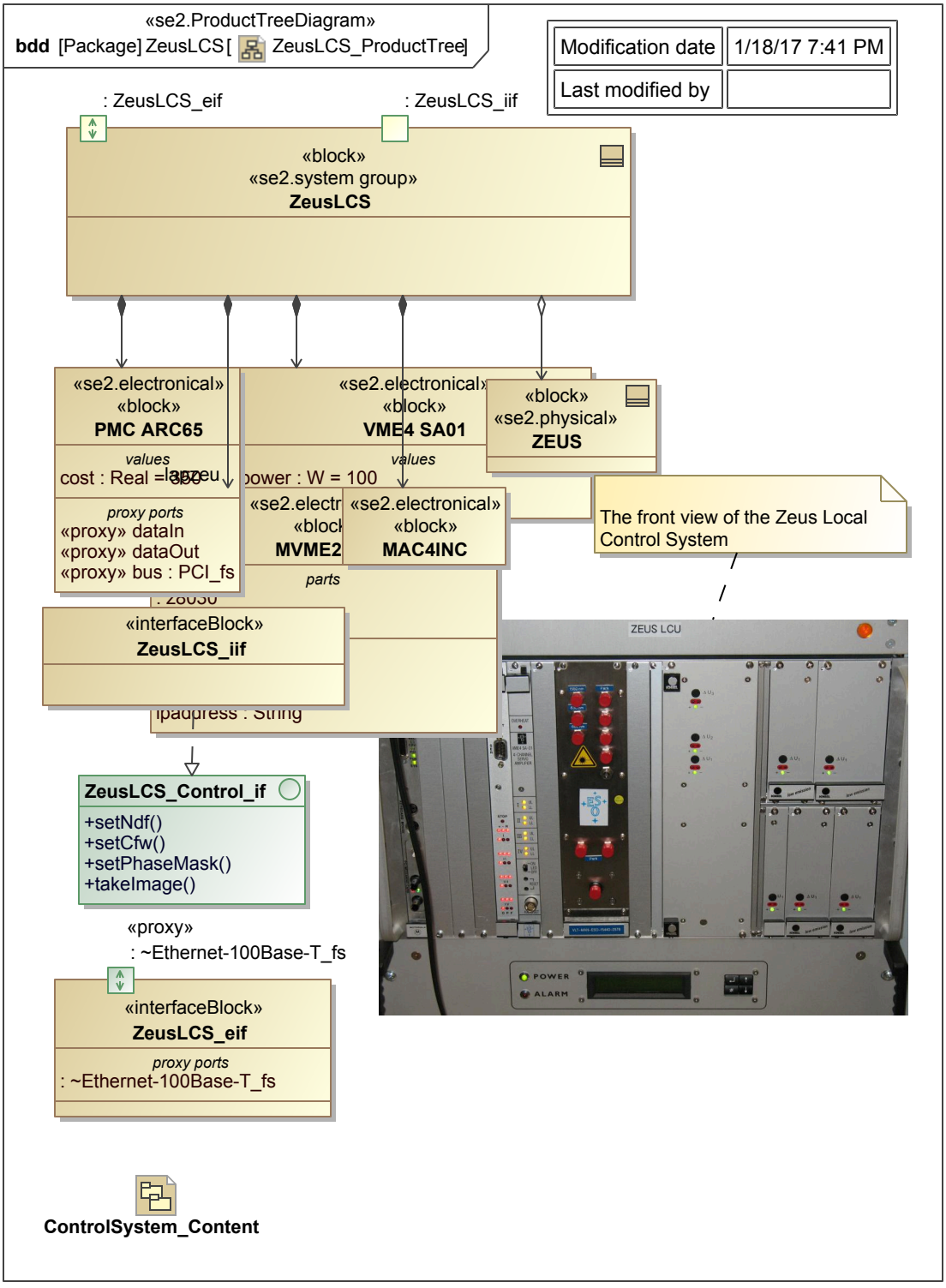


Figure 7. ZeusLCS_ProductTree

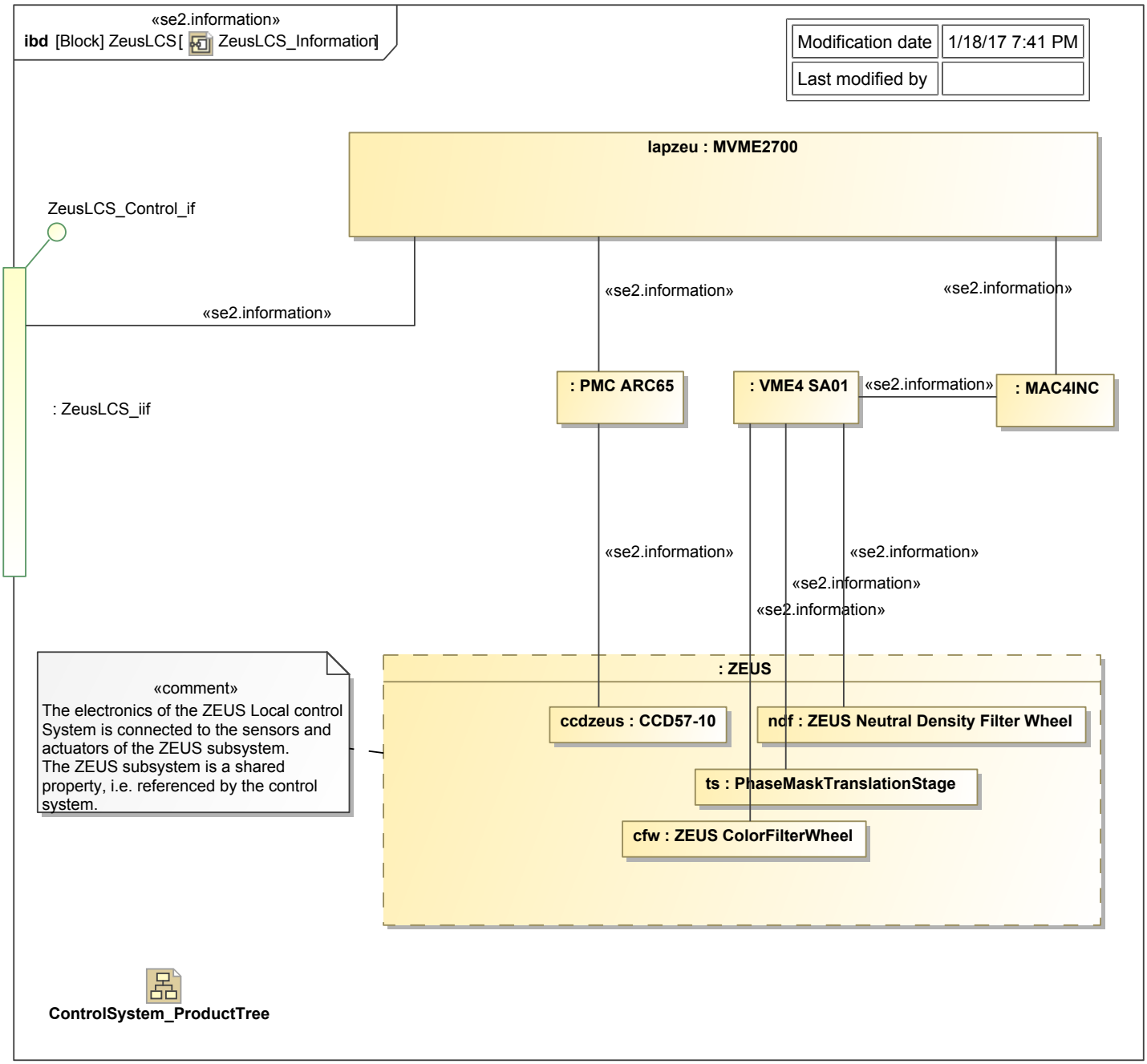


Figure 8. ZeusLCS_Information

3.2.1.1 SysML elements to model connected nested structures

The basic elements to model connected nested structures are ports and connectors. Usually, an interface is seen as a part of a subsystem. e.g. the Nasmyth platform is part of the main structure, the SCPs are part of the telescope, electrical sockets in a building are interface at the outmost layer but are hierarchically very deep. Depending on what needs to be shown there are three different approaches: 1. A block is used to type a standard port and represents at the same time the part property without explicitly modeling as separate property. In the case it is not important to show it as part property and to have a physical property (like a plug) visible at the border of a block. Complex (nested) ports are used (15-N-A:SCP Part A in Figure 42). It's a simple approach, but the element is not listed as a part in the product tree. 2. Connect directly to the part property or the part property's port (15-N-B:SCP Part B in Figure 42). Also a simple approach, but the connection is hidden when the inner parts of the enclosing part are hidden. Relay through the port at the border to a part property in case the part property is important and needs additional modeling. In this case stereotype the port as <>. The stereotype classifies the outer border port simply as representation of an inner port but not as a part of its own right. Junction ports do not delegate connectors, they are rather a window into the part and simply relay the connection (15-N-C:SCP Part C in Figure 42). The modeling of this approach is more complex, but the elements are listed in the product tree and there is a connection between the enclosing elements even when the inner parts are hidden.

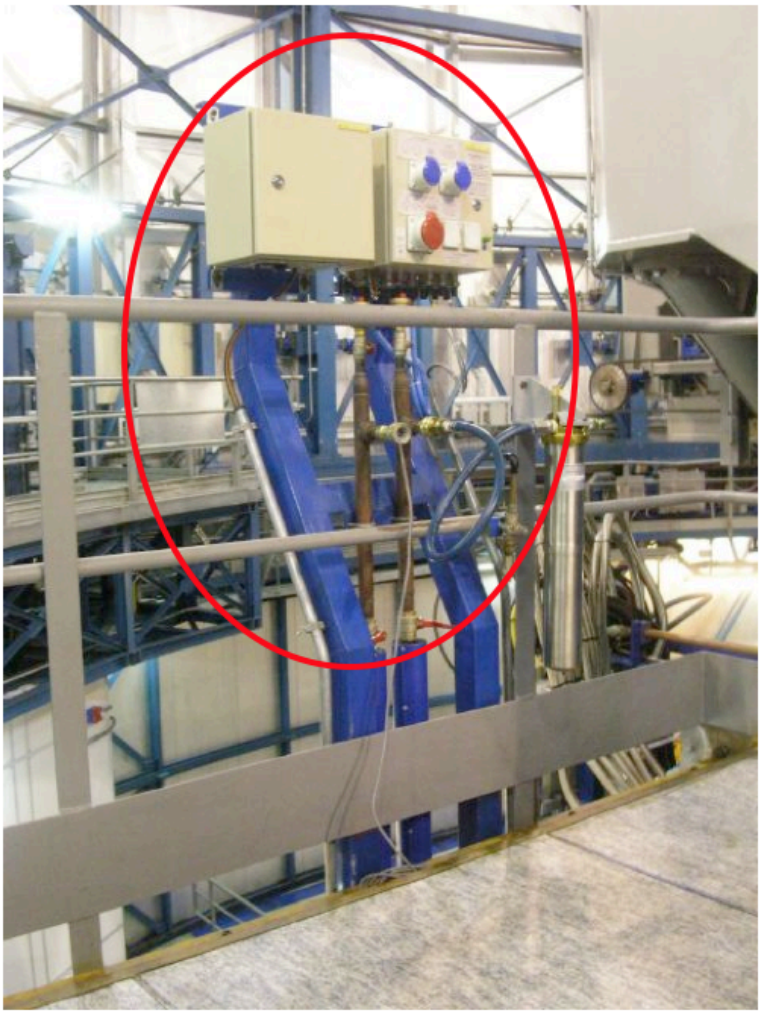


Figure 9. SCP at a Nasmyth Platform

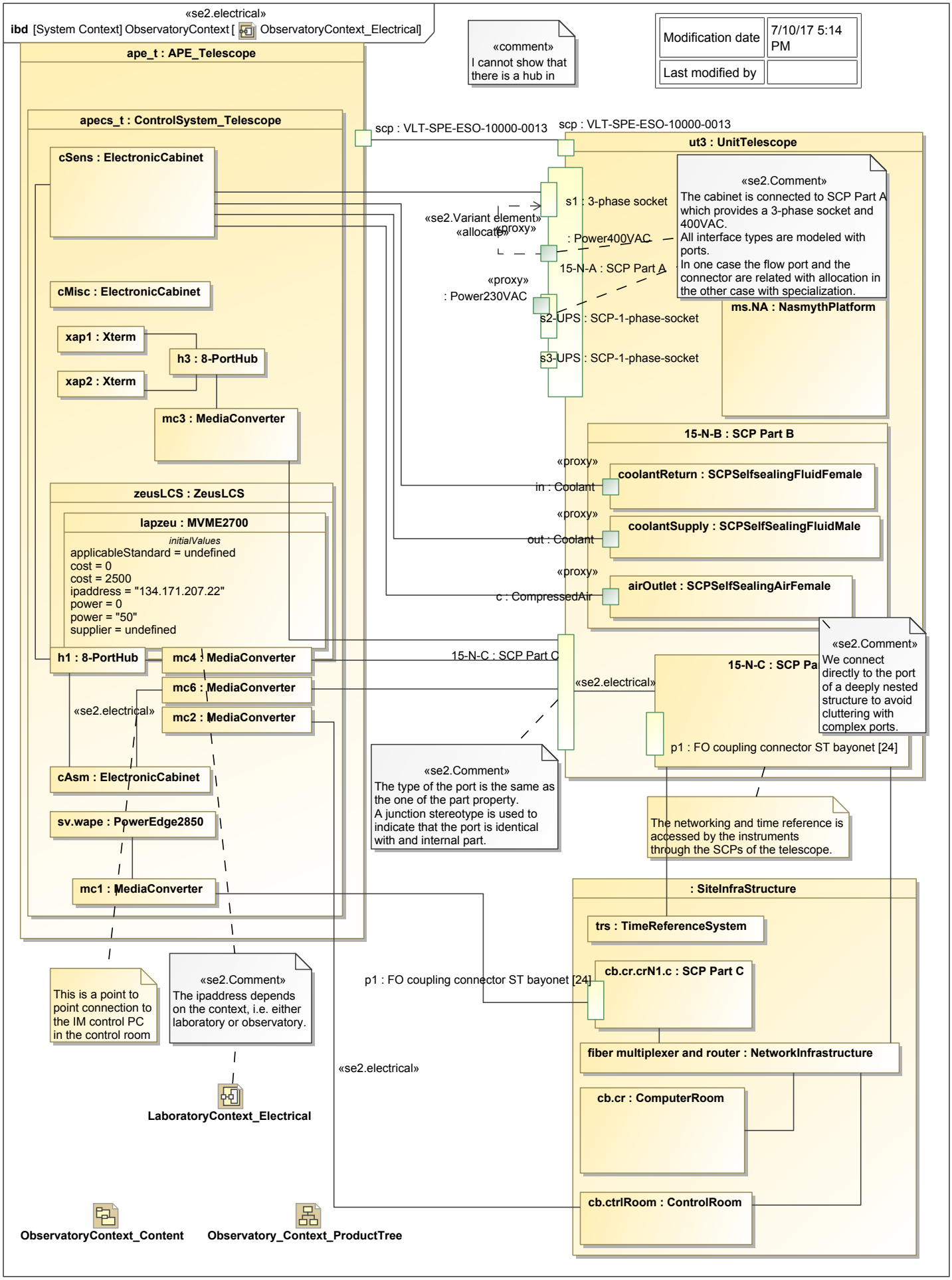

«se2.electrical»
ibd [System Context] ObservatoryContext [ObservatoryContext_Electrical]
«comment»
I cannot show that there is a hub in
Modification date
7/10/17 5:14 PM
Last modified by
ape_t : APE_Telescope
apecs_t : ControlSystem_Telescope
scp : VLT-SPE-ESO-10000-0013
scp : VLT-SPE-ESO-10000-0013
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
cMisc : ElectronicCabinet
xap1 : Xterm
h3 : 8-PortHub
xap2 : Xterm
mc3 : MediaConverter
ms.NA : NasmythPlatform
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]
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.
fiber multiplexer and router : NetworkInfrastructure
«se2.electrical»
cb.cr : ComputerRoom
LaboratoryContext_Electrical
cb.ctrlRoom : ControlRoom
ObservatoryContext_Content
Observatory_Context_ProductTree

Figure 10. ObservatoryContext_Electrical

3.2.2 How do I distinguish a sub system and an assembly

It is sometimes difficult to define what a subsystem or merely an assembly is. Follow this definition: A subsystem is a set of interdependent elements constituted to achieve a given objective by performing a specified function, but which does not, on its own, satisfy the customer's need. Following this definition they are either a package (like the ASM in Figure 43) on their own or simply a part of a package (like LensGroup1 in Figure 44). Do NOT use the stereotypes <> or <> unless it is well defined. The definitions differ from one environment to another. In APE we use only plain block and the stereotype <> to indicate blocks that do not belong to the system under design. In the Organization Ontology we stereotype to those elements either as <> (a separate recursive package structure is created) or as <>, <>, <>, <>, <>. <>, <>, <>, and <> form the leaves of the product tree which are not further decomposed at system model level. A Block stereotyped as <> has a special meaning. It serves as a (modeling) container for leaf elements. Typically, leaf elements are reusable things, out of a catalog. When the system is assembled those parts are connected. The <> container serves as the context where those parts are connected. For example, the ZeusLCS (Figure 38), consists of a VME crate, a backplane, a CPU, IO boards etc. The Block ZeusLCS does not exist as a tangible item in the real world but its parts do. It serves as a container to show how the crate is connected to the CPU, how the CPU is connected to the IO boards, etc.

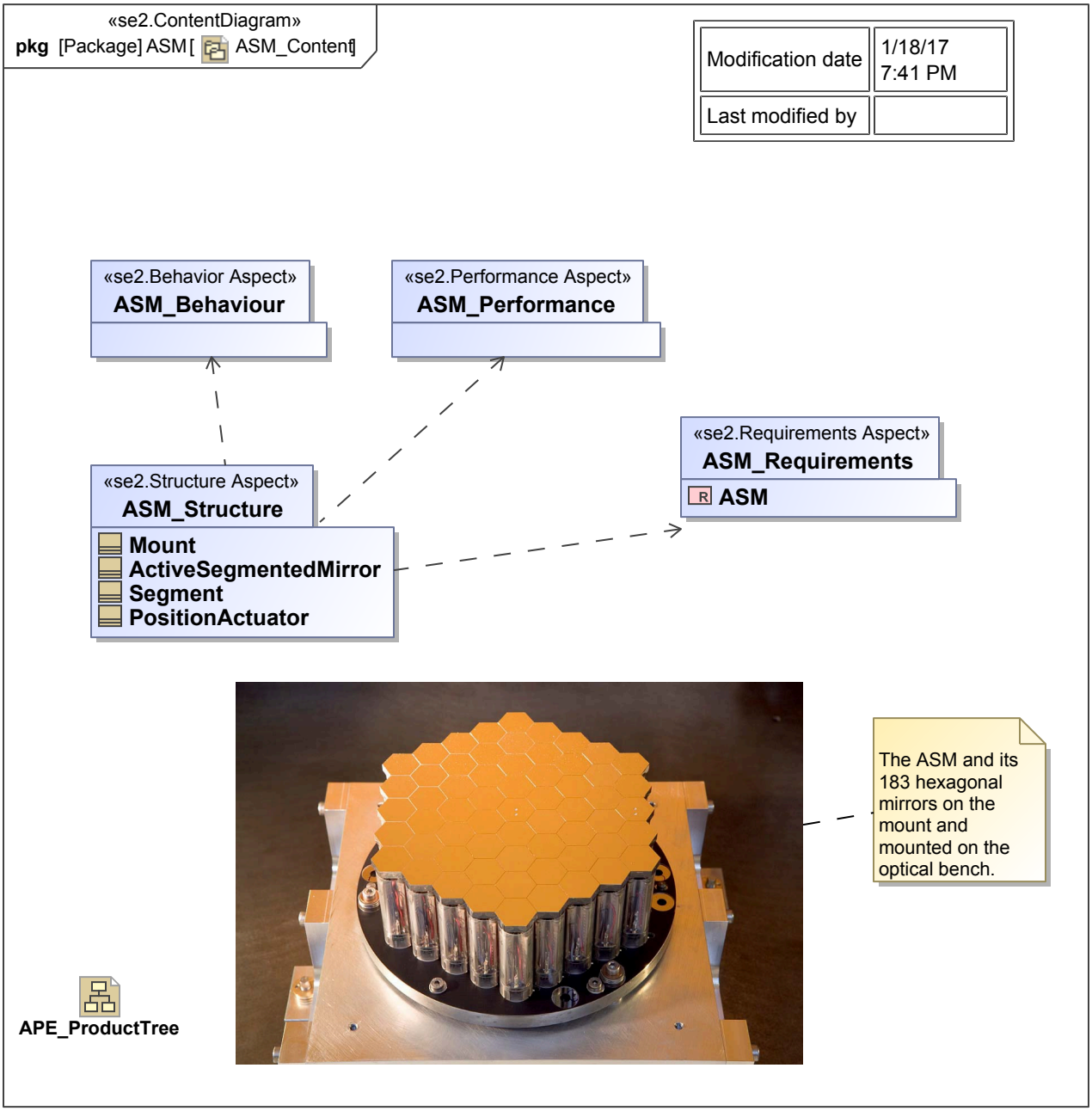


Figure 11. ASM_Content

«se2.ProductTreeDiagram»
bdd [Package] OptoMechanicalBench_Structure[OptoMechanicalBench_ProductTree]

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

«se2.physical»
OptoMechanicalBench

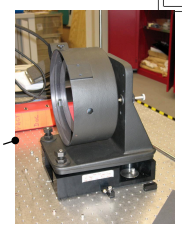
dummyASM → «block» «se2.physical» **SegmentedDummyMirror**

The dummy is only used for alignment and calibration. In operation the real ASM is used.

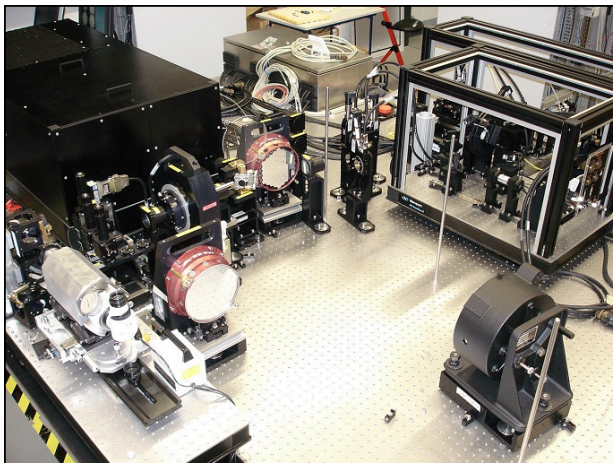
cu → «block» «se2.physical» **CalibrationUnit**

lg1 → «block» «se2.physical» **LensGroup1**

The parabolica is on a mount and screwed to the bench



parabolica → «block» «se2.physical» **OffAxisParabolicMirror**



cover → «block» «se2.physical» **OMBCover**

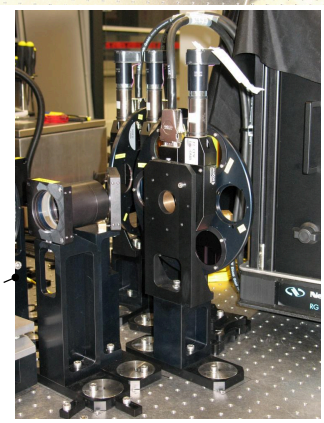
lg3 → «block» «se2.physical» **LensGroup3**

derot → «se2.physical» «block» **Derotator**
values
lightTransmission : Real = 0.975

lg4.1 → «block» «se2.physical» **LensGroup4**

lg4.2 →

lg5 → «block» «se2.physical» **LensGroup5**



fm2 → «block» «se2.physical» **FoldingMirror2**

fs → «block» «se2.physical» **FieldSelector**

The wheels´ mount is screwed to the bench. The motor attached to the mount, and the disc to the motor

smpp → «block» «se2.physical» **SecondaryMirrorPistonPlateWheel**

apcfw → «block» «se2.physical» **APE ColorFilterWheel**

cf 8 → «se2.optical» «block» **ColorFilter**
values
wavelength : nm{unit = nanometer}
bandwidth : nm{unit = nanometer}
lightTransmission : Real = 0.99

«block» «se2.physical» **ColourFilterWheel**

The wheel is the same for sensors and bench only the filters change.

apndf → «block» «se2.physical» **APE Neutral Density Filter Wheel**

«se2.physical» **NeutralDensityFilterWheel**

nf 5 → «block» **NeutralDensityFilter**

TCCDContrSHAPS
TCCDContrPYPS
TCCDContrDIPSI
TCCDContrZEUS
TCCDContrICAM
TCCDContrGCAM
→ «se2.electronical» «block» **TCCDController**
values
power : W = 150
cost : Real = 5000
proxy ports
«proxy» dataIn
«proxy» dataOut

fm1 → «block» «se2.physical» **FoldingMirror1**

jb_left → «se2.physical» «block» **JunctionBox1**

The JBs enable to transport the

Figure 12. OptoMechanicalBench_ProductTree

3.2.3 Where do I put systems which are part of the project and needed in different contexts but not part of the system itself?

It depends what they are used for. • Are they used for operation? Then they ARE part of the system. • Are they truly external? Then they appear only in the context diagram. • Are they specifically built to verify or test the system? Then they go in the Verification Aspect. In the APE case, a specific Star Simulator was built to simulate stars in the lab. It is called MAPS. MAPS is not part of the operational system but built for the APE experiment. MAPS appears as external in the context variant of the lab.

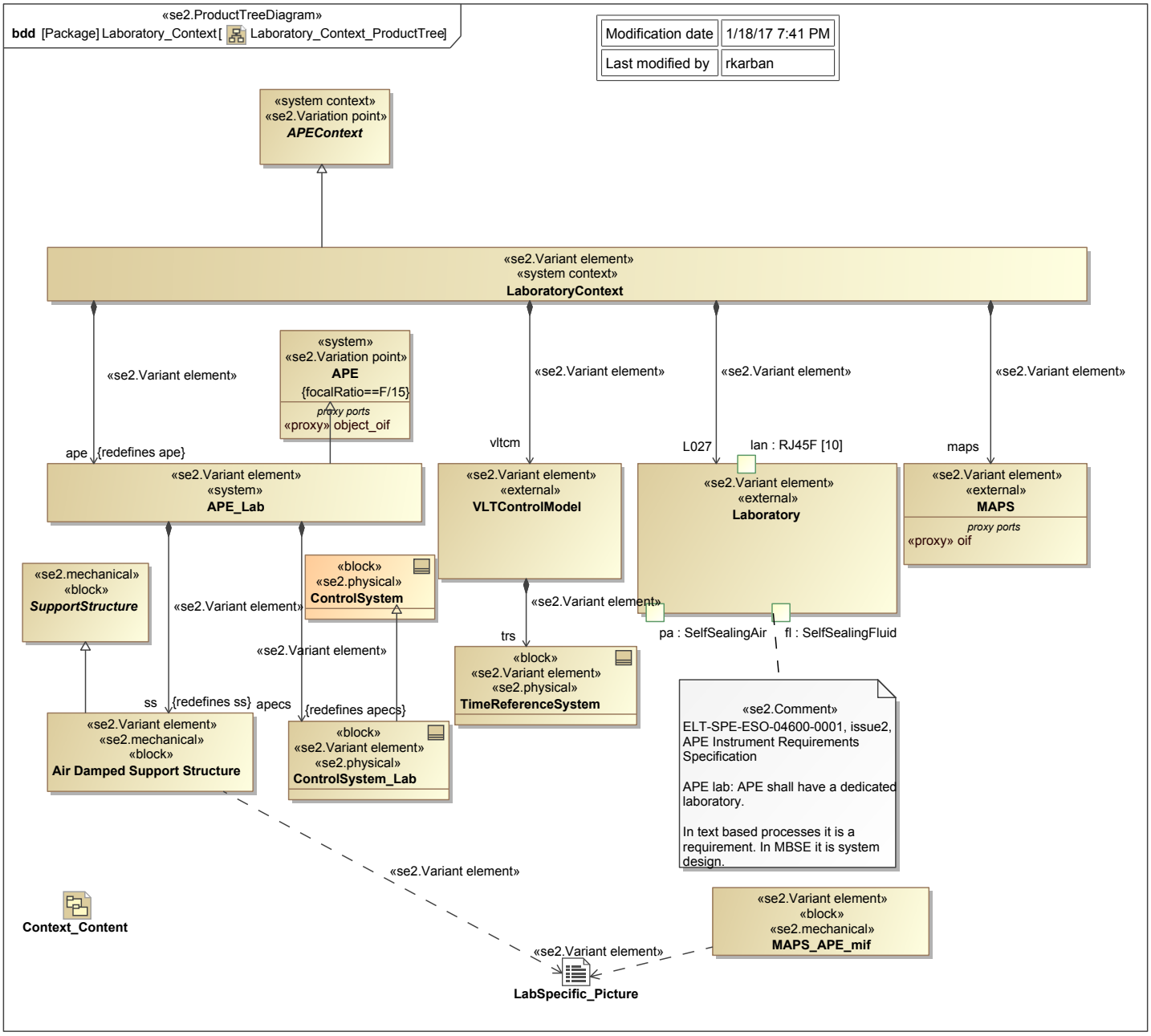


Figure 13. Laboratory_Context_ProductTree

3.2.4 Usage of <<external>>, <<system>>, <<subsystem>>

Let only your domains (top-level contexts) own the <>s. A BDD should definitely not mix up <>s like dataflow into an internal system. If anything, the system to be built should be THE <> or a <>, and anything that is a part of it should not be <>. That is a job for a <> or a domain.

3.3 Structure Relations

3.3.1 What is the relationship between part, property and block?

If A is a block and you drop it on an IBD of block B, A does NOT become a property of block B. A is a block; it remains a block. The association implies the creation of a block property of B that is typed by block A. Blocks dropped into an IBD with its frame on become parts properties (of the context block for the IBD) typed by the dropped Block. This is a very useful feature and a natural way of building progressively hierarchical systems. The fact that a given block is used in another structure is anathema. You can have any number of different contexts for the block of interest. A block is NEVER a part of a structure; a part typed by a block is.

3.4 Structure Properties

3.4.1 Representation of entities flowing in the system

Blocks or value types can be used to represent entities which flow in the system. They are the type of a Pin, an Activity parameter, a Flow Property, or an Atomic Port. • Use blocks to model structures, like systems or discrete entities (e.g. parcels on a conveyor belt). • Use value types to define types for quantitative characteristics (e.g. weight, speed, vector). These are typically things on which one has to operate. Value properties are always typed by a value type. • Model things flowing through the system, like (non-quantifiable) physical entities as <>; e.g compressed air with pressure, water, light beam, magnetic field, or electric current (Figure 46). <> is a stereotype of block .

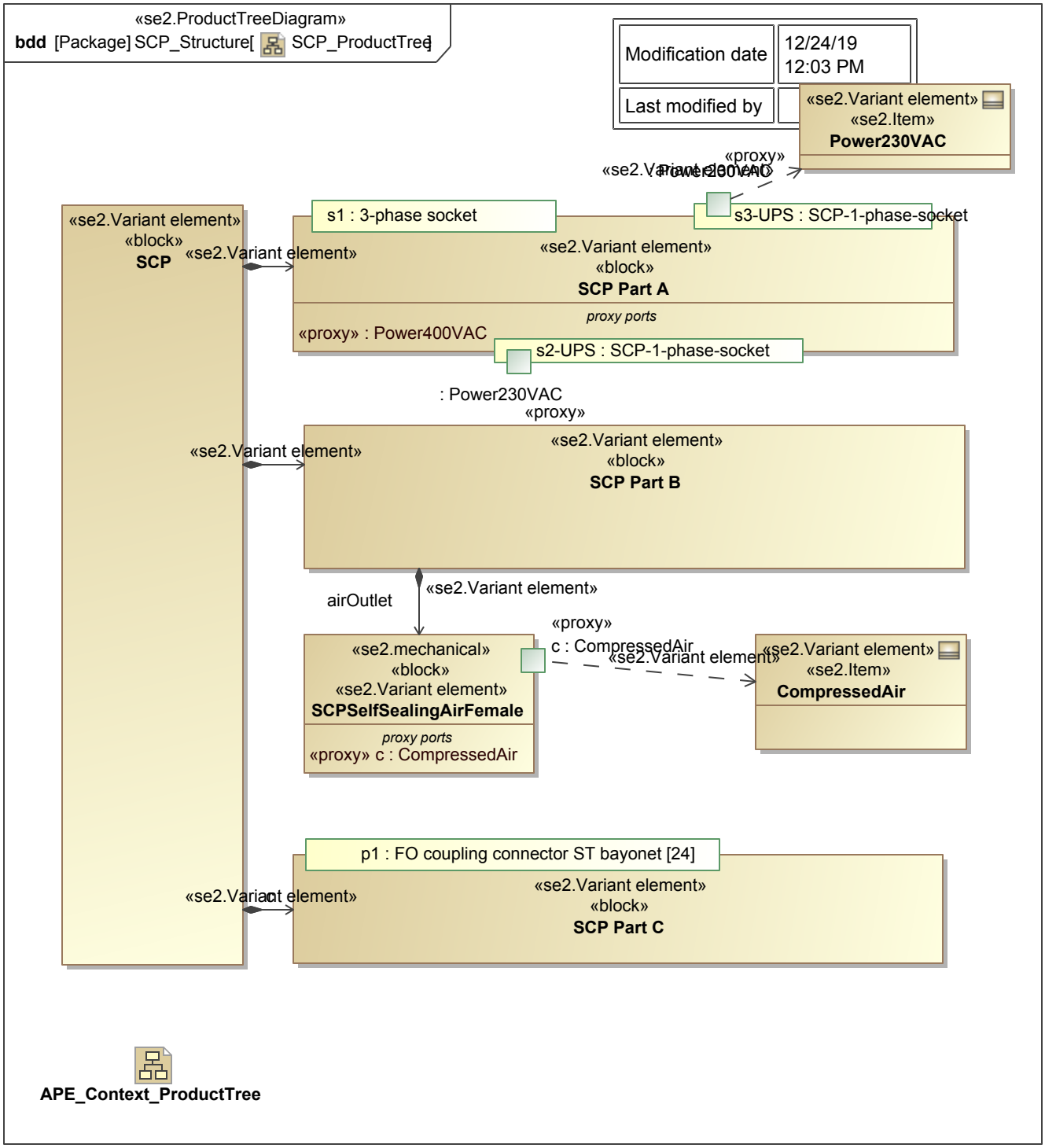


Figure 14. SCP_ProductTree

3.4.2 If I have blocks of the same type (like 10 FPGAs) in the BDD how do I properly use them on the IBD as different properties?

If you define a property with a multiplicity, like an array of FPGA you will not be able to distinguish them in the IBD. They are considered as a single property. This is fine if the system can treat them all the same way. If you want to treat them differently, e.g. there are different algorithm allocated to each of the FPGA or they have different electrical connections then you need to create for each of them a separate property or association with a role. The same block can serve to type different parts in an IBD. Each can have a different role name on the association end in the BDD.

3.4.3 Usage of public and private

Although SysML does not explicitly define visibility, most implementation do have a visibility tag for properties because they are built on top of UML. Private means that those elements are only relevant internally to a block. They are not of any concern externally. Public means that everybody has access to them. In SysML connectors can only cross the parts border and connect to nested parts if the isEncapsulated property of a Block is set to false. For an ICD only features accessible through a port are relevant.

3.5 Reuse of model structural elements

First it is important to realize, that parts are not reused, just types of parts are reused, i.e. a <>. A part is the role of a block in a certain context. If a catalog of re-usable elements (like CPUs, Motors, Cables, IO boards, etc.) is created, those elements are typically read-only and cannot be changed. Abstract types are used as placeholder for specific building blocks. They are classified in different packages. In order to assemble a system out of those re-usable elements a context or container is required. The product tree of this system is composed of all its (re-usable) elements, its parts. The engineering view (IBD) of this system shows how those parts are connected. This container is not a real, existing, physical part. It is used to model how the real parts are connected. For example, if you want to create a new VME computing node you need a VME crate, a CPU, and an IO board. All of them are re-usable elements, better the definition of those elements. In the real world you would plug the CPU and IO board into the crate in order to assemble it. This does not work in the model. If you create a new part of type CPU within the crate it would change the definition of the crate. All crates would suddenly have a CPU board as a part property. Therefore, a modeling container is used which is typed <>. They are a container for grouping of physical elements and their connectors, for reuse purpose. “Plugging” the CPU into the crate means in modeling term to create a connector between the CPU and the crate in the IBD of the containing block. Each of these containers contains one or more IBDs for different view (electrical, information, mechanical, optical). The real physical elements are stereotyped <>, <>, <>, <>. The following example shows a technical CCD (TCCD) system which consists of a PMC board which is mounted on a VME CPU board, a TCCD Controller box, a TCCD head, cables, and a higher level control system. The electronic cables are soldered to the PCB of the TCCD head. They have a plug at their end which is connected to intermediate cables, which are in turn connected to the TCCD controller. The cables soldered to the PCB are modeled as ports of the CCDhead, typed by a block which represents the soldered cable and its plugs going to the controller. The intermediate cables are modeled once as a simple block and once as an association block. Cables can be modeled as simple Connectors, as Blocks, or as Association blocks, as shown in Figure 51.

bdd [Package] TCCDHeadTypes [TCCDHeadTypes_Catalogue]

Modification date	7/10/17 5:14 PM
Last modified by	

«se2.electronical»
«block»
ElectronicPart
values
power : W = 0

«comment»
For the connection to the leftPCB we model the cable and define as type the TCCDHeadcable1

toController : ITT Cannon 2DA31S
toLeftPCB : TCCDLeftPCB
left : TCCDHeadcable1

toController : TCCD ITT Cannon 2DA31P
«proxy» toRightPCB : TCCDRightPCB «proxy»
ctrl : TCCDControl_fs
power : TCCDPower_fs
right : TCCDHeadcable2

«comment»
For the connector to the rightPCB we model only the physical interface but not the cable

«se2.electronical»
«block»
CCDHead
values
housingX : mm{unit = millimeter}
housingY : mm{unit = millimeter}
housingZ : mm{unit = millimeter}
chipX : pix = 0 pixel {unit = pixel}
chipY : pix = 0 pixel {unit = pixel}
pixelSize : nm = 0 nm {unit = nanometer}

mount

«proxy»
lc : Coolant
coolingSupply : TCCDCoolingConnectorSupply

: SelfSealingFluidMale
«proxy»
lc : Coolant
«allocate»
coolingReturn : TCCDCoolingConnectorReturn

«se2.electronical»
«block»
CCD57-10
properties
photonNoise
values
dit : s = 1.0
chipX : pix = 512 pixel {unit = pixel}
chipY : pix = 512 pixel {unit = pixel}
pixelSize : nm = 13 nm {unit = nanometer}
darkCount : e-1/pixel/s = 1
power : W = 50
cost : Real = 2500
quantumEfficiency : Real = 0.9
readoutNoise : e-1 = 9
binning

«se2.electronical»
«block»
CCD47-20
values
chipX : pix = 1024 pixel {unit = pixel}
chipY : pix = 1024 pixel {unit = pixel}
pixelSize : nm = 13 nm {unit = nanometer}
power : W = 50
cost : Real = 2500

TCCD_Content
PartsCatalogue_Content

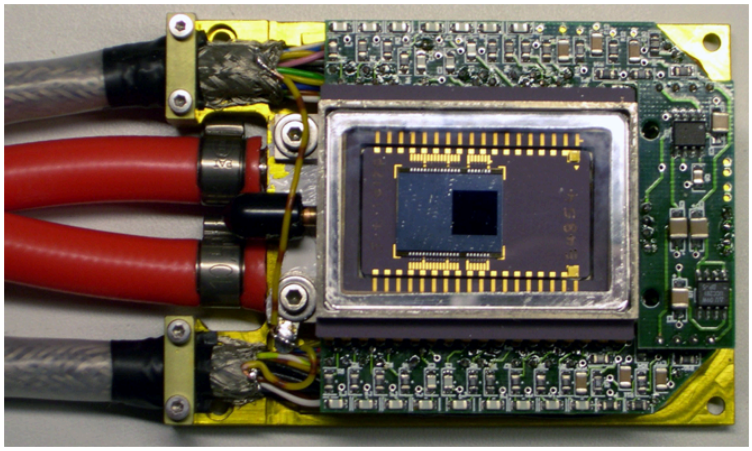


Figure 15. TCCDHeadTypes_Catalogue

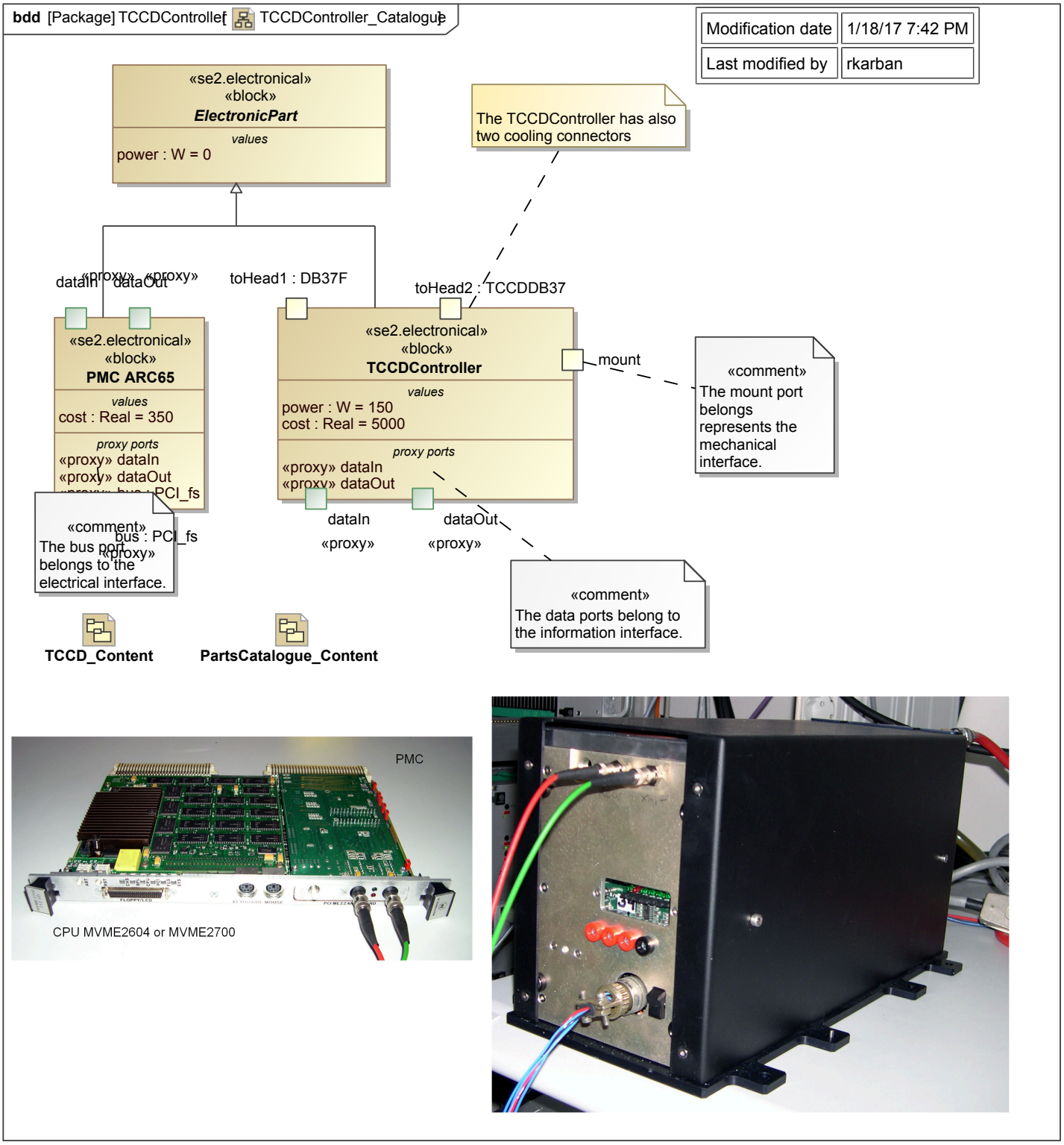


Figure 16. TCCDController_Catalogue

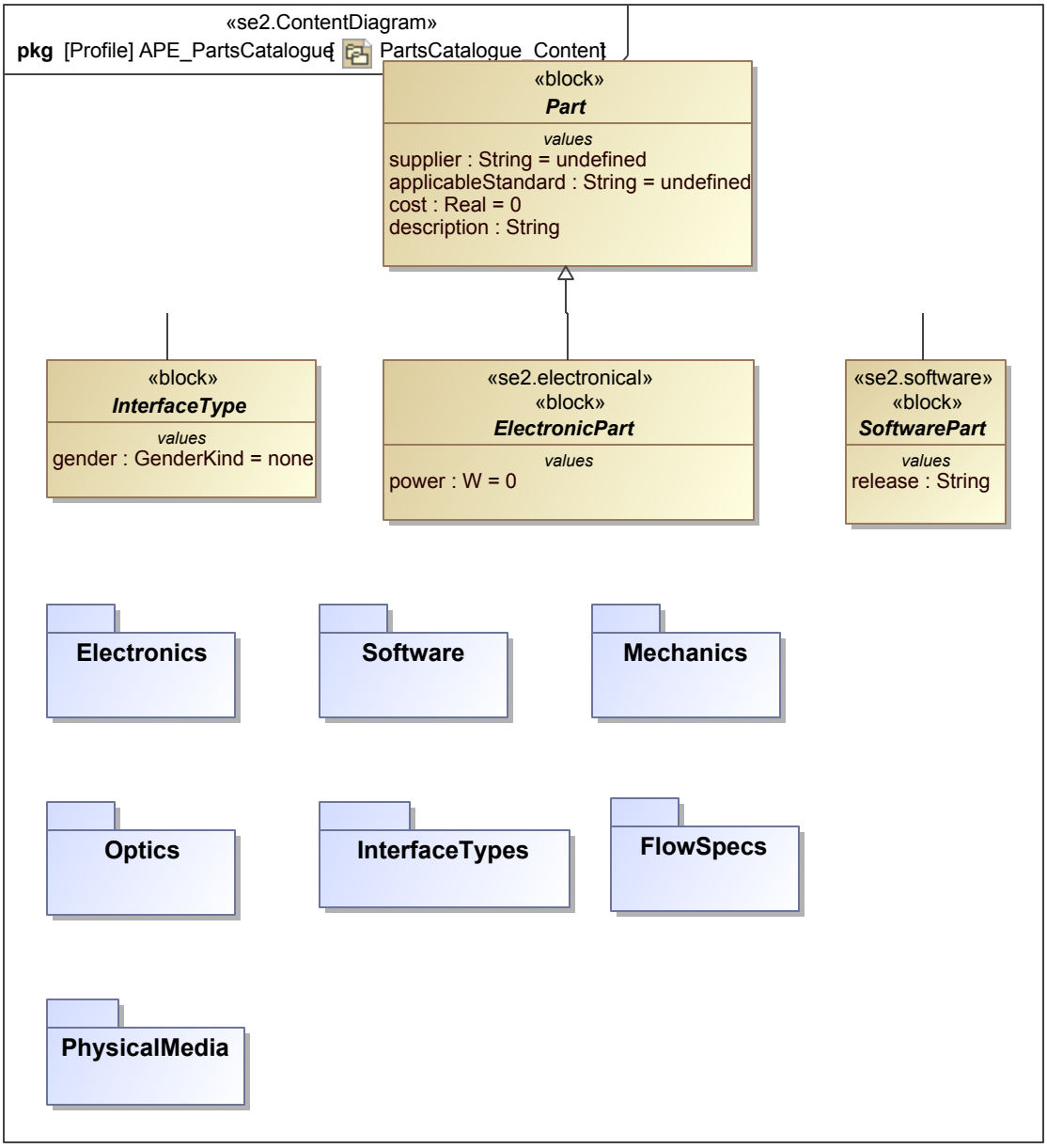


Figure 17. PartsCatalogue_Content

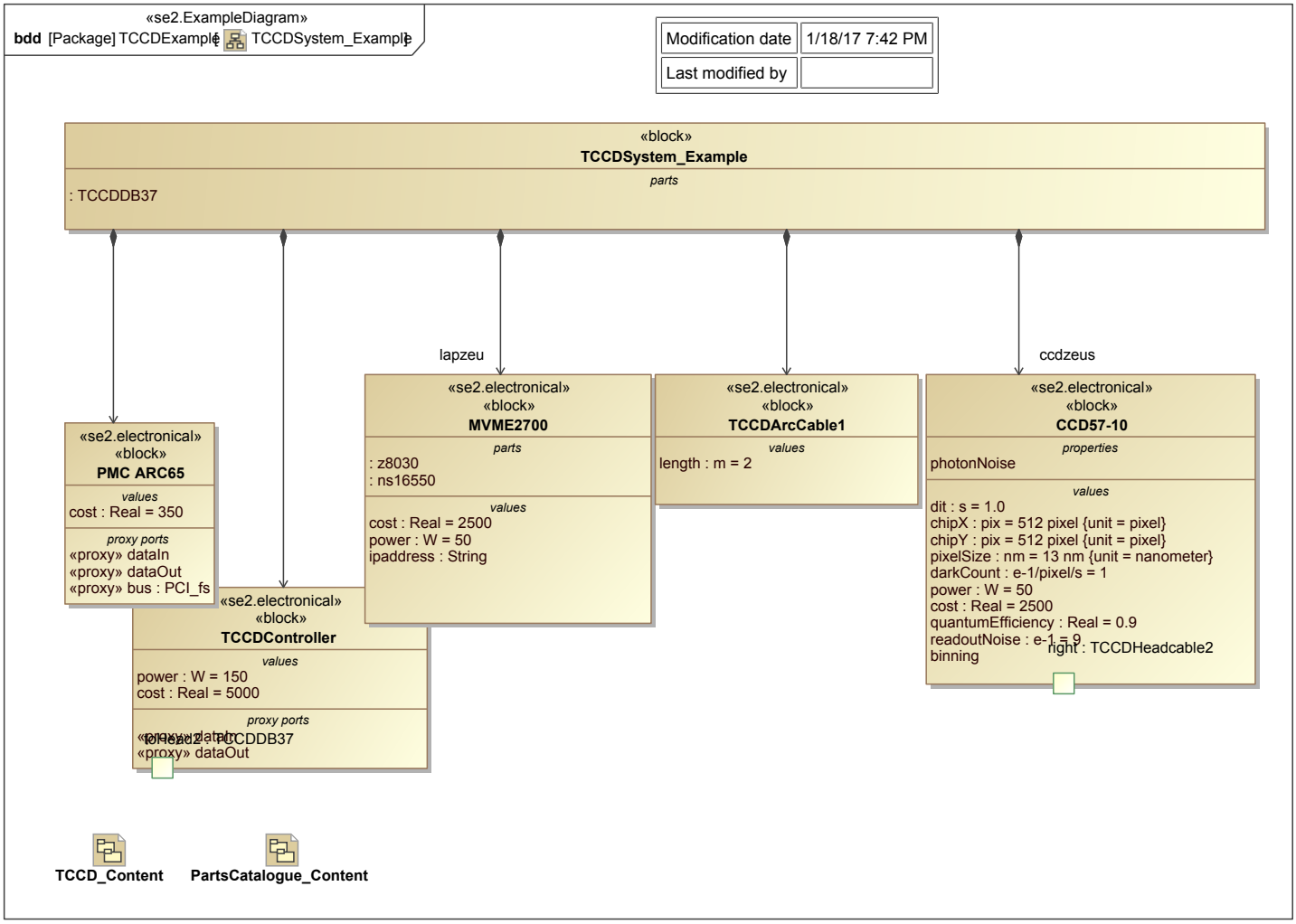


Figure 18. TCCDSystem_Example

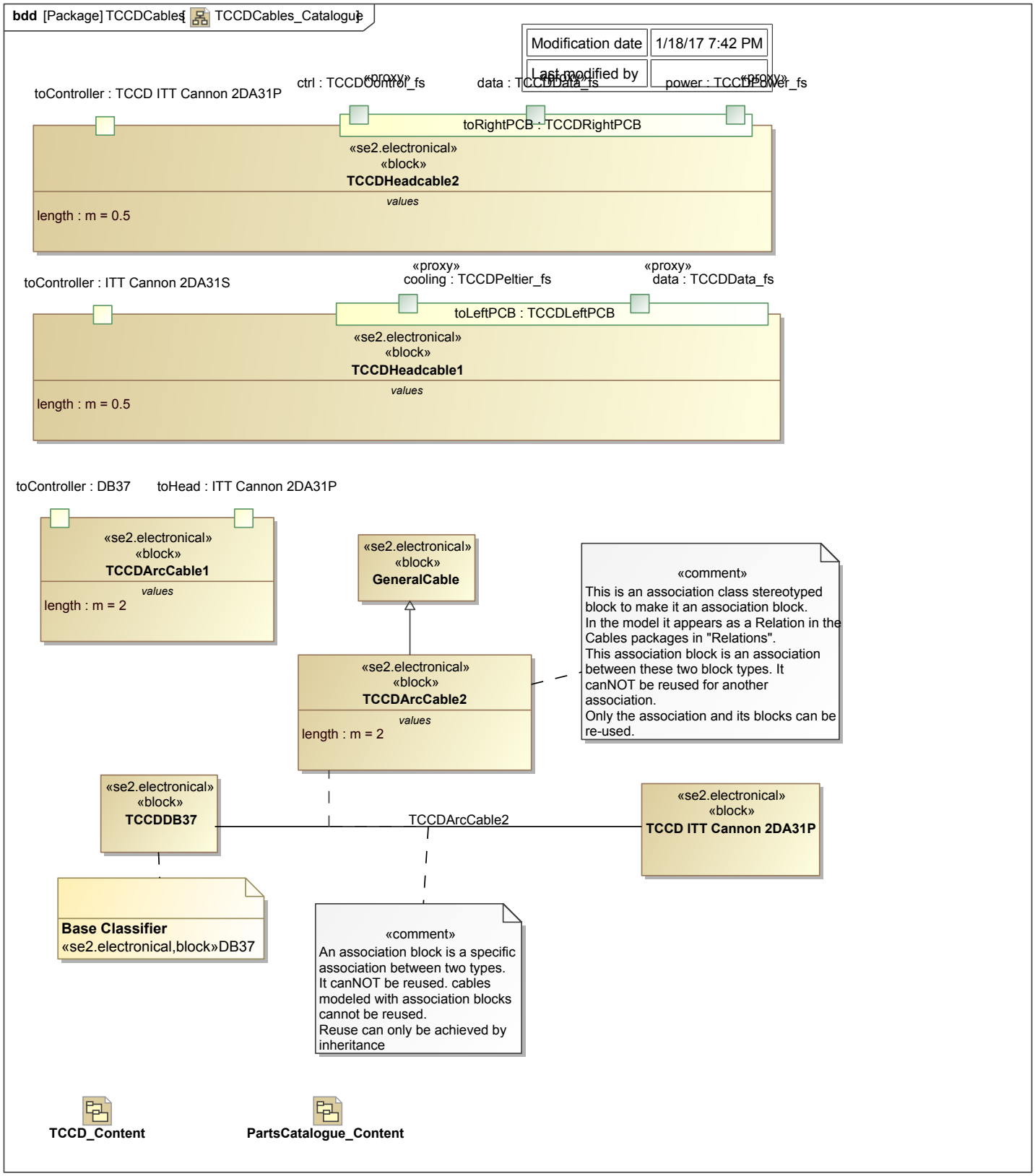


Figure 19. TCCDCables_Catalogue

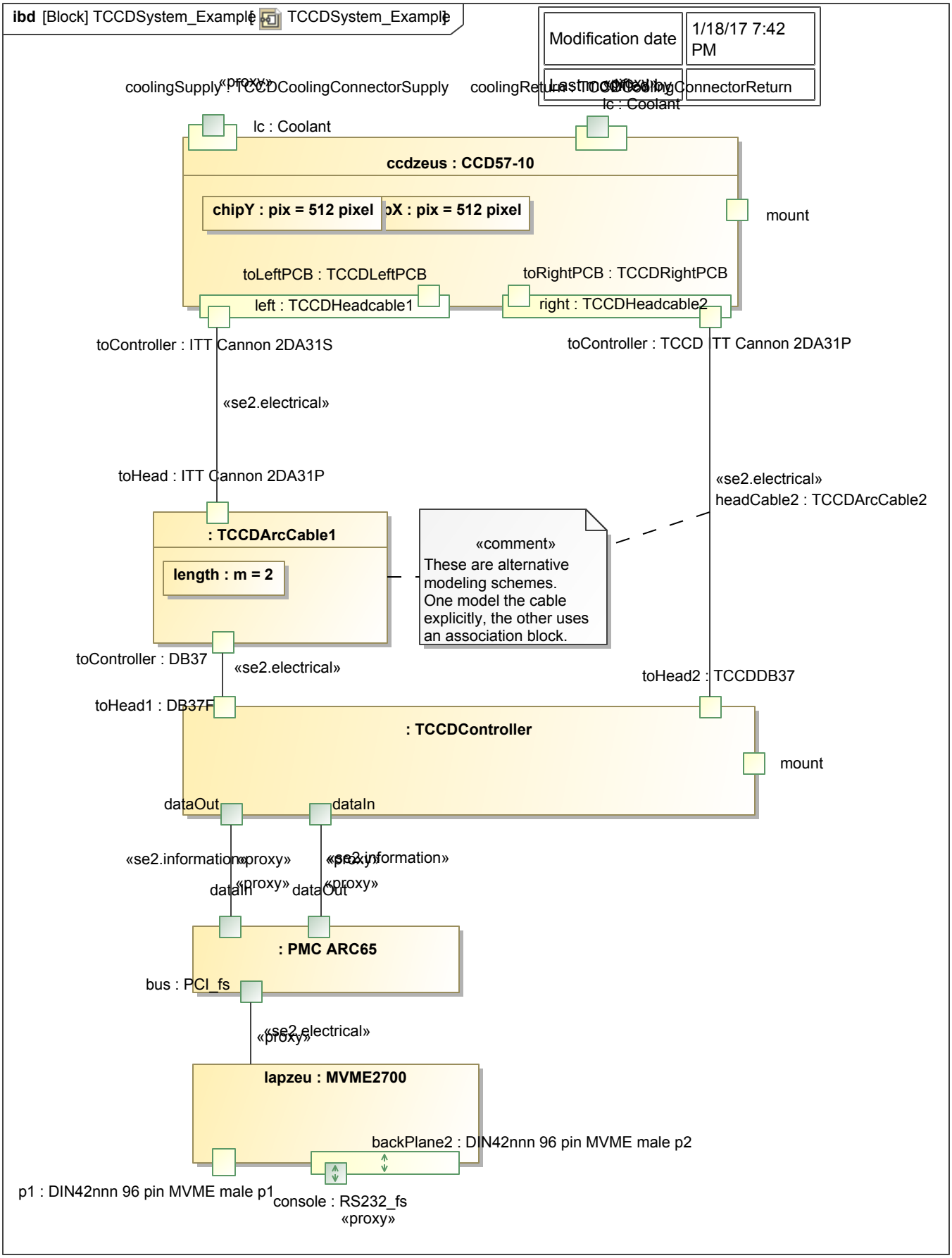


Figure 20. TCCDSystem_Example

3.5.1 Shared parts

Create a package, stereotyped <> for a standard parts catalogue (like motors, CPUs, etc.) to have them separate from the real project model. Add attributes (as value properties) to further specify a part. • references to standards, gender for sockets (male, female - with enum type), supplier id become value properties and are not defined through inheritance • add an attribute (as a placeholder) for a rule on how to connect InterfaceTypes. e.g. only male and female of same type can be connected. This could be interpreted and verified by an engine. In subtypes you can override properties e.g. we want to have the gender of a connector or a standard as value properties but we need different or extended standards in sub-types. In addition we need different gender of the same connector in IBDs (when taking something from a catalogue). • use "redefine property" to override property default values by creating a property in the sub-type of the same name as the super-type and setting its property redefinesProperty. This is used in BDDs to create application specific types. • use context-specific values to set values for certain attributes in IBDs.

3.6 Modeling Physical Aspects

3.6.1 Modeling physical distance between parts

Actuators and sensors might be geographically significantly apart which impacts the rest of the design. For example, the choice of network technology is driven by the physical distance. Those locations usually come from some CAD model (the mechanical design), but they constrain the choice of technology to connect actuator and sensor, e.g. choosing GigaBit Ethernet over a serial line. For example a <> connector shall have a length property.

3.6.2 Model of a magnetic field which exists between two physical entities

Since the magnetic field will have some properties, the simplest is to model the field itself.

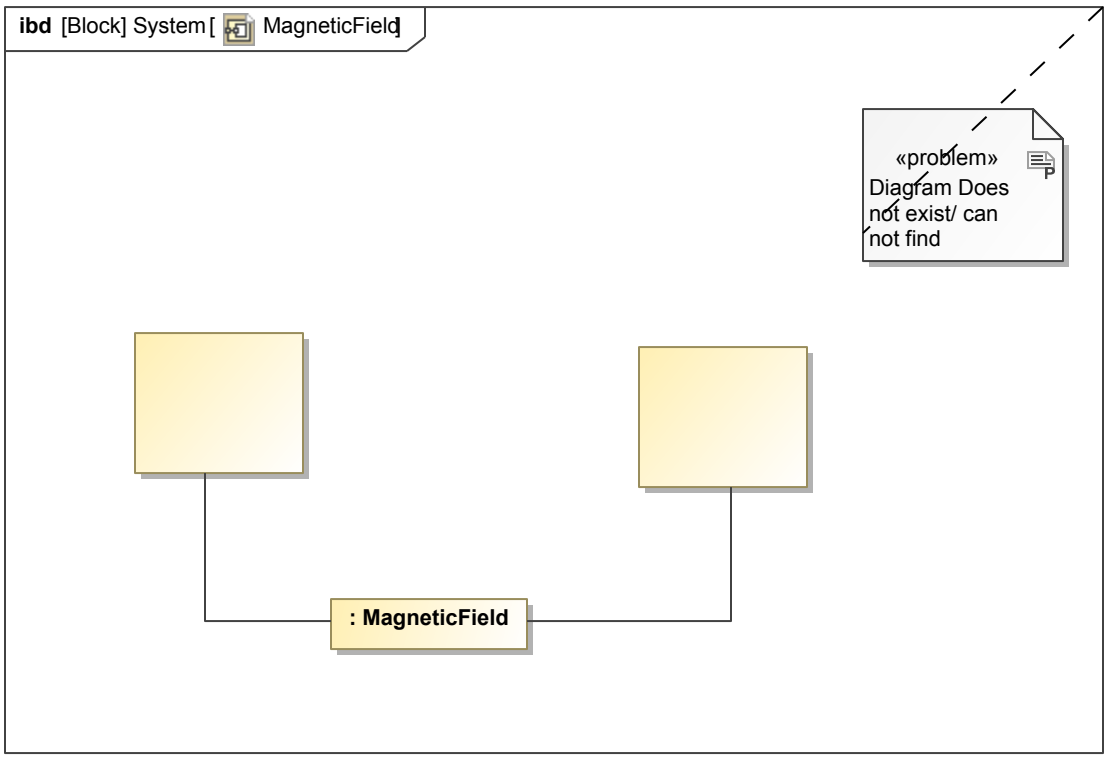


Figure 21. MagneticField

4 Dynamic Constraint Pattern

4.1 Intent

Systems engineers model the state-dependent behavior of components through SysML state machines and activity models. The Dynamic Constraint Pattern describes a method to define the component specific behavior of a component through the dynamic usage of constraints.

4.2 Motivation

The motivation behind the application of the Dynamic Constraint Pattern is to redefine the behavior of a component's behavior to satisfy a system requirement. Previously, there was no obvious solution for systems engineers on how to apply a constraint to a SysML state. In this recommendation, it is suggested that a constraint can be applied to the behavior of a component through the specification of entry behavior. If components in the system need to meet an expectation for analysis, domain specific constraints can be applied dynamically to a component's state through SysML activities and actions.

The Dynamic Constraint Pattern helps to analyze a system model by allowing the user to dynamically apply constraints to the behavioral states of a component. Through the application of the constraints, the modeler can verify whether a value property of a state meets a constraint.

The pattern can be applied in the context where a system's behavior is defined through state machines. If the components in the hierarchy specialize the owning block of the state machine, they will inherit the state machine and can specialize the behavior according to a constraint through entry behavior.

A modeler can apply the Dynamic Constraint Pattern to a block that owns a state machine by creating an activity for the entry behavior of a state, and applying a particular constraint through the action "createObject".

4.3 Concept

The SysML structure of the Dynamic Constraint pattern consists of the relationship between the system and constraint components. In the behavior of the system components, the constraints are applied to constrain value properties of a state.

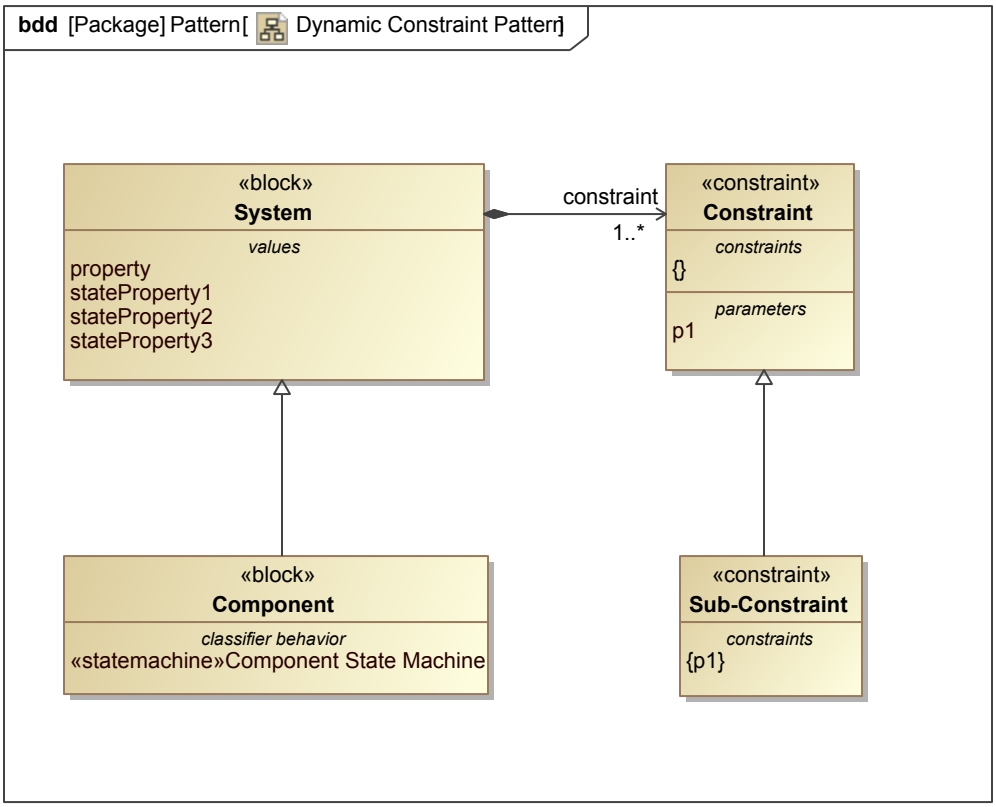


Figure 22. Dynamic Constraint Pattern

The structure of the pattern consists of a system (System block) which owns a value property (property) that will be set equal to the state specific properties (stateProperty1, stateProperty2, stateProperty3) depending on the behavior. These properties are inherited by the Component block through a generalization relationship. The relationship between the system and the constraints that are to be applied are through a composition relationship. The structure of the constraint hierarchy consists of an unspecified constraint block (Constraint). The constraint expression of the block must remain empty. However, the parameters that are to be inherited by sub constraints must be specified here. In the constraint expression of the Sub-Constraint block, the parameter will be applied in the constraint. The purpose of the constraint blocks is to serve as a method for a modeler to apply a requirement on a state of component.

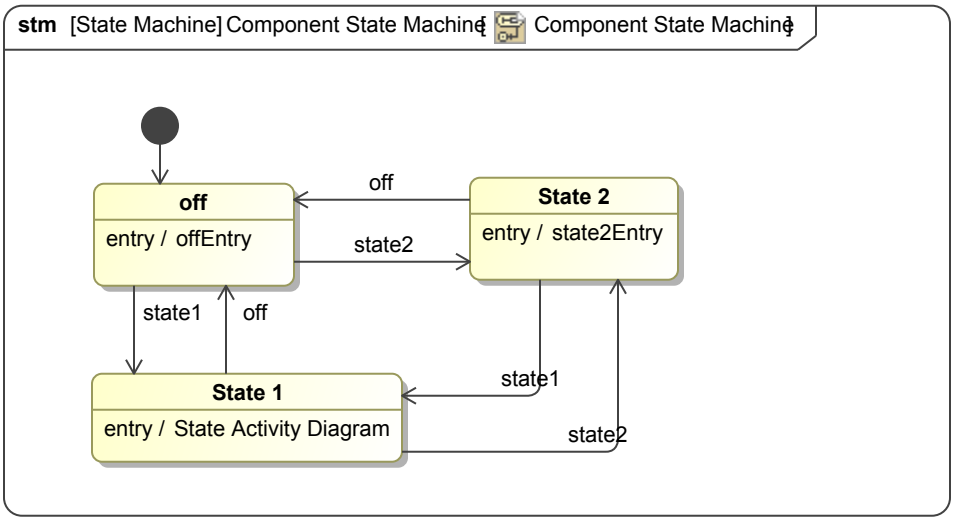


Figure 23. Component State Machine

The state machine is owned by the Component, and is a reusable definition of some state dependent behavior. The state machine represents a change in how the block responds to events and the behaviors it performs. Each state in the state machine has an entry behavior, and is set to the type activity. The value of property becomes equal to the value of the operating state (stateProperty1, stateProperty2) depending on the signal event (state1, state2, off) and the constraint (requirement) that is applied the state.

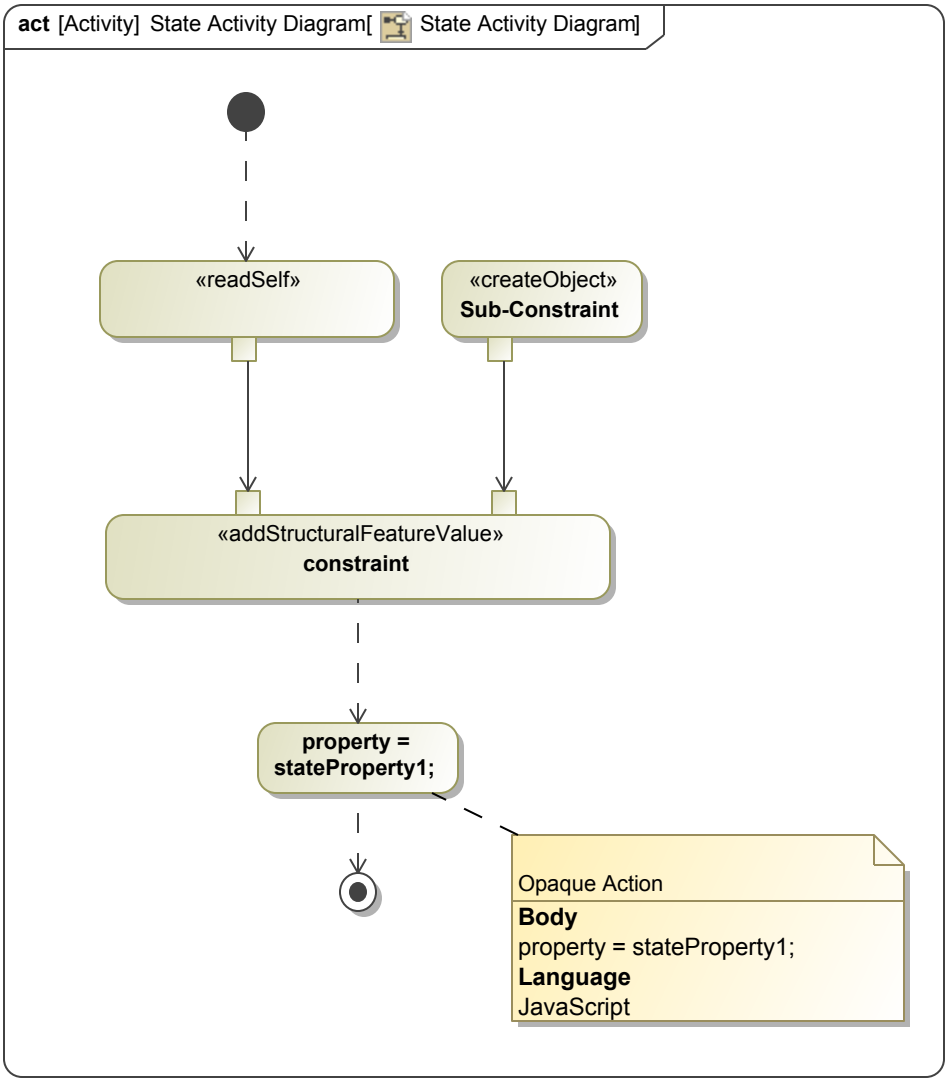


Figure 24. State Activity Diagram

The behavior type of each state in the Component's state machine is specified in an activity diagram. In the activity diagram, the following "Any Actions..." must be present - readSelf, createObject, addStructuralFeatureValue, and an opaque action where the property is set equal to a state property (i.e. stateProperty1).

When the diagram is initialized the action, readSelf, will read behavior and specification of the owning element of the diagram (State 1). The createObject action (Sub-Constraint) will cause an object to be created based on the action's classifier (Sub-Constraint). Note, the classifier is the constraint block that we created earlier. The addStructuralFeatureValue action "constrain" is used to create a structural value that consists of the owning state and the the constraint block Sub-Constraint. In this action, "constrain" is only a name and there are no other changes to the specification properties of the action. Lastly, in the body and language specification property of the opaque action, the value property "property" is set equal to the state property stateProperty1. Note, Javascript was used.

A listing of the model elements used in the pattern and their roles in the Dynamic Constraint pattern.

Table 1. <>

Model Element	
Sub-Constraint	The Sub-Constraint block specializes the Constraint block, and will inherit it's parameters and constraints. In the Sub-Constraint block a systems engineer can apply a requirement to the system as a constraint specification. The Sub-Constraint block will become a classifier of the createObject action in the activity diagram of a component's state. The systems engineer can apply an infinite amount of sub-constraints to the states of a component (see Modeling the Structure of Multiple Constraints).

Model Element	
Component	The Component block represents a general component that specializes a system or other component whose properties are to be inherited. The Component owns a state machine where the state-dependent behavior throughout the block's lifecycle is defined.
System	The System block represents a top level hierarchical component where the value properties are defined. The value properties include state specific properties (stateProperty1, stateProperty2) and the general property. These properties are inherited by any component that specialize the System block. Additionally, the System block owns at least one constraint block (Constraint).

4.4 Consequences

There are several trade-offs to consider when applying the Dynamic Constraint Pattern to the modeler's system. These trade-offs are primarily in respect to applying the pattern to a system that is specialized by a multiple or a hierarchy of components.

Conflicts that occur while applying or using the Dynamic Constraint Pattern.

Usage of the Pattern	Explanation of Conflict
Displaying the relationship between a constraint and a state.	The modeler is unable to create a relation between a constraint and a state unless using a dependency. Therefore the modeler will not be able to display and visualize the relationships. By locating the constraints in the scope of the component block the associations between the constraint and the block can be viewed. However, the association between the constraint and the state in which it is applied can't be seen.

4.4.1 State Machine Redefinition

Action	Consequence
The state machine is owned by the Resource aspect and is inherited by the components of the system	Every component of the system specializes the roll-up aspect which owns a state machine where the behavior of the system is specified. These components will then inherit this state machine. This can be a concern if a component doesn't share the same states that are specified in the state machine. A modeler needs to create a new state machine for this component which will modify the classifier behavior. See State Machine Redefinition as example how to modify the classifier behavior in this case.

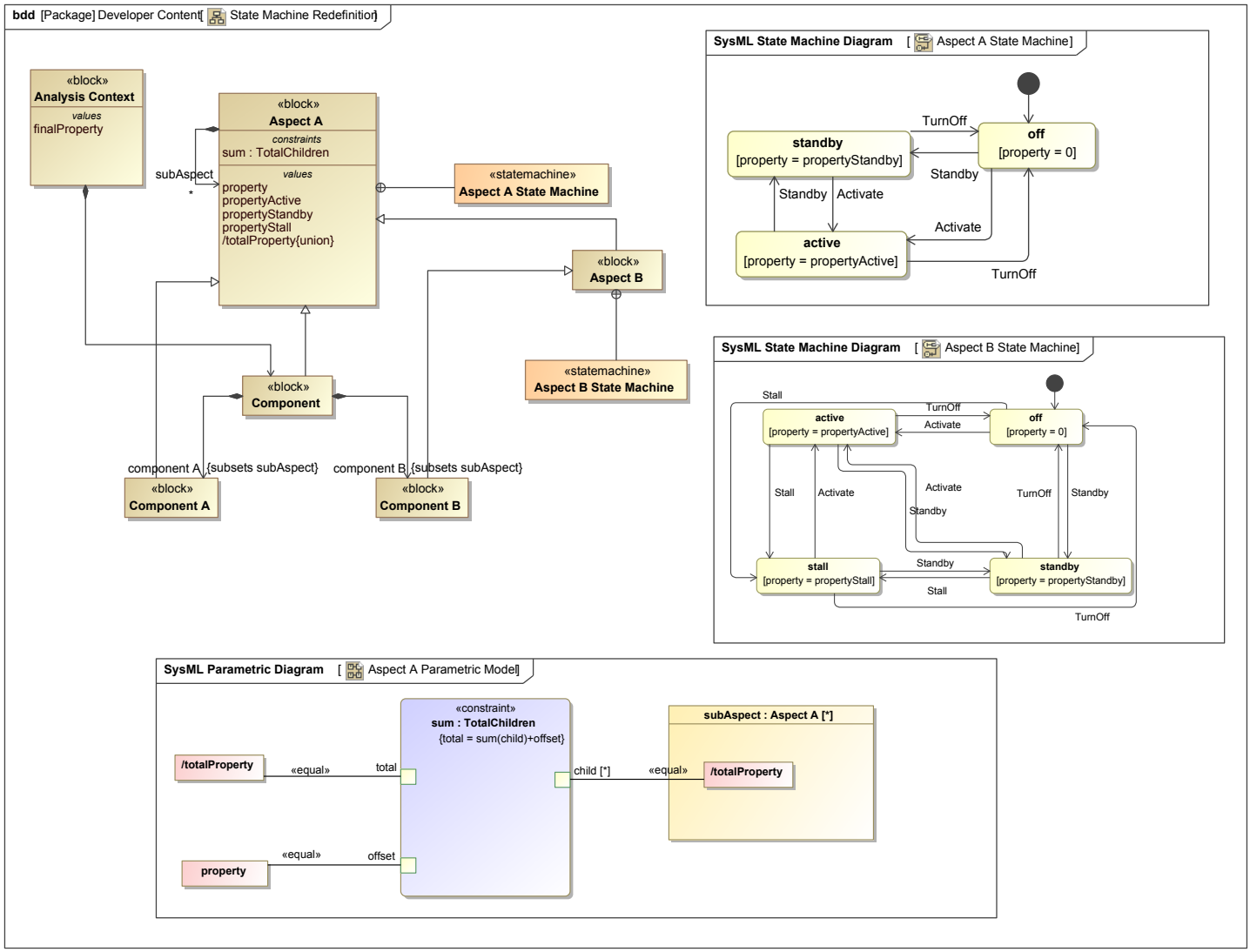


Figure 25. State Machine Redefinition

4.4.2 Modeling the Structure of Multiple Constraints

Action	Consequence
Modeling multiple sub-constraints by further specializing a general constraint block.	If there are additional constraints that need to be applied to specify the behavior of a component, the modeler can choose to specialize the general constraint block. This is only required if the new constraint relies on a parameter that wasn't previously specified. The modeler would add the new parameter to the preexisting general constraint block (Constraint), add another composition relationship from the System block the Constraint block, and lastly add the new sub-constraint block (Sub-Constraint B). SeeMethod A Modeling Multiple Constraints.
Modeling multiple sub-constraints by creating additional constraints of the System block.	If there are additional constraints that need to be applied to specify the behavior of a component, the modeler can choose to create additional general constraint blocks that are specialized by state specific constraints. This is only required if the new constraint relies on a parameter that wasn't previously specified. The modeler would create another constraint part property of the System block (Constraint B) where there is an empty constraint and the parameters of the inherited sub-constraint are owned. The sub-constraint (Sub-Constraint B) specializes the general constraint block (Constraint B) and applies the parameter in a constraint expression that is to be applied in a state of the system's component (Component). See Method B Modeling Multiple Constraints.

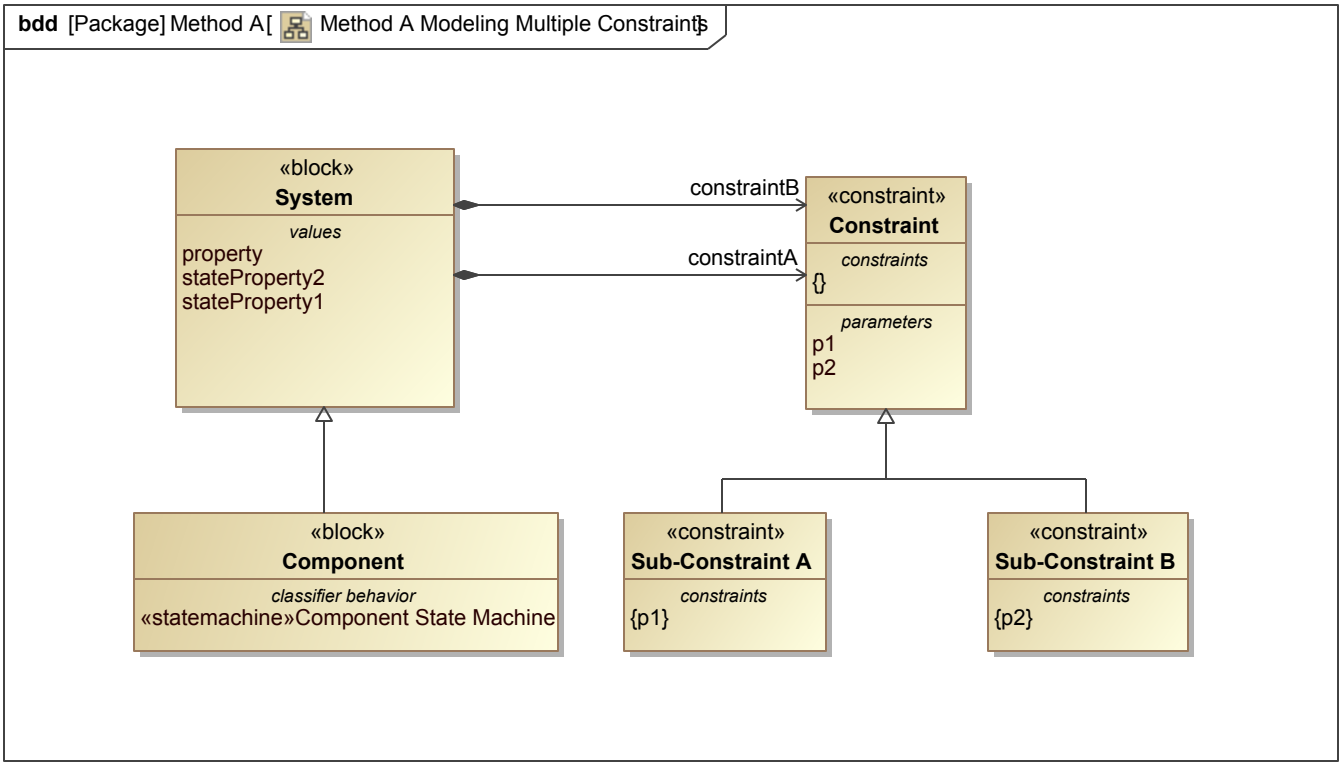


Figure 26. Method A Modeling Multiple Constraints

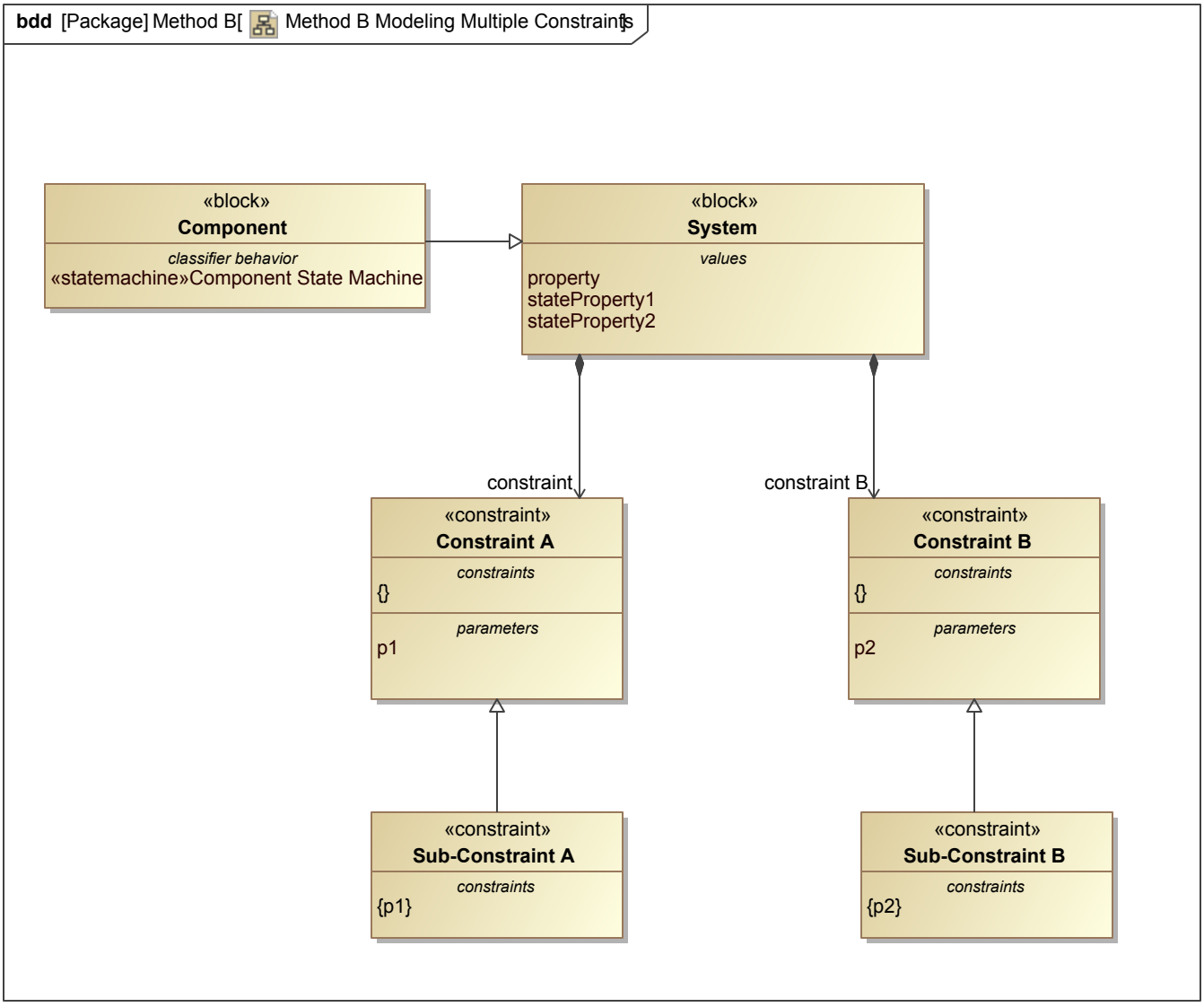


Figure 27. Method B Modeling Multiple Constraints

4.5 Implementation

In this sample model, the Dynamic Constraint Pattern will be applied to perform a data roll-up analysis of a computer. The purpose of the model is to demonstrate the structure and relationship between a component (Computer), and state specific constraints. The state specific constraints will be applied to the behavior of the computer through entry activities. To find further information on how to apply the Dynamic Constraint Pattern for analysis see Requirement Verification.

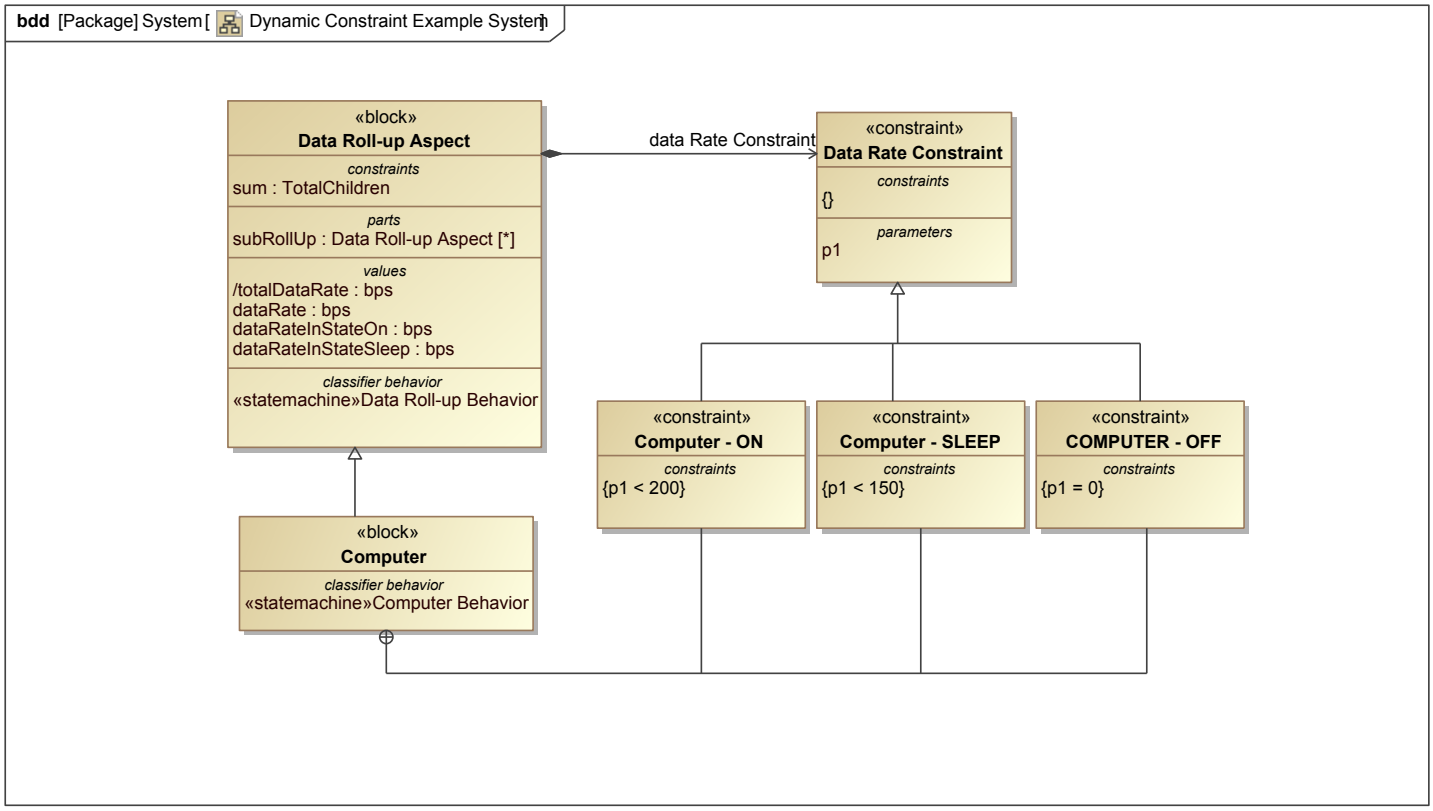


Figure 28. Dynamic Constraint Example System

The system for this example consists of the Data Roll-up Aspect block where the recursive roll-up pattern structure is specialized by a Computer block. The Computer will inherit the value properties, parts, constraints, parametric model, and state machine of the Data Roll-up Aspect. The behavior of the computer is specified in a state machine where the constraints of are applied. The aspect block owns a constraint (Data Rate Constraint) that has an unspecified constraint and a parameter (p1). The parameter is used in the constraints of the sub-constraints (Computer - ON, Computer - SLEEP, Computer - OFF). Each of the sub-constraints specialize the general Data Rate Constraint block, and their names refer to the state they specialize.

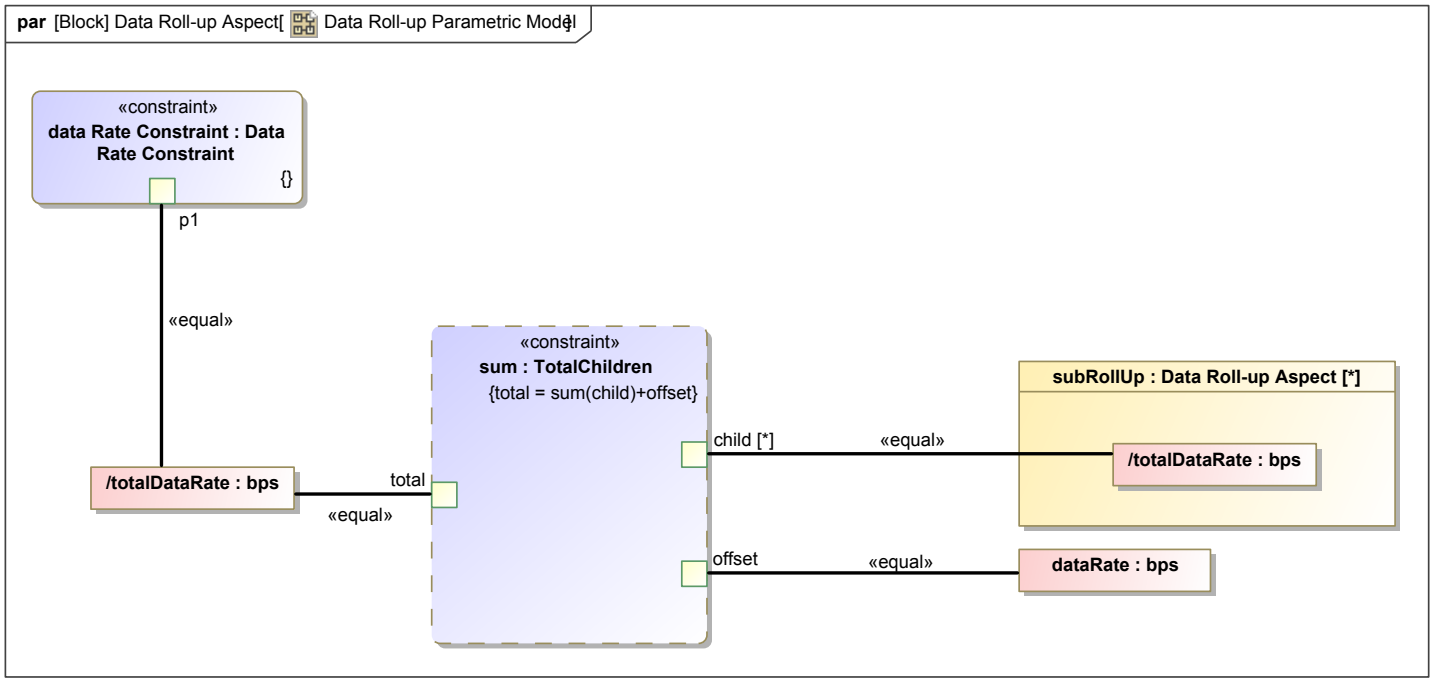


Figure 29. Data Roll-up Parametric Model

The parametric model of the Data Roll-up Aspect block constrains the p1 parameter of the Data Rate Constraint to the totalDataRate value property of the subRollup part property. By constraining these properties the Component block will inherit this internal structure.

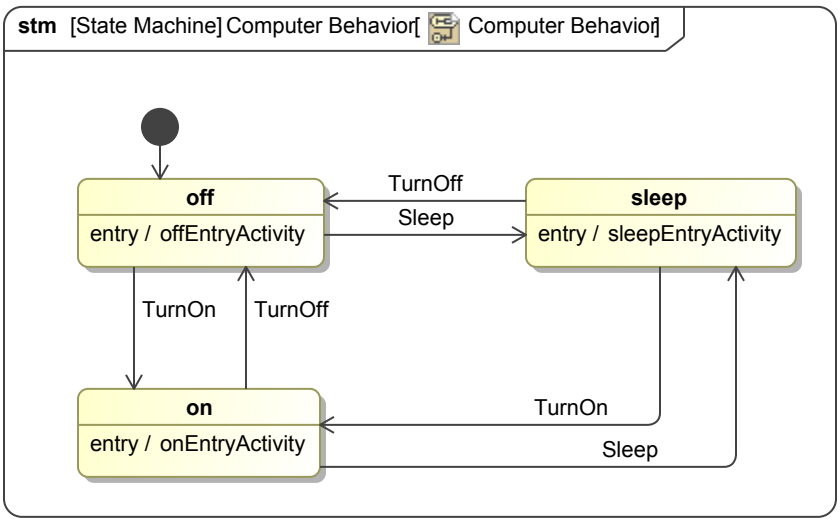


Figure 30. Computer Behavior

A state machine is created for the Computer block to specify the behavior of the component during it's life-cycle. The states (off, on, sleep) have an entry behavior of type Activity. If at least one state in the system needs to apply a constraint, then all states need to have an entry behavior and type as well. During simulation, the computer navigates between the the operational states through the signal events (TurnOn, TurnOff, Sleep).

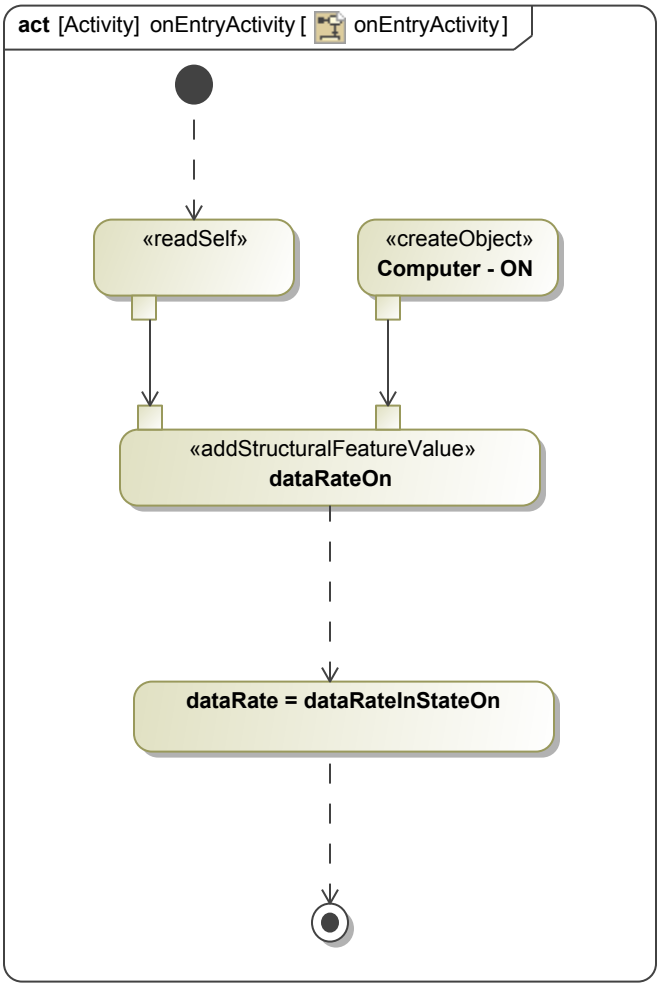


Figure 31. onEntryActivity

In this example, the entry behavior of the on state is defined in an activity diagram, and is used to apply a constraint to the behavior of the state. In the activity diagram the actions readSelf, createObject, addStructuralFeatureValue, and an opaque action is applied. When the diagram is initialized the action, readSelf, will read behavior and specification of the owning element of the diagram (the on state). The createObject action (Computer - ON) will cause an object to be created based on the action's classifier (Computer - ON). Note, the classifier is the constraint block that we created earlier. The addStructuralFeatureValue action (dataRateOn) is used to create a structural value that consists of the owning state and the the constraint block Computer-ON. Note, dataRateOn is just the name of the action and there is nothing do be done in the specifications of the action. Lastly, in the body and language specification property of the opaque action the dataRate is set equal to the state property (dataRateInStateOn). Note, Javascript was used.

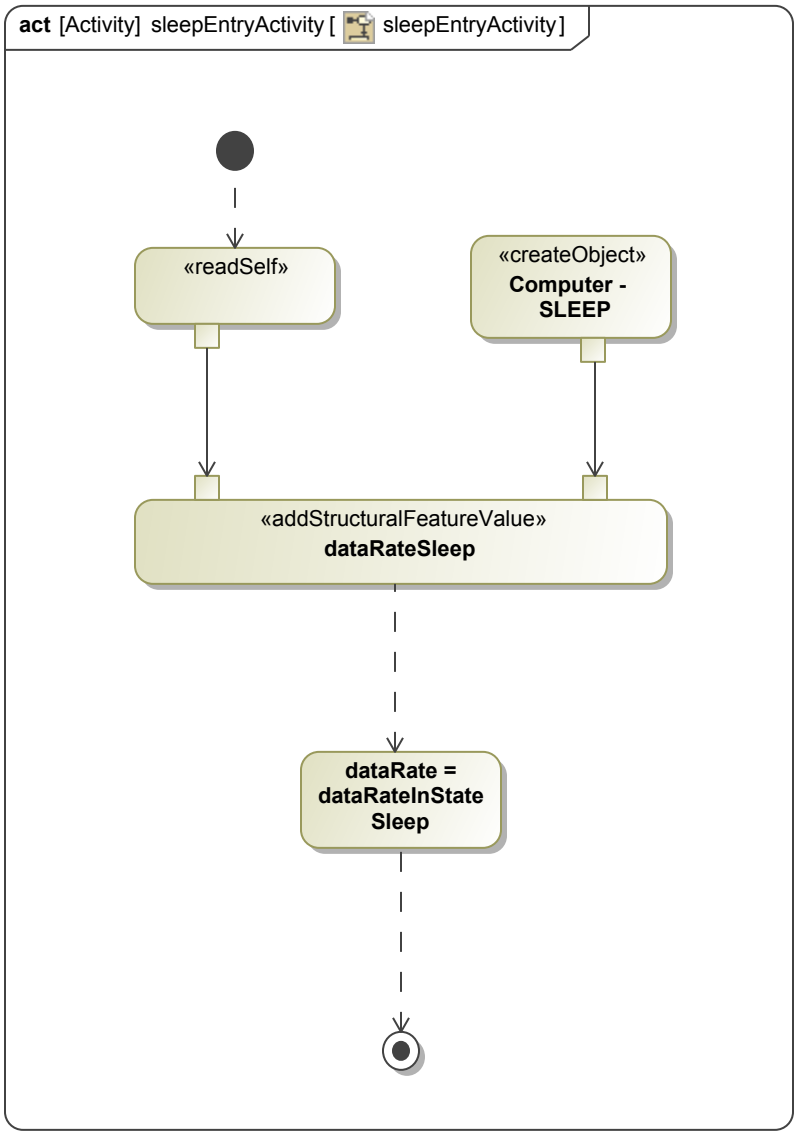


Figure 32. sleepEntryActivity

The same actions as above were taken to specify the entry behavior of the sleep state of the computer. When the diagram is initialized the action, readSelf, will read behavior and specification of the owning element of the diagram (the sleep state). The createObject action (Computer - SLEEP) will cause an object to be created based on the action's classifier (Computer - SLEEP). Note, the classifier is the constraint block that we created earlier. The addStructuralFeatureValue action dataRateSleep is used to create a structural value that consists of the owning state and the the constraint block Computer-SLEEP. Note, dataRateSleep is just the name of the action and there is nothing do be done in the specifications of the action. Lastly, in the body and language specification property of the opaque action the dataRate is set equal to the state property (dataRateInStateSleep). Note, Javascript was used.

4.6 Known Uses

4.7 Related Patterns

List of other patterns that are related to the Dynamic Constraint pattern, and the differences and similarities between them.

Pattern Name	**Similarities**	**Differences**
Dynamic Roll-up Pattern	The Dynamic Roll-up Pattern and the Dynamic Constraint Pattern can both redefine the behavior of a system through state machines.	The Dynamic Roll-up Pattern documents how to perform roll-up calculations of components to find a resource value. The Dynamic Constraint Pattern documents how to apply requirements to a system through constraints and behavior models.
Requirement Verification		

5 Model Organization/Abstraction Pattern

5.1 Intent

5.2 Motivation

5.3 Concept

5.4 Simulation

5.5 Consequences

Action	Consequence

5.6 Implementation

5.7 Known Uses

5.8 Analysis

5.9 Related Patterns

6 Abstraction Layer Traceability Pattern -DRAFT-

6.1 Intent

The intent of the Abstraction Layer Traceability Pattern -DRAFT- is to provide system engineers a replicable design pattern detailing the levels of abstraction and the navigation between them.

The following will be included:
1) Use Case elaboration describing black-box specifications with activities and elaborating to conceptual and realization.
2) How components and behavior are related among layers of abstraction.
3) Relate Use Cases to component behavior (how action becomes do behavior)

6.2 Motivation

Black Box Specification

- Defining the behavior / functionality of the system.
- The black box model defines the interfaces of the system to the outside world without exposing the internal design details (described later in conceptual and realization models).
- These interfaces have been discovered as a result of specifying several system engineering use cases involving the system

Conceptual

- Refinement of the conceptual architecture is performed for each newly identified behavior / function of the system.
- Conceptual activities or actions, and corresponding conceptual components owning and executing these actions are then created.
- Once the internal behavior has been modeled, conceptual components that were created as part of modeling the internal behavior are aggregated whenever sensible to either existing or new conceptual components, with the aim being to build a logical decomposition
- The conceptual model specifies the internal details of the system at a functional level and without commitment to particular technologies (which are rather specified by the realization model). The conceptual model identifies the main functional components of the system, specify their interfaces, and connect such interfaces to each other and to the black box interfaces of the system.

Realization

- Rather than conceptual components, *realization* components (that is, COTS tools, or specific implementations) are used
- Mappings from realization to conceptual components are added. Mappings between realization and conceptual components are added using the **UML Realization** relationship (a *realization* component *realizes* a *conceptual* component)

Black Box

Based on the use case, a black box specification is defined. The process for defining the black box specification is to aggregate identified stakeholders, aggregate identified external systems, add behavior as operations of the black box specification, define interface definitions of the black box specification. Define required operations, provided operations, and flow properties to create an interface of the black box specification.

Conceptual

Conceptual activities or actions, and corresponding conceptual components owning and executing these actions are then created. This is done in a similar fashion to modeling an operational scenario: **Swimlanes** are used to represent conceptual components (which are modeled as **SysML Blocks**) that are owned parts of the OpenCAE Conceptual architecture SysML
block. **Actions**, **CallBehaviorActions** and other SysML behavioral elements are used in the process of detailing the internal behavior. Object flows and pins are used to indicate any data exchange. Once the internal behavior has been modeled, pre- and post-conditions and other invariants are added to the activity (or just to the diagram using SysML Comments).

Once the internal behavior has been modeled, conceptual components that were created as part of modeling the internal behavior are aggregated whenever sensible to either existing or new conceptual components, with the aim being to build a logical decomposition (however, note that all conceptual components must be children (at some depth) of the SysML Block representing the OpenCAE

Conceptual architecture). Thereafter, **SysML InterfaceBlocks**are created (or existing ones refined) to capture the provided and required operations of, as well as the item flows over the boundary / interface of the various conceptual components. **SysML Proxy Ports** are added to the conceptual components, typed by the corresponding interface blocks. Interfaces are defined based on the actions being performed by the conceptual components, and the items flowing across the boundary. Data and information flow between the various conceptual components is shown by creating (or refining) an internal block diagram of the

Realization

Refinement of the realization architecture is performed almost identically to that of the Conceptual architecture. The primary differences are the following:

- Rather than conceptual components, *realization* components (that is, COTS tools, or specific implementations) are used
- Mappings from realization to conceptual components are added using the **UML Realization** relationship (a *realization* component *realizes* a *conceptual* component).

6.3 Concept

An abstraction layer is a generalization of a conceptual model or algorithm, away from any specific implementation. These generalizations arise from broad similarities that are best encapsulated by models that express similarities present in various specific implementations. MODIFY

Black Box Specification

- Components
- Behavior (State Machines)

Characterization Version

- Whole design space
- Component may have multiple possible behaviors
- Need to have system breakdown of all components
- Define the interfaces between blocks
- Classifier behavior of the blocks (<<characterize>>)
- Different types of characterizations that are consoladated
- Decision: Component chooses which characterization to apply. The one you will analyze.
- To execute the specialization of the system then this is the way.
- Refer to Ontology and Modeling Patterns for State-Based Behavior Representation. Point design? Pick one aspect of the model. Particular design in the design space.

Design Version

- Select a subset of a specific design
- Concrete conceptual
- Different conceptual designs based on the black box
- Depends on which subset is chosen

Realization Analysis

- Copy or create block specific types (BST) then apply to the realization design to perform analysis

Multi-Scenario Analysis

- Create BST of the design
- Apply pattern
- Create instances and modify values of value properties according to the scenario
- Execute the specified scenarios

Roll-up Analysis

- Apply Roll-up pattern(s) to system to perform analysis

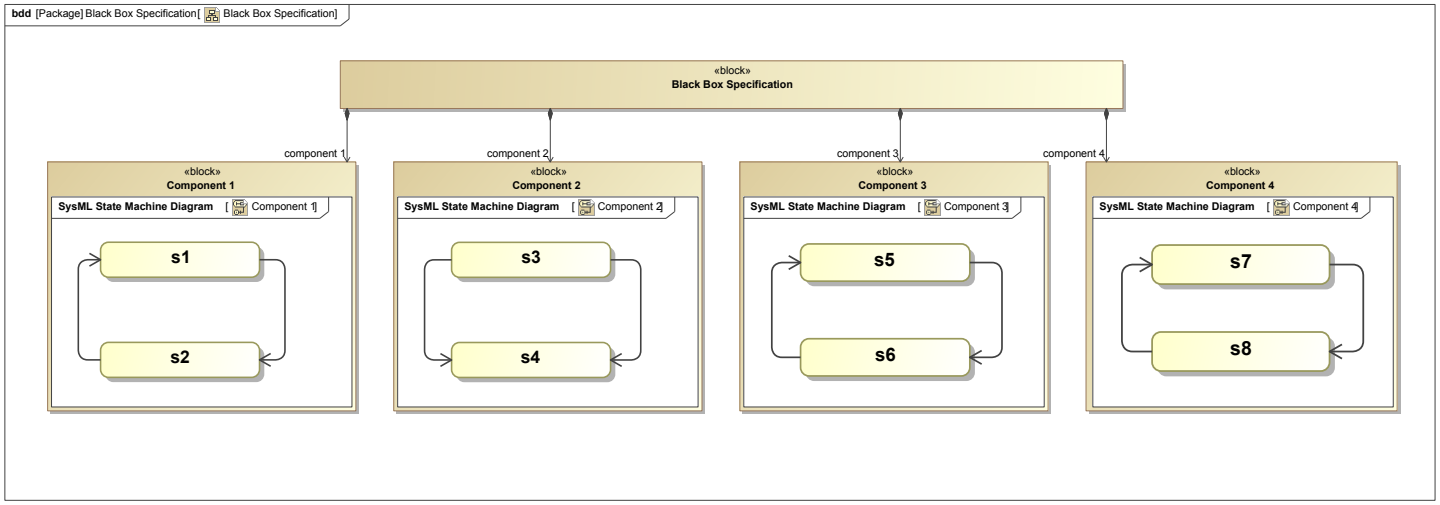


Figure 33. Black Box Specification

6.4 Consequences

There are several trade-offs to consider when applying the Abstraction Layer Traceability Pattern -DRAFT- to the modeler's system.

Action	Consequence

6.5 Implementation

6.6 Known Uses

6.7 Analysis

6.8 Related Patterns

7 Deployment Pattern

The Deployment Pattern is used to describe the relationship between Systems' black box specifications and actual deployed environments.

7.1 Intent

The intent of the Deployment Pattern is to provide system engineers a replicable design pattern detailing the realization of applications/services environment in relation to the physical deployment.

7.2 Applicability

After describing a system in the various levels of abstraction and their corresponding behaviors (see Section 6), the next step is to identify and reveal the actual deployments of said system. The idea is that the specification will define the system and then each deployment should be a specific implementation of it, depending on the intent of the deployment such as automatic integration testing or user activitity testing.

Although this pattern has distinct applications to software specifications and their deployments, it can be used for other applications as well. The main benefit of this pattern is that it provides the ability to define multiple specifications for the multiple environments and their corresponding set of implementations.

7.3 Structure

The view describes the basic structure of the Deployment Pattern. The structure is based on the ability to define and apply the architecture of a system in different contexts. First, the system is defined in the abstraction layers, conceptual and realization. This is where the freedom of designing the system comes into play. Once that engineering process has been iterated through and approved, the next step is implementation. This pattern creates a clear relation between what has been designed and what has been delivered. Refer to the following diagrams for how this relationship can be illustrated.

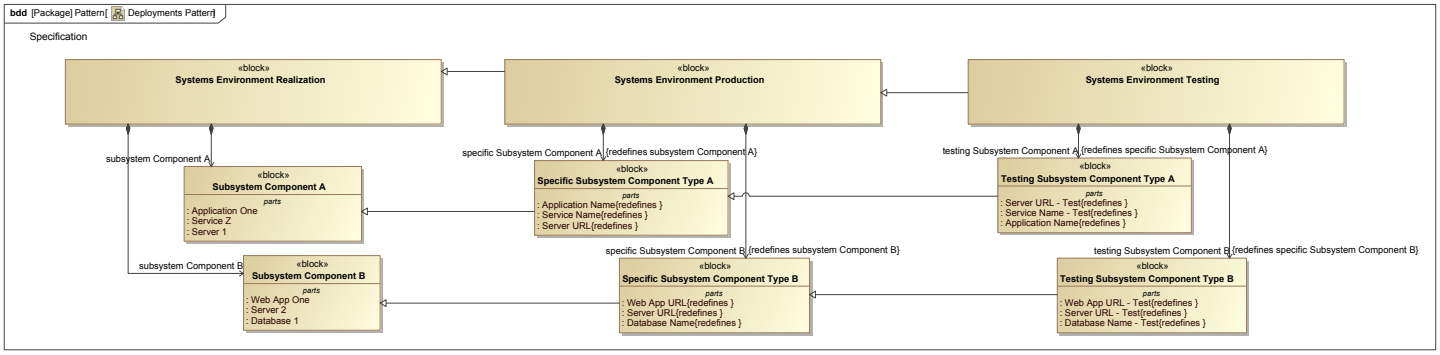


Figure 34. Deployments Pattern

The Block Definition Diagram (BDD) of the Deployments pattern shows the different Blocks that represent a Systems Environment Realization, the Systems Environment (in) Production, and the Systems Environment (in) Testing. The Systems Environment Realization is broken into two subsystem components that also have part properties. Note, the names of these components are arbitrary, however, more can be inferred from the Sample Model later in this document. Once all of the subcomponents were defined, the Production environment was created by specializing the Realization. This not only allows for inheritance, but proper redefining of each part property so that the new environment can reflect what is deployed, which may or may not have everything in the realization. From there, the Testing environment was specialized from the production environment. This is to reflect that the Testing environment should be almost identical to the Production environment for proper testing. The difference however, could be that the testing environment may also have a few changes that will eventually redefine the production environment once those changes have been approved.

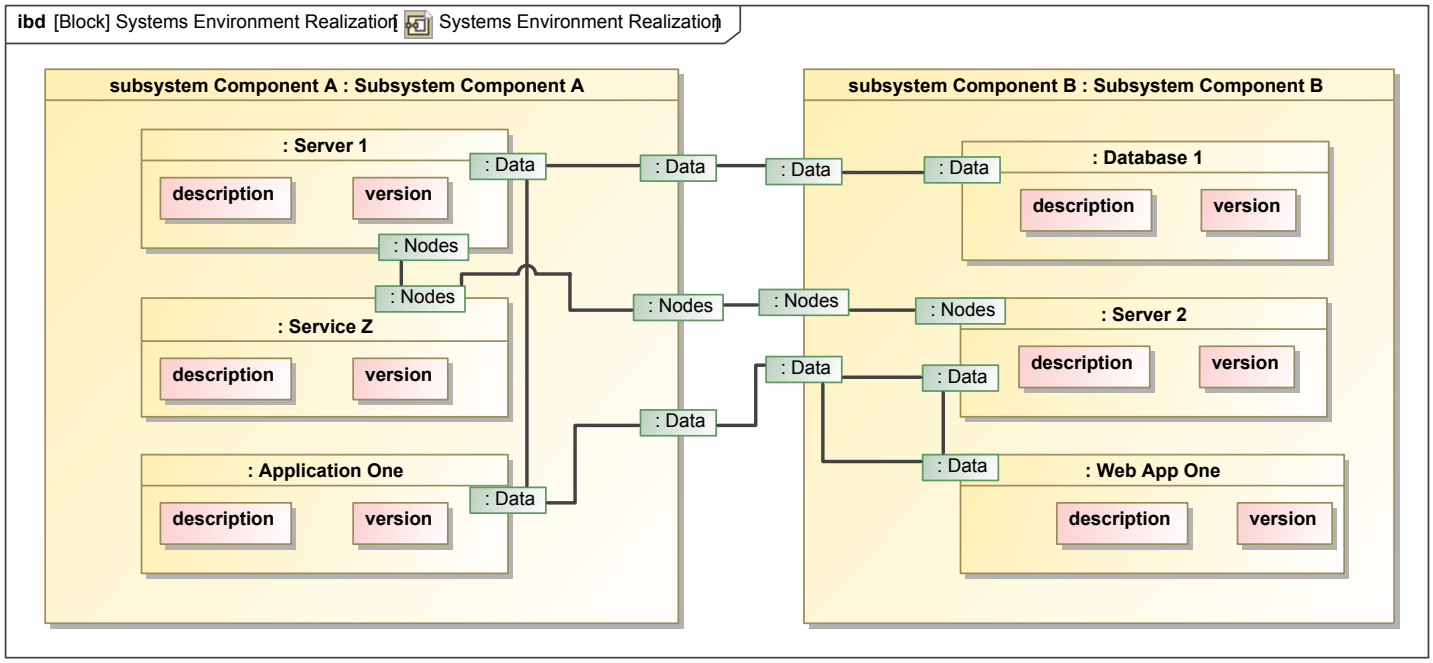


Figure 35. Systems Environment Realization

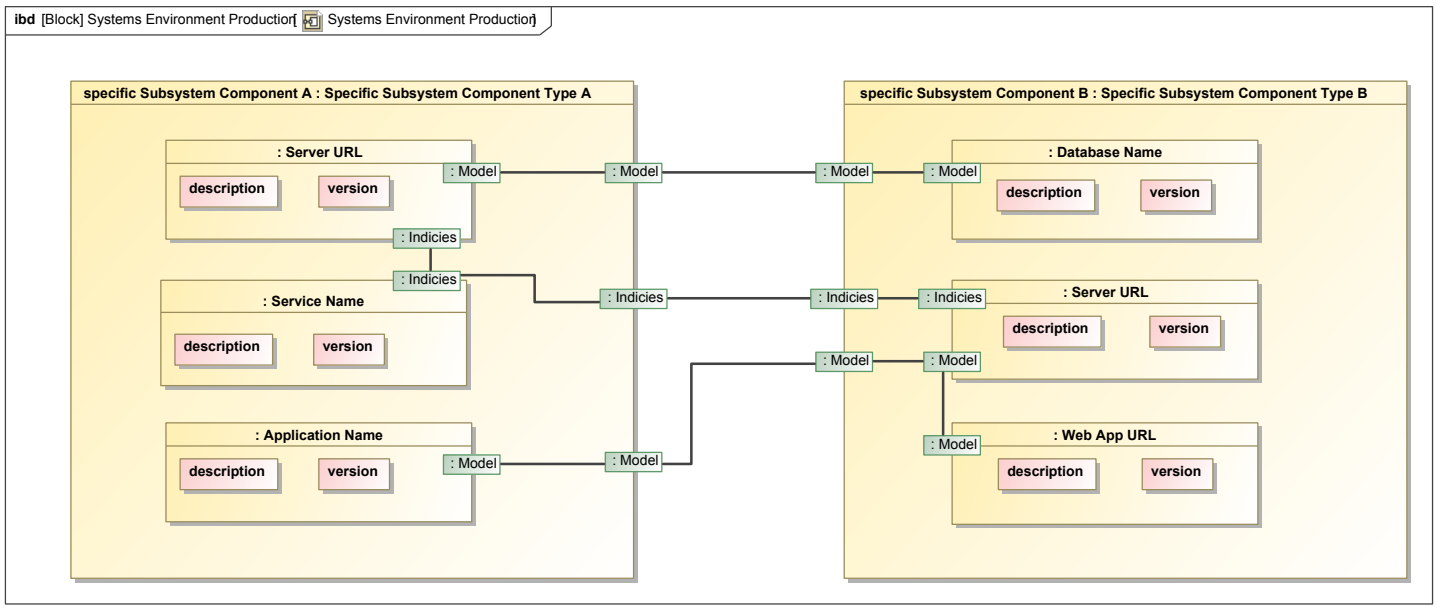


Figure 36. Systems Environment Production

The Systems Environment Production IBD shows almost exactly the same design as the Systems Environment Realization, which is the intent of specializing the Realization element. The design captures all the capabilities and it is distinguished through the deployment block what is exactly implemented. In this case, the Systems Environment Realization shows that there are two connections to the Model port of ":Web App URL", one coming from the Model port of Specific Subsystem Component Type B and one coming from the Model of ":Server URL". In the realization, these connections showed that a model could travel to the Web App either through the server or directly. In the Production deployment, the connection through the latter has been chosen and implemented.

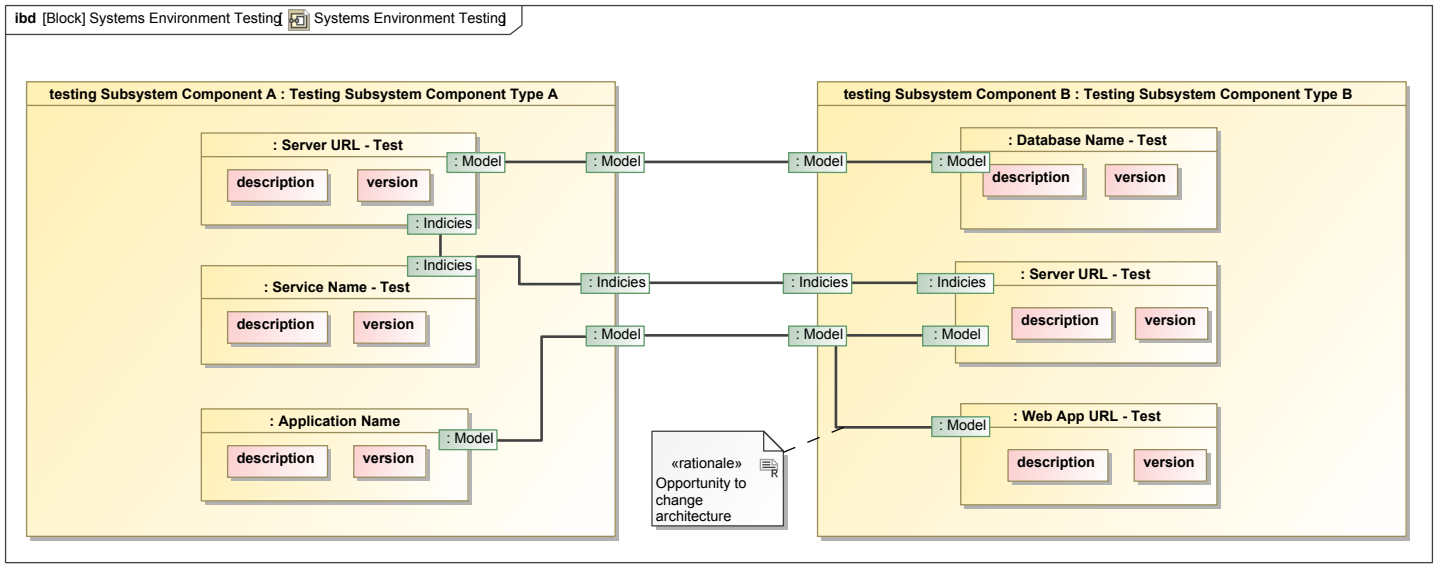


Figure 37. Systems Environment Testing

Alternatively, since the Systems Environment Testing was specialized from Production but is supposed to also represent future changes to the deployment according to the Realization design, the Systems Environment Testing IBD shows that there is only one connection to ":Web App URL - Test", which is through the Testing Subsystem Component Type B interface, typed Model. This change is highlighted in the <rationale> element displayed on the diagram. Again, this IBD is to reflect the appropriate implementation of the design which can differ from deployment to deployment.

7.4 Consequences

Once this Deployment Pattern is in place, updating the system design and propagating those changes throughout the deployments should be the same steps as any other modeling. Depending on the detail of the system and how many deployments there are, it can be somewhat cumbersome to keep all the changes in check. However, following the pattern should make the process smoother. It is definitely beneficial to keep the diagrams up to date for both knowledge of current status and for determining future architectural changes.

7.5 Sample Model

This view describes the sample model for the Deployment Pattern. It demonstrates how a system's specification can be used to define its various deployments, in this case both the Production and User Acceptance Testing (UAT) of a Modeling Environment.

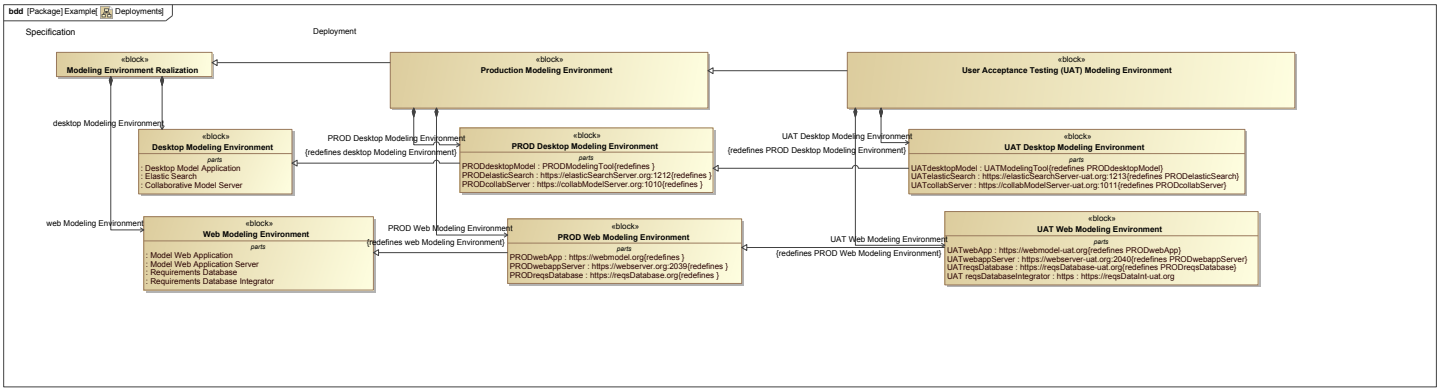


Figure 38. Deployments

As seen in the Deployments diagram, the specification realization has been created and defined to be the base model of the deployments. The realization has all possibilities of parts/ports/and connections for the various deployments. This means that the Production environment was specialized recursively from it and redefined according to the Production attributes. The UAT environment specializes from Production rather than the specification itself because it should only be a variation of what it is actually deployed in the production environment. The UAT environment is mostly used for testing the upcoming releases and so it should

reflect what is currently deployed in production and then whatever changes will happen during the next release. The overall specification should have the attributes for all releases, but Production should only show what is currently live.

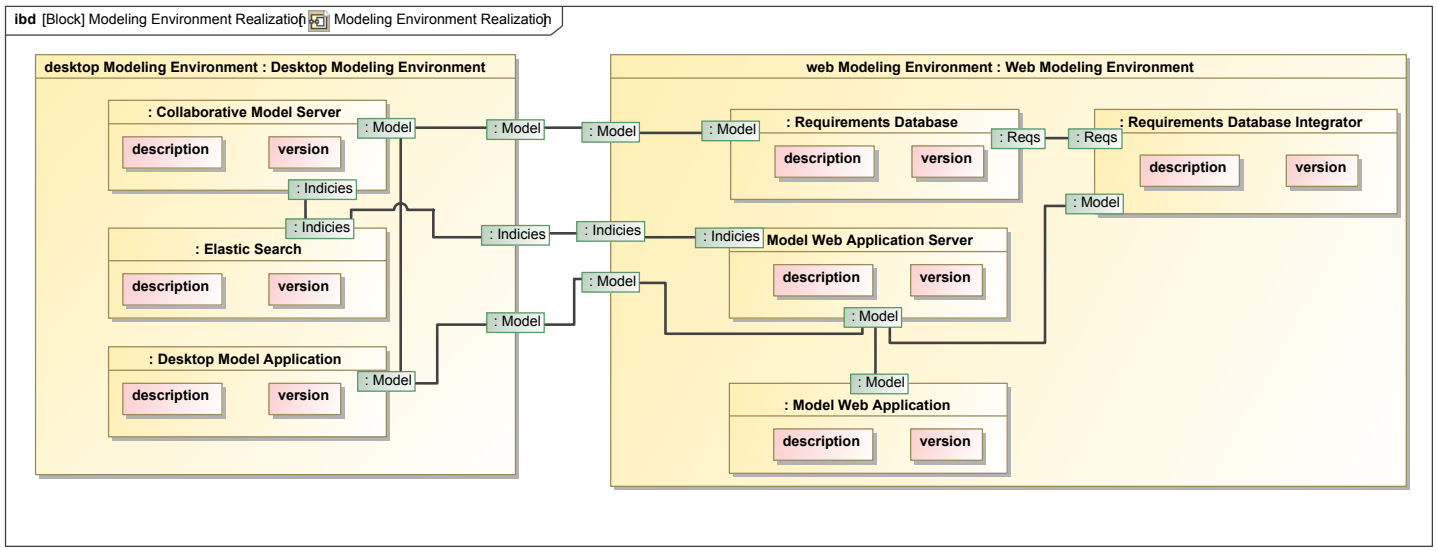


Figure 39. Modeling Environment Realization

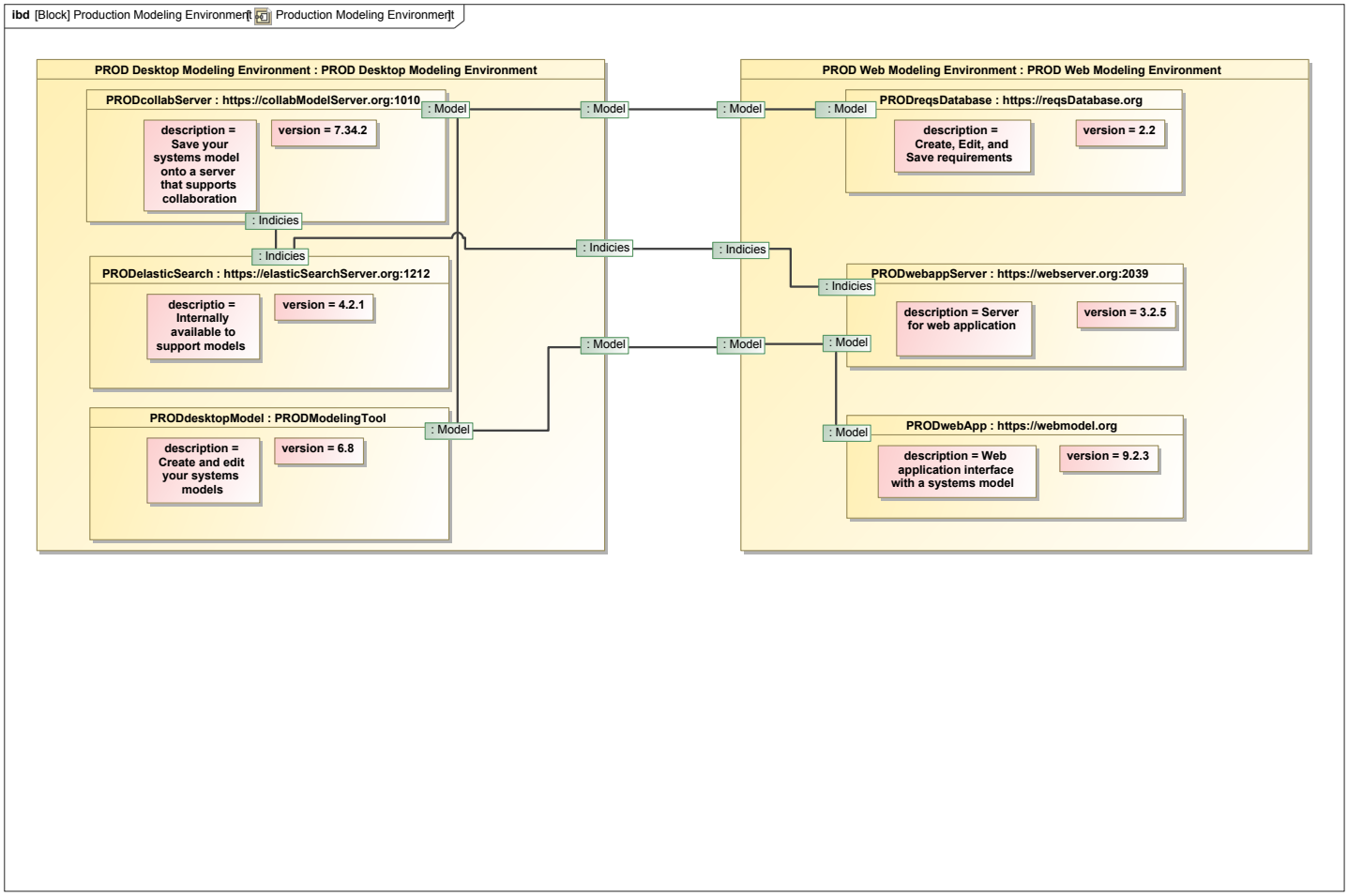


Figure 40. Production Modeling Environment

In the Production environment, the various compnents are officially defined according to the Production environment. This translates to determining the actual servers, tools, and versions for each component. One will notice that the servers are now identified according to their URL, instead of the base type; therefore URLs are used, descriptions filled in, and versions defined. It'll also be noted that the Production environment does not have The Requirements Database Integrator as defined in the specification. This means that although that component is an option for deployments, it is not actually deployed in the production environment. However, this does not limit the architecture for the other deployments, such as UAT.

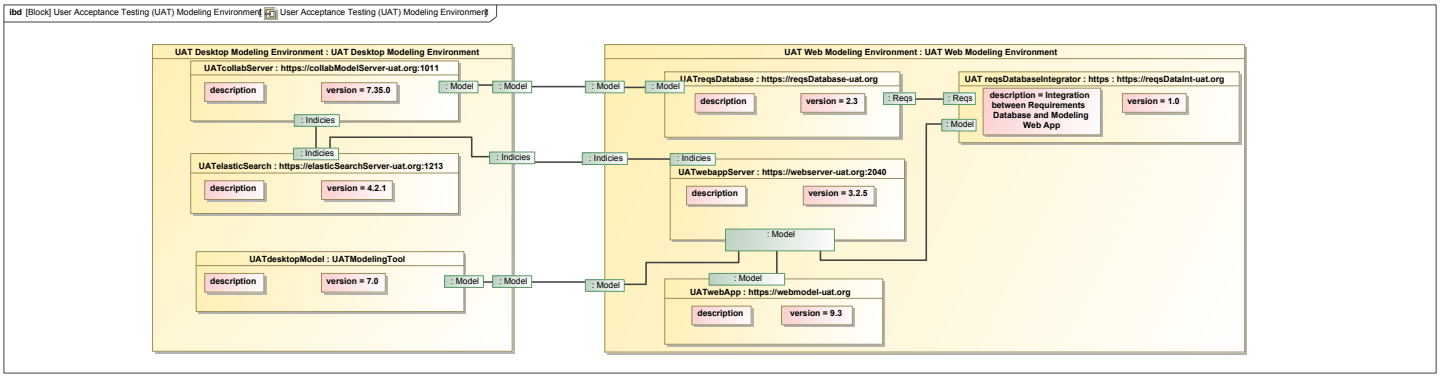


Figure 41. User Acceptance Testing (UAT) Modeling Environment

7.6 Known Uses

This pattern has been used to define deployments of the applications and services that are offered by an internal infrustcture team.

7.7 Analysis

7.8 Conflicting Usages

7.9 Related Patterns

8 Customer/Supplier Pattern

8.1 Intent

The Customer/Supplier Pattern documents the abstraction layers and specification exchange between a customer and a supplier. The pattern provides a replicable construct for an analysis realization architecture in the context of dependent and interdependent analysis between a customer and a supplier. If the modeler desires to perform an analysis of a system where the functions, behavior, specifications, or requirements are given to a supplier (from a customer) as the general workflow the Customer/Supplier Pattern can be utilized in their domain specific construct.

8.2 Motivation

The Customer/Supplier pattern addresses the concerns and motivations of a customer and supplier. The role of a systems engineering customer is to provide the resources to systems engineering organizations and individuals, and in return they receive systems engineering products and services. These stakeholder's express their needs, concerns, and expectations to which the system engineer or supplier addresses. The motivation of the customer is to provide a comprehensive specification that will be used for implementation by the supplier. While the role of the supplier is to provide products and related services that meet the stakeholder's needs and requirements. The motivation of the supplier is to construct and provide a product that complies to the customer's specifications, differential margins, and satisfies the system design.

A general workflow and supply chain between a customer and a supplier starts with the analysis realization architecture hand off to the supplier. The comprehensive specification includes the requirements for the system, predefined components, a conceptual design, and a realization design. From the specification can the system engineer produce specializations of the specification margins for the provided requirements.

The primary motivation of the pattern is that it provides a structure that allows for the supplier to have full knowledge of the structure, features, and behavior of system specification that the consumer has provided to construct a product. Through the patterns unique relationships between a consumer and a supplier, the supplier can inherit and redefine conceptual and realization components. Additionally, the pattern captures the motivations and relationship of the customer and the supplier which addresses the requirements of a stakeholder in the development of the system.

The Customer/Supplier Pattern can be applied to the development of a system where a comprehensive specification has been provided by a customer and is to be implemented by a supplier. In the implementation, the system engineer/supplier can specify the required system functionality, interfaces, and other quality characteristics to meet the customer's requirements. The pattern can be applied to a system design that is divided into components that can adhere to system requirements. These multiple levels of abstraction begin at a system of systems level. Variants of this process are applied recursively to each intermediate level of the design, and downward until which the components are procured. The Customer/Supplier Pattern captures these layers of abstraction to satisfy a customer's system design, specification, and requirements.

8.3 Concept

The context for the pattern to which it can be applied is of an analysis realization architecture. This analysis realization architecture consists of a conceptual design component and a realization design component which is produced by the supplier. The structure of the pattern consists of customer and supplier system specifications, and relationships between realization and conceptual components to each system specification.

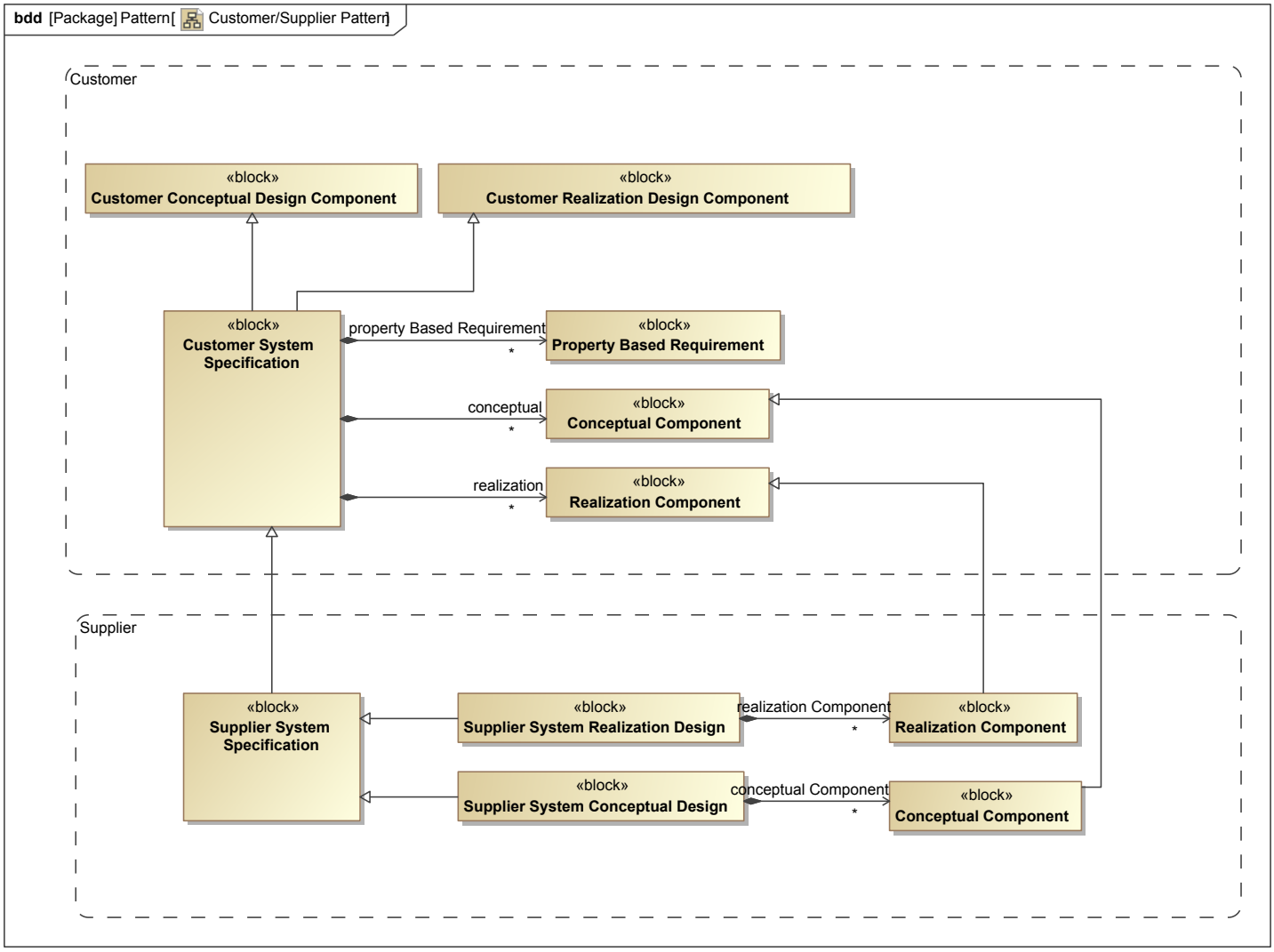


Figure 42. Customer/Supplier Pattern

In the Customer Conceptual Design Component, the functional components of a system is modeled. In the Customer Realization Design Component, the surface representation of the system is modeled. The Customer System Specification and Supplier System Specification are inherited classifiers of the these components.

The Customer System Specification block is a representation of the customer's specification for the system they want to characterize. The block specializes the design components, and will inherit any properties that are specified. The modeler can either redefine the inherited properties or create new properties that weren't included in the design components. This may be due to the design components not having knowledge of the properties at that level of abstraction. In the system specification of the customer, any and all design knowledge of the system will be specified. This can include any Property Based Requirement's the customer wants to provide the supplier so the product can be designed to satisfy the requirement(s). The customer can also provide any specific Conceptual Component's or Realization Component's to elaborate on the system specification. This is done by creating a composition relationship between the Customer System Specification to the Property Based Requirement, Conceptual Component, and/or the Realization Component.

A unique quality of the Customer/Supplier Pattern is the relationship between the Customer System Specification and the Supplier System Specification. The Supplier System Specification specializes the Customer System Specification which allows the supplier to have full knowledge of the customer's properties and requirements.

From the generalization relationships, the supplier can perform analysis on a system/component, and create their own design components. The supplier can specify a Supplier System Realization Design and a Supplier System Conceptual Design to represent the functional and physical aspects of the system. These designs specialize the Supplier System Specification block, and elaborate another level of abstraction and organization for specifying structure, function, and behavior. Additionally, each of these designs can own Realization Component and Conceptual Component part properties where the supplier can specialize the initial

component's that were provided by the customer. Through this fashion, an engineer can create functional or physical components to conform to more stringent margins for analysis. The margins of specified by the Property Based Requirement.

Through this framework, the supplier can meet and exceed the expectations of the system specification and address the concerns of a stakeholder in the development of the system.

Table 2. <>

Model Element	
Customer System Specification	The Customer System Specification represents a design black box specification which doesn't have knowledge of how the system or component is structured inside the box. The Customer System Specification generalizes the Supplier System Specification which will inherit and redefine property values. Note, formalized requirements can be applied to block's decomposed components.
Realization Component	The Realization Component block is a part property of the Customer System Specification. The block represents a physical component where the customer can attribute value properties, behavior, etc to the component. The modeler can create an infinite amount of realization components to meet the requirements of the customer.
Conceptual Component	The Conceptual Component block is a part property of the Customer System Specification. The block represents a conceptual or functional component of the system. The modeler can create an infinite amount of conceptual components which will be inherited by the supplier's Conceptual Component block, and can attribute value properties, behavior, etc to the component.
Customer Conceptual Design Component	In the Customer Conceptual Design Component, the functional component of a system is modeled. The block is generalized by the Customer System Specification, and will share its structure, function, and behavior of the system (if specified).
Customer Realization Design Component	In the Customer Realization Design Component, the surface representation of the system is modeled. The block is generalized by the Customer System Specification, and will share it's structure, function, and behavior of the system (if specified).
Property Based Requirement	The customer's needs and concerns will be defined in the Customer System Specification through properties. The Property Based Requirement represents the customer's requirements that need to be applied to the system. The customer can provide any Property Based Requirements the customer wants to provide the supplier so the product can be designed to satisfy the requirement(s). This is done by creating a composition relationship between the Customer System Specification to the Property Based Requirement.
Supplier System Specification	The Supplier System Specification represents a design black box specification which doesn't have knowledge of how the system or component is structured inside the box. The Supplier System Specification specializes the Customer System Specification which can inherit and redefine property values. Note, formalized requirements can be applied to block's decomposed components.
Supplier System Realization Design	The supplier can perform analysis on a system/component by creating their own design. The supplier can specify a Supplier System Realization Design to represent the physical aspects of the system. The Supplier System Realization Design block specializes the Supplier System Specification, and will inherit the block's properties and behavior.
Supplier System Conceptual Design	The supplier can perform analysis on a system/component by creating their own design. The supplier can specify a Supplier System Conceptual Design to represent the functional and physical aspects of the system. The Supplier System Conceptual Design block specializes the Supplier System Specification, and will inherit the block's properties and behavior.
Conceptual Component	The Conceptual Component block is a part property of the Supplier System Conceptual Design. The block represents a conceptual or functional component that can specialize and inherit the properties of the customer's Customer Conceptual Design Component. The modeler can create an infinite amount of conceptual components to meet the requirements of the customer, and can attribute value properties, behavior, etc to the component.

Model Element	
Realization Component	The Realization Component block is a part property of the Supplier System Realization Design. The block represents a physical component that can specialize and inherit the properties of the customer's Realization Component. The modeler can create an infinite amount of realization components to meet the requirements of the customer, and can attribute value properties, behavior, etc to the component.

8.4 Participants

8.4.1 Supplier Participants

A listing of the model elements used on the Supplier side, and their roles in the Customer/Supplier Pattern

Table 3. Supplier Participants

Element	Role
Realization Component	The Realization Component block is a part property of the Supplier System Realization Design. The block represents a physical component that can specialize and inherit the properties of the customer's Realization Component. The modeler can create an infinite amount of realization components to meet the requirements of the customer, and can attribute value properties, behavior, etc to the component.
Supplier System Realization Design	The supplier can perform analysis on a system/component by creating their own design. The supplier can specify a Supplier System Realization Design to represent the physical aspects of the system. The Supplier System Realization Design block specializes the Supplier System Specification, and will inherit the block's properties and behavior.
Conceptual Component	The Conceptual Component block is a part property of the Supplier System Conceptual Design. The block represents a conceptual or functional component that can specialize and inherit the properties of the customer's Customer Conceptual Design Component. The modeler can create an infinite amount of conceptual components to meet the requirements of the customer, and can attribute value properties, behavior, etc to the component.
Supplier System Specification	The Supplier System Specification represents a design black box specification which doesn't have knowledge of how the system or component is structured inside the box. The Supplier System Specification specializes the Customer System Specification which can inherit and redefine property values. Note, formalized requirements can be applied to block's decomposed components.
Supplier System Conceptual Design	The supplier can perform analysis on a system/component by creating their own design. The supplier can specify a Supplier System Conceptual Design to represent the functional and physical aspects of the system. The Supplier System Conceptual Design block specializes the Supplier System Specification, and will inherit the block's properties and behavior.

8.4.2 Customer Participants

A listing of the model elements used on the Customer side, and their roles in the Customer/Supplier Pattern.

Table 4. Customer Partcipants

Element	Role
Customer System Specification	The Customer System Specification represents a design black box specification which doesn't have knowledge of how the system or component is structured inside the box. The Customer System Specification generalizes the Supplier System Specification which will inherit and redefine property values. Note, formalized requirements can be applied to block's decomposed components.
Customer Realization Design Component	In the Customer Realization Design Component, the surface representation of the system is modeled. The block is generalized by the Customer System Specification, and will share it's structure, function, and behavior of the system (if specified).

Element	Role
Customer Conceptual Design Component	In the Customer Conceptual Design Component, the functional component of a system is modeled. The block is generalized by the Customer System Specification, and will share its structure, function, and behavior of the system (if specified).
Realization Component	The Realization Component block is a part property of the Customer System Specification. The block represents a physical component where the customer can attribute value properties, behavior, etc to the component. The modeler can create an infinite amount of realization components to meet the requirements of the customer.
Conceptual Component	The Conceptual Component block is a part property of the Customer System Specification. The block represents a conceptual or functional component of the system. The modeler can create an infinite amount of conceptual components which will be inherited by the supplier's Conceptual Component block, and can attribute value properties, behavior, etc to the component.
Property Based Requirement	The customer's needs and concerns will be defined in the Customer System Specification through properties. The Property Based Requirement represents the customer's requirements that need to be applied to the system. The customer can provide any Property Based Requirements the customer wants to provide the supplier so the product can be designed to satisfy the requirement(s). This is done by creating a composition relationship between the Customer System Specification to the Property Based Requirement.

8.5 Consequences

There are several trade-offs to consider when applying the Customer/Supplier Pattern to the modeler's system.

Table 5. Consequences Table

Action	Consequence
Supplier redefines the value based properties that were provided by the customer to meet tighter margins.	By redefining existing value properties, the supplier can perform analysis of the system based on more stringent requirements. An instance in which this can benefit the supplier is if the supplier provides a requirement that has variable range in the value of the property. Based on the supplier's black box specification, the supplier can modify the redefined value property to meet a narrower range or value of a property for analysis purposes.
The supplier's system specification specializes the customer's system specification.	The relationship between the two shows designates that the customer's black box specification and any parts/properties are inherited by the supplier's black box specification. This relationship allows for the decoupling of the supplier and the customer. Through this fashion, both the customer and the supplier are able to specify their own system designs depending on their requirements and needs.
Performing V&V checks of the customer's system against the specification.	Through the independent structure of the customer and supplier's system specifications, verification and validation can be performed on the solely the customer's black box specification design to check whether the system meets the requirements and specifications
Performing V&V checks of the supplier's system against the specification.	Through the independent structure of the customer and the supplier's system specifications, verification and validation can be performed on solely the supplier's black box specification design to check whether the system meets the requirements and specifications of its intended design.

Listing of possible conflicts that can occur while applying or using the Customer/Supplier Pattern.

Usage of the Pattern	Explanation of Conflict

8.6 Implementation

To display the relationship and exchange of information between a customer and a supplier, a sample model has been created. The Customer/Supplier Pattern can be applied to a customer/supplier relationship between a roller-coaster park and a roller-coaster

manufacturer for the production of a roller-coaster. The purpose of the model is to show how through the generalization relationship the manufacturer can expand on requirements and system design of the customer in order to perform analysis of the system.

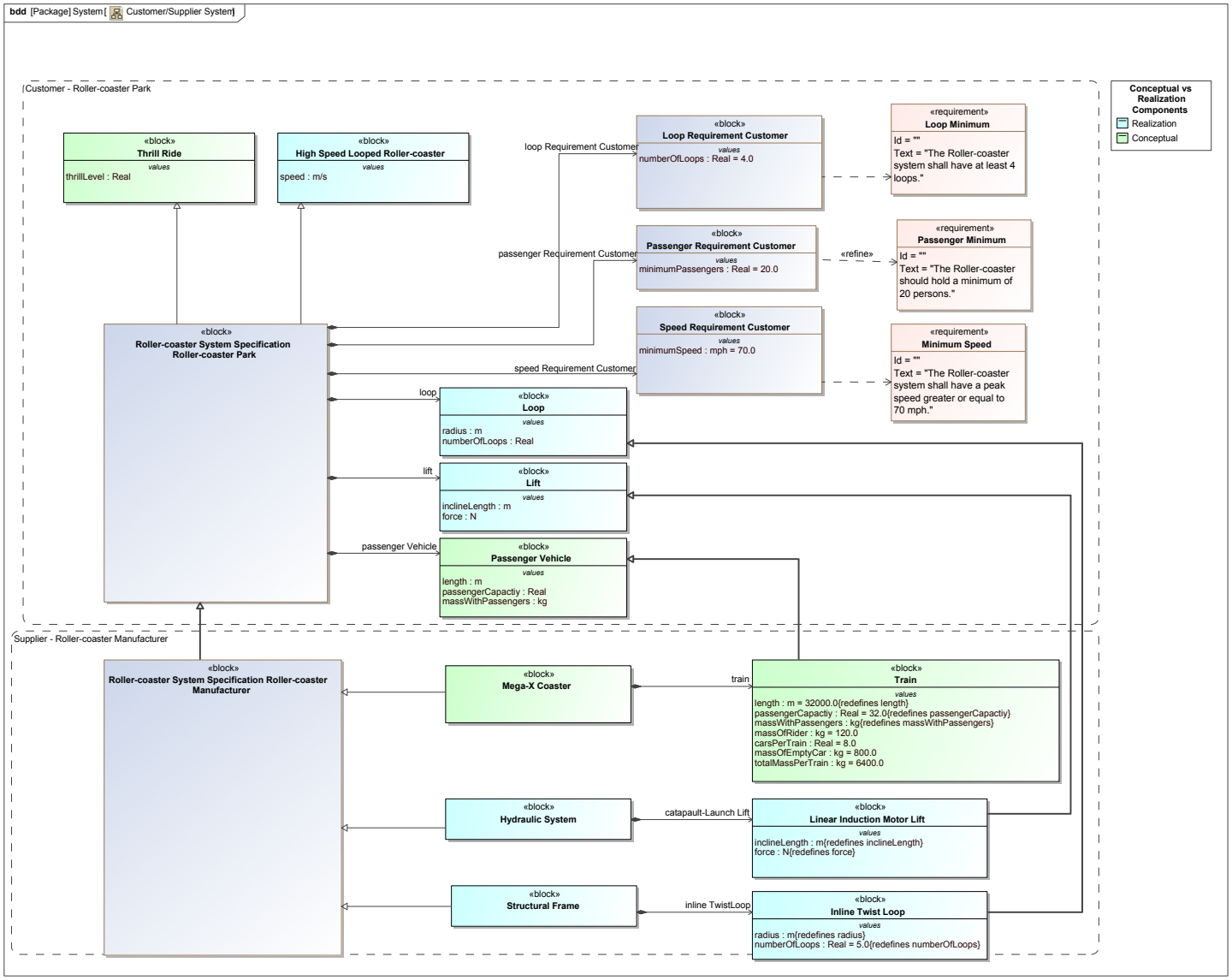


Figure 43. Customer/Supplier System

The customer of the sample model is a Roller-coaster Park, which requires a product to be supplied by a supplier. The park's conceptual design for the product is that they require a Thrill Ride for the park. The realization design for the product is that the park requires a High Speed Looped Roller-coaster . Additionally, the property based requirements that were provided by the customer were a Loop Minimum, a Passenger Minimum, and a Minimum Speed. The Roller-coaster System Specification Roller-coaster Park black box specification also is composed of a conceptual component Passenger Vehicle with the value properties of length, passenger capacity, and mass. The realization design components Loop and Lift have the properties of radius, inclineLength, and force. These properties can be redefined by the supplier. The Roller-coaster System Specification Roller-coaster Manufacturer black box specification is composed of a system conceptual design roller-coaster Mega-X Coaster that is composed of a Train. The Train block redefines the properties from the supplier's Passenger Vehicle, and has additional properties are defined by the manufacturer that have an affect on the uphill and downhill speeds of the roller-coaster. The system realization design for the Hydraulic System consists of a Linear Induction Motor Lift. The Linear Induction Motor Lift redefines the properties of the supplier's Lift that can be used to calculate the speed. The system realization design for the Structural Frame consists of a Inline Twist Loop, and redefines the properties of the supplier's Loop.

8.7 Known Uses

The Thirty Meter Telescope shares a similar method for modeling the relationship between a customer and supplier of the telescope's Alignment and Phasing System (APS). The customer, TMT, provides a black box specification of the APS for the supplier, JPL.

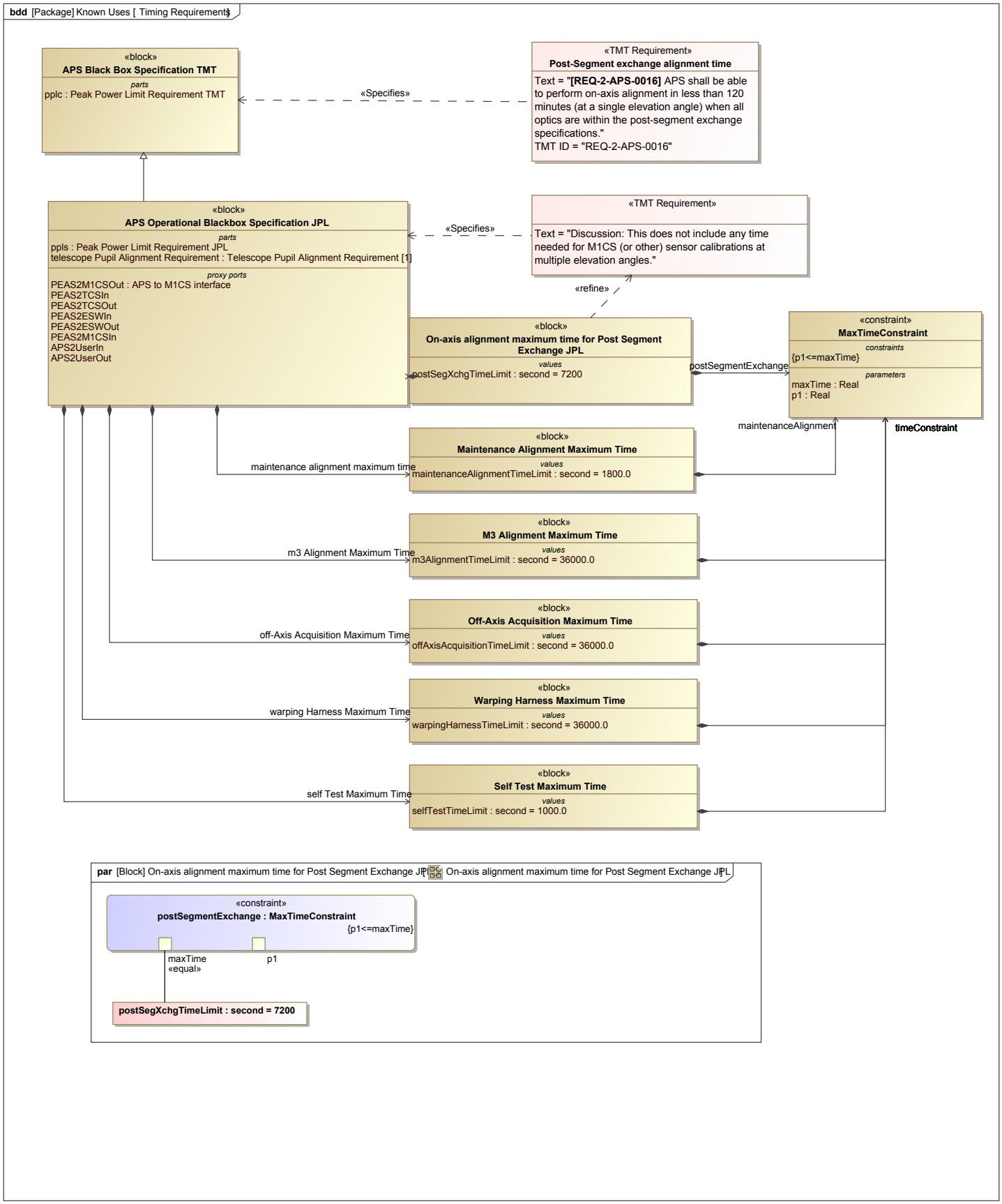


Figure 44. Timing Requirements

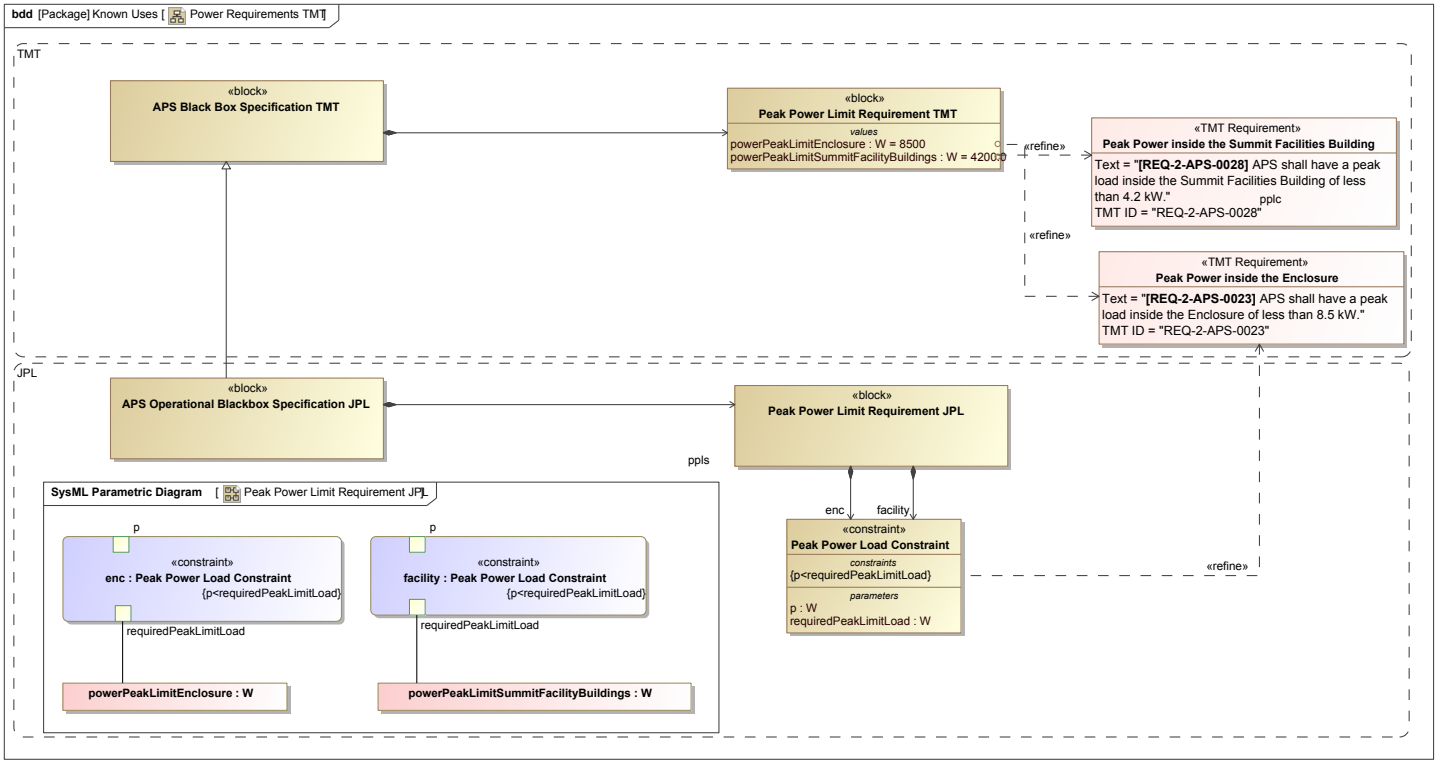


Figure 45. Power Requirements TMT

8.8 Related Patterns

List of other patterns that are related to the Customer/Supplier Pattern, and the differences and similarities between them.

Pattern Name	Similarities	Differences
Property Based Requirements Pattern	During the exchange of information between a customer and a supplier, the customer can provide property based requirements that are to be applied in the supplier's system analysis. Requirements are used to establish a contract between the customer (or other stakeholder) and those responsible for designing and implementing the system. The Property Based Requirements Pattern is a good source for customer's to describe textual requirements through properties that can then be applied in the Customer/Supplier Pattern.	The purpose of the Customer/Supplier Pattern is to provide a context for the exchange of system knowledge from customer to supplier. Meanwhile, the Property Based Requirements Pattern describes a method to transform textual requirements to a model based fashion through SysML properties.

9 Spatial Reference Frame -DRAFT-

9.1 Intent

In systems engineering it is important to establish a Frame of Reference in which to identify a system's position in relation to another system's position. A Frame of Reference is a position in a specified space with a coordinate system imposed on it. Through a series of geometrical relationships, an engineer using this pattern would be able to compare the position of components in a system to one another using a vector.

9.2 Motivation

Frame of Reference can be used to verify the positions of both various systems and the components that make up these sub-systems. This pattern should allow a user to accomplish two goals: the transformation of a CAD model to a SysML model and a basic Frame of Reference pattern used to determine the positions of system's components with respect to components in another system or components in it's own system. This could mean having a system such as a Conference Room set in a Spatial Reference frame that lies in 3D Euclidean Coordinate Space and comparing it's position to that of a doorknob which lies in a 3D Spherical Coordinate space. By comparing their given position vectors and applying geometric math and relationships defined in IBDs and Parametrics, the user should be able to relate the position of the two objects despite their different coordinate systems.

Observational frames of references is used when the emphasis is upon the state of motion rather than upon the coordinate choice or the character of the observations or observational apparatus. This reference frame allows the study of effect of motion upon an entire family of coordinate systems attached to the frame.

A mathematical concept for coordinate systems is amounting to a choice of language that is used to describe observations. An observer in an observational frame of reference can choose to employ any coordinate system (i.e. Cartesian, polar, curvilinear, generalized,...) to describe observations made from that frame of reference. A change in the choice of this coordinate system does not change an observer's state of motion, and so does not entail a change in the observer's observational frame of reference.

There is no necessary connection between coordinate systems and physical motion (or any other aspect of reality). However, coordinate system can include time as a coordinate, and can be used to describe motion. In engineering, they could be angles of relative rotations, linear displacements, or deformation of joints. Deciding what to measure and with what observational apparatus is a matter separate from the observer's state of motion and choice of coordinate system.

9.3 Concept

The structure for the Spatial Reference Frame pattern is derived from the ISO-18026 Standard: Spatial Reference Model. Using this ISO Standard, it is easy to create a SysML model from the classes and definitions specified throughout the document. The pattern specified here merely scrapes the surface of the information laid out in ISO-18026 but gives a good starting point for anyone who may want to model Spatial Reference Frame in more depth.

All diagrams provided in this Chapter come from various portions of ISO-18026. To download the Standard follow the hyperlink above. The .zip file provides dozens of PDFs that are hyperlinked in the Table of Contents (ISOIEC_18026_TOC.PDF). The Table of Contents will be what is referenced throughout this documentation.

The frame of reference has several properties that can also be considered patterns. The most important one is the origin. Always having a defined origin is a very important pattern for the frame of reference. The origin is a reference point for the reference frame that establishes a zero point. A distance of 100 yards has meaning in the frame, which is understood that the distances is measured relative to the frame's origin.

In weapon systems, target direction from the origin is measured in angular quantities. To give meaning and a zero reference to angular quantities, reference lines were established. Reference lines pass through the origin and establish the axes of the reference frame.

In addition to reference lines, angular measurements require the definition of planes in which angles are to be measured. For example, an object 100 yards away from the origin and 45 degrees.

Stability

9.3.1 Spatial Reference Frame MetaModel

The SRFMetaModel provides a basic layout for the generic classes of the Spatial Reference Model. This diagram is referenced directly from **Annex C**, **Figure C.1-SRM concepts and their relationships**. The black diamond relationships represent a composition relationship i.e SpatialReferenceFrame is composed of an AbstractCoordinateSystem and an ObjectReferenceModel. This means that SpatialReferenceFrame has the part properties abstractCoordinateSystem typed by the class AbstractCoordinateSystem. The open arrow line depicts a generalization relationship between a general classifier and specialized classifier i.e. a Coordinate is a more specilized classifier of AbstractCoordinateSystem and will inherit the common features from the general classifier. The part properties contained in the general classes and not shown on this diagram will shown in the subchapters of this section starting with Spatial Reference Frame Classes. For simplicity, the description and modeling of the general classes refer to the **Concepts (Chapter 4)** of ISO-18026.

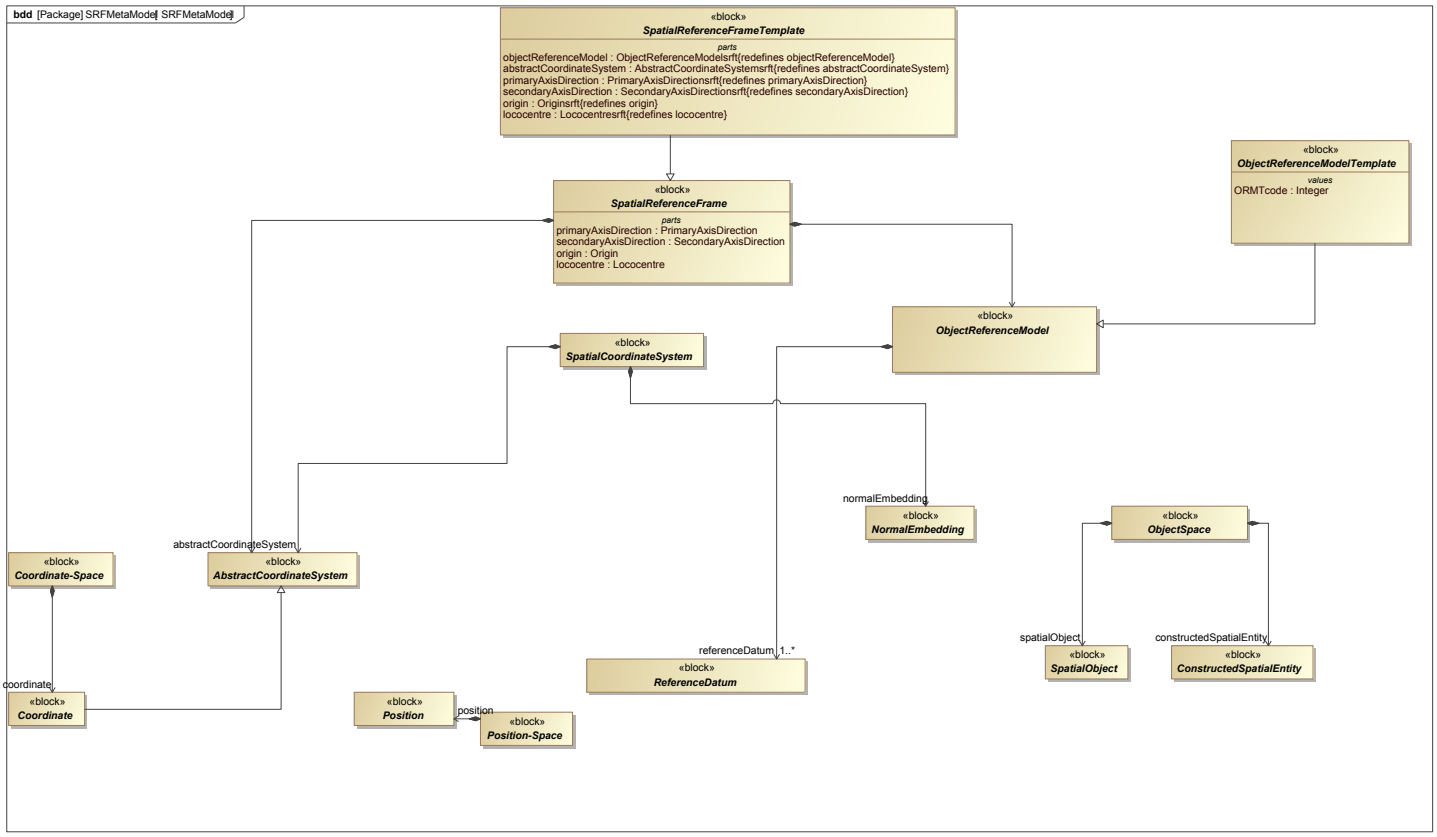


Figure 46. SRFMetaModel

9.3.1.1 Spatial Reference Frame Classes

Be aware that ISO-18026 starts by defining the composition of a SpatialReferenceFrame and then describing the concept of a SpatialReferenceFrame. In this document, SpatialReferenceFrame will be the first concept outlined.

This section will be referring to **4.7 Spatial reference frames** of the ISO-18026 Standard. A SpatialReferenceFrame will also be referred to as a SRF. A SRF is defined as "a means of specifying a spatial coordinate system for an object space. The spatial reference frame concept uses an object reference model to specify a normal embedding. That normal embedding combined with an abstract coordinate system comprises the spatial coordinate system." To further understand this concept refer to **Figure 4.1-SRM conceptual relationships**:

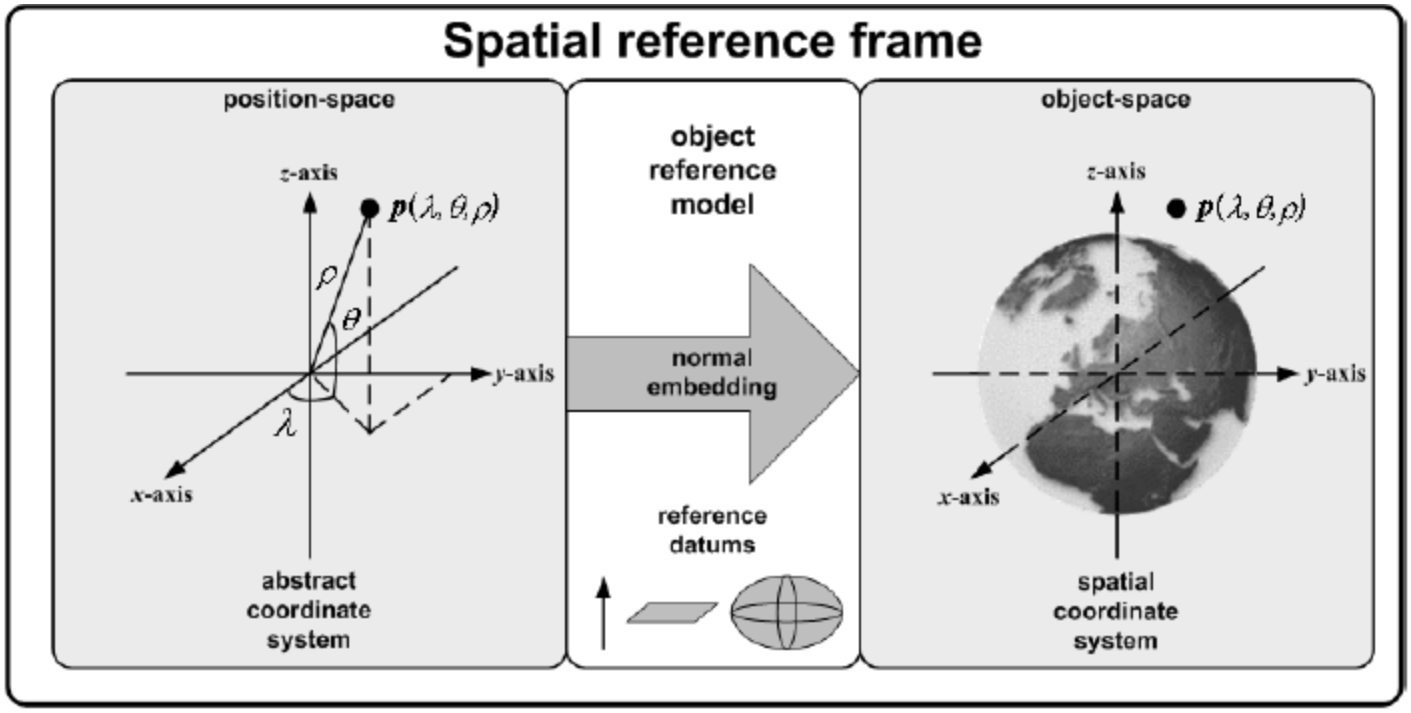


Figure 4.1 — SRM conceptual relationships

An AbstractCoordinateSystem is imposed on a Position-Space and the Reference Datums that comprise the ObjectReferenceModel determine how Position-Space relates to Object-Space and this relationship is expressed mathematically through the normal embedding.

The pattern used in this Cookbook uses only some of these concepts for simplicity sake. The SpatialReferenceFrame in the following model is composed of five main parts:
objectReferenceModel, abstractCoordinateSystem, primaryAxisDirection, secondaryAxisDirection, origin, and lococentre. Each part property is Typed by it's general class meaning that each part property has the properties and behavior of it's Type. For Example, secondaryAxisDirection will have the value properties *sx,sy* and *sz* just like the Class SecondaryAxisDirection. SRFclasses shows this composition relationship between the part properties and SRF. The part properties can now be a new usage of it's type in a context.

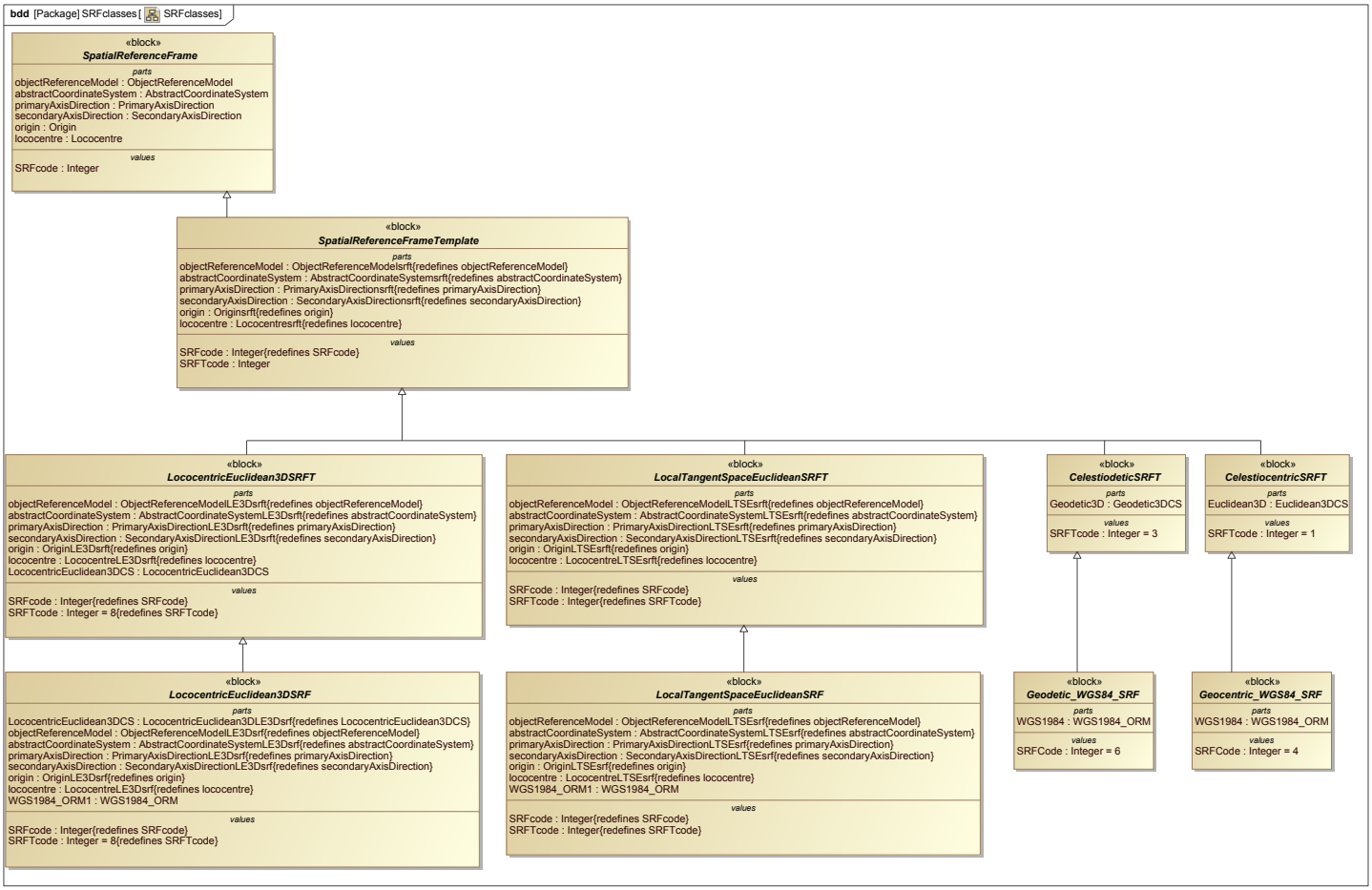


Figure 47. SRFclasses

A SpatialReferenceFrameTemplate or SRFT is "an abstraction of a set of spatial reference frames that share the same abstract coordinate system, coordinate system parameter binding rules, and coordinate-component names and symbols." This is shown as a generalization relationship between the SRF and SRFT where SRFT is the specialization of an SRF. An SRFT inherits the properties of an SRF and redefines them to give them a different context. Each part property gives the SRFT specific attributes which can then be further specialized to be used for different contexts. For example, the diagram above shows that the LococentricEuclidean3DSRFT is a specialization of SpatialReferenceFrameTemplate. All of the part properties that compose LococentricEuclidean3DSRFT are redefined to a new context and thus can have new Default Values or in the case of this model, can be further redefined for a specific SRF. There is a defined set of standardized SRFT's and their corresponding SRF's in **Chapter 8-Spatial reference frames** of ISO-18026.

The SRFs modeled in the example pattern were specified in the SRM API User's Guide which is modeled in Implementation. SRFs such as LococentricEuclidean3DSRF are specializations of their SRFTs. Their part properties are the same as their general classifier's but are now Typed for new usage in the new context. A component such as an AircraftBody will be composed of a SRF which will allow the AircraftBody's position to be in a defined context. The component's position is determined by the part properties' various attributes which have Default Values assigned to them.

It was determined to use these specific part properties based off of ISO-18026 User's Guide. The snips of C code define certain properties that each SRF are comprised of. The part properties are then redefined for each SRF to give it a new usage and context but with the same attributes as the original part properties. The part properties that make up an SRF compose the general classes of the model and are explained in the following subviews.

9.3.1.2 Reference Datums

ReferenceDatum "is a geometric primitive in position-space. Reference datums are points or directed curves in 2D position-space or points, directed curves or oriented surfaces in 3D position-space." Due to the abstract nature of this concept, the SysML model only includes these concepts as Classes and does not attempt to show the mathematical relationship of RDs. The diagram below shows the generalization relationship between the various kinds of Reference Datums. The only one modeled down to it's components is Point which is a specialization of BaseCoordinate3D. These various RD's are then used to compose

different ObjectReferenceModelTemplate. This is shown in the following subview, Object Reference Frame Classes. The diagram below is a SysML interpretation of the picture from **Annex C**, **Figure C.2-3D RDs**.

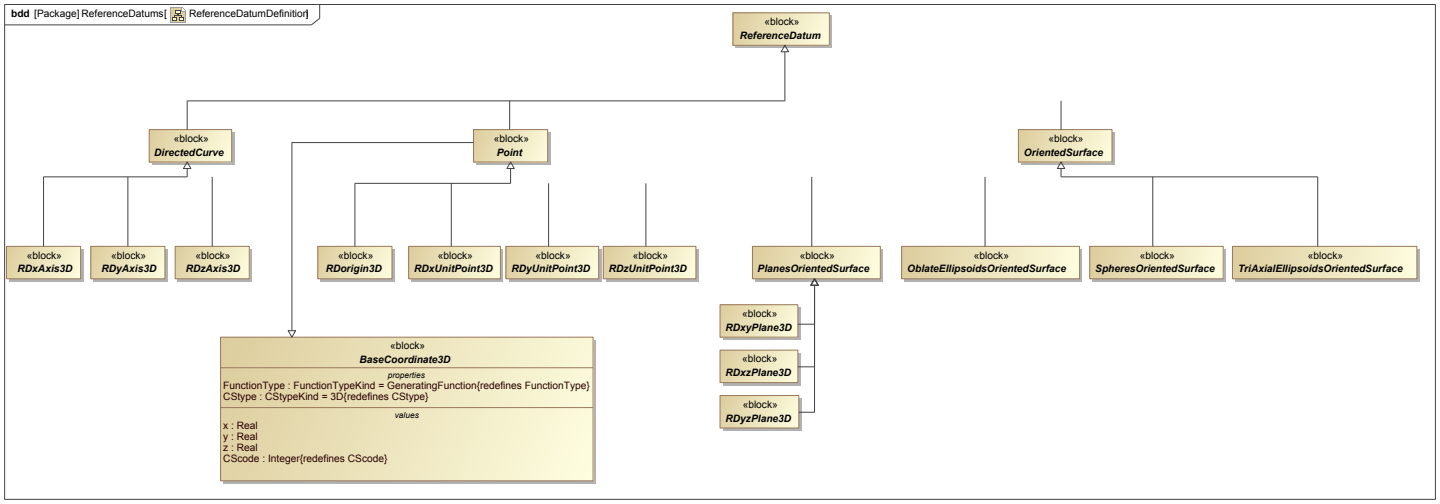


Figure 48. ReferenceDatumDefinition

9.3.1.3 Object Reference Frame Classes

An ObjectReferenceModel "for a spatial object is a set of bound reference datums for which there exists exactly one (right-handed in the 3D case) normal embedding of position-space into object-space that is compatible with each reference datum binding in the set." The supplied pattern shows a specific ORM as a specialization of an ObjectReferenceModelTemplate which is composed of various ReferenceDatum which are defined in Reference Datums .

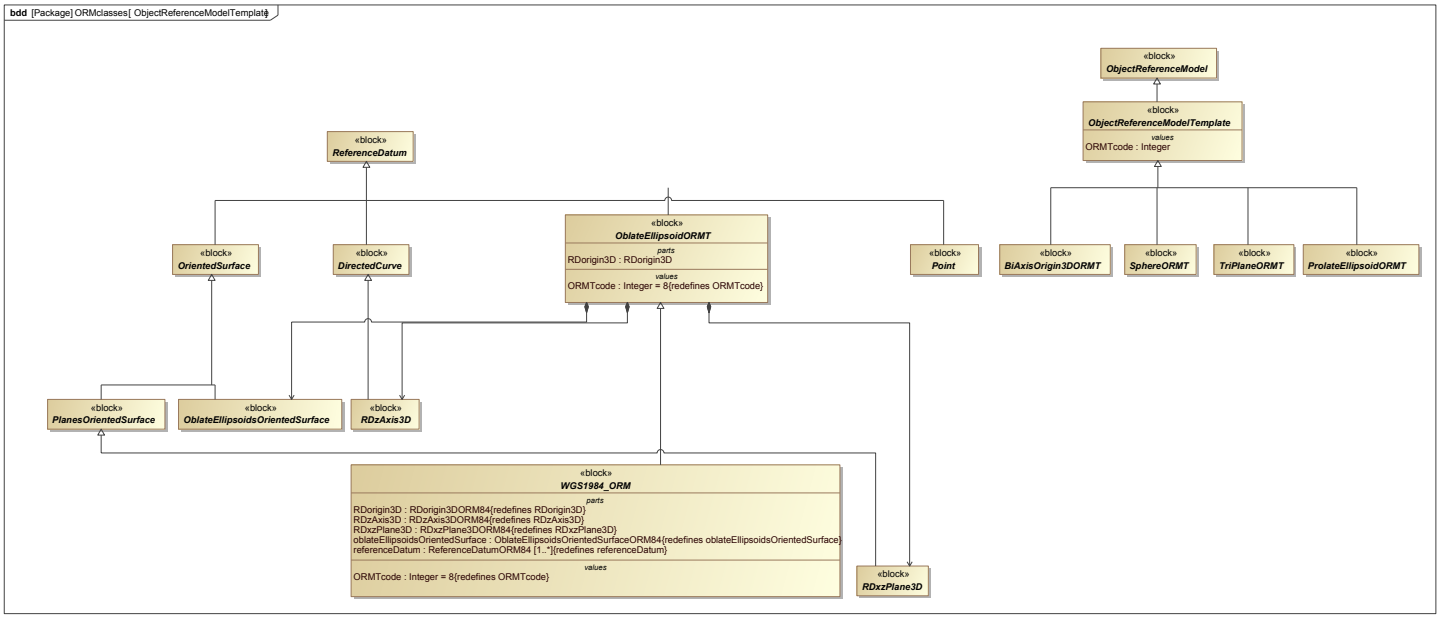


Figure 49. ObjectReferenceModelTemplate

The WGS1984_ORM is a specialization of the OblateEllipsoidORMT and is used as the standard ORM in the SRM User Guide. Each ORMT is composed of several Reference Datums which can be specializations of Point, DirectedCurve or OrientedSurface. The diagram above is a SysML interpretation of the picture in **Annex C**, **Figure C.2- 3D RDs**.

9.3.1.4 Abstract Coordinate Systems

An AbstractCoordinateSystem, as defined by the ISO-18026 standard, **Chapter 5 Abstract coordinate Systems**:

> An abstract Coordinate System (CS) is a means of identifying a set of positions in an abstract Euclidean space that shall be comprised of:
> a) a CS domain,

b) a generating function, and
c) a CS range,
where:
a) The CS domain shall be a connected replete domain in the Euclidean space of n-tuples (1 ? n ? m), called the coordinate-space.
b) The generating function shall be a one-to-one, smooth, orientation-preserving function from the CS domain onto the CS range.
c) The CS range shall be a set of positions in a Euclidean space of dimension m (n ? m ? 3), called the position-space. When n = 2 and m = 3, the CS range shall be a subset of a smooth surface. When n = 1 and m = 2 or 3, the CS range shall be a subset of an implicitly specified smooth curve.

For the purposes of this model only Function Type and the components that make up a CS, i.e. x, y and z are modeled properties.

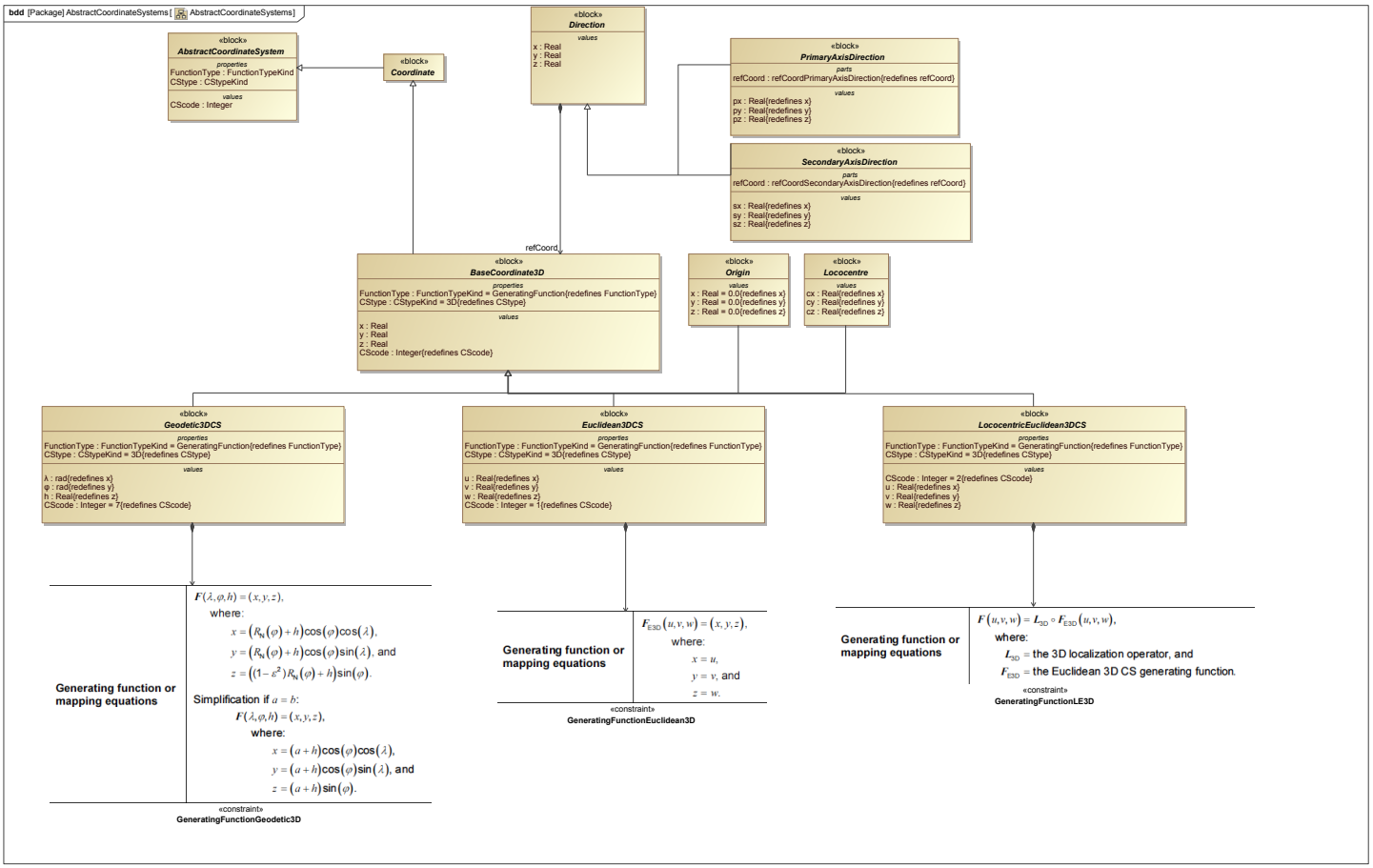


Figure 50. AbstractCoordinateSystems

The diagram shows a generalization relationship between an AbstractCoordinateSystem and a Coordinate which is further specialized to be BaseCoordinate3D.

9.4 Simulation

A description of how the model elements interact with each other when simulated.

9.5 Consequences

A description of the results, side effects, and trade offs caused by using the pattern.

9.6 Implementation

9.6.1 Spatial Refernce Frame Users Guide

9.6.1.1 Battlefield Composition

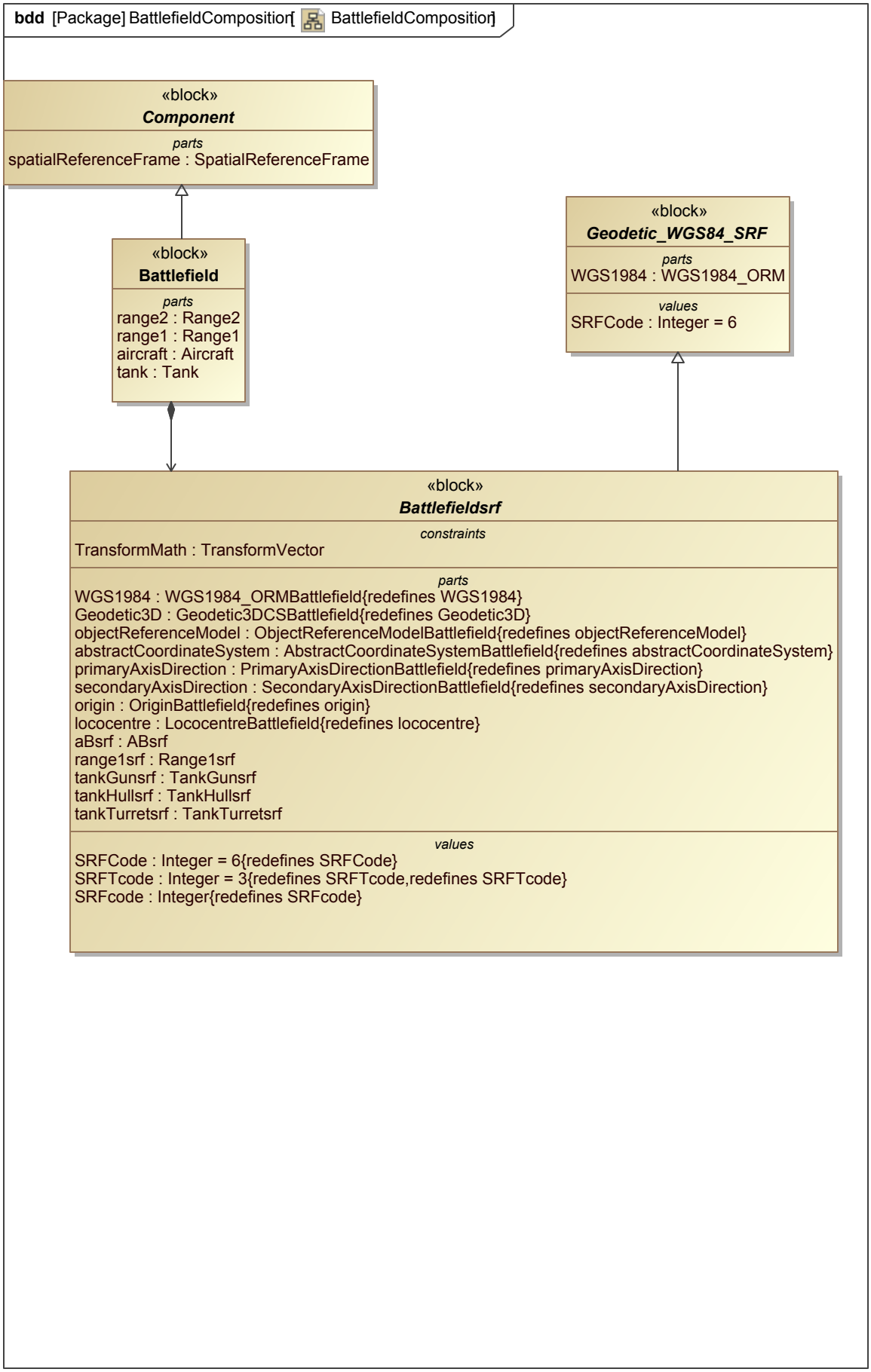


Figure 51. BattlefieldComposition

9.6.1.2 Range Composition

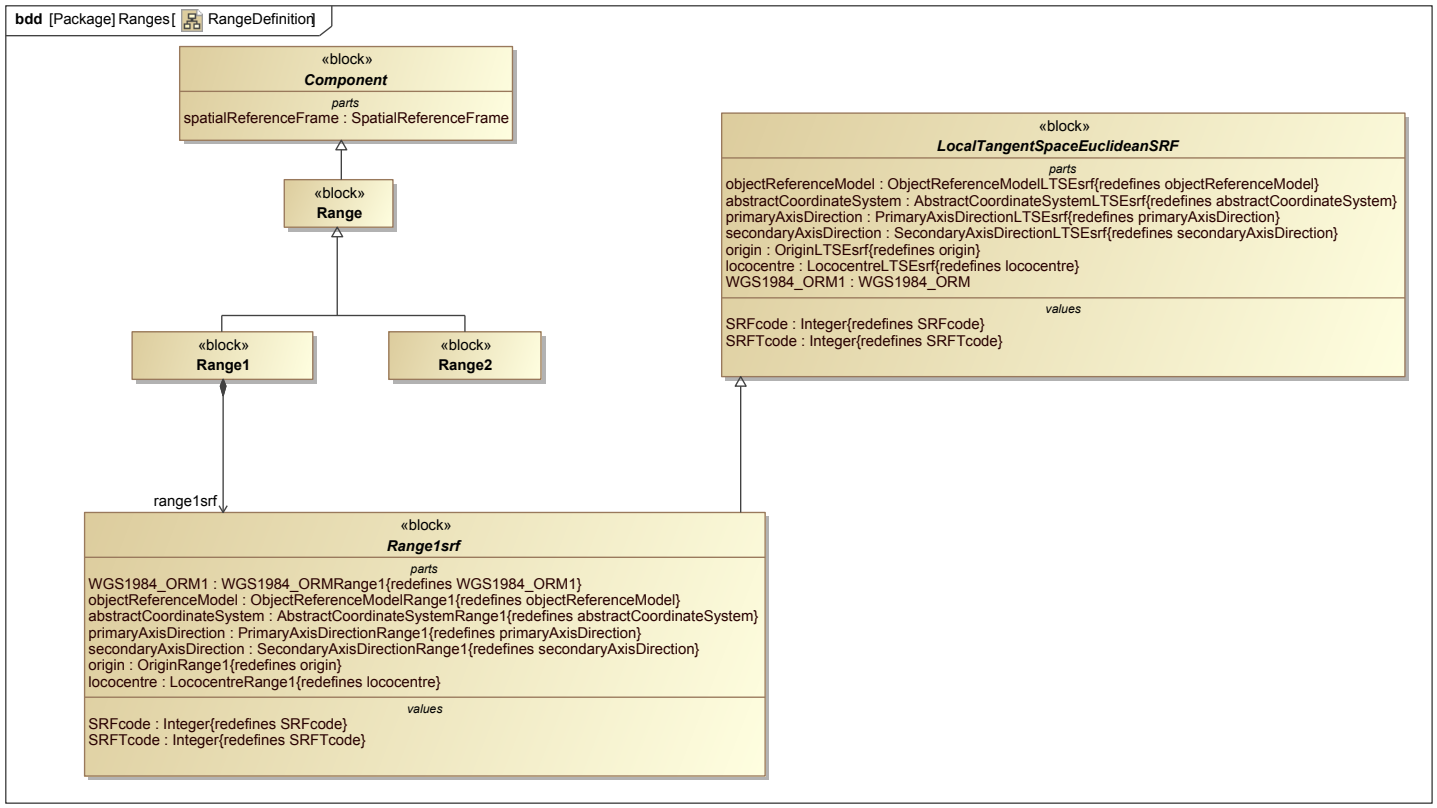


Figure 52. RangeDefinition

9.6.1.3 Aircraft Compostion

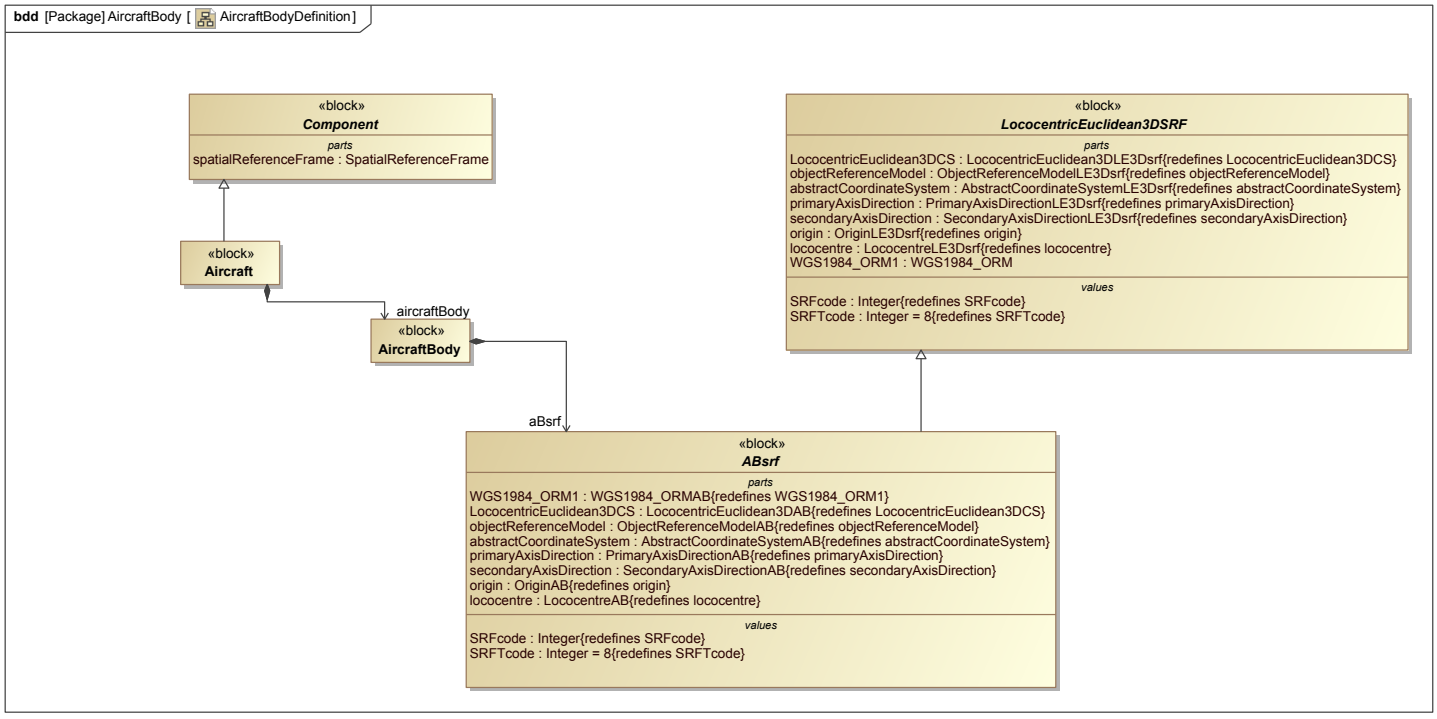


Figure 53. AircraftBodyDefinition

9.6.1.4 Tank Compostion

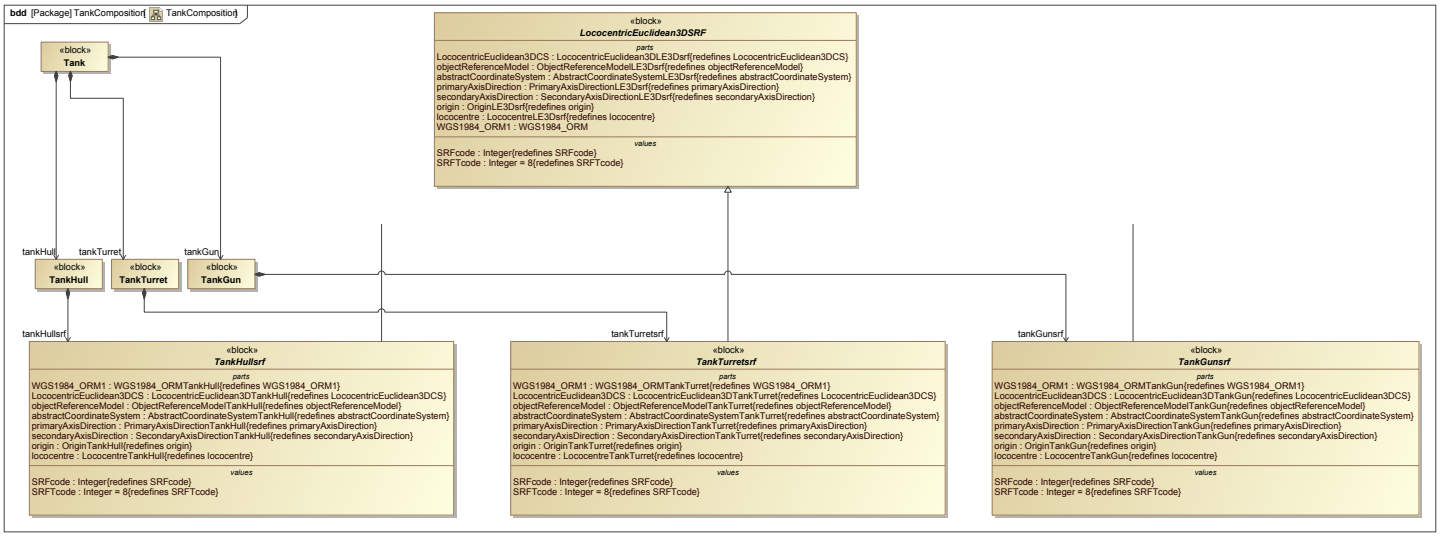


Figure 54. TankComposition

9.7 Known Uses

Examples of real usages of the pattern.

9.8 Analysis

How this pattern helps to analyze a system model

9.9 Related Patterns

10 Enabling Activity Reusability in States Pattern -DRAFT-

10.1 Consequences

Action	Consequence

10.2 Analysis

10.3 Concept

10.4 Implementation

10.5 Motivation

The motivation for using this pattern is to enable usage of activities and states to be owned by any element of the model. An example is a state with a block as its context. The block has the Owned behavior and classifier behavior associated with that state.

10.6 Intent

The purpose of this pattern is to utilize a pre-defined behavior in a different location of a SysML model. This is done by creating a state machine and setting it as an owned and classifier behavior for a component of the model.

10.7 Known Uses

10.8 Related Tooling

The main tool used for this pattern was MagicDraw 18.5.

10.9 Related Patterns