



02 Systems Requirements Management

Table of Contents

1 Introduction.....	6
2 Property Based Requirements Pattern	7
2.1 Intent	7
2.2 Motivation.....	7
2.3 Concept	7
2.4 Consequences.....	11
2.5 Implementation	12
2.6 Known Uses	15
2.7 Tooling	16
2.7.1 Model Construction.....	16
2.8 Related Patterns.....	17
3 Requirement Verification Pattern	19
3.1 Intent	19
3.2 Motivation.....	19
3.3 Concept	20
3.4 Consequences.....	22
3.5 Implementation	23
3.6 Known Uses	32
3.7 Analysis.....	37
3.8 Tooling	38
3.8.1 Cameo Simulation Toolkit	38
3.9 Related Patterns.....	41

List of Tables

1. <>11

2. PBR Consequences11

3. Related Patterns17

4. <>22

5. Consequences Table.....22

6. Post-Segment Exchange Alignment Timing Analysis Results36

7. Analysis Results.....37

8. Related Patterns Table41

List of Figures

1. Property Based Requirement Pattern	8
2. Requirement Constraint Block Pattern	9
3. Property Based Requirement Hierarchy	10
4. Property Based Requirement Pattern SysML 1.4	10
5. Property Based Requirements Time Example	13
6. Property Based Requirements Area Example	14
7. System Verification Example	15
8. Power Requirements TMT	16
9. Systems Reasoner	17
10. Requirements Verification Structure	20
11. Analysis Context	21
12. Analysis Context Scenarios	21
13. Domain Architecture	23
14. Structural Decomposition	24
15. Communication Interface Definition	24
16. Conceptual Node Design	25
17. Communication Context	25
18. Autonomous Ship Modes	26
19. Transmit Data	27
20. Consume Power	28
21. Send Location Data	28
22. Recharge Batteries	29
23. Remote Navigation Control	29
24. Receive Data	30
25. Data Requirement	30
26. Analysis Context	31
27. Operational Scenario	32
28. APS Realization Structure	33
29. Use Case: Post-Segment Exchange Alignment	33
30. Coarse Tilt Alignment	34
31. Post-Segment Exchange Duration Scenario Analysis	34
32. Peak Power Limit Scenario Online	35
33. APS Realization Configuration Scenario Online	35
34. Simulation	38
35. Simulation	39
36. Communication Context	39
37. Analysis Context	39
38. Transmit Data	40
39. Simulation UI	41

1 Introduction

The Systems Requirements Management Case Study focuses on developing requirements and providing methods for analyzing requirements.

2 Property Based Requirements Pattern

2.1 Intent

The Property Based Requirements Pattern provides a method for generating formal expression requirements to model the quantitative specification of numerical parameters, relationships, equations and/or constraints. The intent of the Property Based Requirements Pattern is to supply a solution to represent numerical requirements more precisely, to reduce ambiguity, and to facilitate system verification by analysis and other methods.

The defining qualities of a Property Based Requirement is that the requirements shall have numerical properties (properties capable of representing numerical values), the numerical properties shall be typeable (preferably by `ValueType`) to account for quantity kind and units, and the numerical properties shall be bindable (preferably using `BindingConnector`) to other model elements (e.g., `ConstraintParameters`) so they can be evaluated using analysis tools. The proposed property-based requirement is intended to be used with the overall system model to assist in specifying and architecting systems.

2.2 Motivation

Formally, a requirement specifies a condition that must or should be satisfied. A requirement may specify a function of a system that is to be performed or a performance condition a system must satisfy. Relationships to or from requirements can be specified, and enable the modeler to relate requirements to another and to other model elements. The modeler can create these relationships by either defining a requirements hierarchy, deriving requirements, satisfying requirements, verifying requirements, or refining requirements. In the specification of a standard SysML requirement, a unique identifier and text requirement properties can be specified. Additionally, the user may need to specify additional properties such as verification status.

The motivation behind the Property Based Requirements Pattern is to provide a method to transform the text and id properties of a formal requirement into an element that inherits the properties, attributes, and functions of the requirement and block classes. Formal requirements of a system/project are often provided by engineers in a textual format. However, by modeling these formal requirements in SysML the engineer has the extended capabilities to support the specification, analysis, design, verification and validation of the system. Through the process of extracting the content of textual requirements into value and constraint properties, the engineer can provide a more detailed view and can perform a requirement analysis of their system.

Requirements are typically used to establish a contract between a stakeholder and those responsible for the design and implementation of the system. SysML provides the modeling constructs to represent formal text-based requirements, and to relate them to other modeling elements through the derivation of formal requirements. A systems engineer can apply the Property Based Requirements Pattern to a system specification in which formal requirements of a system have been provided in some format by a stakeholder. The system model can be used as a model-based specification which is composed of block instances that own property values to represent the physical properties of a derived requirement. Additionally, SysML constraint blocks can be applied by the modeler to represent engineering equations, and to provide a context for simulation processes.

The general process for the application of a Property Based Requirement to a system specification is the following:

1. A system requirement can be modeled as a formal requirement with text and ID attributes.
2. The textual requirement can be applied in a unique element that inherits the attributes of a block, requirement, and abstract class as a value property.
3. The value property, which are a representation of a textual requirement, can be bound to a constraint parameter of a typed constraint block in a parametric model.
4. The Property Based Requirement can be applied throughout a system's specification decomposition (if needed).
5. The system engineer can constrain the value properties of the Property Based Requirement in a Verification Context to perform a verification and requirement analysis of their system.

2.3 Concept

The structure of the Property Based Requirements Pattern consists of the relationships between the following model elements:

A `cf name([cf:Generic Property Based Requirement .name]) does not exist` that is stereotyped by `<<cf name([cf:library:propertyBasedRequirement.name]) does not exist >>`, a general requirement that is stereotyped by `<<requirement>>`, a specialized element Domain Property Based Requirement that is stereotyped by `<<cf`

name([cf:library:propertyBasedRequirement.name])) does not exist >> , and a Constraint block that types a constraint property of the Domain Property Based Requirement element.

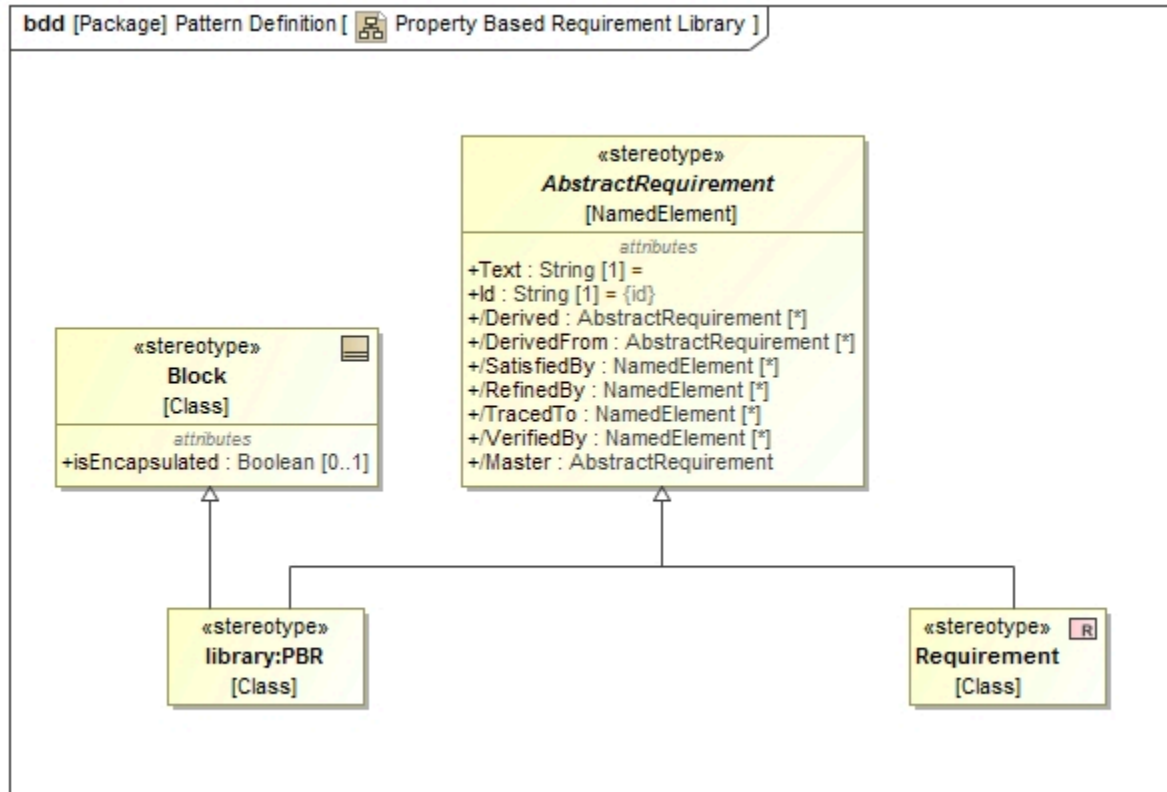


Figure 1. Property Based Requirement Pattern

The pattern for creating a property based requirement in SysML 1.5 is shown above. The **AbstractRequirement** stereotype specializes the **Requirement** stereotype. The usage of both **AbstractRequirement** and **Block** to define a new PBR stereotype name `<<library:PBR>>`.

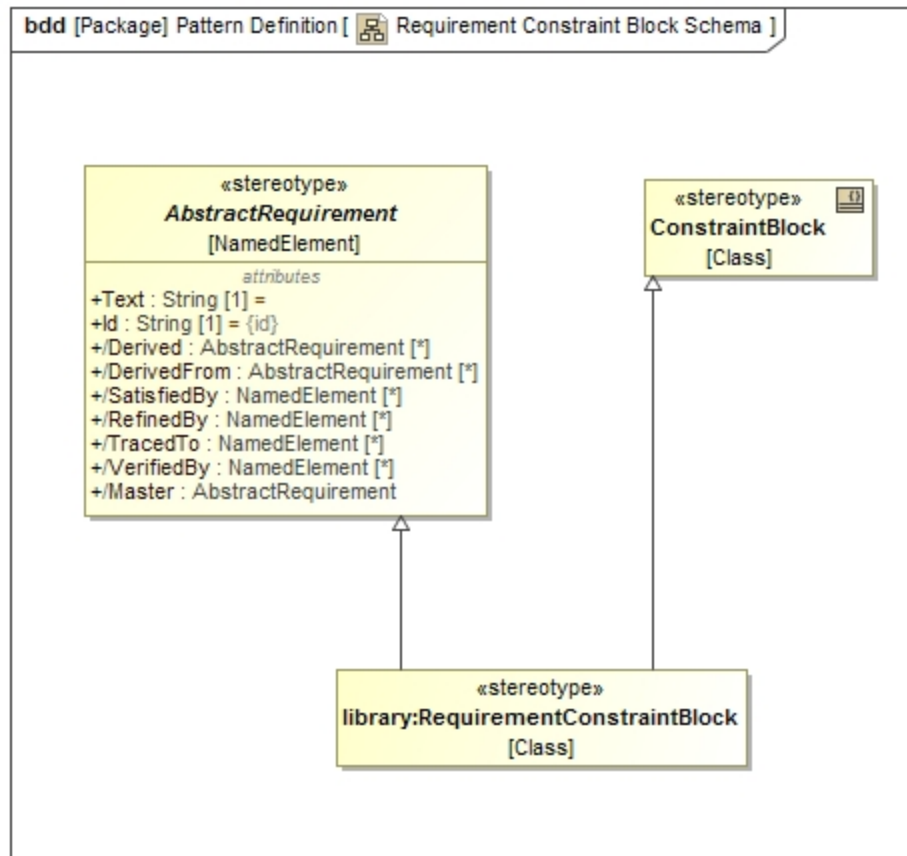


Figure 2. Requirement Constraint Block Pattern

The use of both the AbstractRequirement and ConstraintBlock stereotype to define a new PBR stereotype named RequirementConstraintBlock. By inheriting from the ConstraintBlock, it allows for additional flexibility in expressing the name of required numerical values as ConstraintParameters, which can be typed by ValueTypes and related to properties or parameters of other model elements using binding connectors.

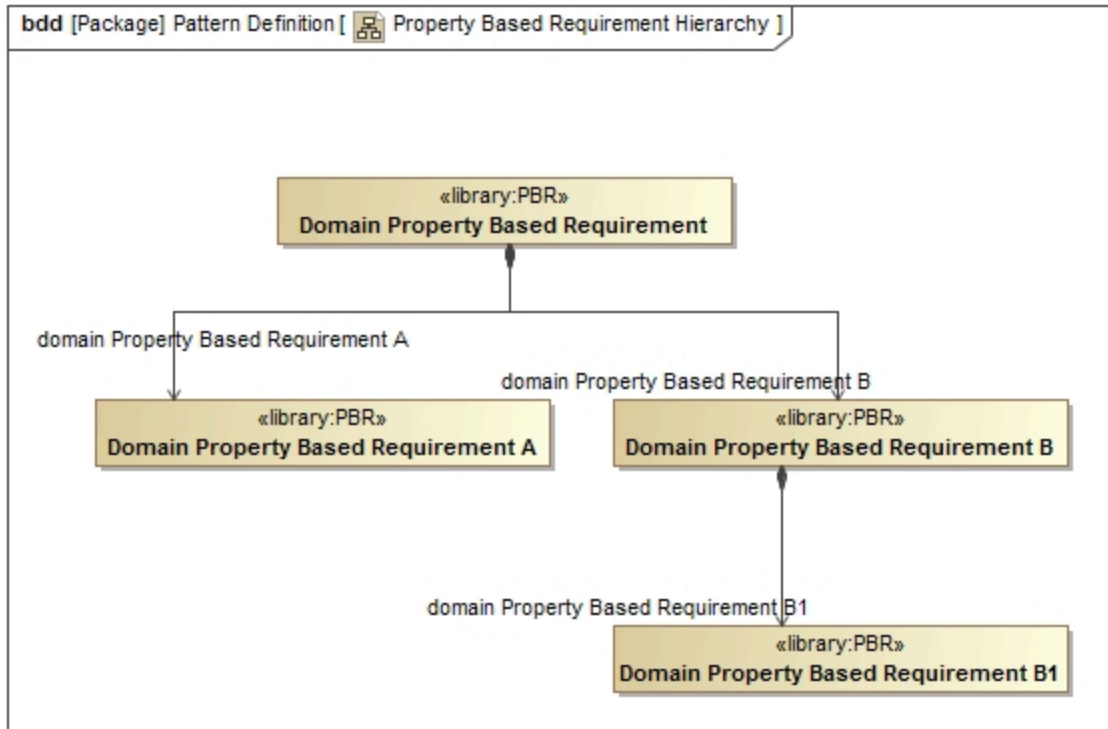


Figure 3. Property Based Requirement Hierarchy

A requirement hierarchy is represented with a containment relation. However, a property based requirement hierarchy would be represented with composition relations.

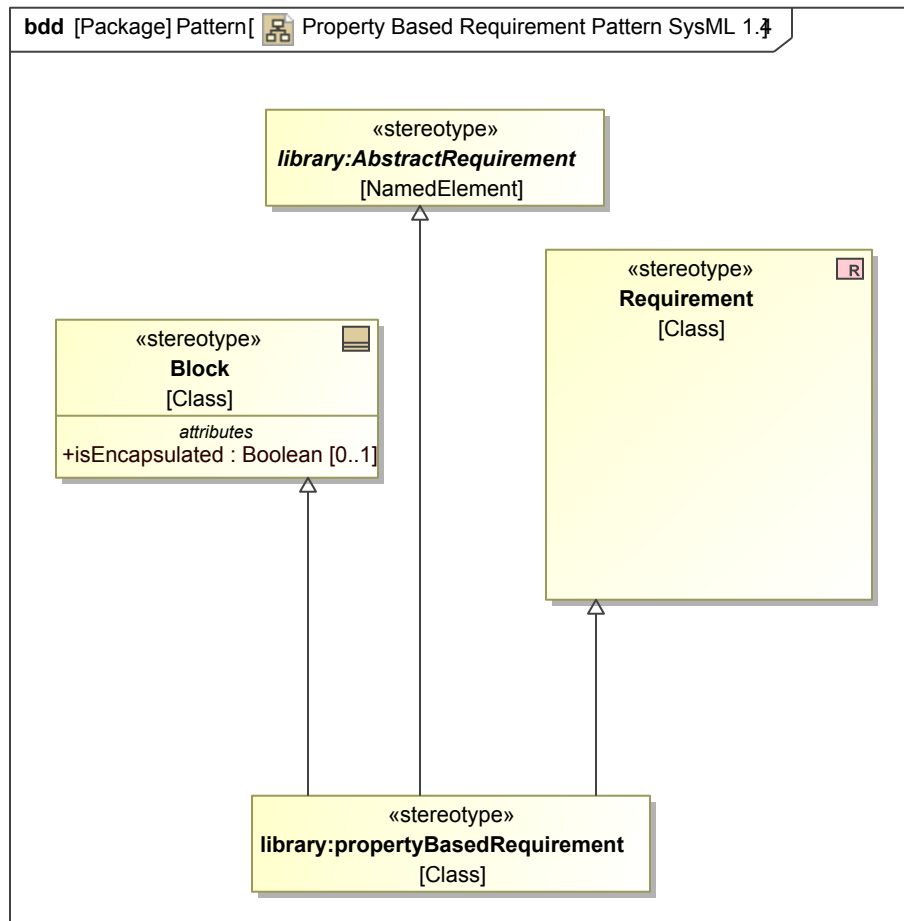


Figure 4. Property Based Requirement Pattern SysML 1.4

The Generic Property Based Requirement is a unique model element that shares the characteristics of a block, a requirement, and an abstract requirement. The resulting element owns a text and ID attribute which is inherited by a standard requirement, and can own value, reference, part, and constraint properties which is inherited by a Block. The Domain Property Based Requirement element was generated using the System Reasoner tool's action "Specialize Structure" on the Generic Property Based Requirement element. The generated element is a block instance that has the capability to own property values that are derived from a formal requirement. The Property Based Requirement allows engineers to model a more formal expression of requirements. The Constraint block constrains the physical property of based on the text of the derived requirement into a mathematical expression. The constraint property of the Domain Property Based Requirement is typed by the Constraint block. In the parametric model of the property based requirement block, Domain Property Based Requirement the value property is constrained to the parameter of the constraint property.

A listing of the model elements and their roles in the Property Based Requirements Pattern.

Table 1. <>

Model Element	
General Requirement	The General Requirement element owns the stereotype <<requirement>>, and consists of a textual representation and a unique id of the requirement. A requirement specifies a condition that must or should be satisfied. The text of a requirement may specify a function, property, or operation that affects the performance of a component. A requirement may specify a function of a system that is to be performed or a performance condition a system must satisfy. The relationship between the Domain Property Based Requirement and the General Requirement is an incoming derive requirement relationship.
Domain Property Based Requirement	The Domain Property Based Requirement element is representative of the domain specific and concrete property based requirement. To create a property based requirement, create an element and stereotype it with <<cf name([cf:pbr.name]) does not exist >>. By applying the stereotype, the Domain Property Based Requirement will have the attributes of a block and requirement.
Generic Property Based Requirement	The Domain Property Based Requirement element has the characteristics of a block, a requirement, and an abstract requirement which results in an element that inherits of each of these components. This is due to the cf name([cf:library:propertyBasedRequirement.name]) does not exist stereotype. The owned attributes of the property based requirement are a unique ID and text. If a user wants to create a property based requirement they can use the Systems Reasoner feature, "Specialize Structure" to create a specialization of the Domain Property Based Requirement .

2.4 Consequences

There are several trade-offs to consider when applying the Property Based Requirements Pattern to the modeler's system.

Table 2. PBR Consequences

Action	Consequence
The user specializes the structure of the Generic Property Based Requirement to create an element that inherits the properties of a requirement, block, and abstract requirement.	The stereotype of the Generic Property Based Requirement element is library:propertyBasedRequirement. The relationships and classes applied to the Property Based Requirement element is based on the SysML 1.4 specification. However, any preexisting Property Based Requirement elements that are created prior to the SysML 1.5 specification release will need to be refactored to meet the new spec. The stereotypes and properties of the element that owns the property based requirement will have new attributes.

Listing of possible conflicts that can occur while applying or using the Property Based Requirements Pattern.

Usage of the Pattern	Explanation of Conflict
----------------------	-------------------------

<p>A user can create their own domain specific property based requirements, by specializing the structure of the Generic Property Based Requirement element.</p>	<p>By performing the Systems Reasoner function "Specialize Structure" of the Generic Property Based Requirement , the new element will own an outgoing generalization relationship to to the Generic Property Based Requirement . One would expect to see the generalization relationship when selecting to "Display Paths". However, the user is unable to visually display the relationship. The only method to denote the relationship between the two elements is by adding a note/comment element and anchoring the elements to another.</p>
<p>A user can attribute their own domain specific property based requirements that were specialized by theGeneric Property Based Requirement element with value properties.</p>	<p>The property based requirement is based on a Block which allows the user to define additional properties like value properties. The user would expect to follow the standard procedure of selecting their Domain Property Based Requirement, select the + symbol, and select value property. However, due to the unique quality of the library:propertyBasedRequirement stereotype the user is unable to follow this procedure. To create a new value property the user must select the Domain Property Based Requirement in the Containment tree and select Create Element "Value Property". The resulting value property can only be viewed in the attribute compartment of the property based requirement. The same procedure should be followed when creating constraint properties, value properties, etc.</p>
<p>A user can attribute their own domain specific property based requirements that were specialized by the Generic Property Based Requirement element with constraints.</p>	<p>The property based requirement is based on a Block which allows the user to define additional properties like constraint properties. Due to this relationship, the user would expect to have the capability to create a composition relationship between a Domain Property Based Requirementelement to a Constraint block. However, altogether the property based requirement is unable to manually create composition relationships. An alternative solution to creating a composition (part-whole) relationship is to define a constraint property in the specification of the Domain Property Based Requirement element. The user can specify a constraint in this specification or they have capability to type the constraint property by a Constraint block. Through the application of this method, the user can specify parameters that were unable to do through a constraint property.</p>

2.5 Implementation

The Property Based Requirements Pattern can be applied to create property based requirements for the system specification of an automated robotic vacuum. During the robot's vacuum operation, the requirements MaximumVacuumTime and MaximumCleaningArea must be met to satisfy the system design.

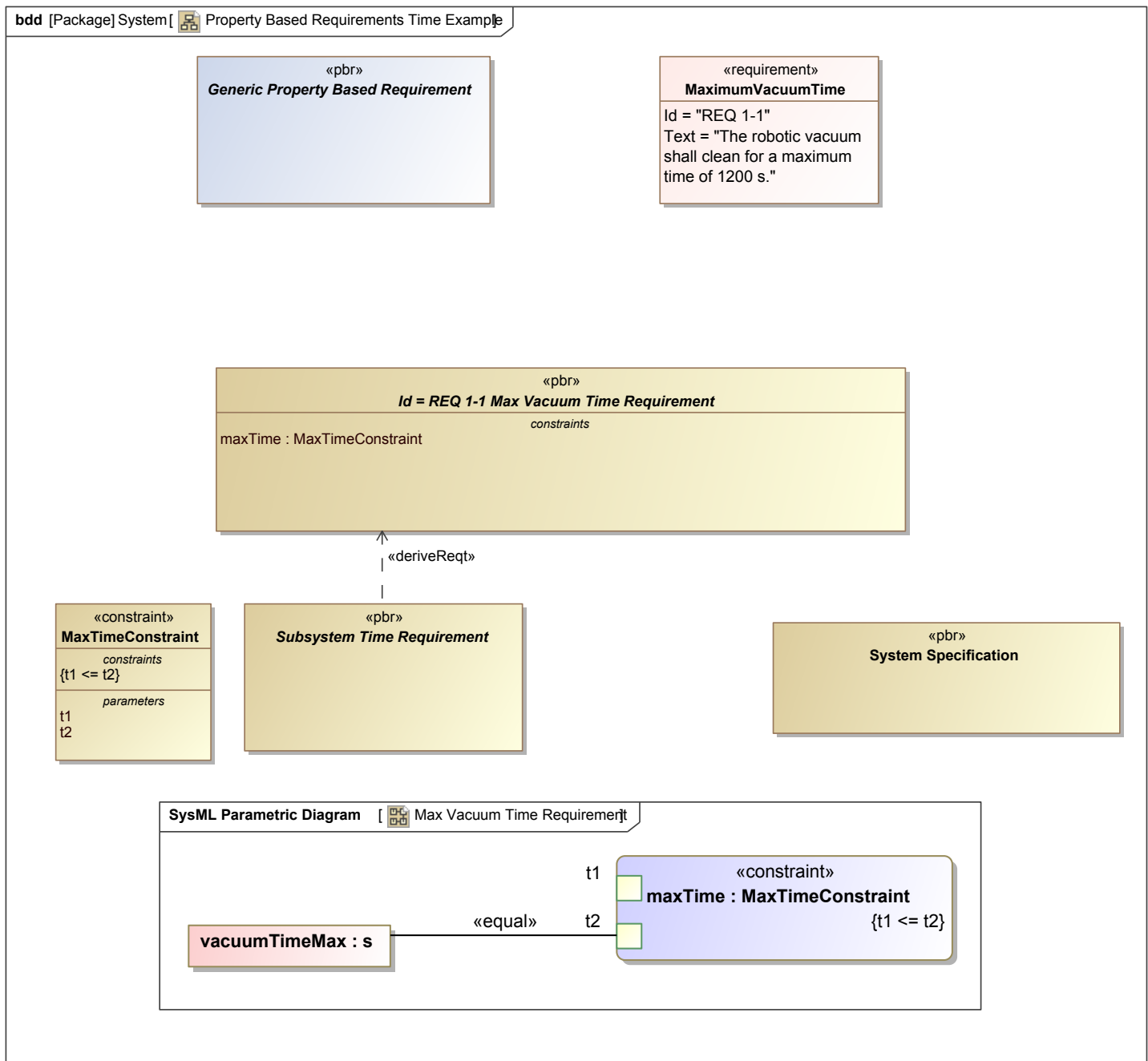


Figure 5. Property Based Requirements Time Example

The derived requirement MaximumCleaningArea requires the robotic vacuum have a vacuum time limit of 1200 seconds. To create a property based requirement that derives MaximumCleaningArea, the user must specialize the structure of the Generic Property Based Requirement element. The resulting generated element Max Vacuum Time Requirement, redefines the ID and text properties with the content from the MaximumCleaningArea requirement. In Max Vacuum Time Requirement, the value property vacuumTimeMax is defined with the value from the requirement ("1200 s"). A constraint property is created, MaxTimeConstraint, to constrain the vacuumTimeMax property as specified in the requirement ("...clean for a maximum time of..."). The property based requirement is a derived requirement of a Subsystem Time Requirement, and a part property of the System Specification of the robotic vacuum.

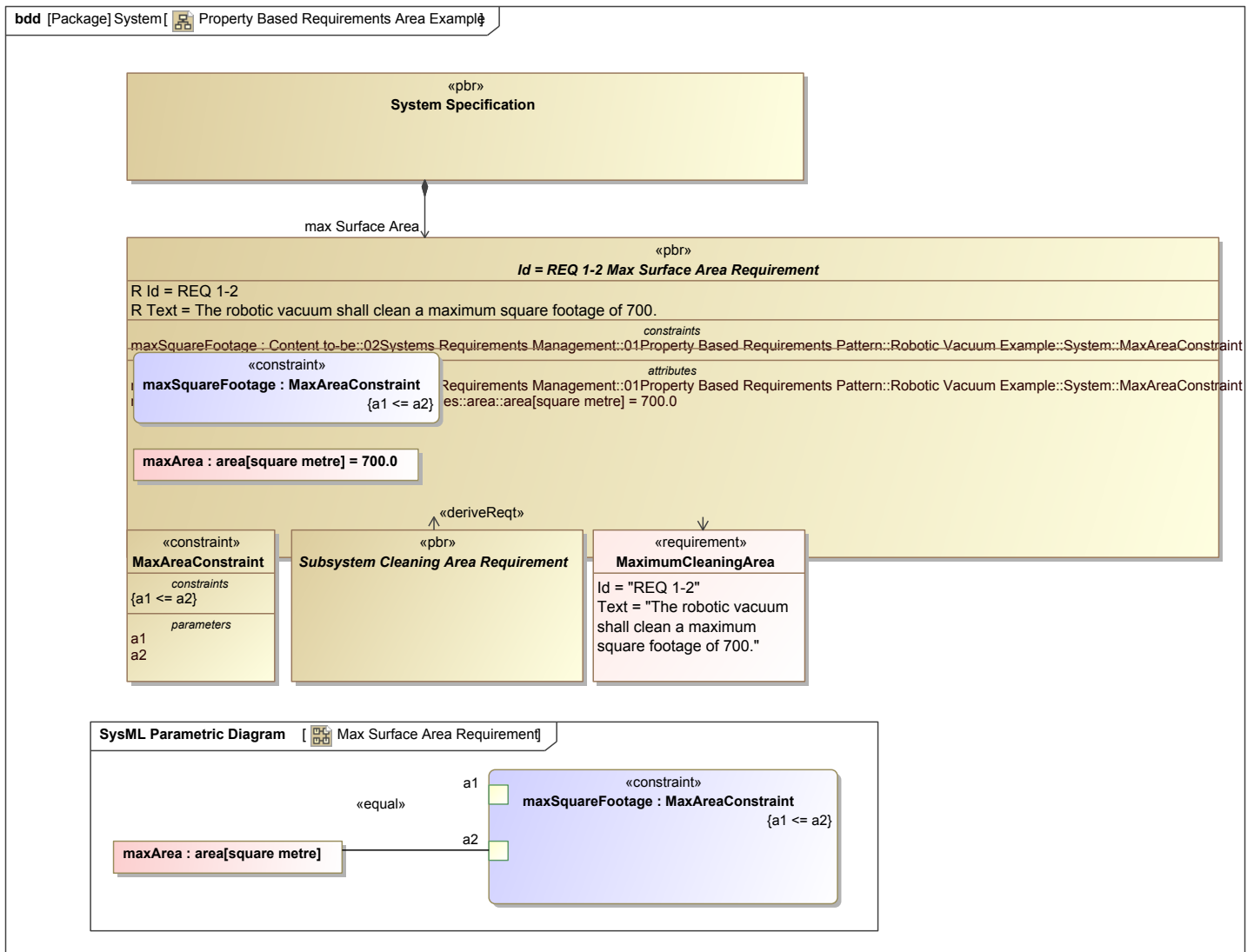


Figure 6. Property Based Requirements Area Example

The derived requirement MaximumCleaningArea requires the robotic vacuum have an area limit of 700 sq ft. To create a property based requirement that derives the MaximumCleaningArea requirement, the user must specialize the structure of the Generic Property Based Requirement element. The resulting generated element Max Surface Area Requirement, redefines the ID and text properties with the content from the MaximumCleaningArea requirement. In Max Surface Area Requirement, the value property maxSquareFootage is defined with the value from the requirement ("square footage of 700"). A constraint property is created, MaxAreaConstraint, to constrain the maxSquareFootage property as specified in the requirement ("...clean for a maximum of..."). The property based requirement is a derived requirement of a Subsystem Cleaning Area Requirement, and a part property of the System Specification of the robotic vacuum.

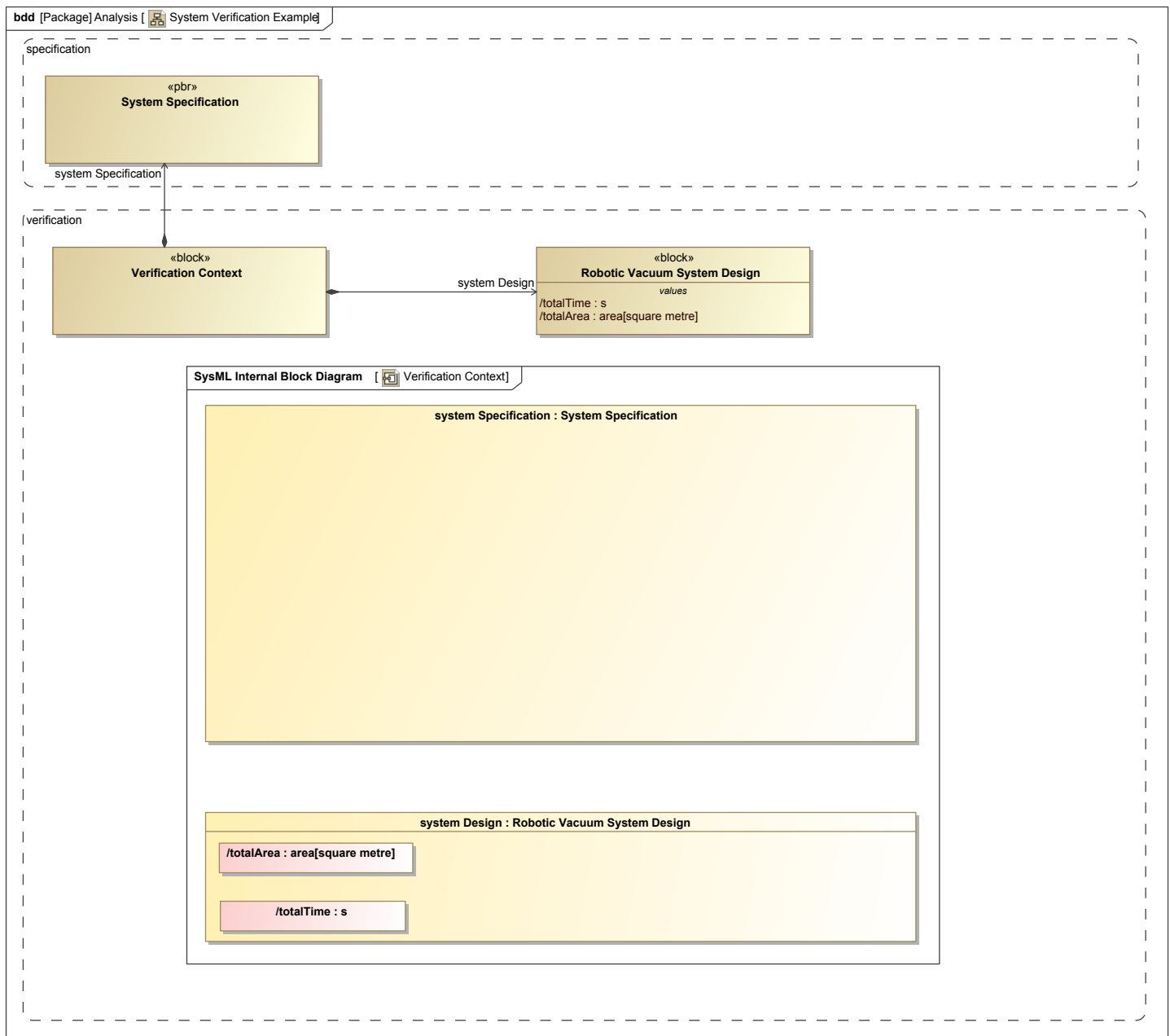


Figure 7. System Verification Example

The requirement is contextualized in the block System Specification. The block Verification Context contextualizes the block Robotic Vacuum System Design which holds the as-designed totalArea and totalTime value properties. In this context the as-designed values are bound to the requirement constraint for the purpose of analysis. The system engineer can verify that the designed value satisfies the required values.

2.6 Known Uses

The Thirty Meter Telescope model, transforms textual requirements into property based requirements in order to perform analysis of the Alignment and Phasing System (APS). A requirement may specify a function that a system must perform or a performance condition that a system must satisfy.

However, note that these "property based requirements" are not stereotyped by `<<cf name([cf:library:propertyBasedRequirement.name]) does not exist >>`, but rather `<<block>>`. Therefore, they do not share the properties, behavior, and functions of a block and a requirement as the `cf name([cf:Generic Property Based Requirement .name]) does not exist` does (which is applied in theProperty Based Requirements Pattern. Although for this known use case, the "property based requirements" serve the same function in the impression that they represent a requirement that is defined in a formal requirement. Since the "property based requirements" are owned by blocks, the relationship between a formal requirement is different.

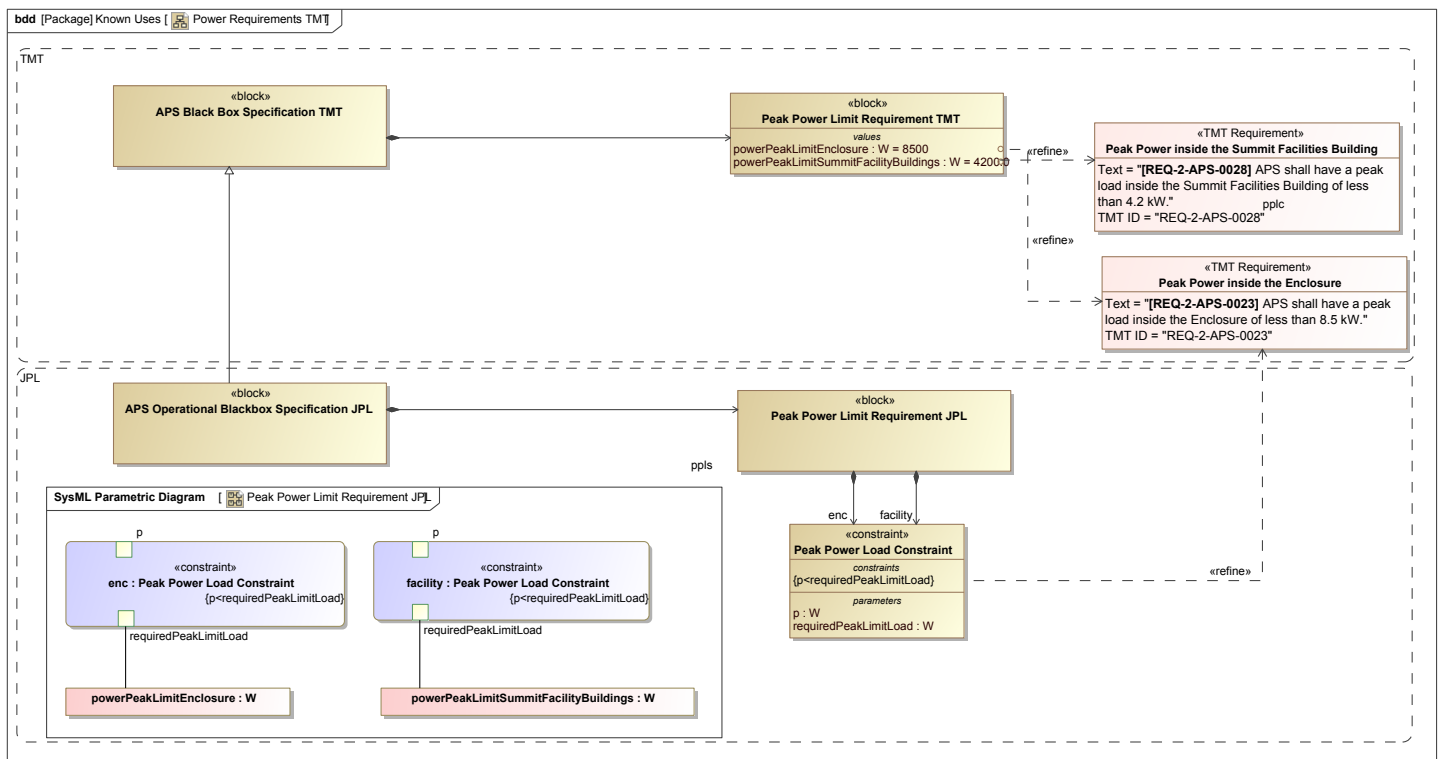


Figure 8. Power Requirements TMT

In the APS Black Box Specification TMT, the Thirty Meter Telescope defines a design black box specification where the customer can provide a comprehensive outline of the requirements of the system.

The formal requirements that were specified by the TMT are "Peak Power inside the Summit Facilities Building" and "Peak Power inside the Enclosure". These formal requirements are represented as "property based requirements" in the block Peak Power Limit Requirement TMT. Since, the Peak Power Limit Requirement TMT is of block stereotype, the element can't form a derivereq relationship that is shown in this pattern. The value properties, value properties powerPeakLimitEnclosure and powerPeakLimitSummitFacilityBuildings, of the Peak Power Limit Requirement TMT block were formulated based on the formal textual requirements.

Through the generalization relationship between the Peak Power Limit Requirement TMT and the Peak Power Limit Requirement JPL blocks, the Peak Power Limit Requirement JPL block will inherit the powerPeakLimitEnclosure and powerPeakLimitSummitFacilityBuildings value properties. Since the JPL black box specification can redefine the TMT's black box specification for tighter margins, the value properties of the Peak Power Limit Requirement JPL block are also redefined.

Similar to the Property Based Requirements Pattern, the "property based requirements" of the TMT model, Peak Power Limit Requirement TMT and Peak Power Limit Requirement JPL, own constraint properties. The Peak Power Load Constraint refines the "Peak Power inside the Enclosure", and is used for constraining the physical properties of the specialized property based requirement, Peak Power Limit Requirement JPL. The Constraint block represents the text of the derived requirement, General Requirement, into a mathematical expression $\{p < \text{requiredPeakLimitLoad}\}$.

2.7 Tooling

The requirements modeling constructs are intended to provide a bridge between traditional requirements management tools and the other SysML models.

2.7.1 Model Construction

The Systems Reasoner tool is used for creating structural elements that inherit properties and aspects from a general element. The new elements will share the same structure and behavior, and its owned properties become "redefined". In the Property Based Requirements Pattern, the Systems Reasoner tool is used to create domain specific property based requirements.

The procedure to create a property based requirement is the following:

1. Right click the cf name([cf:Generic Property Based Requirement .name]) does not exist element that is located in the cf name([cf:OpenSE Common Library.name]) does not exist package, and select Specialize Structure
2. In the specification window, select the owning package for the generated element.
3. Navigate to the previously selected package to locate the generated property based requirement.
4. Verify in the specification window the generalization relationship to the cf name([cf:Generic Property Based Requirement .name]) does not exist and the redefined properties ID and text.

See the [Model Construction](#) documentation for more details.

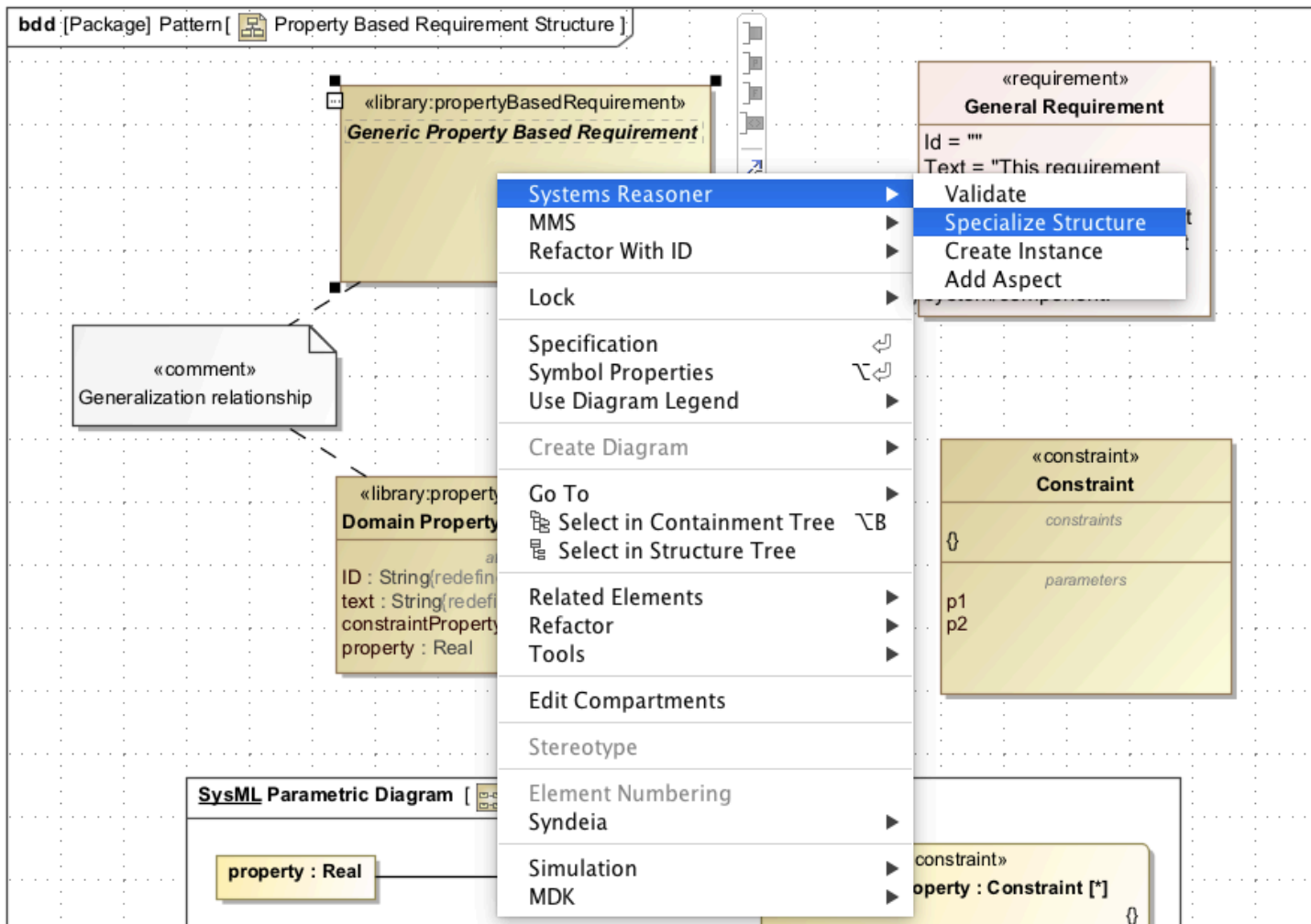


Figure 9. Systems Reasoner

2.8 Related Patterns

List of other patterns that are related to the Property Based Requirements Pattern, and the differences and similarities between them.

Table 3. Related Patterns

Pattern Name	Similarities	Differences
Customer/Supplier Pattern	The Property Based Requirements Pattern can be applied in the application of the Customer/Supplier Pattern to describe textual requirements in SysML properties. Requirements are used to establish a contract between the customer (or other stakeholder) and those responsible for designing and implementing the system. 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.	The purpose of the Property Based Requirements Pattern, is to describe the transformation of textual requirements to a model based fashion through SysML properties. The purpose of the Customer/Supplier Pattern is to provide a context for the exchange of system knowledge from customer to supplier.

Black Box Specification with Property Based Requirements Pattern		
---	--	--

3 Requirement Verification Pattern

3.1 Intent

The intent of the Requirement Verification Pattern is to provide a standard practice for requirements modeling and model checking as methods for requirements verification. Verification is the procedure of establishing a truth of the correspondence between a software product and its specification. Requirement verification is the process of verifying the correspondence between the systems engineering product and its requirement specification. Informally, Verification can be defined through the question- "Am I building the product right?"

Verification tools are used to verify requirements are satisfied at each level of the system hierarchy throughout the lifecycle of system development. Verification tools address a breadth of requirements that can include functional, interface, performance, and physical requirements. Tools can have the capability to support different methods of verification, such as inspection, analysis, demonstration, and test.

3.2 Motivation

Verification ensures that a system's elements and interfaces conform to its requirements, and ensures "you built it right". Verification encompasses the tasks, actions, and activities performed to evaluate the progress and effectiveness of the evolving system solution and to measure compliance with requirements. The motivation is to completely verify the system's capability to meet all requirements prior to the operational stage of the program's life-cycle.

Textual requirements of the system are formalized in SysML as Property-Based Requirements. Through Property Based Requirements, system engineers are enabled a methodology to relate requirements to design elements, and to perform automatic design evaluations to verify requirements. The conceptual and/or realization design can automatically be verified against the changed requirement, resulting in a pass or fail.

The general workflow for the automated requirements verification described in this pattern is the following: the formalization of requirements (as property based requirements), defining an Analysis Context, connecting system value properties to constraint parameters, evaluating system configurations by analysis and testing, verifying the system requirements, and capturing the verification test results.

The primary verification method that is applied in the Requirement Verification Pattern is analysis; the use of analytical data or simulations under defined conditions to show theoretical compliance. Analysis (including simulation) is used where verifying to realistic conditions cannot be achieved and when such means establish that the appropriate requirement, specification, or derived requirement is met by the proposed solution. Requirements verification is an important analysis commonly performed in the context of MBSE. To perform this analysis, the requirements, executable behavior, and models predicting the performance of the specified system must be integrated. In ESEM, this is done using standard SysML modeling constructs.

Requirement verification can be done through the means of a verification plan. A verification plan consists of defining the verification owner whose responsible for conducting the verification activities, verification method (Test, Demonstration, Analysis, and Inspection), and verification level for each requirement. The primary objective of verification planning is to define how each requirement will be verified to ensure requirements are verifiable early in the project's life-cycle. Typically, verification begins at the lowest level of the requirement hierarchy to which they have been decomposed.

After the engineer has formalized the requirements of the system as property based requirements an Analysis Context should be created. The context is applied to specify the structure, apply the usage of constraint blocks to evaluate numerical characteristics of the system, to store test results and other data, and to avoid any impact to the system. The Analysis Context is defined in a BDD and applied in a parametric diagram where the value properties of a system are binded to constraint block parameters for mathematical evaluation.

The system value properties are binded to constraint parameters through the application of parametric diagrams. Parametric diagrams are used to create systems of equations that can constrain properties of blocks. The diagram connects these value properties to the parameters of a constraint block using binding connectors. A boolean result of the constraint expression can be treated as a verdict to indicate whether a requirement test has passed or failed. The engineer can bind the formalized requirement to value properties of the system under test.

System engineers can evaluate system configurations through testing. Parametric models are used as a standard-based tool to enable automatic requirements verification in a defined analysis context.

Values needed for the evaluation of the constraint can be passed in a form of an instance specification of the system under test or inputted during run-time. Every single instance specification can be specified with different set of input parameters to represent variants of the system under test.

Requirement verification analysis can be performed in a simulation environment which allows users to observe how the change of an input influences the result of verification by automatically highlighting requirements that fail (based on the set-up traceability), and allows the user to view the text of the failing requirement.

Lastly, the results of the requirement verification simulation can be stored to the model in the form of the instance specifications. The instance specifications can then be represented in the tabular format (Instance Table).

3.3 Concept

To verify whether a system satisfies a requirement, the user should specify and analyze the designs and components of the Enterprise's System Specification. In the System Specification, a requirement (modeled as a property based requirement) can be specified as a part property. The properties of the requirement will be verified through simulation of either the Conceptual Node Design, Physical Design, or Conceptual Design's components.

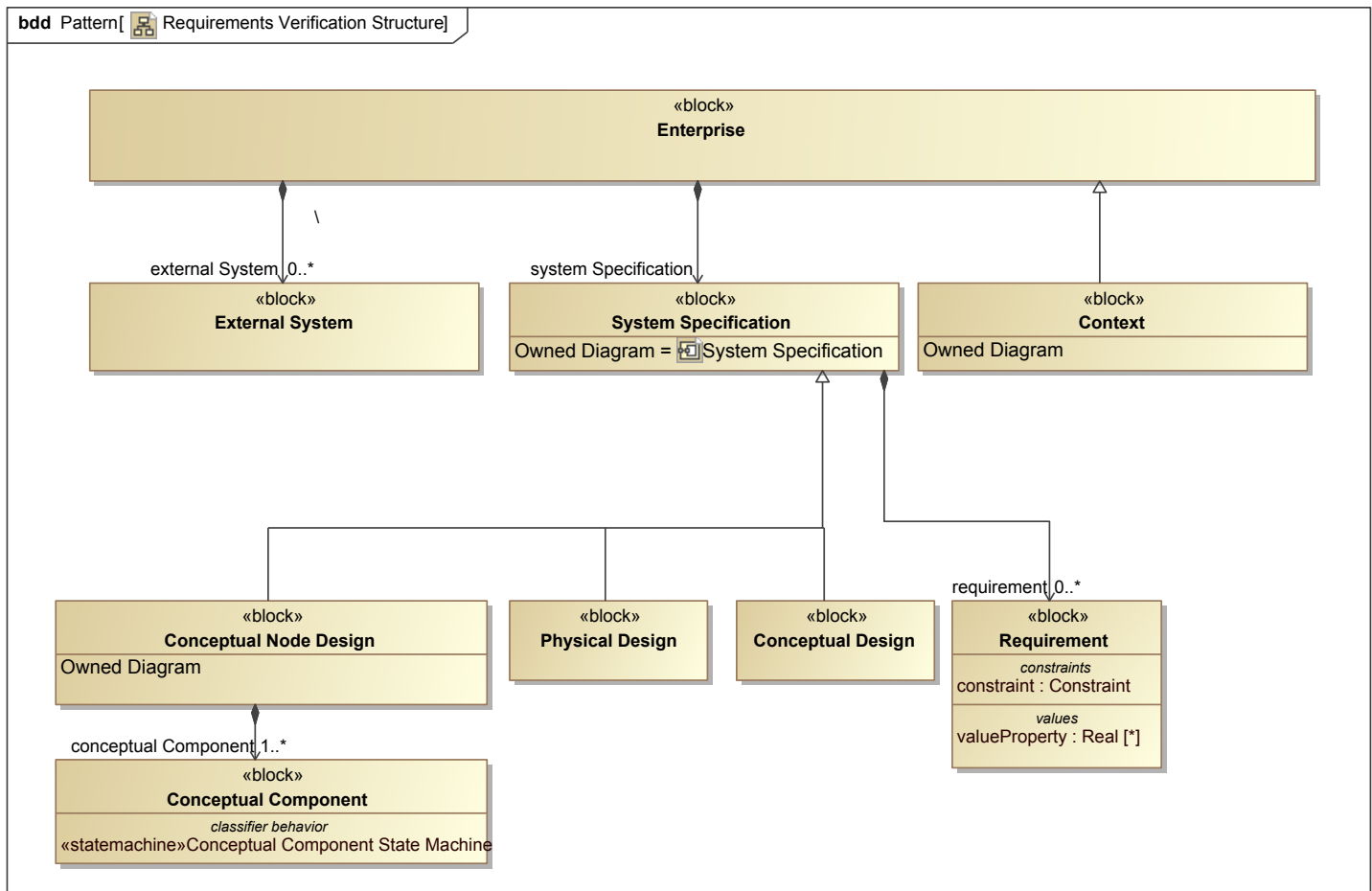


Figure 10. Requirements Verification Structure

The system engineer would specify the properties and behavior of the components that would address the System's Requirement. The Conceptual Node Design specifies functional, technology-independent system components and captures their specified behavior. This part of the model is used to analyze characteristics such as duration of operational scenarios. The behavior of various components can be captured using state machines and activity diagrams. The communication across system components and with components external to the system-of-interest is completed through signals over SysML ports. These SysML ports can be typed by system specific interface blocks where a flow property can be specified.

In order to contextualize a concrete conceptual, physical, or conceptual node design in the context of the Enterprise the Context block is applied. The Context allows for system engineers to carry out analysis of the concrete designs in the context of the Enterprise to verify if components of the design satisfy the system's specification.

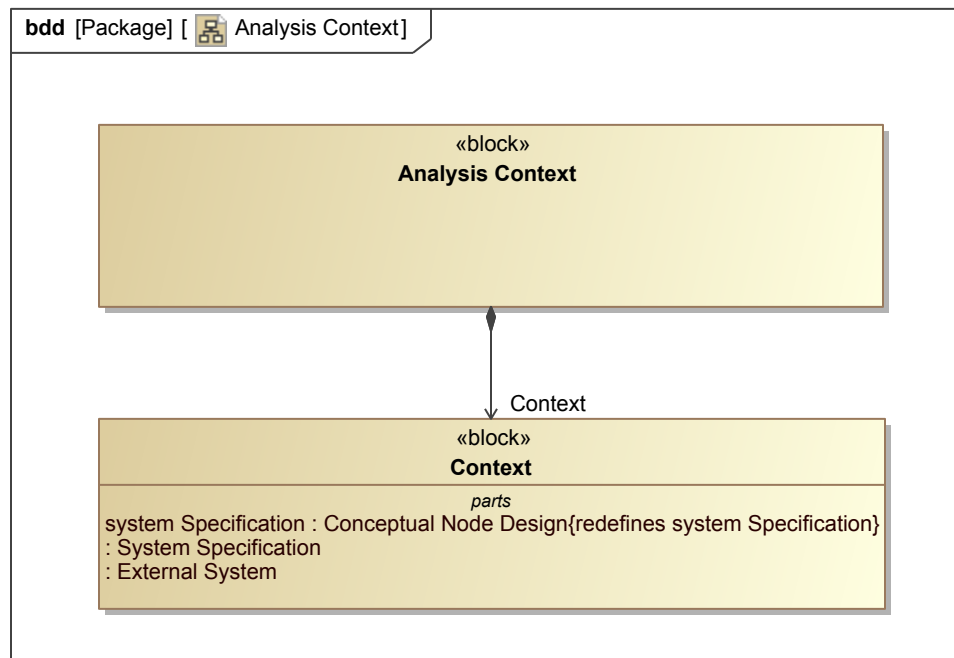


Figure 11. Analysis Context

The Analysis Context block is applied to specify the structure, apply the usage of constraint blocks to evaluate numerical characteristics of the system, to store test results, and to avoid any impact to the system. The Analysis Context block has a composition relationship to the Context block because the element is applied to contextualize the Enterprise block. To perform an automated simulation of the system, the modeler could specify a parametric diagram to constrain system properties, a sequence diagram, and set the execution target property of a simulation configuration in the Analysis Context block.

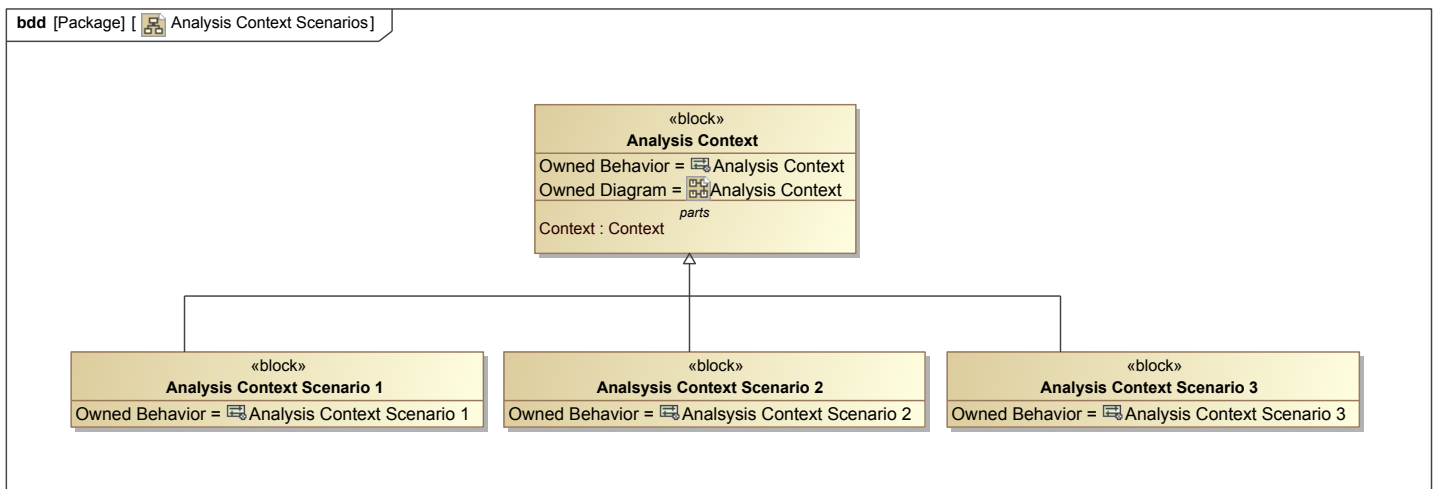


Figure 12. Analysis Context Scenarios

If the modeler has multiple verification scenarios they should model them as specializations of the Analysis Context block. The Analysis Context Scenarios would inherit the properties of the Analysis Context, but allows the modeler to create a sequence diagram for each specific scenario. In the sequence diagram the modeler can specify lifelines, state invariants, messages, and constraints.

A listing of the model elements and their roles in the Requirement Verification Pattern.

Table 4. <>

Model Element	
Conceptual Node Design	The Conceptual Node Design follows the OOSEM pattern, and represents the conceptual design of the system specification. The element is composed of a conceptual component(s). The internal structure of the Conceptual Node Design can be captured in an IBD, to display the elements inner elements and parts. For requirement verification, the communication among the parts of the Conceptual Node Design can interact through ports and connectors to the Enterprise.
Conceptual Component	The Conceptual Component element represents an element which is specified in the System Specification. The element is not included in the Enterprise of the system as opposed to the External System Element. The structure, function, and behavior of the Conceptual Component needs to be defined the systems engineer.
System Specification	The System Specification is a part of the Enterprise, and represents the functional specification of a system design. Depending on the system design, the System Specification can generalize a Conceptual Design, Physical Design, and a Conceptual Node Design. Property Based Requirements can be specified in the System Specification through a composition relationship. In the Context block, the System Specification block is inherited and redefined.
Conceptual Design	The Conceptual Design block specializes the System Specification. If the system engineer wanted to verify a property of the Conceptual Design, this can be done by redefining the System Specification and setting the type to the Conceptual Design block.
Physical Design	The Physical Design block specializes the System Specification. If the system engineer wanted to verify a property of the Physical Design, this can be done by redefining the System Specification and setting the type to the Physical Design block.
Context	The Context model element contextualizes a concrete conceptual, physical, or conceptual node design in the context of the Enterprise. The purpose of the element is to carry out analysis of the concrete designs in the context of the Enterprise. The Enterprise is composed of a System Specification which generalizes the following components: Conceptual Node Design, Physical Design, and Conceptual Design. The element is able to provide this context by redefining the System Specification part property, and changing the type. The type of the part property should be changed to the design element (Conceptual Node Design, Physical Design, Conceptual Design). The Context will then inherit the parts System Specification and External System.
Enterprise	According to the OOSEM, in the Enterprise, analysis is performed to assess capabilities and limitations, and to identify potential improvements. In this pattern the Enterprise is composed of an External System, the System Specification, and a Context.
External System	According to the OOSEM, the External System, is a part of the Enterprise and is specified by the stakeholders of the Enterprise. The function, structure, and behavior of the element is assumed. The Conceptual Components verify requirements the user must specify
Requirement	The Requirement of the system that is to be verified is specified in the System Specification. The Requirement is modeled as a Property Based Requirement that is to be verified is modeled as part of the System Specification.
Analysis Context	After the engineer has formalized the requirements of the system as property based requirements an Analysis Context should be created. The context is applied to specify the structure, apply the usage of constraint blocks to evaluate numerical characteristics of the system, to store test results and other data, and to avoid any impact to the system. The Analysis Context is defined in a BDD and applied in a parametric diagram where the value properties of a system are binded to constraint block parameters for mathematical evaluation.

3.4 Consequences

There are several trade-offs to consider when applying the Requirement Verification Pattern to the modeler's system.

Table 5. Consequences Table

Action	Trade-Off
--------	-----------

Specifying the values of properties through redefinitions before runtime instead of creating instances for components of the system.	By redefining the values of the system's components the modeler can easily change attributes by redefining and modifying the values. However, one could argue that redefinition is the proper approach to specifying component's attributes because the modeler can view and modify the inherited aggregated value properties of it's classifier. These modifications can be done through instance tables and parametric diagrams.
--	--

Listing of possible conflicts that can occur while applying or using the Requirement Verification Pattern.

Usage of the Pattern	Explanation of Conflict

3.5 Implementation

The following sample model serves to provide an example of how to verify a requirement of a system by applying it to a Autonomous Ferry Transportation Enterprise. The Autonomous Ferry Transportation Enterprise consists of a System Specification and a Satellite . In the System Specification, an Autonomous Ferry and Remote Navigation Control are specified and interact with the Satellite. Companies can send location and navigation directions remotely through a Remote Navigation Control to a Satellite which will relay these messages to an Autonomous Ferry. The Autonomous Ferry can then travel to a precise location to perform an operation, experiment, delivery, etc. The Autonomous Ferry is electric powered and will need to recharge it's batteries due to the communication and flow of data between the Autonomous Ferry and the Satellite. Therefore the main operating states of the Autonomous Ferry are transmitting data (the location of the Autonomous Ferry) and recharging the batteries from a certain battery level to a full battery level. The vessel features an all-electric powertrain, lithium-ion battery packs, and an electric propulsion system. The Autonomous Ferry runs on renewable energy, and can recharge the lithium-ion battery pack while transporting through solar panels until it has reached a pier.

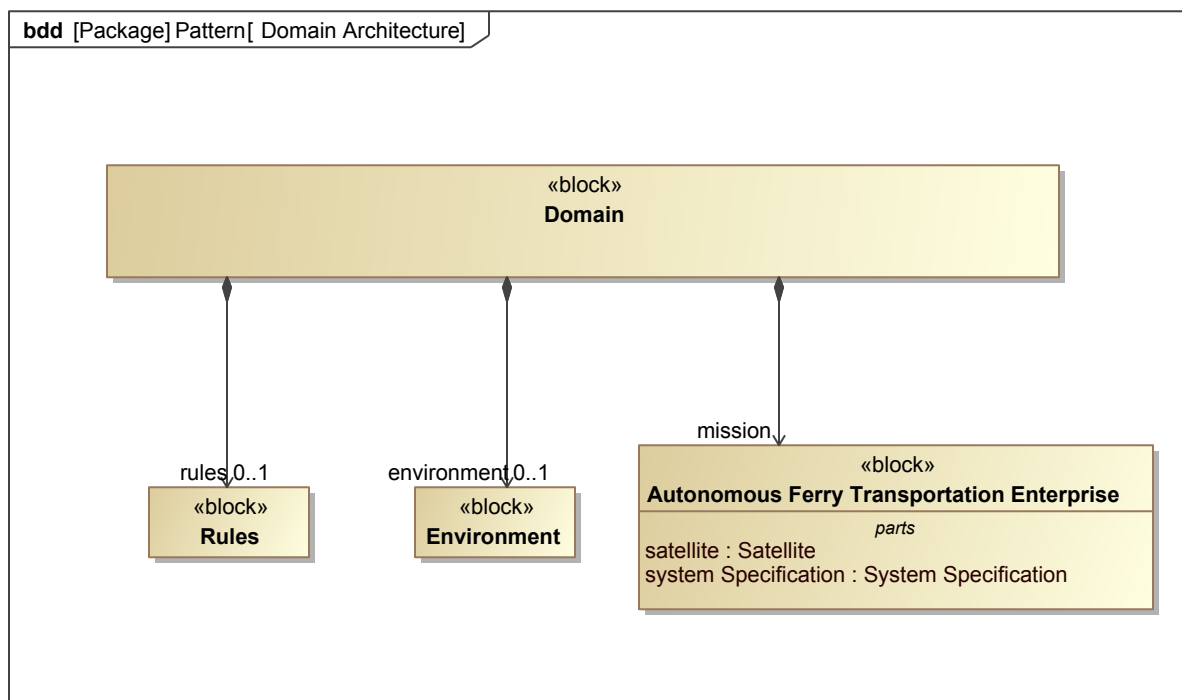


Figure 13. Domain Architecture

In the domain element of a system domain model helps establish the boundary between the system of interest and the external systems and users that the system either directly or indirectly interacts with. The sample Domain is composed of a Autonomous Ferry Transportation Enterprise, Environment, and Rules. The focus of this example is to verify a requirement of the Enterprise.

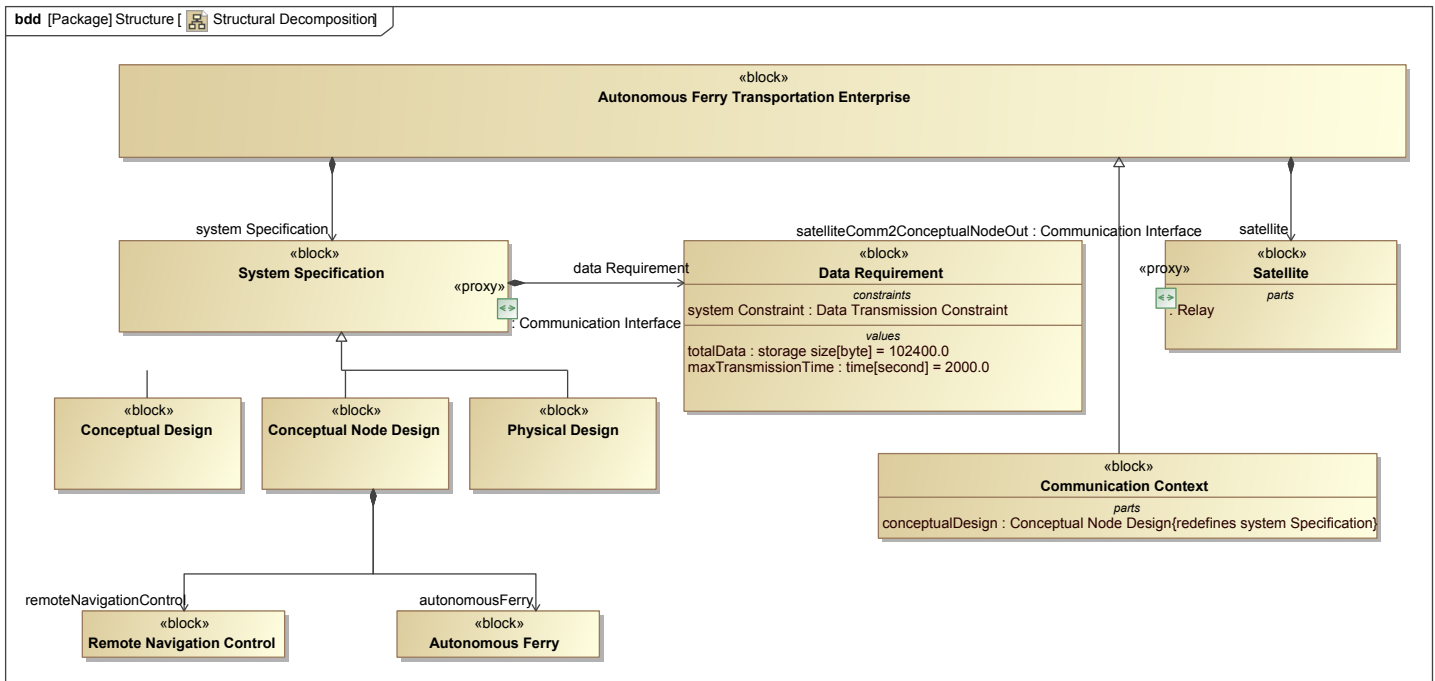


Figure 14. Structural Decomposition

The Solar Powered Vehicle Enterprise is composed of a System Specification and a Home Charging Base. In the System Specification a Conceptual Design and Physical Design can be specified. However, for this example we want to compare the behavior and properties of an Autonomous Electric Vehicle and a Solar Panel as they communicate with the Home Charging Base. After analysis, it can be determined whether the Power Requirement is satisfied. Any requirements of the system are modeled as part properties of the System Specification. The Power Requirement is a type of "Property Based Requirement", and is constrained with the constraint block Data Transmission Constraint in order to compare the designed time with the maximum charging time. The Communication Context block generalizes the Solar Powered Vehicle Enterprise which will result in the Communication Context inheriting the all the parts, properties, and behavior of the Enterprise.

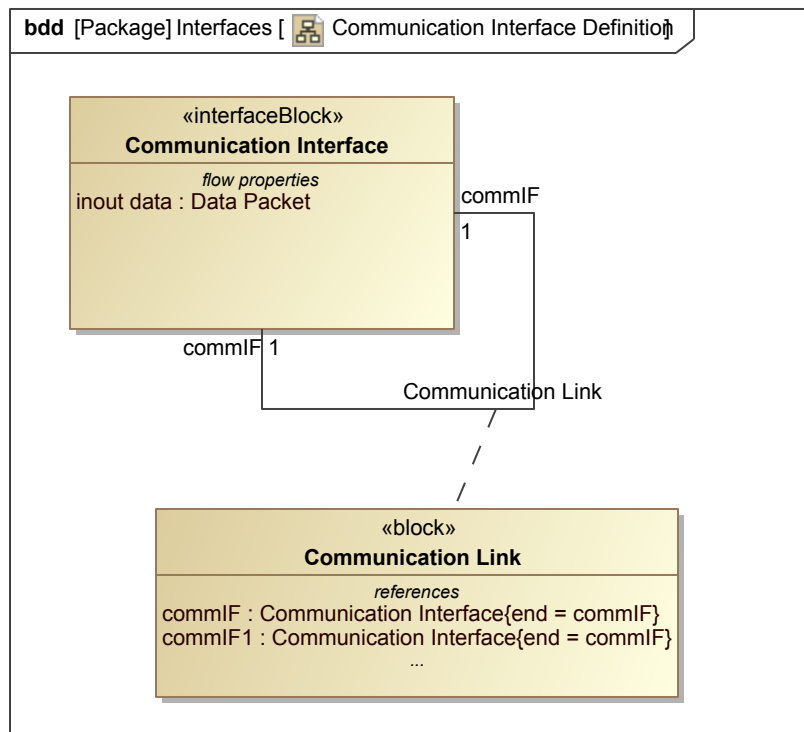


Figure 15. Communication Interface Definition

A proxy port is a port that specifies features of owning blocks or internal parts that are available to external blocks through external connectors to the port. The port can be typed by the Communication Interface block with the flow property data. The flow property signifies power can flow to and from a block. The flow property's values are either received from or transmitted to an external block.

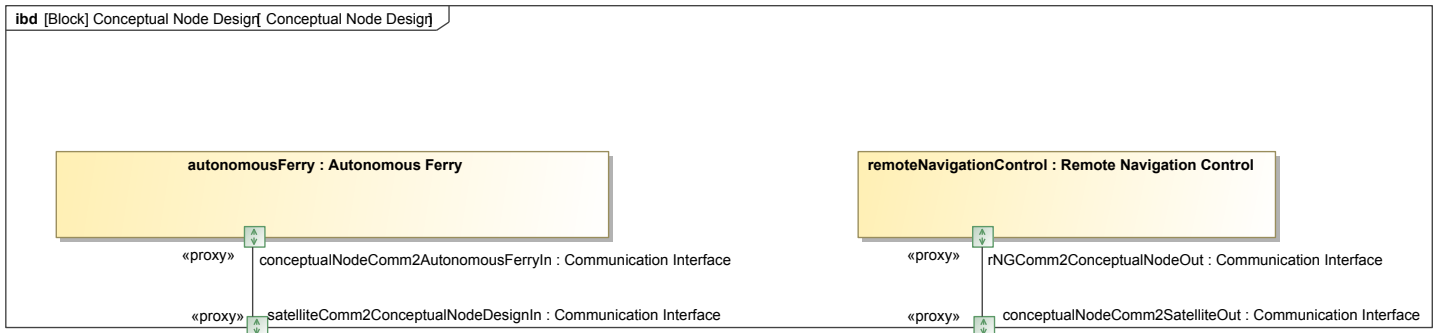


Figure 16. Conceptual Node Design

The Autonomous Ferry interacts with the Home Charging Base for power through the proxy ports typed by Communication Interface. In the Conceptual Node Design, the components (Autonomous Ferry and Remote Navigation Control) connect to the Conceptual Node Design block in order to communicate with the Remote Navigation Control.

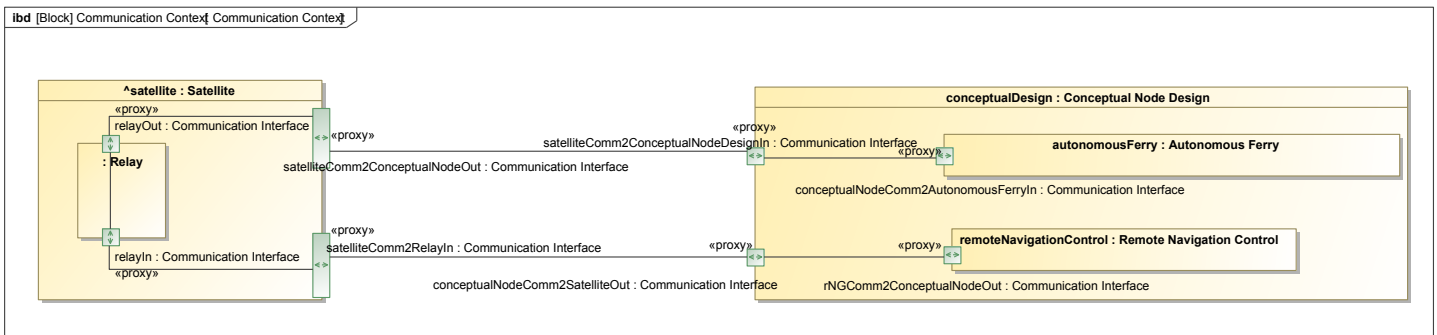


Figure 17. Communication Context

The power flow property from the proxy ports flow from the Conceptual Node Design to the ports of the Home Charging Base. To enable the Solar Panel and the Autonomous Electric Vehicle to communicate with another a connector part is used.

The behavior of the Autonomous Ferry block is specified through the application of state machines and activities.

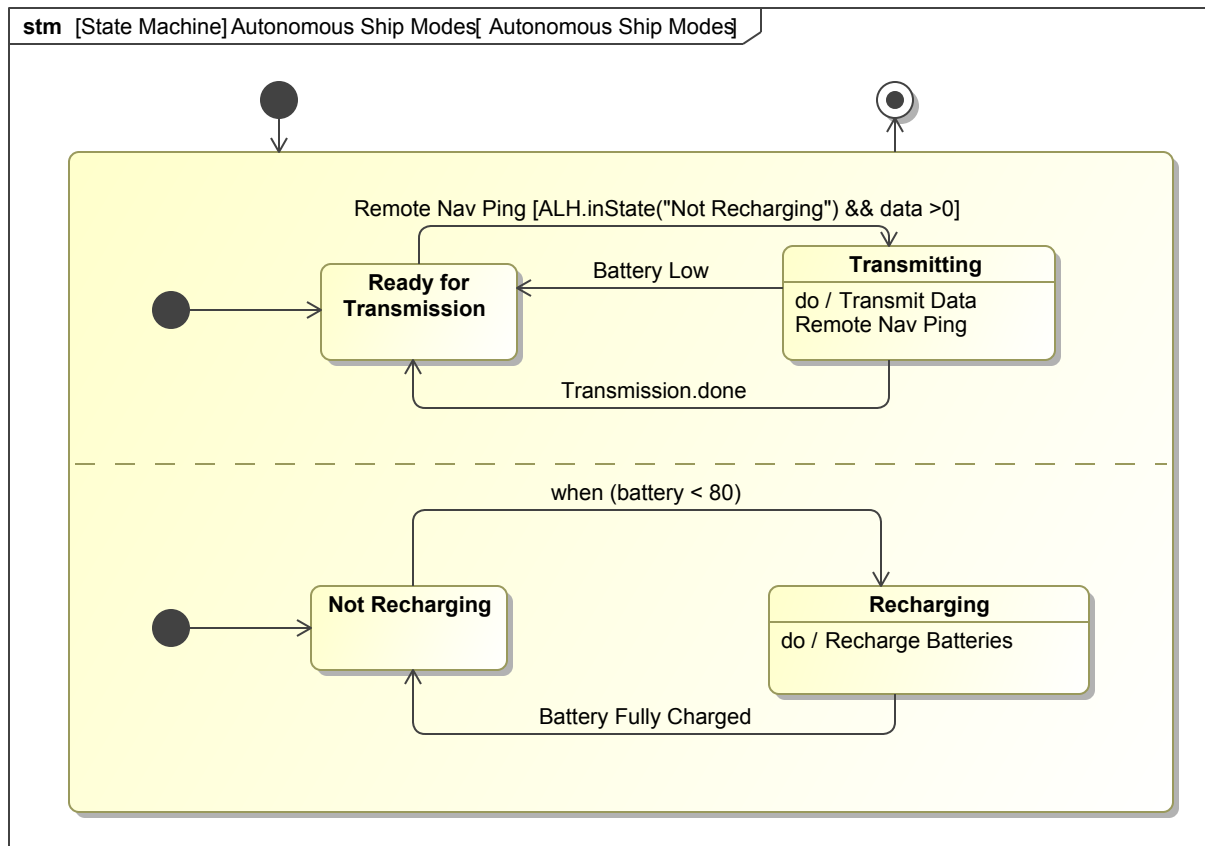


Figure 18. Autonomous Ship Modes

The behavior of the Autonomous Ferry is modeled through a combination of state machines and activity modeling. The modes of the Autonomous Ferry are modeled within a composite state with two regions. Each region has its own active states that are independent of the others, and any incoming event is concurrently consumed within each region. A completion event for the composite state will occur when all the regions are in their final state. The behavior of the Autonomous Ferry's charging mode cycles between Charging and Not Charging. When the value property battery is less than 80 there will be a transition from the Not Recharging state to the Recharging state. In the Recharging state, a do behavior of type activity, Recharge Batteries, is specified. The do behavior will continue to execute until it has been completed or the state is exited.

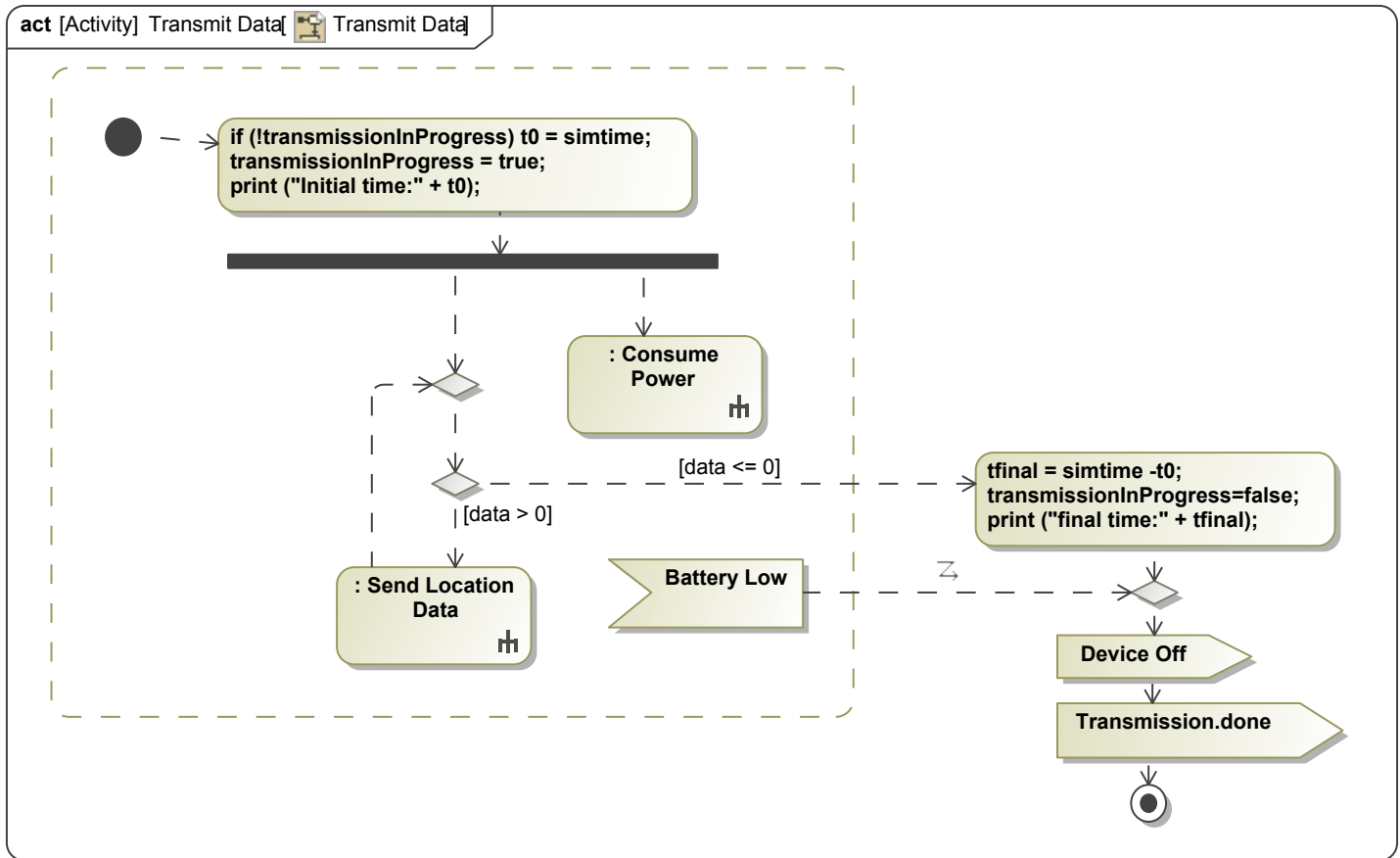


Figure 19. Transmit Data

In the do behavior of the Transmitting action the activity, Transmit Data, will be performed. During this activity the Autonomous Ferry will Consume Power and depending on the data property will either Send Location Data or signal to turn the device off. An interruptible region is used because only a subset of the action executions should be terminated. This means the Consume Power action, Send Location Data action, and the send signal action Battery Low will be terminated before transitioning to the opaque action and signaling to turn the Device Off. The opaque action is used to retrieve the time at a certain event in order to calculate the overall time between two events (start and end of a sequence). Opaque actions are not specified by the standard which allows users to specify a language and pass an argument. In the body of the opaque action, the variables refer to value properties of the Autonomous Ferry (i.e. `tfinal`, `t0`, `transmissionInProgress`). Transition guards are specified with expressions that must evaluate to true for the transition to occur. When an event satisfies a trigger, the guard on the transition is evaluated. Initially the Autonomous Ferry's data is greater than 0 so the guard on the transition will be satisfied and continue to the Send Location Data action. This cycle will continue until the data is less than or equal to zero in which it will then transition to the opaque behavior. In the opaque behavior the final time `tfinal` is calculated, the value property `transmissionInProgress` is set equal to false, and transitions to the signal Device Off.

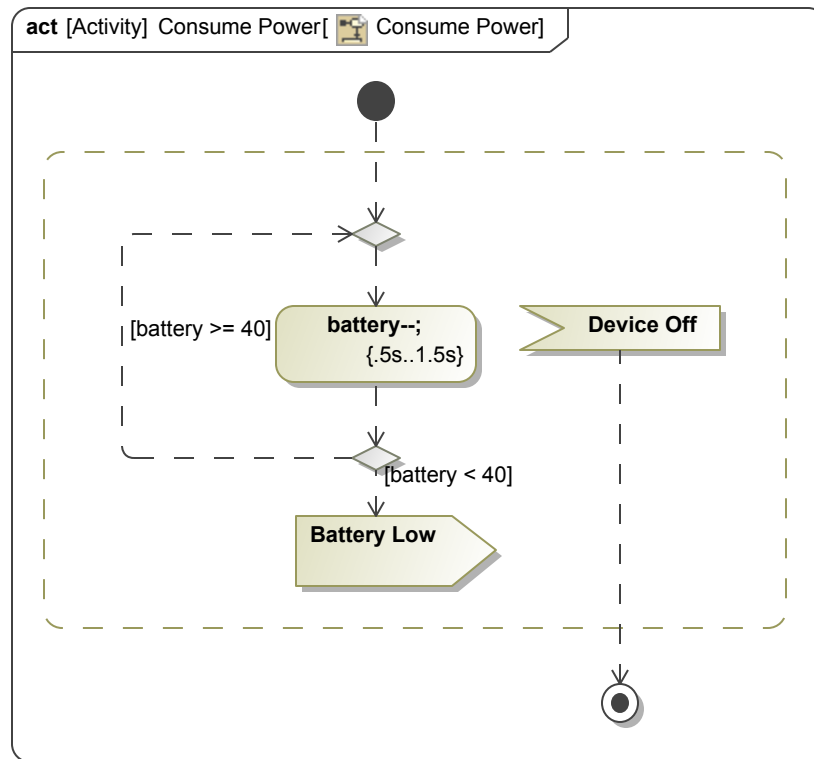


Figure 20. Consume Power

The fork node of the Transmit Data activity has one input flow and two output flows, and replicates every input token it receives onto each of its output flows. This allows for the Consume Power action to occur concurrently. In the nested action Consume Power, the battery value property of the Autonomous Ferry is evaluated until the signal Device Off. Due to the expression specified in the body of the opaque action, the battery value property will decrease. Since these model elements are located inside an interruptible region the elements will be terminated.

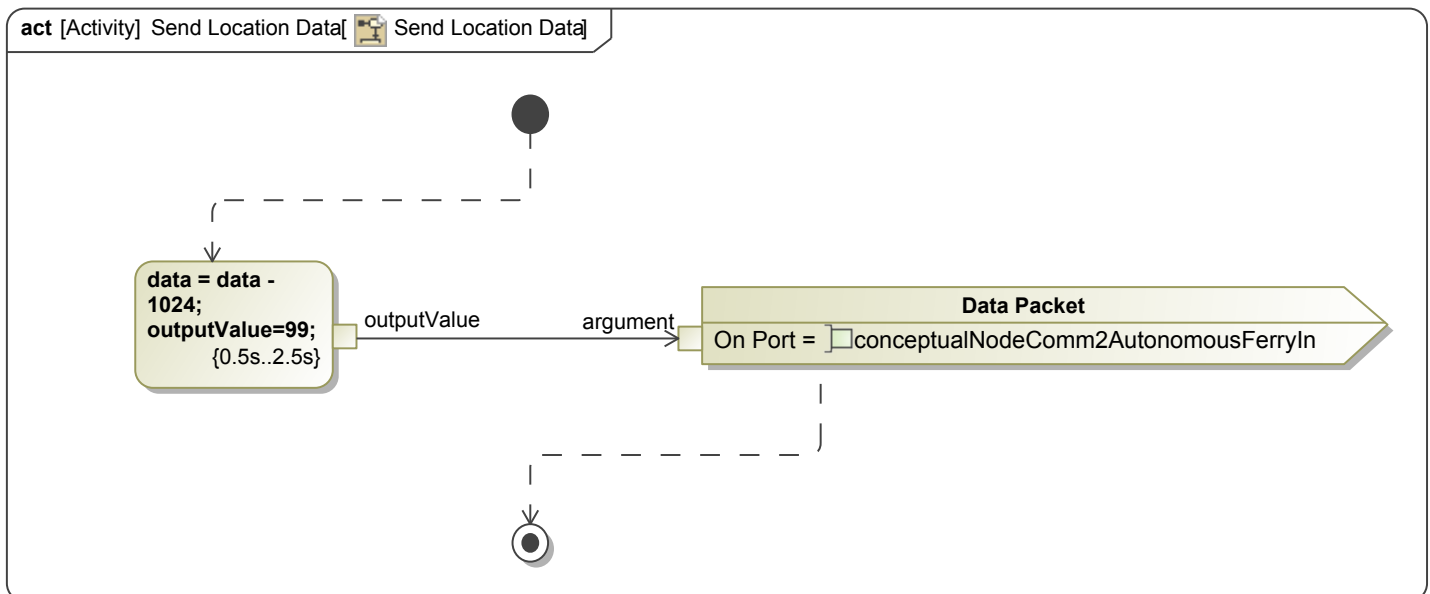


Figure 21. Send Location Data

If the data value property is greater than 0 then the Send Location Data activity will be performed. During this activity, an opaque action and a send signal action Data Packet will be concurrently performed. In the opaque action, an expression is specified to calculate the value property data. An object flow exists to route the input/output tokens that represent information between object nodes. The send signal action has one input pin per attribute of the signal to be sent and one input pin to specify the target for the signal. The target pin has been deleted so instead a signal is sent over the On Port. The Data Packet action is located on the proxy port

of the conceptualNodeComm2AutonomousFerryIn. Send signal actions support communication between executing behaviors using messages.

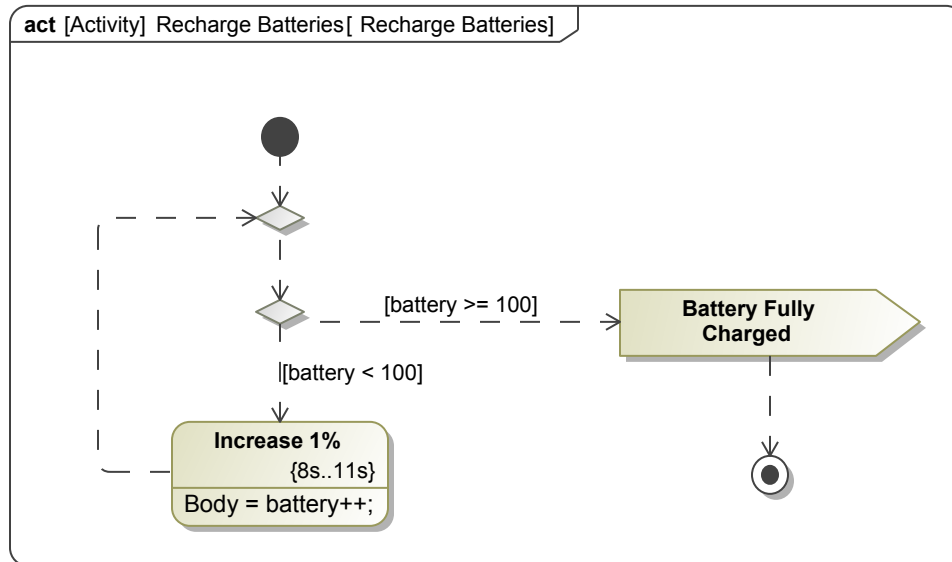


Figure 22. Recharge Batteries

Based on the Autonomous Ship Modes state machine, a change event is specified in the transition from the Not Recharging state to the Recharging state. The Recharge Batteries activity causes the Autonomous Ferry to return back to it's fully charged state. The change expression indicates that a certain condition expression has been satisfied. After the value of the property battery is less than 80 there will be a transition to the state Recharging where the Autonomous Ferry will perform the do behavior, Recharge Batteries. In the Recharge Batteries activity, transition guard expressions must evaluate to true for a transition to occur. If the battery value property is greater than or equal to 100 a transition to the Battery Fully Charged send signal action will occur. If the battery value property is less than 100 a transition to the Increase 1% opaque action will occur. In the opaque action, an expression is specified to increase the value of the battery value property (battery++), and a duration constraint specifies the min and max time for this action to occur (8s ..11s). The Recharge Batteries process will occur until the battery is greater than or equal to 100 to be terminated.

The behavior of the Remote Navigation Control block is specified through the application of state machines and activities.

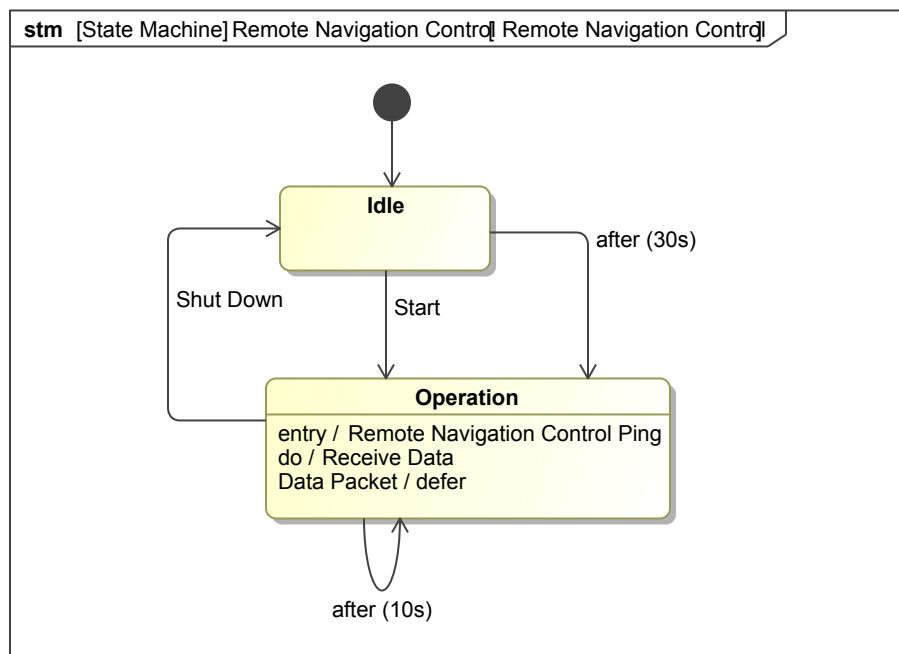


Figure 23. Remote Navigation Control

The system conceptual component Remote Navigation Control, owns a state machine as the classifier behavior Remote Navigation Control where the states Idle and Operation are specified. Depending on the time or Start signal, a transition to the Idle state will

occur. Upon entry of the Idle state the activity Remote Navigation Control Ping will be performed. After the entry behavior has been terminated, the do behavior Receive Data will execute. A time event trigger of 10s is specified on the self transition which will cause the Receive Data activity to terminate and reenter the Operation state which will cause the entry state, Remote Navigation Control Ping, to execute. The self transition is required to terminate the Receive Data activity.

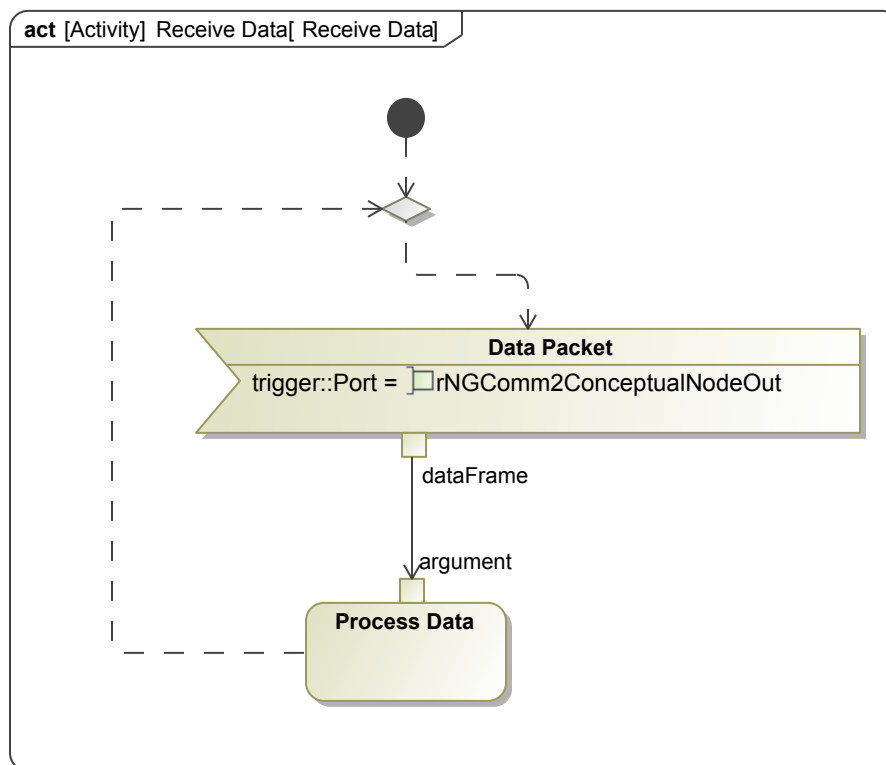


Figure 24. Receive Data

After the entry behavior has been terminated, the activity Receive Data will begin execution. An activity can accept signals through the usage of accept event action for a signal event. Communication can then be achieved between activities by including a send signal action in one activity and an accept signal action for a signal event representing the same signal in another activity. Communication through signals takes place asynchronously- the sender doesn't wait for the signal to be accepted by the receiver before proceeding to other actions. The AcceptEventAction is triggered by the reception of a dataPacket Signal on the port rNGComm2ConceptualNode. Through the application of signals, the Receive Data activity and the Remote Navigation Control Ping activity are able to communicate with another. The flow of data will begin on the port that is specified in both actions, rNGComm2ConceptualNode.

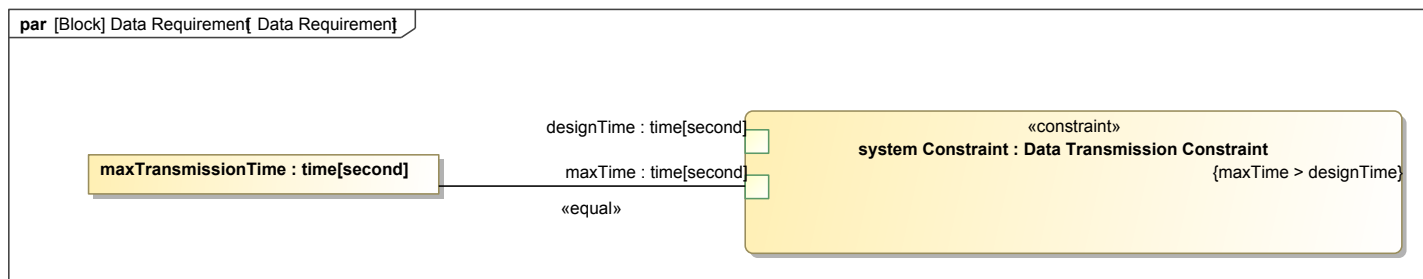


Figure 25. Data Requirement

Parametric diagrams are used to create systems of equations that can constrain the properties of blocks. A constraint block encapsulates a constraint to enable it to be defined once and then used in different contexts. The parametric model Data Requirement is owned by the Data Requirement constraint block to constrain the value property of the requirement (maxTransmissionTime) equal to the maxTime parameter. The constraint parameters of the system Constraint are maxTime and designTime. The constraint expression of the constraint block is maxTime > designTime. In the property based requirement, Data Requirement, the values of the system's properties are specified. The value property, maxTransmissionTime, is set equal to 300 seconds and the totalData is set equal to 10,400 bytes. Through the parametric model, the modeler can verify if the system design meets the maximum transmission time requirement.

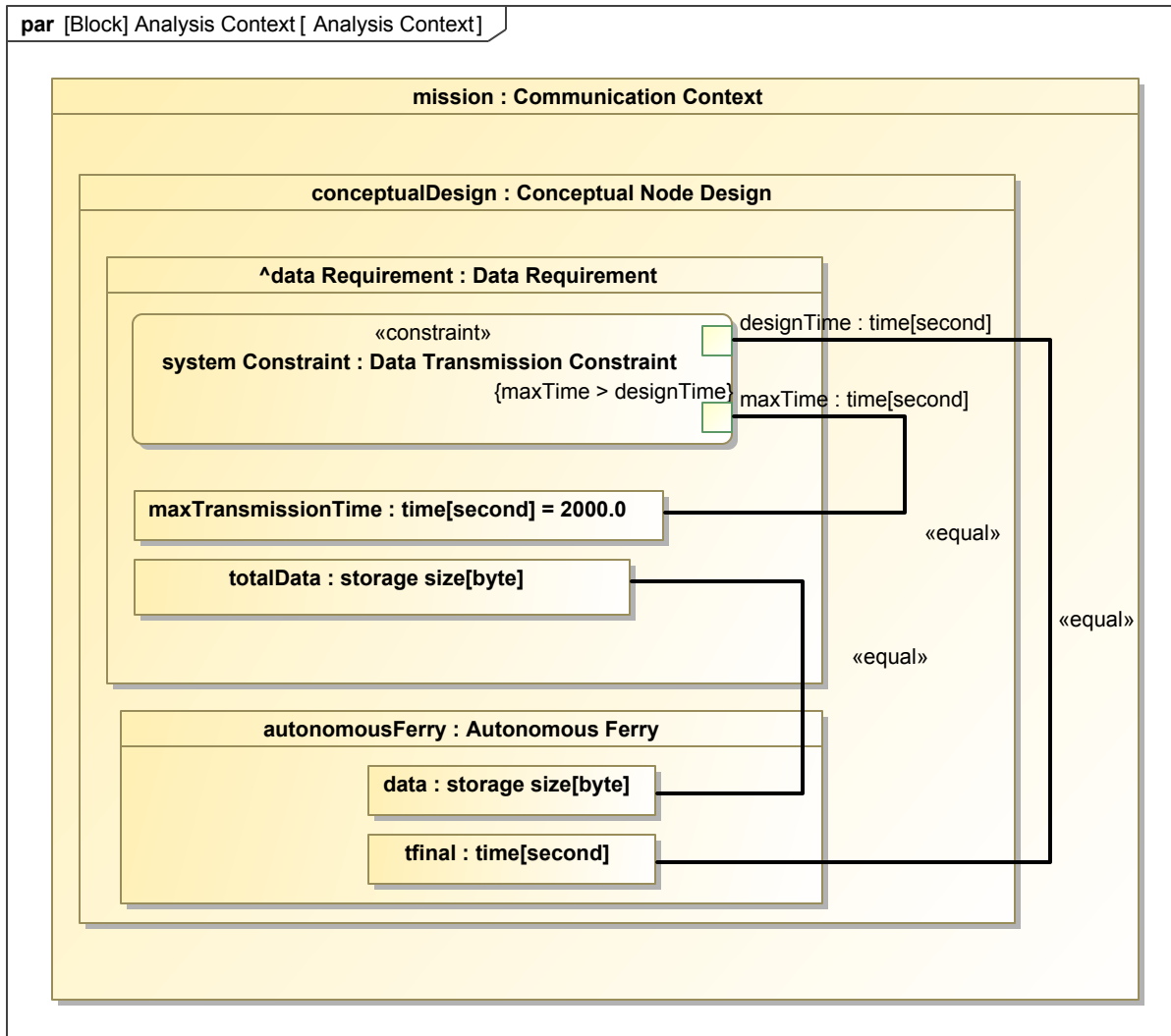


Figure 26. Analysis Context

Through the Analysis Context parametric model, the modeler can verify if the system design meets the maximum transmission time requirement for the given amount of data. The value properties of the Autonomous Ferry data and tfinal are constrained through connectors to calculate the total time and total data. The property data is constrained to the value property totalData of the Data Requirement. The tfinal property is connected to the system Constraint parameter designTime. The maxTransmissionTime property of the Data Requirement is constrained to the parameter maxTime. In the constraint expression of the system Constraint, the maxTime is defined to be greater than the designTime parameter.

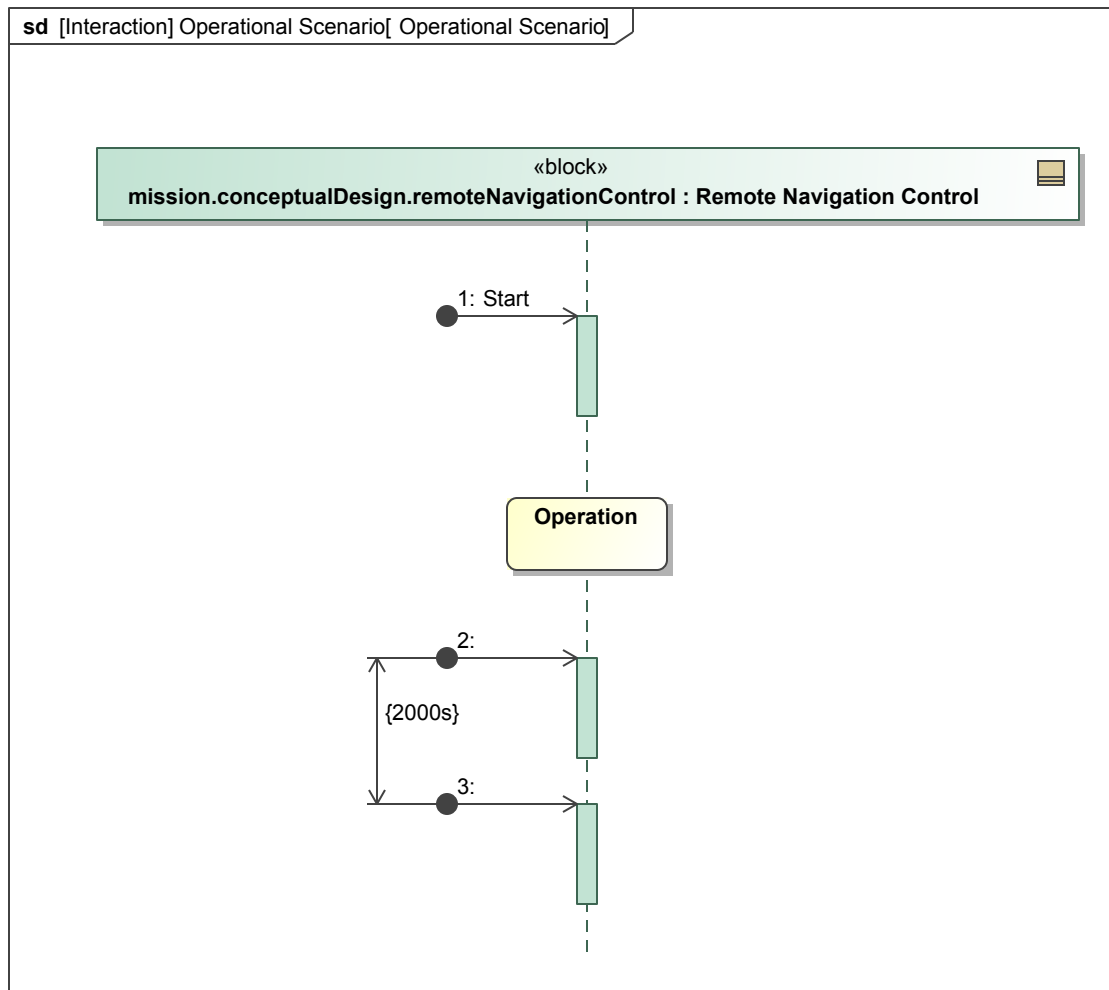


Figure 27. Operational Scenario

The Operational Scenario is the classifier behavior of the analysis context and specifies the verification scenario for the analysis context. The sequence diagram is applied to represent the interaction between structural elements in the model as a sequence of message exchanges. The lifeline must be in the context of the system (i.e. mission.system Specification.remoteNavigationControl) which represents the Remote Navigation Control block. For the simulation, a found message signal is used to send the Start signal of the Remote Navigation Control, and will cause the Operation action to begin execution where the entry and do behavior will be performed until terminating. The purpose of the duration constraint is to serve as a timeout for the simulation. The timeout terminates the simulation if there is a delay in the model and is unable to continue execution. The value specified is the expected time the simulation will run until completion.

3.6 Known Uses

The APS (Active Phasing System) team of the Thirty Meter Telescope applies an MBSE approach to analyze requirements, derive an architecture design, and ultimately implement the system. Particularly, through simulation of the APS system design, components can be determined to meet or fail the system's requirements. The Post-Segment Exchange Alignment Use Case, utilizes a combination of state machines, parametric diagrams, and activity diagrams to verify whether the the system can conform to time requirement.

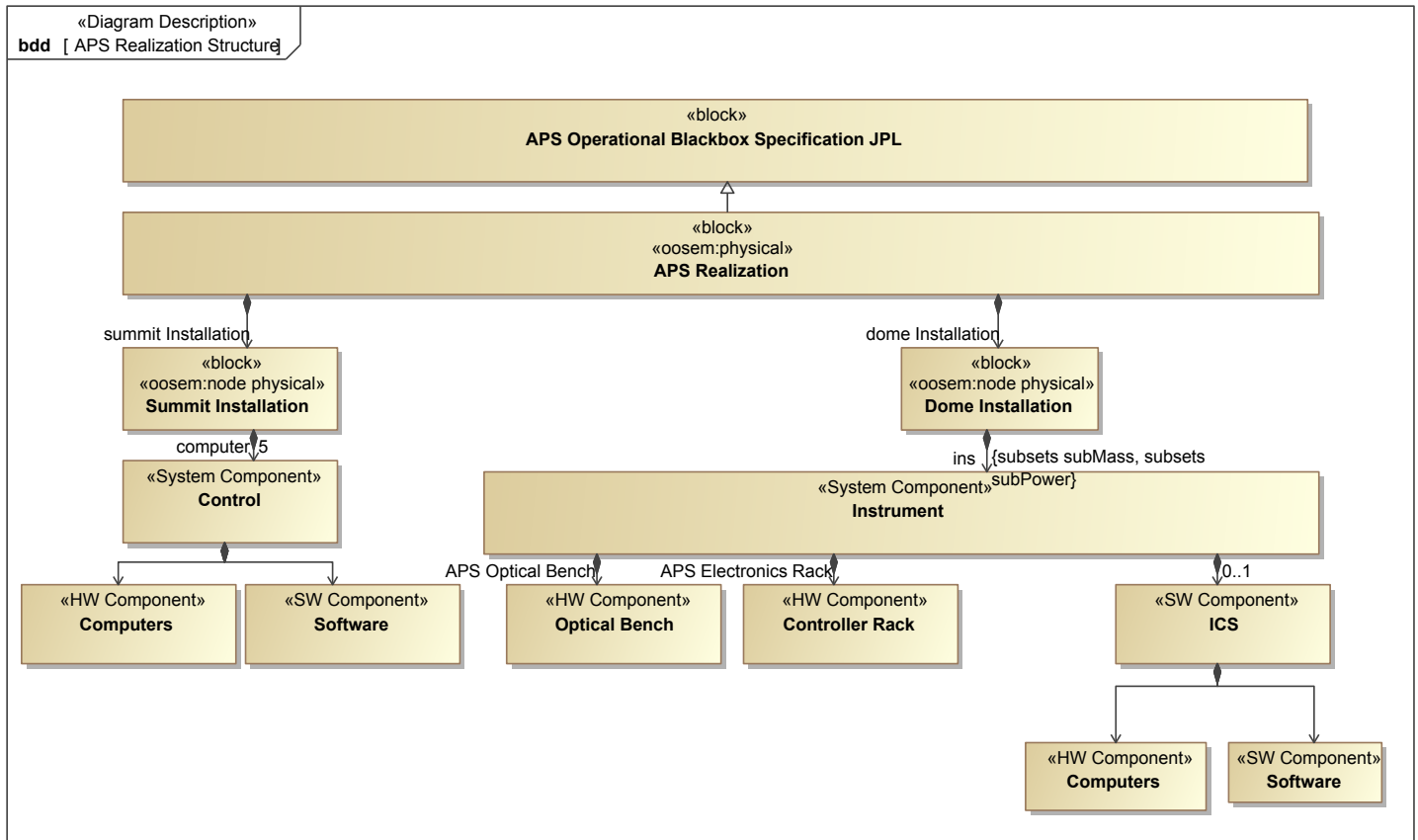


Figure 28. APS Realization Structure

The APS Realization, consists of a conceptual node design where physical and conceptual components reside. The relationship between the node design and the APS black-box specification is a generalization relationship. Therefore, the APS Realization inherits the properties, behavior, and requirements of the black-box specification.

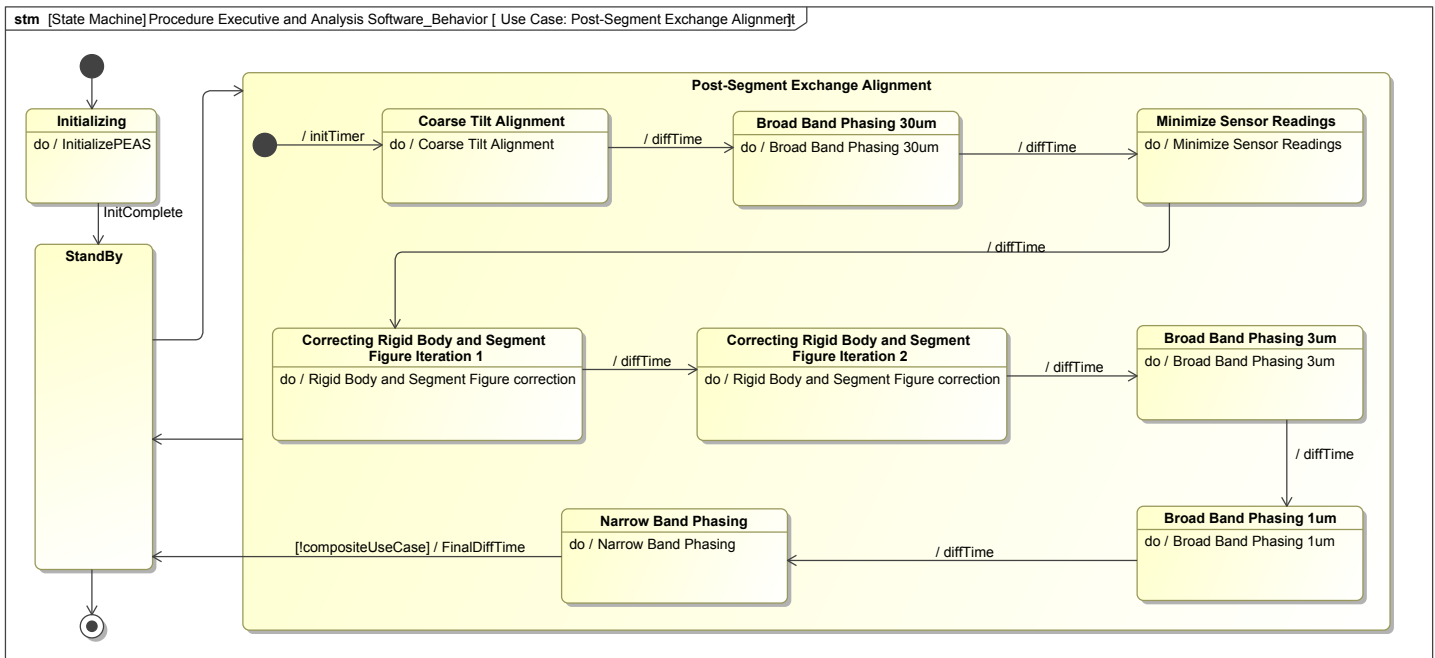


Figure 29. Use Case: Post-Segment Exchange Alignment

The behavior of the Post-Segment Exchange Alignment is specified through state machines and do behavior activities.

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

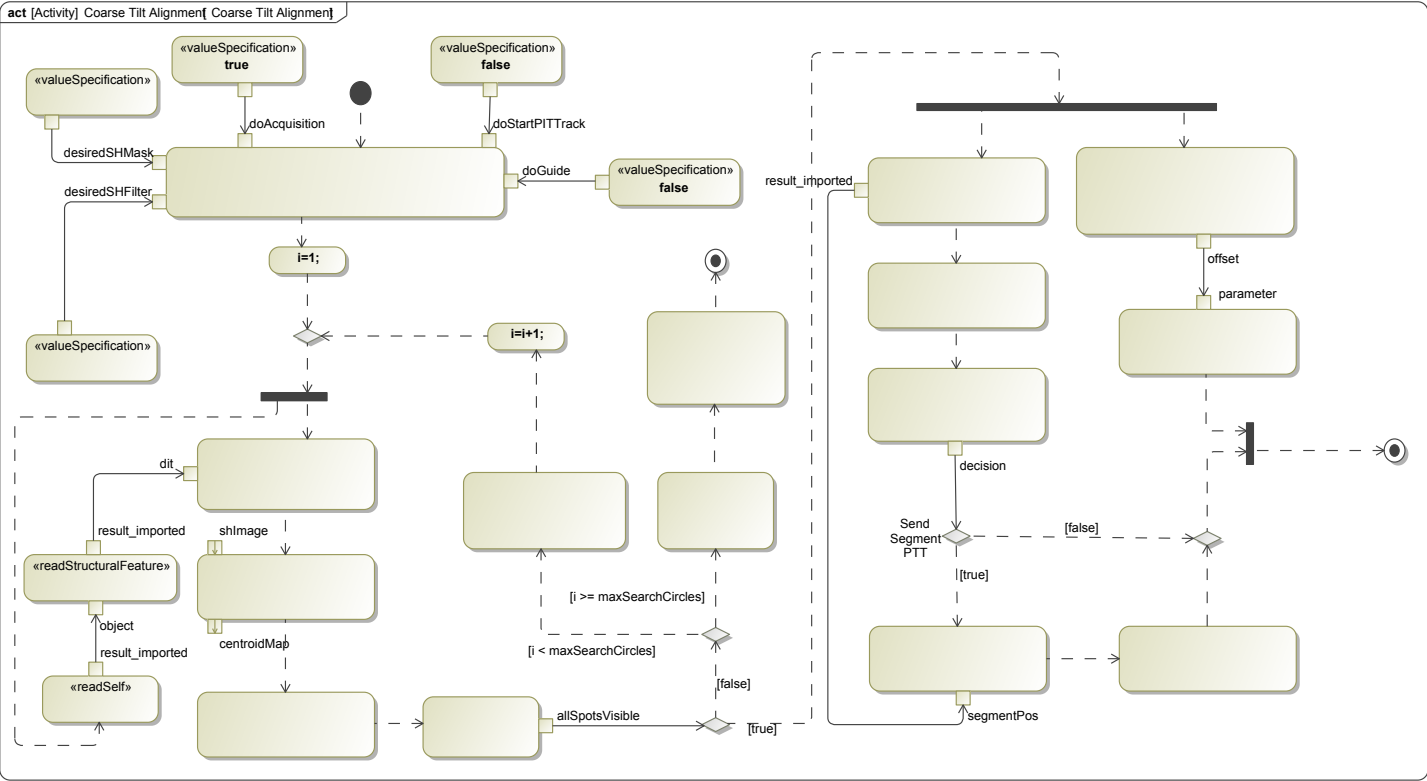


Figure 30. Coarse Tilt Alignment

Activity modeling emphasizes the inputs, outputs, sequences and conditions for coordinating other behaviors. It can be used to specify in detail the requested sequence of actions performed by the system during a specific Use Case. In the Coarse Tilt Alignment diagram, the actions that are required to perform the Coarse Tilt Alignment activity are modeled.

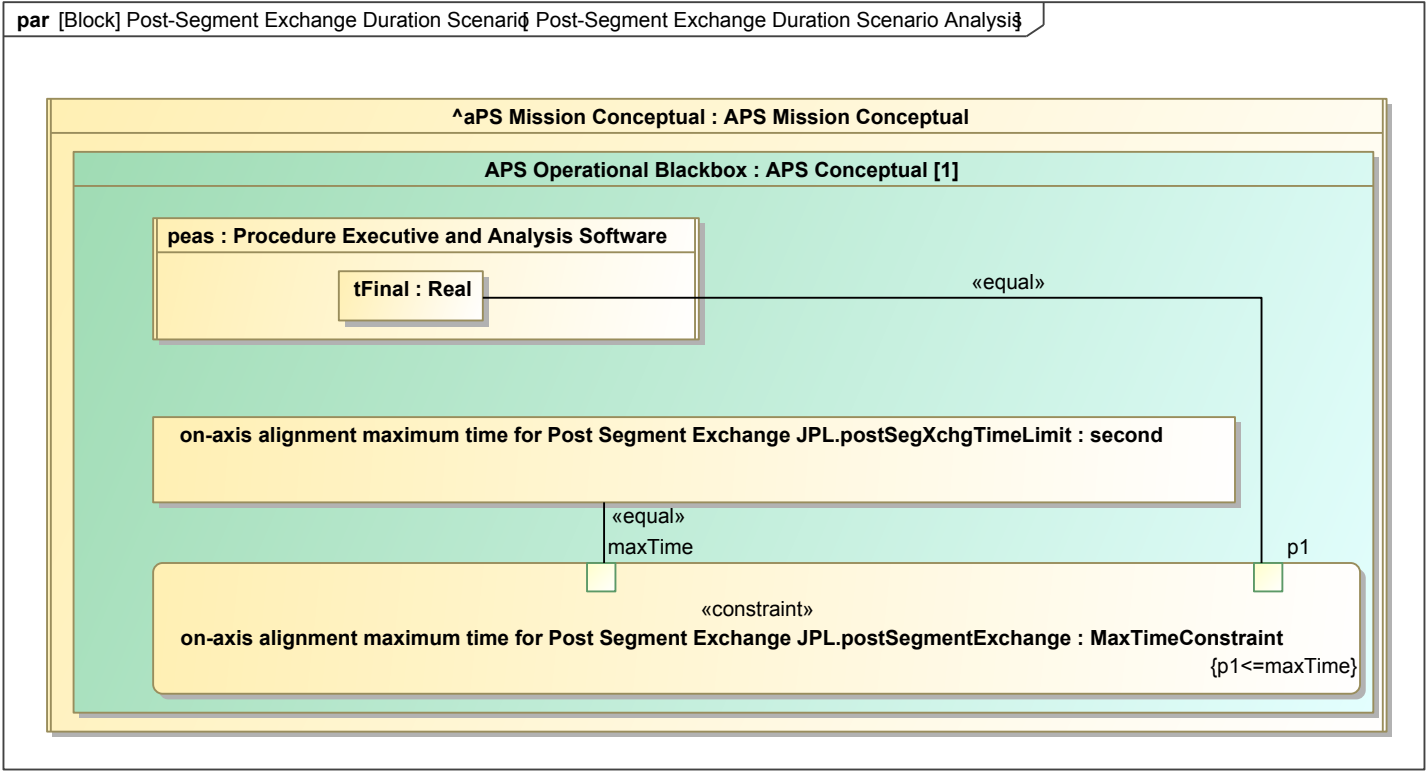


Figure 31. Post-Segment Exchange Duration Scenario Analysis

All Use Case Duration Scenarios should own a parametric diagram. The purpose of the Use Case duration scenarios is to determine the final time and overall duration of a particular use case. Therefore, each duration scenario block should own a parametric diagram.

Depending on the use case, the tFinal value property of the Procedure Executive and Analysis Software could be constrained to a constraint of the APS Operational Blackbox for requirement verification purposes. All current and future constraints in APS should be found in the model at the following location: TMT::Project::Work Packages::Telescope::WorkPackages::Control::Alignment and Phasing System::Systems Engineering::01 Components::Operational::APS::Black Box Specification::Requirements::10 Formalized Requirements.

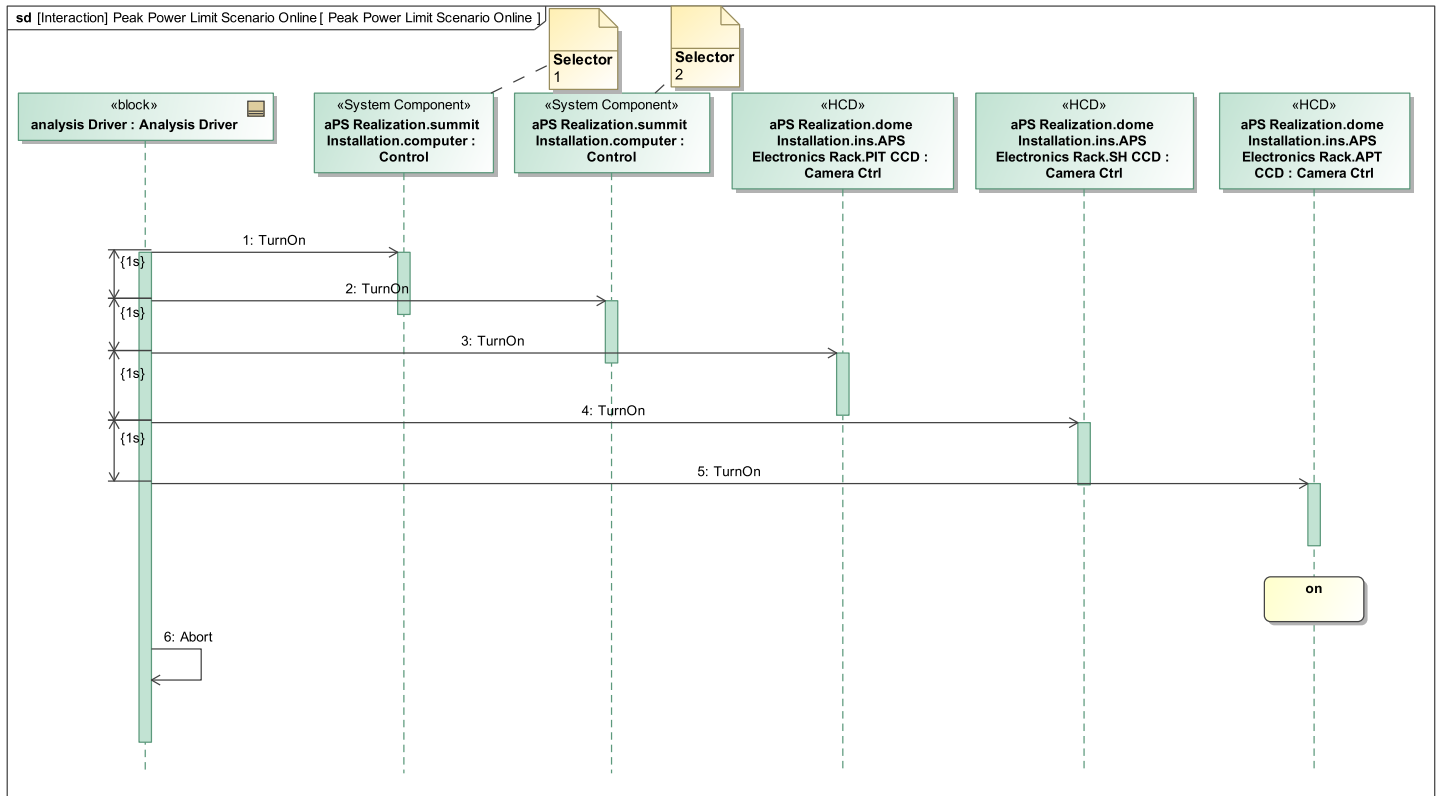


Figure 32. Peak Power Limit Scenario Online

Sequence diagrams are used to specify the interaction between blocks of the system. Messages are exchanged between the instance lifelines (i.e. computer, PIT CCD, SH CCD, APT CCD), and duration constraints are used to specify the timing between each signal.

#	Name	Classifier	Operating Power : W	Standby Power : W
9	peak Power Limit Scenario Online.aPS Realization.dome Installation.	Slide Wheel Ctrl	10.0	
10	peak Power Limit Scenario Online.aPS Realization.dome Installation.	Slide Wheel Ctrl	10.0	
11	peak Power Limit Scenario Online.aPS Realization.dome Installation.	Camera Ctrl	150.0	0.0
12	peak Power Limit Scenario Online.aPS Realization.dome Installation.	MassRollUpPattern		
13	peak Power Limit Scenario Online.aPS Realization.dome Installation.	PowerRollUpPattern		
14	peak Power Limit Scenario Online.aPS Realization.dome Installation.	MassRollUpPattern		
15	peak Power Limit Scenario Online.aPS Realization.dome Installation.	PowerRollUpPattern		
16	peak Power Limit Scenario Online.aPS Realization.dome Installation.	Digital IF	5.0	
17	peak Power Limit Scenario Online.aPS Realization.dome Installation.	Digital IF	5.0	
18	peak Power Limit Scenario Online.aPS Realization.dome Installation.	Camera Ctrl	150.0	150.0
19	peak Power Limit Scenario Online.aPS Realization.dome Installation.	Rotate Wheel Ctrl	10.0	
20	peak Power Limit Scenario Online.aPS Realization.dome Installation.	Rotate Wheel Ctrl	10.0	
21	peak Power Limit Scenario Online.aPS Realization.dome Installation.	Digital IF	10.0	

Figure 33. APS Realization Configuration Scenario Online

To configure the physical and conceptual components of the APS system an instance table is applied (APS Realization Configuration Scenario Online). For example, the Camera Ctrl is configured to have an operating power of 150.0 W and a Standby Power of 150.0 W. Instances should be used to specify the values of properties instead of redefining each component.

Table 6. Post-Segment Exchange A

Name	Classifier	postSegXchgTimeLimit : second	tFinal : Real	postSegmentExchange : MaxTimeConstraint	broadbandPhasingSteps : Integer	narrowbandFilter : Integer
post-Segment Exchange Duration Scenario at 2017.10.19 07.28	Post- Segment Exchange Duration Scenario					
post-Segment Exchange Duration Scenario at 2017.10.24 09.01	Post- Segment Exchange Duration Scenario					
post-Segment Exchange Duration Scenario.aps mission conceptual.aps conceptual	APS Conceptual					
post-Segment Exchange Duration Scenario.aps mission conceptual.aps conceptual.on-axis alignment maximum time for Post Segment Exchange JPL	On-axis alignment maximum time for Post Segment Exchange JPL	7200.0		pass		
post-Segment Exchange Duration Scenario.aps mission conceptual.aps conceptual.procedure Executive and Analysis Software	Procedure Executive and Analysis Software		4804.0		11	2
post-Segment Exchange Duration Scenario.aps mission conceptual.executive Software	Executive Software					
post-Segment Exchange Duration Scenario.aps mission conceptual.1.aps conceptual	APS Conceptual					
post-Segment Exchange Duration Scenario.aps mission conceptual.1.aps conceptual.on-axis alignment maximum time for Post Segment Exchange JPL	On-axis alignment maximum time for Post Segment Exchange JPL	7200.0		pass		

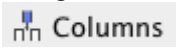
Name	Classifier	postSegXchgTimeLimit : second	tFinal : Real	postSegmentExchange : MaxTimeConstraint	broadbandPhasingSteps : Integer	narrowbandFilter : Integer
post-Segment Exchange Duration Scenario.aps mission conceptual1.aps conceptual.procedure Executive and Analysis Software	Procedure Executive and Analysis Software		4782.0		11	2
post-Segment Exchange Duration Scenario.aps mission conceptual1.executive Software	Executive Software					

3.7 Analysis

After configuring the Requirement Verification Pattern to the system an analysis context needs to be specified. In this analysis context, any operational scenarios need to be specified (in either sequence or activity diagrams), and the values of the configurations need to be specified either through redefinition or instance specifications.

Table 7. Analysis Results

Name	Classifier	mission.conceptualDesign.autonomousFerry.tfinal : time[second]	mission.conceptualDesign.data Requirement.maxTransmissionTime : time[second]	mission.conceptualDesign Requirement.system Constraint : D Transmission Con
analysis Context at 2017.08.02 08.31	Analysis Context	1620.0	300.0	fail
analysis Context at 2017.08.02 08.36	Analysis Context	1622.5	2000.0	pass
analysis Context at 2017.08.02 09.07	Analysis Context	1604.0	2000.0	pass
analysis Context at 2017.09.13 13.03	Analysis Context	0.0	2000.0	pass
analysis Context at 2017.09.13 13.45	Analysis Context	1629.0	2000.0	pass

Verification ensures that the system design satisfies a corresponding requirement for every defined specified scenario. The resulting instance specifications of the simulation can be displayed in an instance table. To display whether the average time satisfies the system requirement, a column for the constraint is created through the "Select Nested Columns" option of the  **Columns** tab. After selecting the constraint, the value of the constraint slot is displayed. From the table we are able to verify if the value of the average time satisfies the property based requirement.

3.8 Tooling

The following tooling supports the modeler in the implementation of the Requirement Verification Pattern in MagicDraw.

3.8.1 Cameo Simulation Toolkit

NoMagic's Cameo Simulation Toolkit (CST) is implemented and supports the Requirement Verification Pattern by providing a method to perform analysis of a modeler's system. After the pattern has been augmented to the system's design, the value properties of the components participating in the simulation need to be specified either through redefinition or instance specifications, and then analysis can be performed on the system by running a simulation with CST. In the variables panel, the user will be able to immediately view properties that have been initialized and the traceability to the system requirement. After the simulation has been completed the variables of the simulation will be denoted as a pass or fail depending on the color. The color green means the system design satisfies the requirement, and red means the requirement is not satisfied.

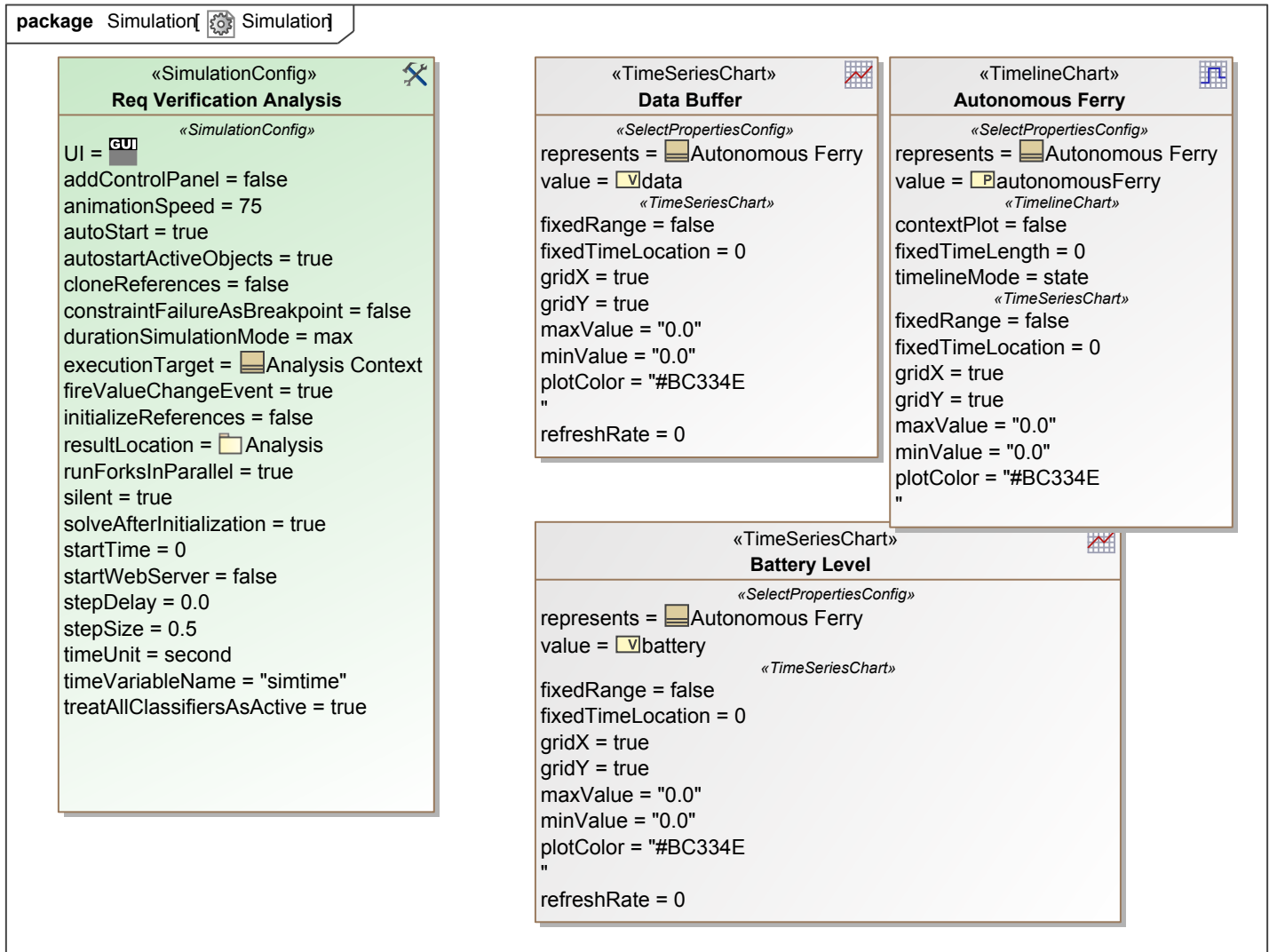


Figure 34. Simulation

To automate the simulation process, a simulation configuration can be specified. The executionTarget property would be set equal to the Analysis Context block which would allow the modeler to execute their system by running the simulation configuration. The output results of the simulation could then be stored in an instance table.

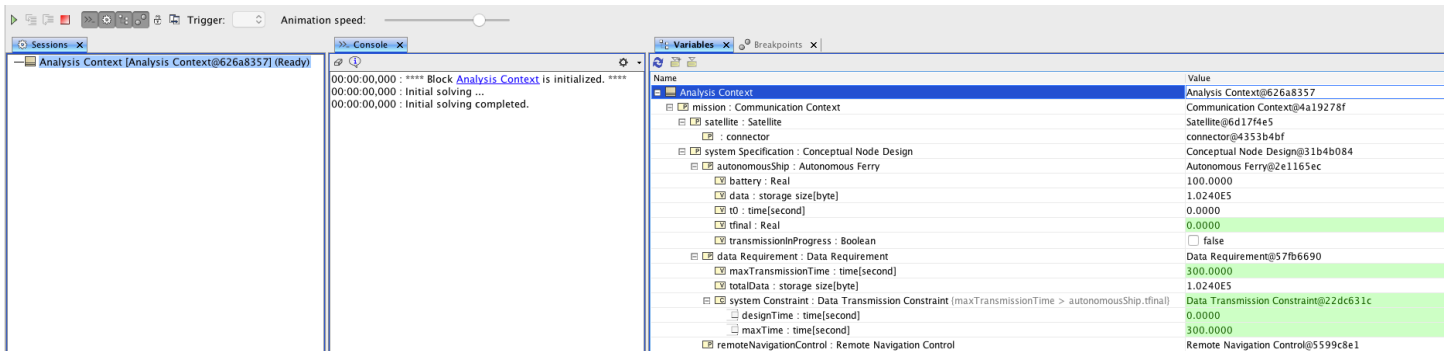


Figure 35. Simulation

After the operational scenario and simulation configuration have been configured, the modeler is ready to perform a simulation of the system using Cameo Simulation Toolkit. During initial solving CST will highlight the value properties tfinal and maxTransmissionTime to confirm that maxTransmissionTime is greater than the tfinal value property. The initial value of the maxTransmissionTime property is equal to 300 seconds, the battery property is equal to 100, and the data property is equal to 1.024E5.

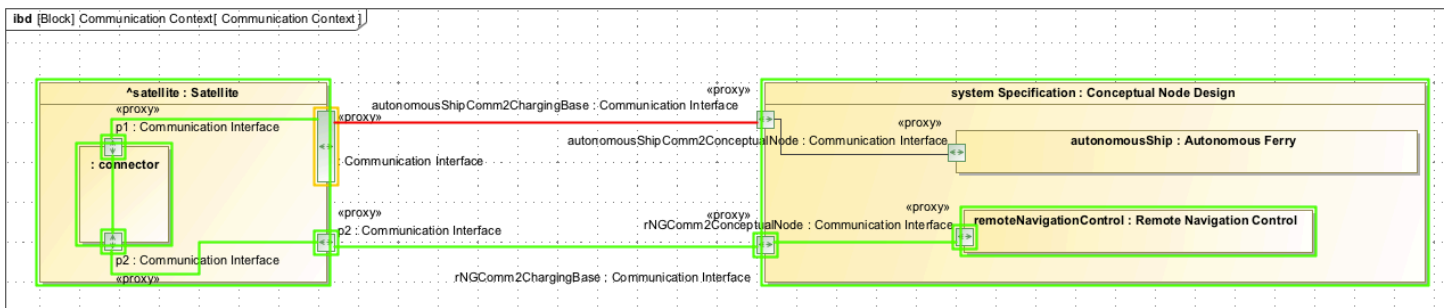


Figure 36. Communication Context

Once the simulation has been started, the system will begin execution of the Autonomous Ship Modes and the Remote Navigation Control state machines. After the Remote Navigation Control has transitioned to the Operation state the entry behavior Remote Navigation Control Ping to be executed. The send signal action in the Remote Navigation Control Ping activity starts the flow of data from the rNGComm2ConceptualNodeOut proxy port until it has reached the conceptualNodeComm2AutonomousFerryIn proxy port. The above diagram shows the communication from the Remote Navigation Control to the Satellite to the Autonomous Ferry.

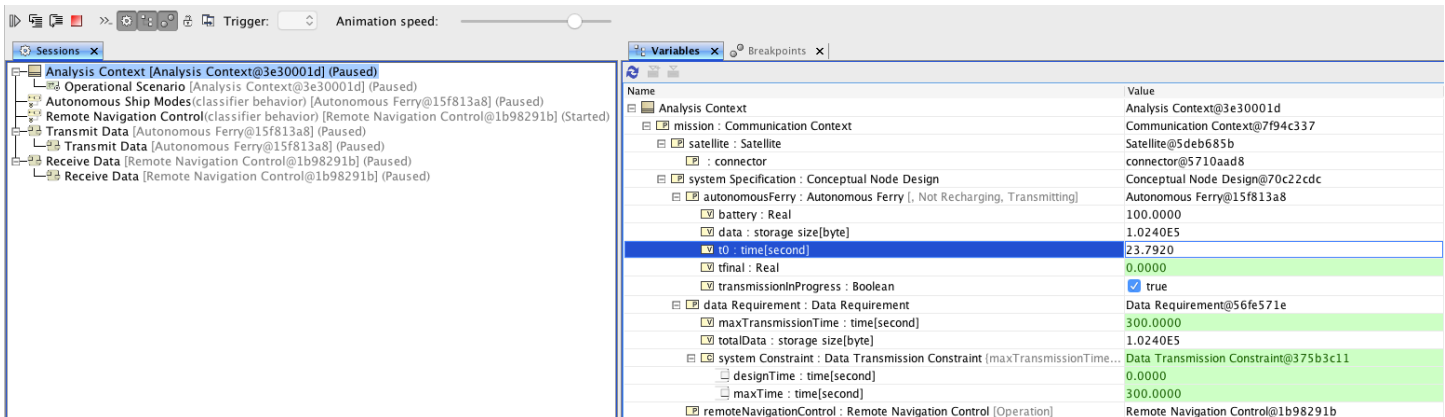


Figure 37. Analysis Context

After the entry behavior of the Remote Navigation Control has been terminated, the do behavior Receive Data will be performed. Additionally, in the Autonomous Ship Modes state machine the Remote Nav Ping signal will cause a transition to the Transmitting state where the entry behavior Transmit Data will begin execution. In the Transmit Data activity, the first opaque action will set the t0 value property equal to the simulation time (23.7920s) and will set the transmissionInProgress to true.

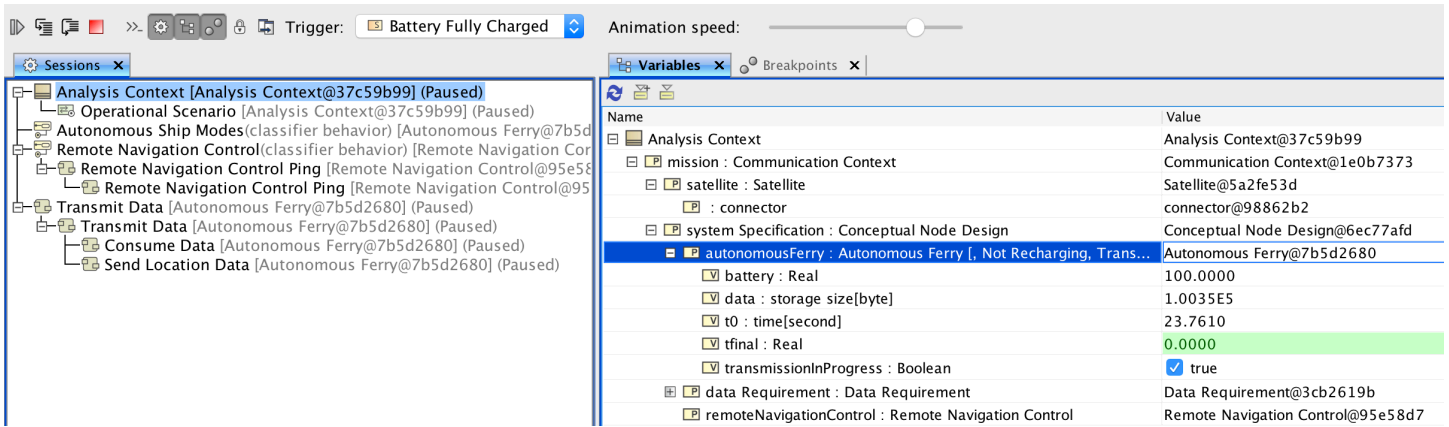


Figure 38. Transmit Data

During the Transmit Data activity of the Autonomous Ship Modes state machine, the Autonomous Ferry's data value property will decrease as it performs the Send Location Data activity. The opaque action of the activity will cause the value of the data property to decrease by 1024 each time the Send Location Data activity is performed. This will occur until the data is less than or equal to 0.

Upon running the Req Verification Analysis simulation configuration, the output of the simulation will be displayed in timeSeries and timelineSeries charts.

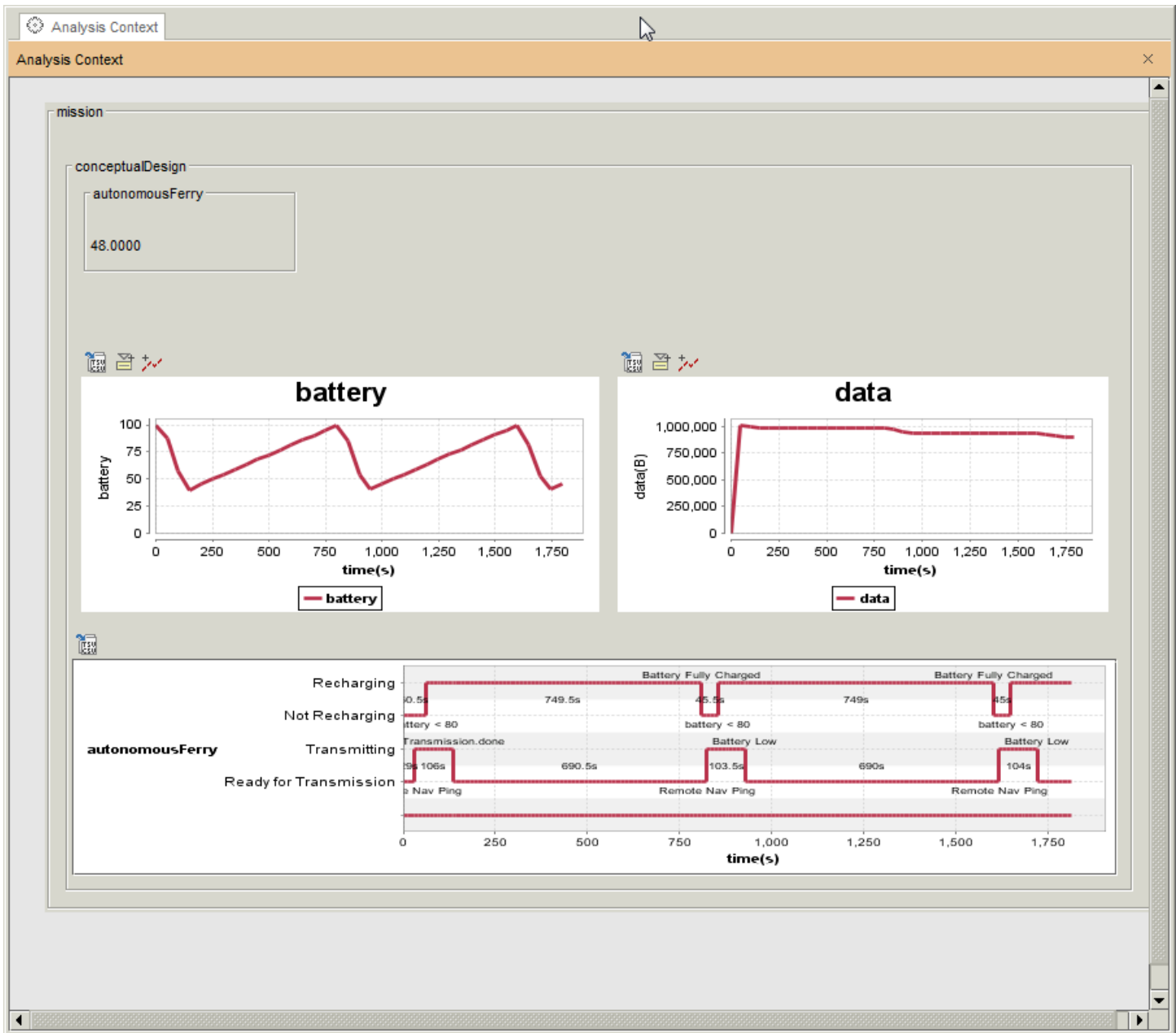


Figure 39. Simulation UI

3.9 Related Patterns

List of other patterns that are related to the Requirement Verification Pattern , and the differences and similarities between them.

Table 8. Related Patterns Table

Pattern Name	Similarities	Differences
Property Based Requirements Pattern	The requirements of the Requirement Verification Pattern are modeled as Property Based Requirements. The Requirement represents a contextual requirement, and is attributed value properties and constraints. The value properties of the requirement transform the textual requirement into an executable element. In the constraint property, parameters conform to a constraint property.	The structure of the Property Based Requirements Pattern describes the attributes and behavior of a property based requirement. However, it doesn't include how the property based requirement can be applied to a system. In the Requirement Verification Pattern, the requirements are applied to an Enterprise as a part property of the System Specification.

Black Box Specification with Property Based Requirements Pattern	<p>The system specification of the Requirement Verification Pattern is modeled identically as described in the Black Box Specification with Property Based Requirements Pattern. The System Specification of the Requirement Verification Pattern is a specialization of OOSEM's "System Of Interest Blackbox Design".</p>	<p>The structure of the Black Box Specification with Property Based Requirements Pattern shows that a System Component is a part of the System Specification. However, in the Black Box Specification with Property Based Requirements Pattern the Conceptual Node Design owns all Conceptual Components. These are two different methods to model a component of a system specification.</p>
--	--	---