05 Systems Analysis Management

Table of Contents

1 Introduction 8
2 Multi-Scenario Analysis Pattern 9
2.1 Intent 9
2.2 Motivation 9
2.3 Concept 9
2.4 Consequences 13
2.5 Implementation 13
2.6 Known Uses 22
2.7 Related Patterns 23
2.8 Tooling 24
2.8.1 Cameo Simulation Toolkit 24
3 Duration Analysis Pattern 27
3.1 Intent 27
3.2 Motivation 27
3.3 Concept 27
3.4 Consequences 29
3.5 Implementation 29
3.6 Analysis 33
3.7 Related Patterns 33
3.8 Conflicting Usages 34
3.9 Tooling 34
3.9.1 Cameo Simulation Toolkit 34
3.10 Known Uses 35
4 Black Box Specification with Property Based Requirements Pattern 37
4.1 Intent 37
4.2 Motivation 37
4.3 Concept 37
4.4 Consequences 40
4.5 Implementation 42
4.6 Known Uses 44
4.7 Tooling 46
4.7.1 Model Construction 46
4.8 Related Patterns 47
5 Monte Carlo Simulation Pattern 49
5.1 Intent 49
5.2 Motivation 49
5.3 Concept 49
5.4 Consequences 54
5.5 Analysis 55
5.6 Known Uses 55
5.7 Related Patterns 62
5.8 Implementation 63
5.8.1 Example 1 63
5.8.2 Example 2 68
5.9 Tooling 72
5.9.1 Cameo Simulation Toolkit 72
5.9.1.1 Implementation Simulation Configurations 78
5.9.2 Monte Carlo Plugin 78
6 Scenario Modeling Pattern 80
6.1 Pattern Name and Classification 80
6.2 Intent 80
6.3 Motivation 80
6.4 Concept 80
6.5 Consequences 82
6.6 Sample Model 82
6.7 Known Uses 85
6.8 Related Patterns 86

6.9 Tooling 86
6.9.1 MagicDraw 86
6.10 Augmented Scenario Pattern 86
6.11 Augmented Scenario Pattern 2 88

List of Tables

1. <> 12
2. Consequences Table 13
3. <> 29
4. T1 29
5. <> 33
6. T1 34
7. Analysis Context Instances 35
8. <> 40
9. Consequences Table 40
10. <> 53
11. Consequences Table 54
12. Timing Results 66
13. Calculation Results 68
14. results 71
15. Timing Analysis Results 77

List of Figures

1. Multi-Scenario Structure....10
2. Muliple Test Case Structure....11
3. Analysis Coverage Activity....12
4. Fidget Spinner Bearing Analyses....14
5. Fidget Spinner System....15
6. Fidget Spinner....16
7. Fidget Spinner....17
8. Friction Slows Clockwise Spin....18
9. Spin Clockwise....18
10. Friction Slows Counter Clockwise Spin....19
11. Spin Counter Clockwise....19
12. Fidget Spinner Spin Clockwise Analysis Scenario....20
13. Fidget Spinner Spin Both Directions Analysis Scenario....20
14. Fidget Spinner Direction Analysis Coverage....21
15. TMT Domain to-be....22
16. TMT Mission....23
17. Fidget Spinner Spin Clockwise Scenario Simulation....24
18. Fidget Spinner Velocity Clockwise Direction....25
19. Fidget Spinner Spin Both Directions Scenario Simulation....25
20. Fidget Spinner Velocity Both Directions....26
21. Analysis Coverage Simulation....26
22. Duration Analysis Structure....28
23. Analysis Context....28
24. Washingmachine definition....30
25. Washing Machine Analysis Context....31
26. WashingMachine....31
27. Controller....32
28. doWashing....32
29. initiateShort and completion Specifications....33
30. WashingMachineSimulation....34
31. Cameo Simulation Results....35
32. Cycle Analysis....35
33. CC_SH....36
34. SendAck_Take_Exposure....36
35. Maintenance Alignment Duration Analysis....36
36. Black Box Specification with Property Based Requirements Pattern Structure....38
37. Verification Context....39
38. Property Based Requirement Association Relationship....39
39. Property Based Requirement Association Relationship....41
40. Drone System....42
41. Drone Components....43
42. Drone System Requirements Diagram....44
43. Drone System Property Based Requirements Diagram....44
44. APS Operational Blackbox Specification....45
45. Timing Requirements....46
46. Model Construction View....47
47. Monte Carlo Simulation Components....50
48. System Activity....51
49. Pattern....52
50. Analysis Contexts....53
51. Acquire a target with IRIS and NFIRAOS - Conceptual Actual....56
52. Acquire and Lock TT OIWFS/ODGW Conceptual Actual....57
53. Acquire and Lock LGS Conceptual Actual....58
54. Acquire and Lock PWFS Conceptual Actual....60
55. Acquisition Simulation....61
56. Block TMT Mission-to-be Min at 2017.02.15 15.11....62
57. Probability Instance Table....62

58. System Hierarchy....64
59. Move Robot....65
60. Turn on sensory system....65
61. Analysis Context Instance....66
62. Multi Element Analysis Context....67
63. Analysis Context Constraints....68
64. Turkey Dinner....69
65. Evaluate Dinner....70
66. AnalysisDef....70
67. Scenario Testing....71
68. Histogram....72
69. System....73
70. System Behavior....74
71. Action A Behavior....75
72. MonteCarlo....76
73. Body and Language....76
74. System Timing Configuration....77
75. MD 19.0 Simulation Configuration....78
76. Run Monte Carlo....78
77. Generic Layout....81
78. Context....81
79. Scenario....82
80. System of Interest....82
81. Scenario Group 1....83
82. Food Truck....84
83. Power Roll-up....84
84. Traveling....85
85. Serving....85
86. Generic Layout....86
87. Variable Remove....87
88. Variable Add....87
89. Generic Layout 2....89
90. Context 2 Parametric....90

1 Introduction

The System Analysis Case Study focuses on the needs of analysts supporting the full lifecycle of a system. The focus on the full system for analysis requires consideration of multiple contributors and data sources. Because the full system is the topic of analysis, the system's uses, supporting components, envisioned scenarios, and the like all need to be considered. These will typically be inputs from other stakeholders, or if developed at the system level, provided to them for their own analysis and understanding.

2 Multi-Scenario Analysis Pattern

2.1 Intent

The intent of the Multi-Scenario Analysis Pattern is to provide a method to model variations, configurations, or scenarios of a system, and to perform systems engineering analyses on them. The differences between configurations can range from value properties, numerical values, and the analysis performed. The Multi-Scenario Analysis Pattern provides two methods/applications for analyzing scenarios. The first method is to individually execute a scenario, and the second method is to group analysis scenarios based on their commonalities into one component and executing the coverage. This pattern may be used in conjunction with the Requirement Verification Patternin order to analyze and verify nominal, off-nominal, and boundary values.

2.2 Motivation

The motivation for this pattern is to capture multiple states within the model for reasons such as

- represent system states during operational scenarios (e.g., nominal and off-nominal) for analysis and/or reports
- analyze system changes over time
- use within the Requirement Verification Pattern to show theoretical compliance under defined conditions
- perform trade studies
- establish model "regression tests," including checks to assist in model refactoring.

Note that this general pattern is implemented with lower-level variations depending on the type of analyses being performed. For example, some analyses compare the same sequences (i.e., state machines) and/or behaviors with different element property values, whereas other analyses may compare results of different sequences with the same property values.

The Multi-Scenario Requirements Pattern enables representing multiple states within the model, particularly to perform analyses. For example, what range of values for a system property will still satisfy a requirement? This pattern is a useful approach to gaining insight into the model and refining it. By decoupling the system specification from the analysis, insight is not at the expense of breaking the system specification. The same analysis can also be reused. For example, a trade study to establish a valid property value range can be adapted to verify requirement(s), and also be used as a regression test. Note that analysis design should be done with special attention to minimize potential for breakage as the model evolves.

2.3 Concept

TheMulti-Scenario Analysis Pattern provides two methods/applications for analyzing scenarios. The first method is to individually execute a scenario (i.e. Analysis Scenario). The second method is to group analysis scenarios based on their commonalities into a component (i.e. Analysis Coverage), and executing the group/coverage.

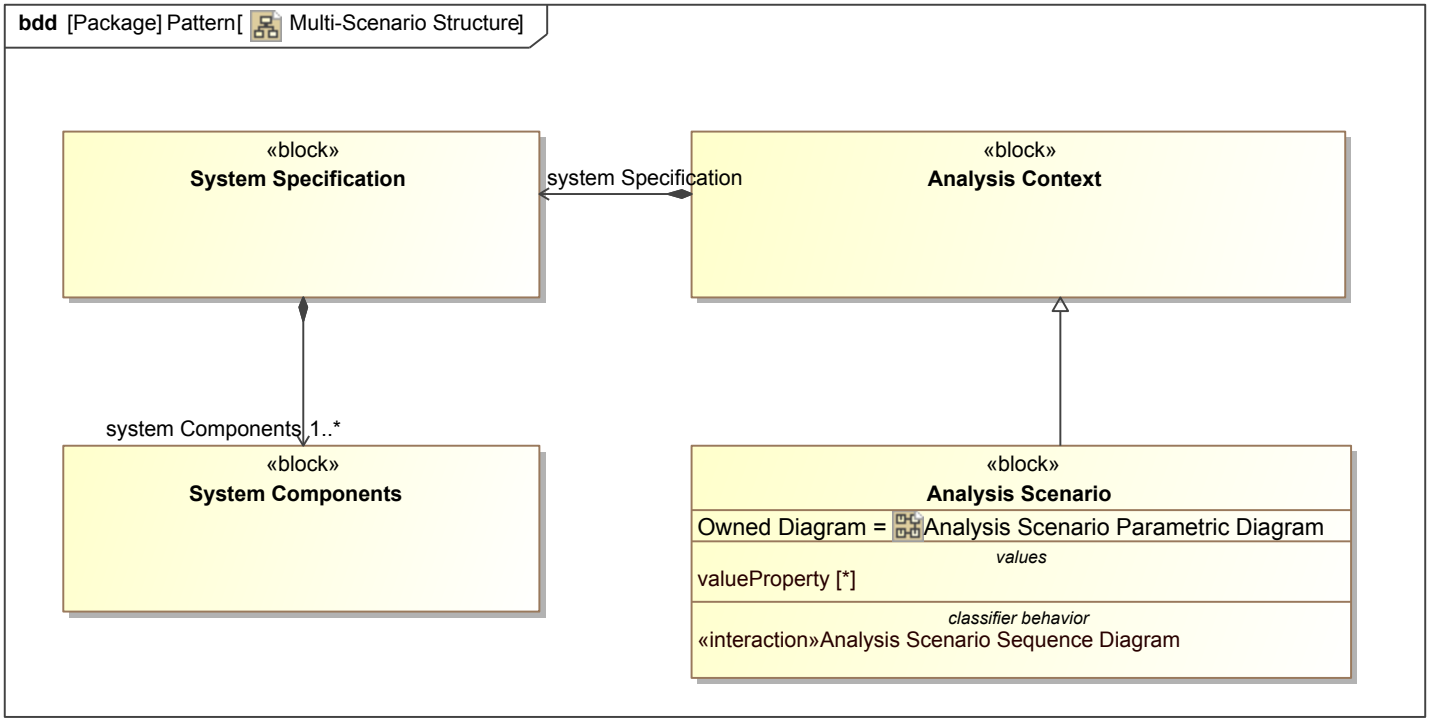

Figure 1. Multi-Scenario Structure

The components that comprise the structure of the Multi-Scenario Analysis Pattern are the following model elements: System Specification, System Components, Analysis Context, and Analysis Scenario. The behavior and function of the System Specification will be defined through its System Components, and is a part of the Analysis Context. The Analysis Context block contextualizes the System Specification content for execution purposes. Each Analysis Scenario defined would specialize the Analysis Context block. These Analysis Scenarios will inherit the structure, function, and behavior of the Analysis Context. The elements that will need to be defined in the Analysis Scenario is a Sequence Diagram, scenario specific value properties, and a parametric diagram to constrain the value properties of the Analysis Scenario to the Analysis Context. To analyze a particular scenario, the modeler will execute the particular Analysis Scenario block.

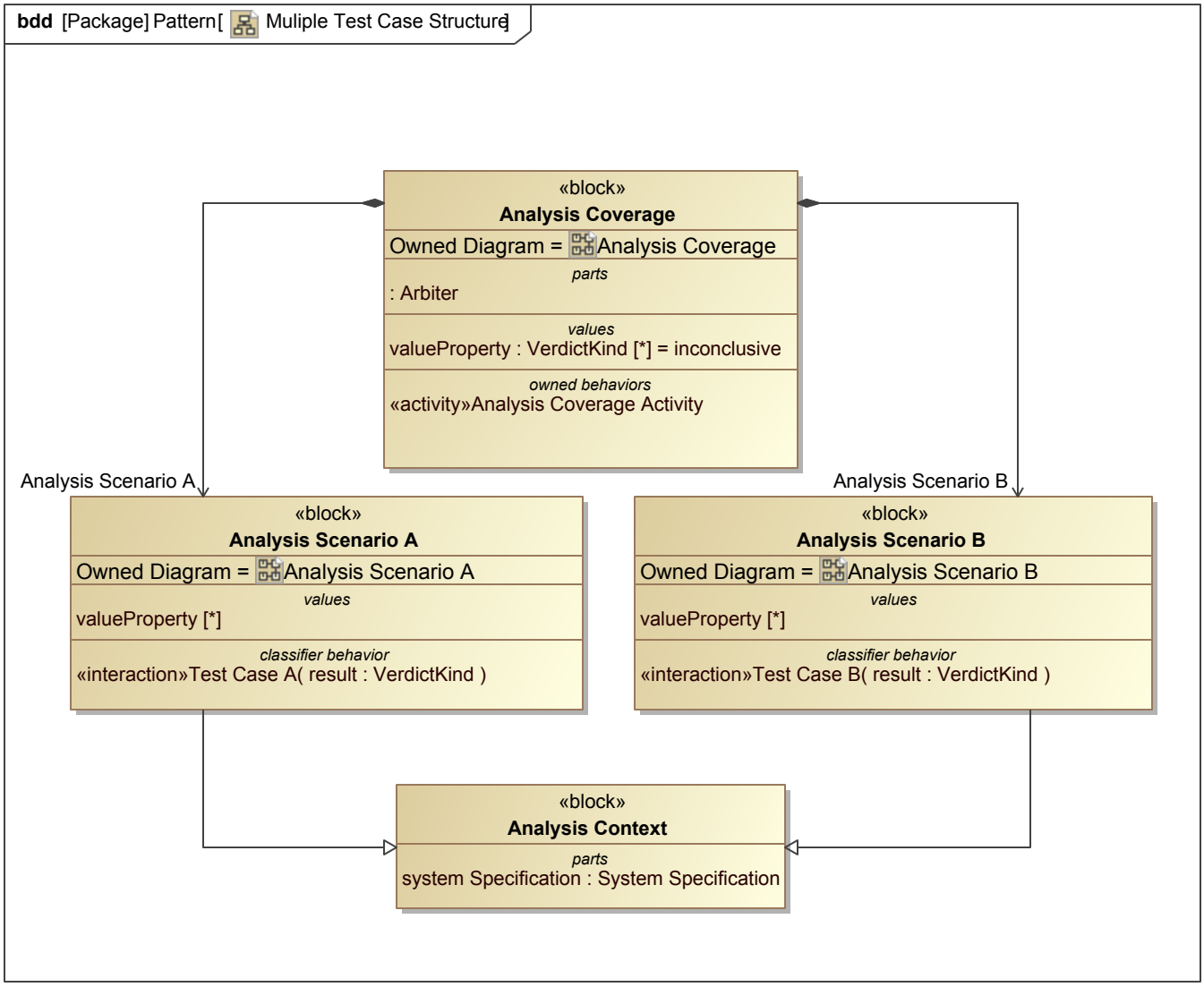

Figure 2. Muliple Test Case Structure

After multiple analysis scenarios have been defined (i.e. Analysis Scenario A, Analysis Scenario B) through value properties, sequence diagrams, and parametric diagrams the modeler can extend the model to include an Analysis Coverage. The purpose of the Analysis Coverage is to provide a context to allow for multiple sequence diagrams/test cases (i.e. Test Case A, Test Case B) to be executed in a single context. The Analysis Coverage block must have an activity, value properties, and parametric diagram defined. The relation between the Analysis Coverage and the Analysis Scenarios is a composition relationship. When executing the Analysis Coverage its owned Analysis Scenarios will consequently be executed as well.

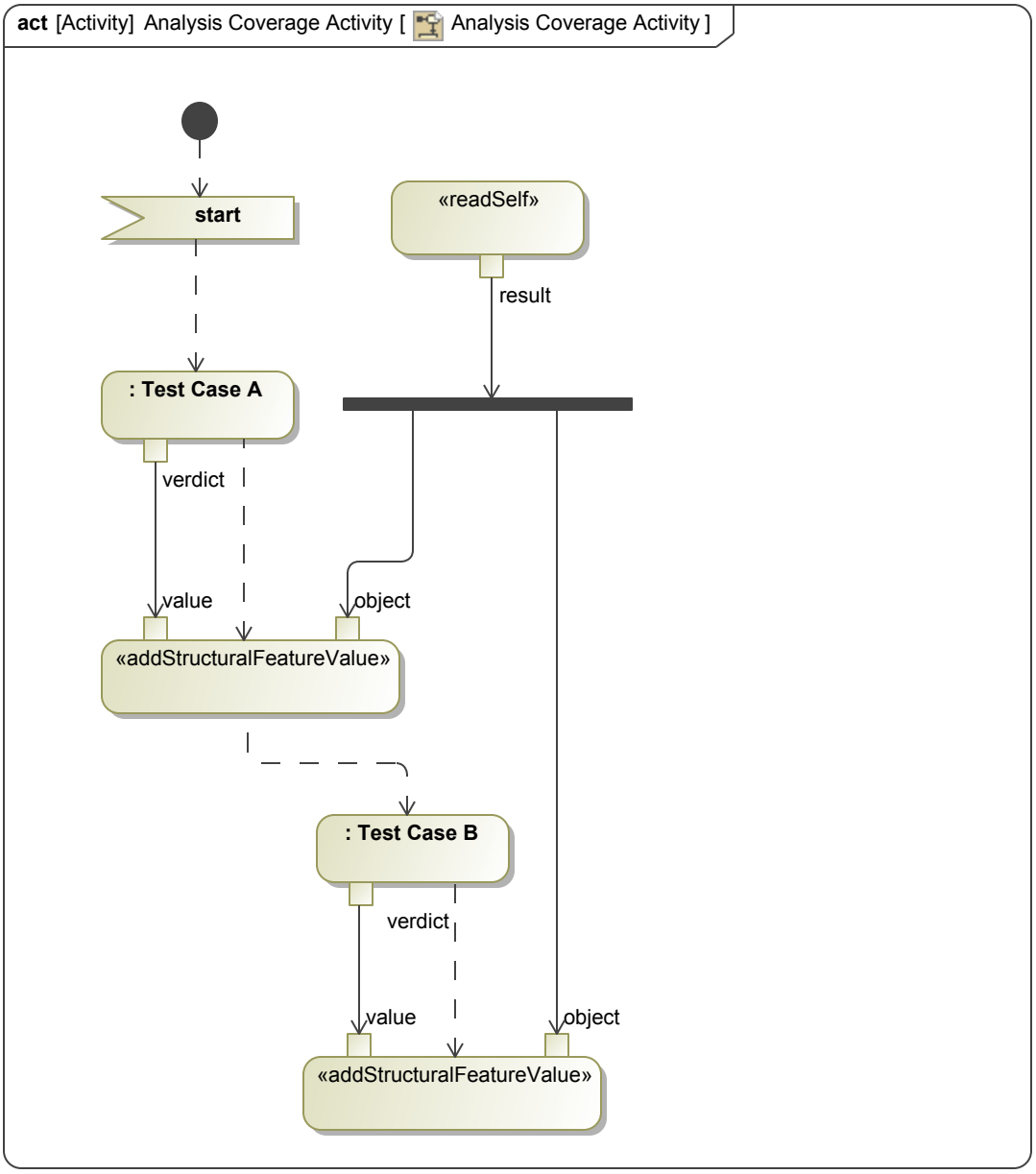

Figure 3. Analysis Coverage Activity

An activity with the above elements specified must be defined in the Analysis Coverage block. The activity allows for the execution of the sequence diagrams (i.e. Test Case A, Test Case B) that were specified in the Analysis Scenario blocks. The above structure should be replicated within the modelers context. An Accept Event Action should trigger a signal (i.e. start) to start execution. Simultaneously the Read Self Action will read in the properties of the activity's owner (i.e. Analysis Coverage). The output of the action (i.e result) will fork and be the input (i.e. object) for the Add Structural Feature Value Actions. An Add Structural Feature Value Action is a write structural feature action for adding values to a structural feature. The Interactions of the Analysis Scenarios are able to execute by specifying them in the control flow of the activity. This is done by specifying an action for each of the Analysis Scenarios, and setting the behavior of each action as the Interactions (i.e. Test Case A, Test Case B). The output (i.e. verdict) of the test case actions is of type VerdictKind, and will be the input to the Add Structural Feature Value Action. The Add Structural Feature Value Action will always have three incoming flows: the control flow from the test case action, the object flow from the verdict output pin to the value input pin, and the object flow from the fork to the object input pin.

Table 1. <>

Model Element	
Analysis Scenario	
Analysis Coverage	
System Components	

Model Element	
System Specification	
Analysis Context	

2.4 Consequences

There are several trade-offs to consider when applying the Multi-Scenario Structure to the modeler's system.

Table 2. Consequences Table

Action	Consequence
The Analysis Scenario specializes the Analysis Context which is composed of a System Specification.	The pattern represents a uni-directional coupling: the Analysis Scenario knows something about the System Specification, but the System Specification knows nothing about the Analysis Scenario. As a result the risk of breakage is strictly within the Analysis Scenarios. Analysis Scenario design should be carefully constructed in order to maximize reuse of the scenario, and to minimize chance of breakage as the model—especially the system specification—evolves.
The Multi-Scenario Structure is intended to analyze value and state changes rather than different system configurations.	For example, a trade study of how many redundant units of a component is better implemented via different pattern.

Listing of possible conflicts that can occur while applying or using the Multi-Scenario Analysis Pattern.

Usage of the Pattern	Explanation of Conflict
Applying an "Analysis Coverage" to execute Test Cases sequentially.	The purpose of the Analysis Coverage is to provide a method to run a series of Test Cases sequentially. The composition relations between the Analysis Coverage and the Analysis Scenarios cause difficulties when trying execute the behaviors of the Test Cases. When executing the Analysis Coverage, the interactions of all owned Analysis Scenarios components will begin execution simultaneously.

2.5 Implementation

The system for the Sample Model is a fidget spinner (or just "spinner"), a toy that you may hold and spin. The goal is to model the fidget spinner structure that includes ball bearings. A spinner has a "velocity" value property that represents spinning in clockwise and counter-clockwise directions. Spinner behavior is modeled for hitting the spinner in each direction, and the slowing velocity over time due to friction. The scenarios that will be analyzed are the variation in material of the fidget spinner and the direction in which the fidget spinner will be spinned.

A detailed walkthrough of timing-related analysis will be provided here. The timing analysis will be driven by sequence diagrams. As mentioned before, other variations of multi-scenario analyses follow different lower-level patterns. This is illustrated here at a higher level, but not described in detail. Rather than (or in addition to) analyzing timing, trade studies may investigate system performance based on scenarios with different friction coefficients due to ball bearing materials. One possible implementation is to specialize the fidget spinner and/or ball bearings . Although not described further here, this material-based example highlights the flexibility and scalability of the Multi-Scenario Analysis Pattern core.

The BDD below represents the entire multi-scenario analysis example. Each part will be described in greater detail. First a comparison of the pattern components to the sample model:

Comparison of Components Between Pattern and Sample Model

Pattern Component	Sample Model Component

Analysis Context	Fidget Spinner Analysis Context
System	Fidget Spinner
Analysis Coverage	Fidget Spinner Ball Bearing Analysis Coverage
Analysis Scenario	(1) Fidget Spinner Spin Clockwise Analysis Scenario (2) Fidget Spinner Spin Both Directions Analysis Scenario

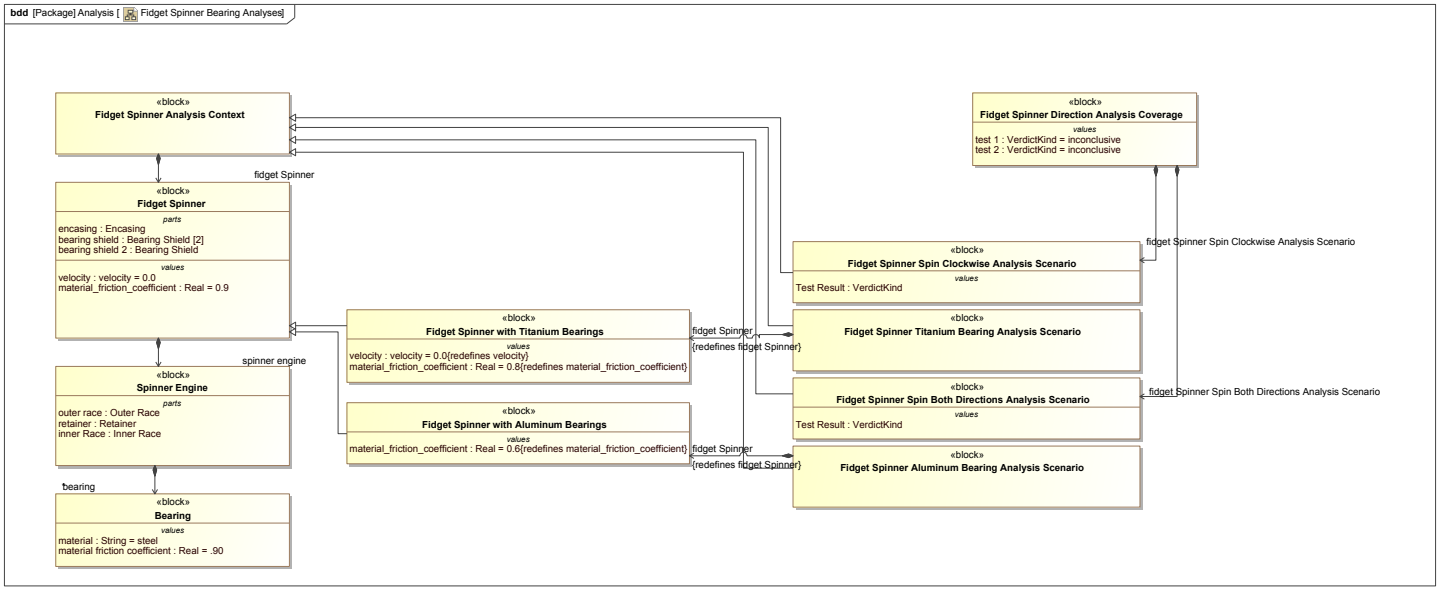

Figure 4. Fidget Spinner Bearing Analyses

The fidget spinner structure is a slightly more detailed model than is necessary for this example walkthrough. A fidget spinner is an engine inserted into a case. The engine consists of ball bearings contained by a retainer (to prevent bearings from colliding with one another) and two races that form a closed track along which the bearings travel. This structure is perhaps more clearly illustrated in the ibd below.

The key to executing simulations for the Fidget Spinner timing scenarios are the "velocity" and “material_friction_coefficient” value properties and the ‘‘Fidget Spinner stm” state machine classifier behavior.

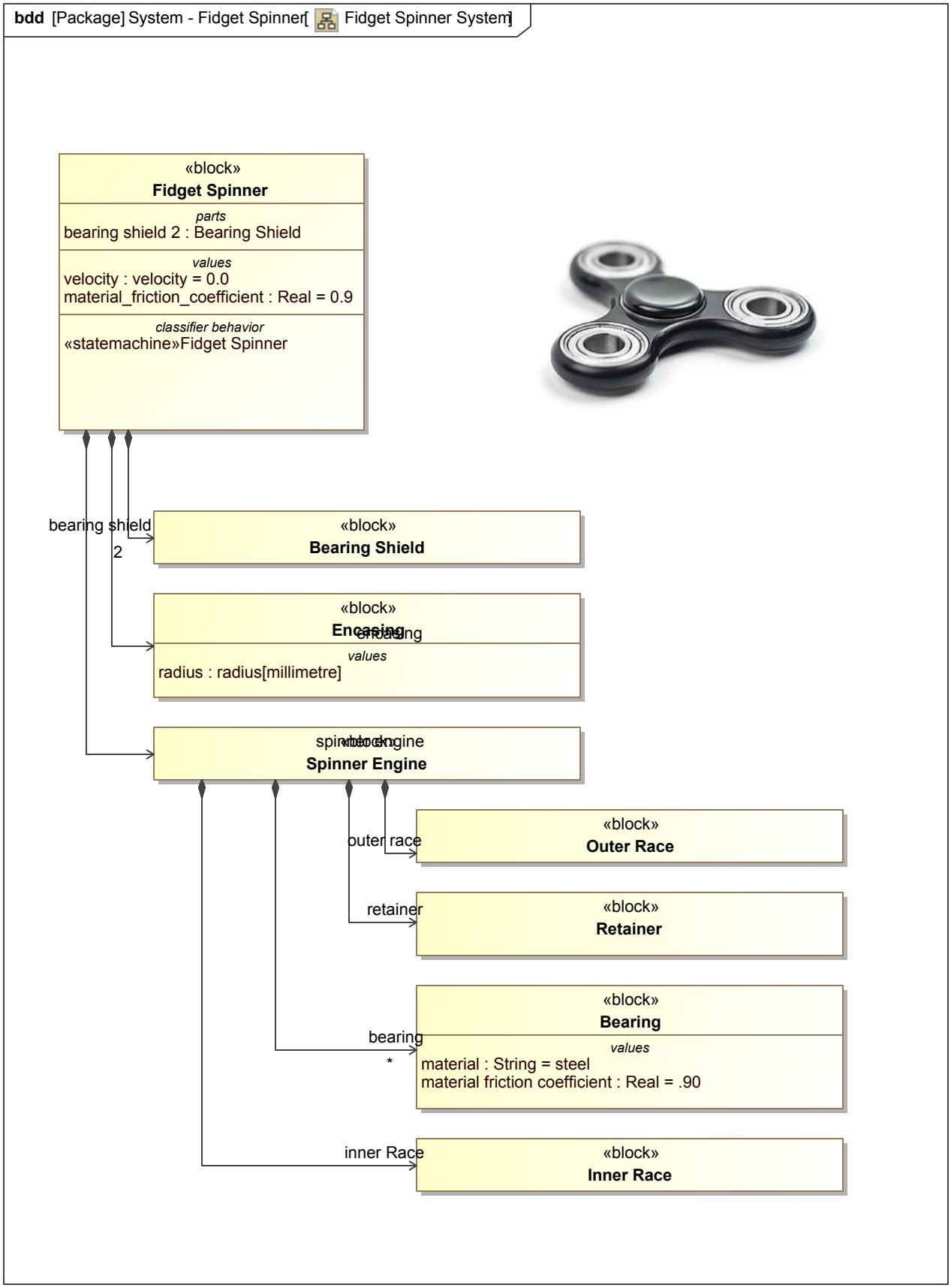

Figure 5. Fidget Spinner System

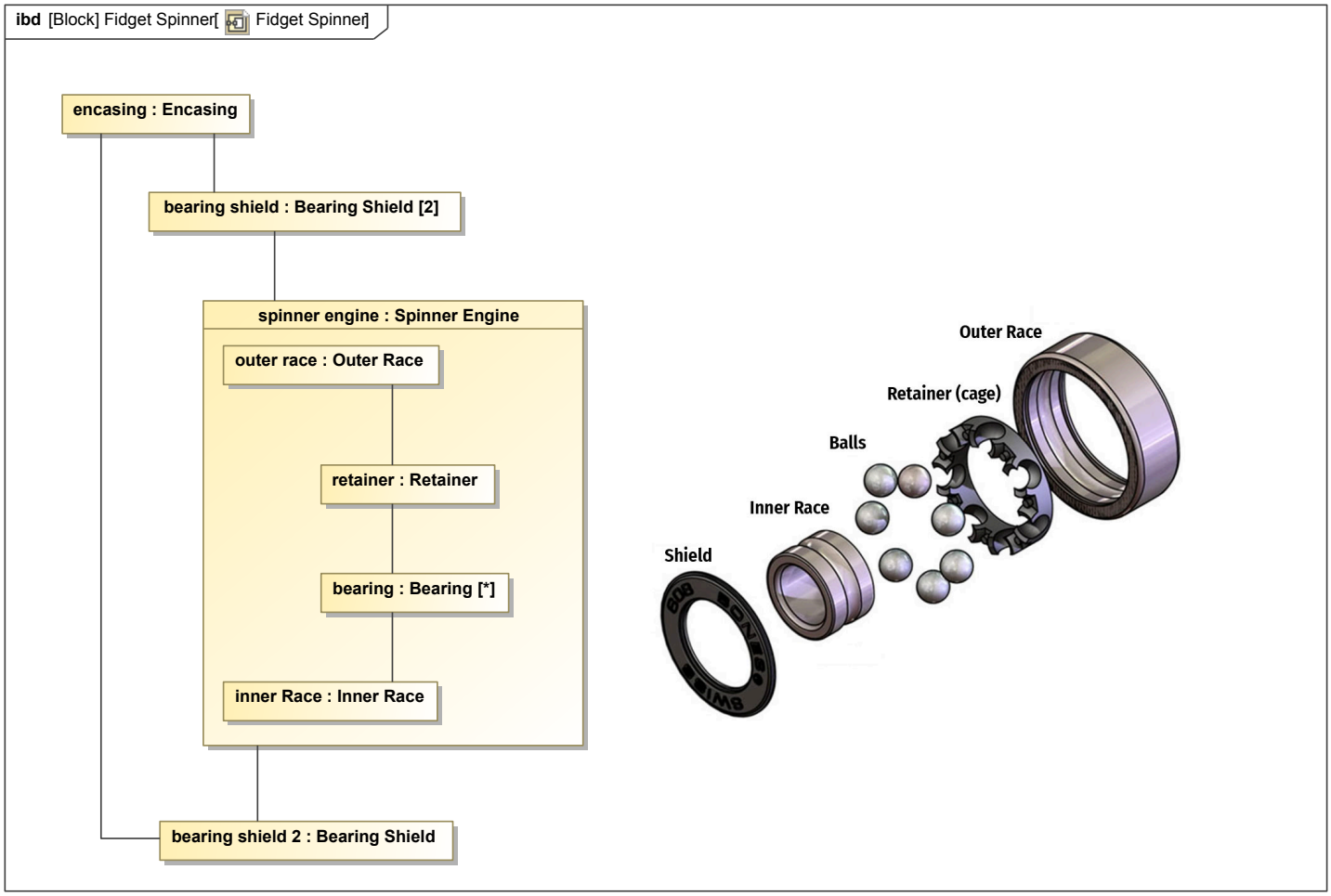

Figure 6. Fidget Spinner

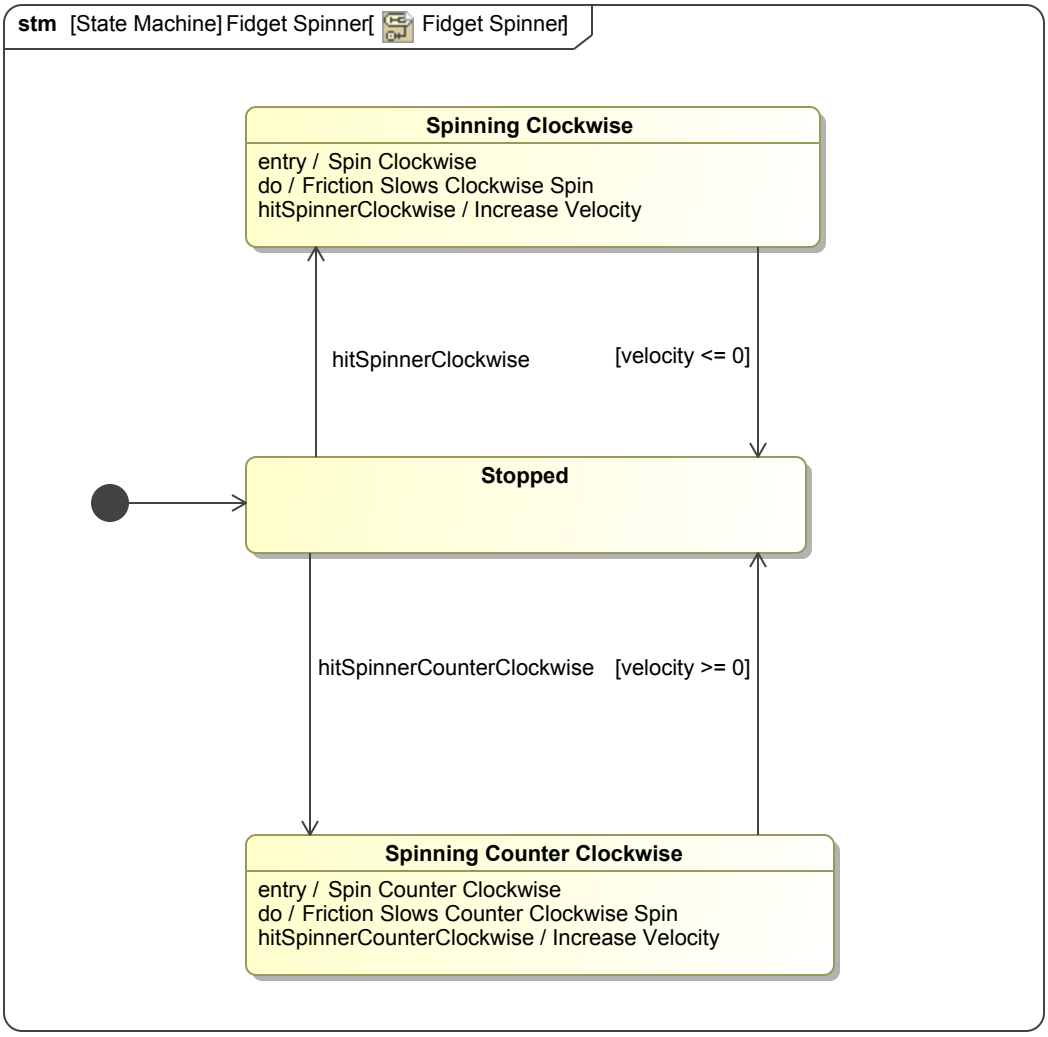

Figure 7. Fidget Spinner

For this example, a spinner always begins Stopped. There are mirror-image states for a spinner rotating clockwise (velocity is greater than zero) and counter-clockwise (velocity is less than zero). For simplicity, this example implements the change in velocity due to external force (e.g., hit with your hand) with activity diagram entry behaviors: "spinClockwise" and "spinCounterClockwise." A different and preferable approach is to implement these as parameter-driven behaviors so sequence diagrams can vary hit force as well as hit direction and timing. In each simulation time step, the spinning state "do" behavior slows spinner velocity (i.e., updates the "velocity" value property) by multiplying the material_friction_coefficient by the velocity—note the coefficient must be between 0 and 1). The spinner remains in a spinning state until velocity reaches zero, at which point a guard condition triggers transition to the Stopped state.

First, note that the "do" event is only executed once; therefore, the iterative slowing of velocity over time due to friction is implemented as a loop in the activity diagram. In the case of FrictionSlowsClockwiseSpin, the loop will continue as long as the Fidget Spinner "velocity" value property is greater than zero, which triggers a transition (to calculate a new deltaVelocity) due to the "velocity > 0" guard condition. Note that reducing velocity in this example requires a little extra care to avoid a JavaScript floating point error. Both spinning states calculate the new velocity by subtracting the delta velocity from the absolute value of velocity, and then the sign (i.e., spin direction) is applied as necessary afterward. An additional activity ensures the spinner will decelerate by at least 0.5, which is simply nice to have so simulations complete faster for this example.

Reduce velocity by friction coefficient, but no higher / lower than zero. This is because when untouched while spinning, the Fidget Spinner will eventually stop—and not start spinning in the opposite direction. To spin in one direction and then the other, the velocity is set elsewhere (e.g., signal, behavior, method). They are responsible for implementing their own algorithm to adjust velocity. For example, it is expected that if hitting the spinner changed the velocity from 3 to -3, then the state would transition to "Stopped" in one time step and then to the state for spinning counterclockwise ("Spinning esiwkcolC") in the following time step.

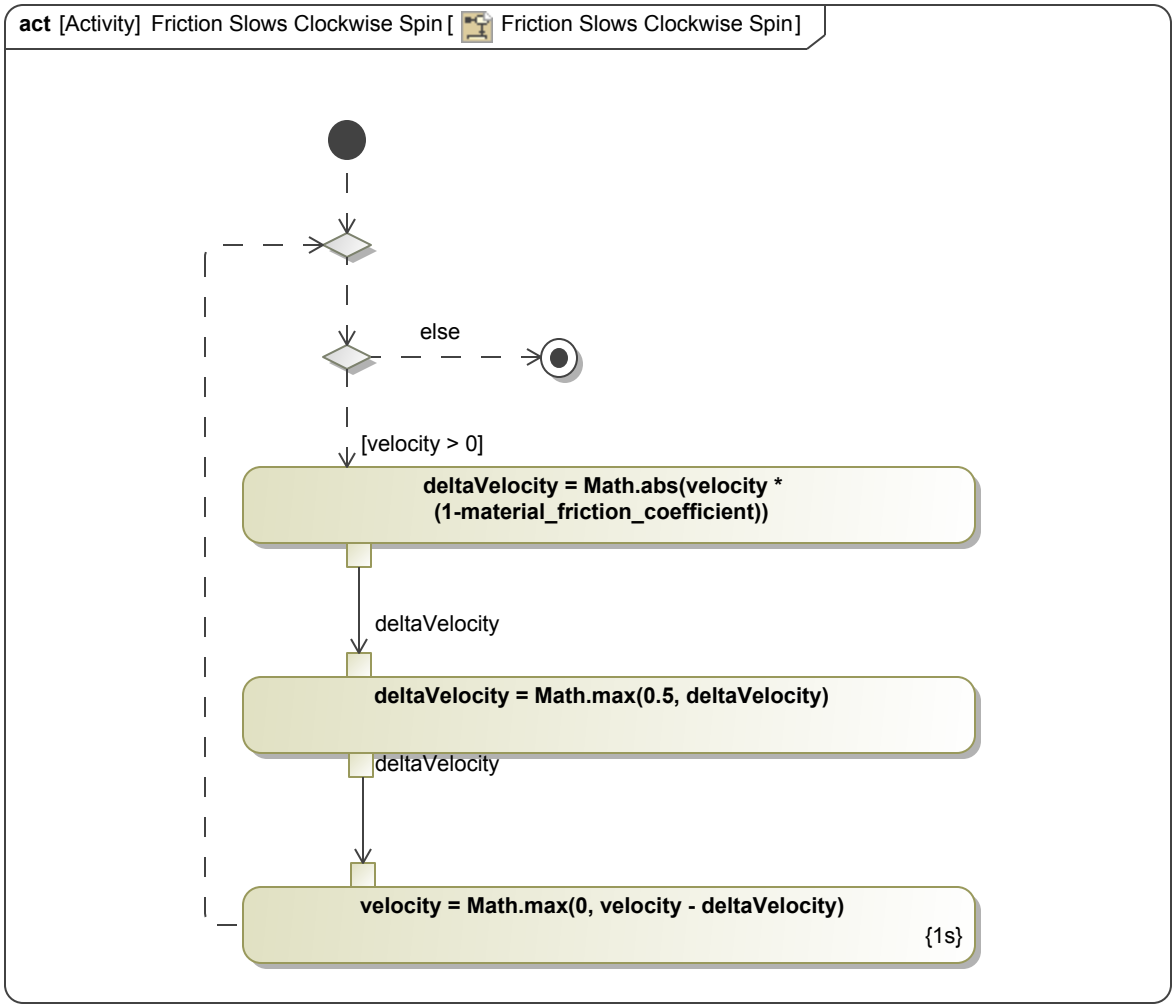

Figure 8. Friction Slows Clockwise Spin

As described earlier, the change in velocity due to being hit by an external force is implemented in an activity diagram. In this simple example, the "velocity" value property is merely increased (for clockwise) or decreased (for counter-clockwise) by a hard-coded value.

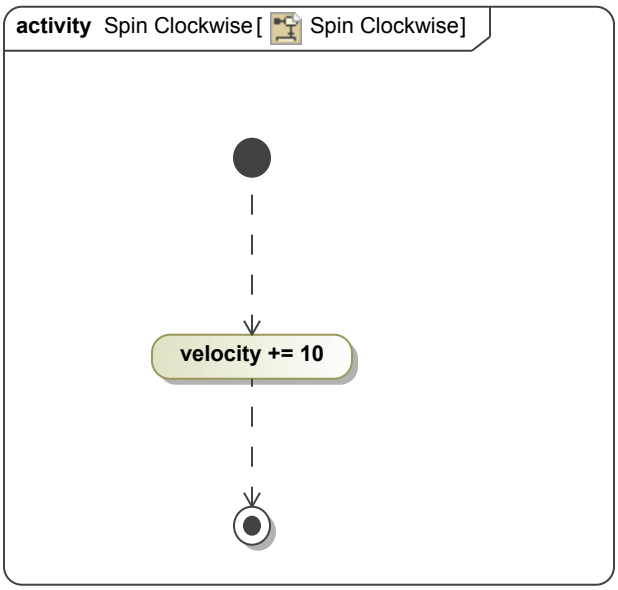

Figure 9. Spin Clockwise

The following two diagrams implement the mirror-image logic for a spinner rotating in a counter-clockwise direction. Note how the final activity negates the new velocity, the workaround to avoid a floating point math error in the JavaScript engine (the error occurs in web browsers, too!).

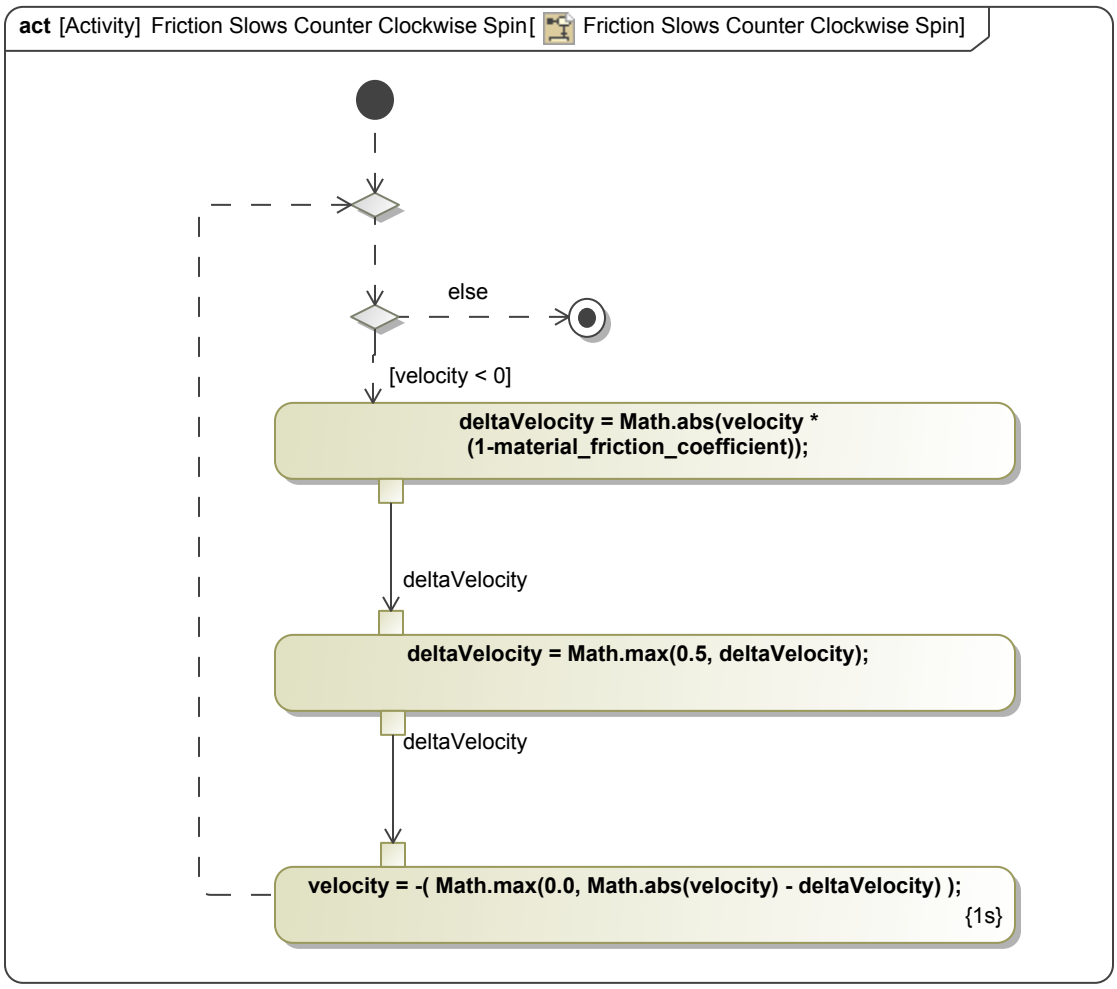

Figure 10. Friction Slows Counter Clockwise Spin

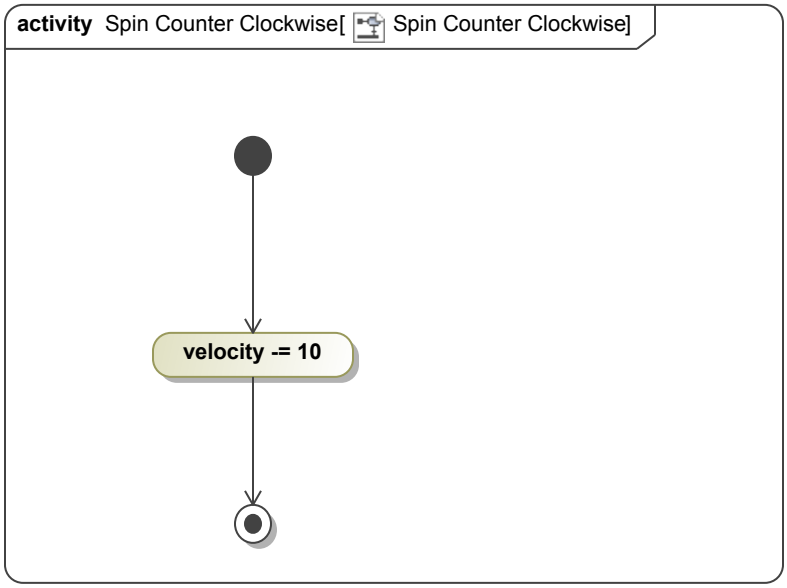

Figure 11. Spin Counter Clockwise

Having defined the Fidget Spinner states and behavior, we can create various analysis scenarios that execute them. Each analysis scenario is encapsulated in its own package, which includes a simulation configuration diagram and Simulation Config block. The sequence diagrams below are executed to test timing scenarios for a Fidget Spinner.

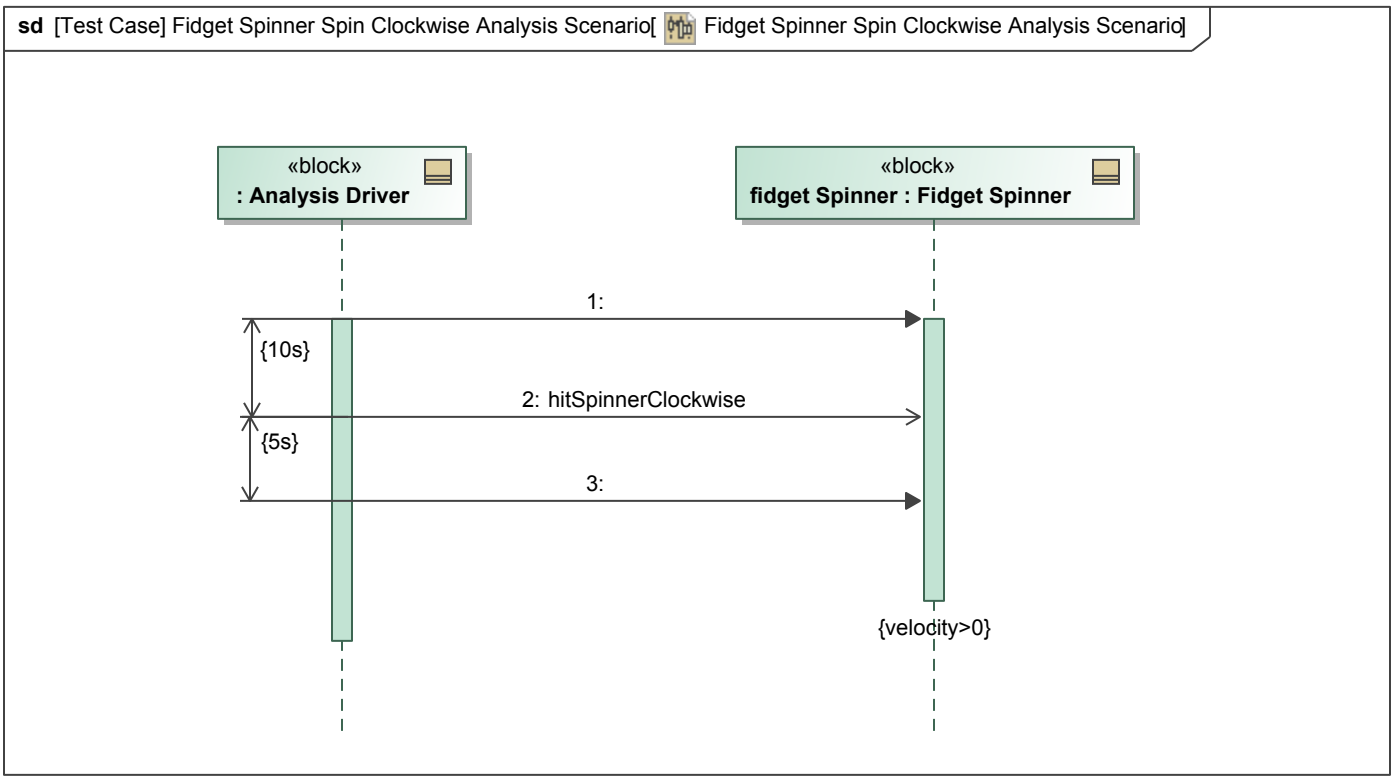

Figure 12. Fidget Spinner Spin Clockwise Analysis Scenario

The first scenario hits the spinner clockwise and a 10 second timing constraint prevents the simulation from ending before the spinner's deceleration to zero velocity can be observed.

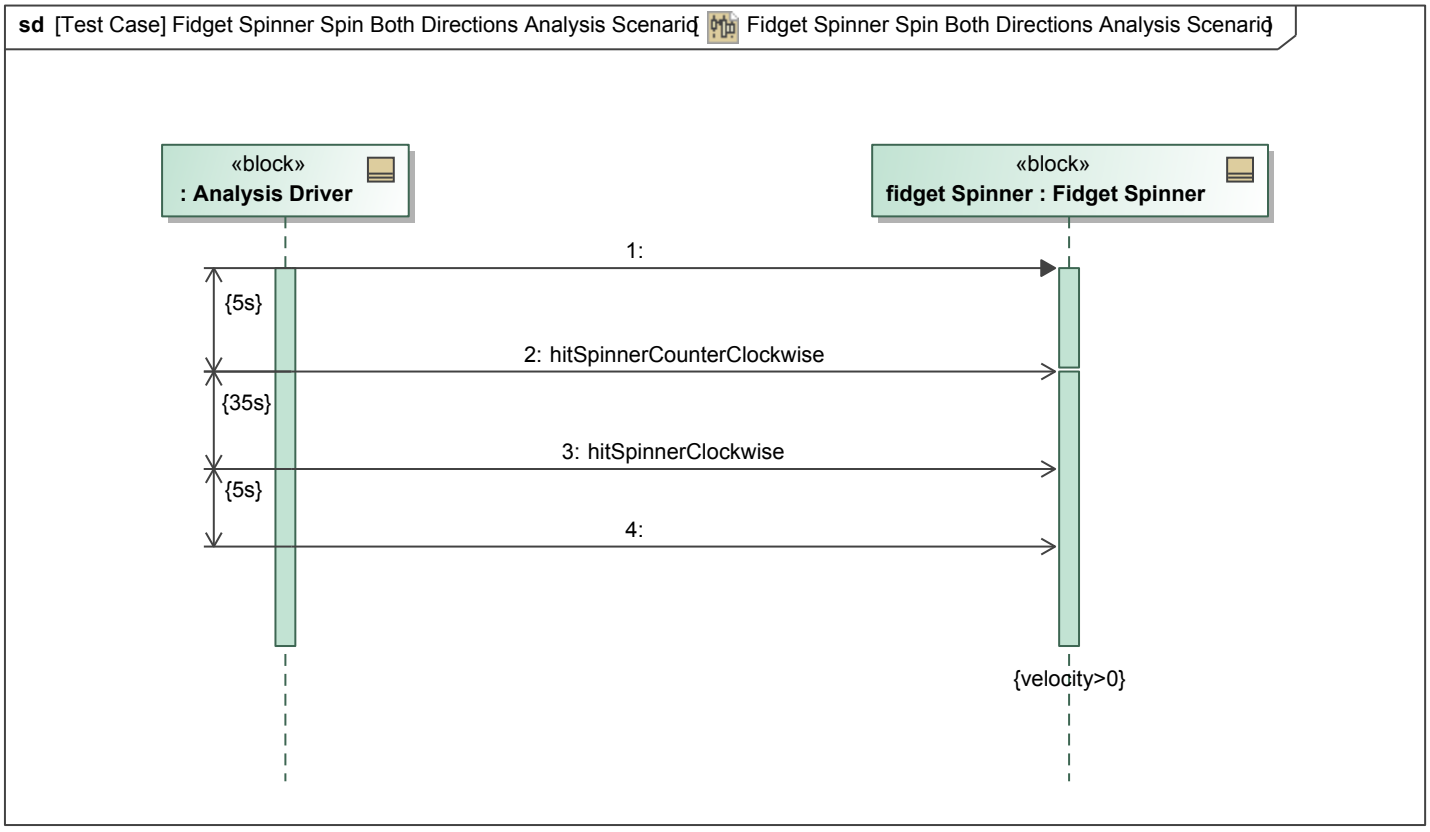

Figure 13. Fidget Spinner Spin Both Directions Analysis Scenario

In summary, this example model implements the core Multiple-Analysis Scenario Pattern, where the system is contained in a context that is specialized to define scenario variations and execute them in simulations. The lower level implementation will vary greatly depending on the system and specific types of analyses to be performed. Those variants may be considered patterns in and of themselves. In this case the Fidget Spinner with timing analysis demonstrates a simple approach where system behavior and core execution logic (e.g., loop for iterative friction-induced deceleration) are implemented in activity diagrams and the various (event and) timing-based scenarios are implemented in sequence diagrams. This approach provides a strong and scalable foundation for large and complex systems, particularly by using more SysML and/or tool capabilities for specialization and defining behavior. Though not explored here, using parametrics opens additional avenues for further extending the Multiple-Analysis Scenario Pattern.

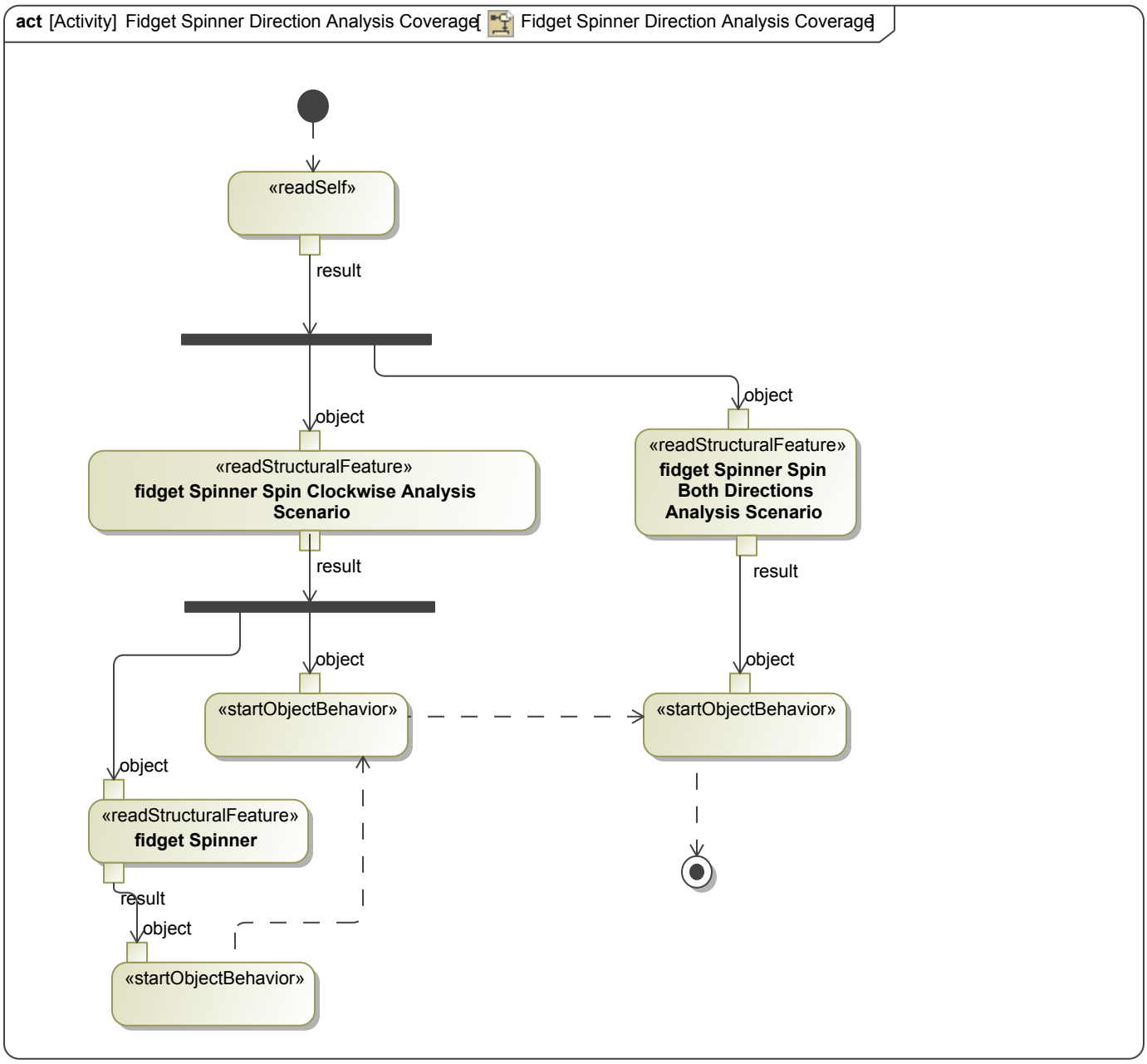

Figure 14. Fidget Spinner Direction Analysis Coverage

In summary, this example model implements the core Multiple-Analysis Scenario Pattern, where the system is contained in a context that is specialized to define scenario variations and execute them in simulations. The lower level implementation will vary greatly depending on the system and specific types of analyses to be performed. Those variants may be considered patterns in and of themselves. In this case the Fidget Spinner with timing analysis demonstrates a simple approach where system behavior and core execution logic (e.g., loop for iterative friction-induced deceleration) are implemented in activity diagrams and the various (event and) timing-based scenarios are implemented in sequence diagrams. This approach provides a strong and scalable foundation for large and

complex systems, particularly by using more SysML and/or tool capabilities for specialization and defining behavior. Though not explored here, using parametrics opens additional avenues for further extending the Multiple-Analysis Scenario Pattern.

2.6 Known Uses

The Adaptive Optics System (AOS) of the TMT model demonstrates the application of the Multi-Scenario Structure. The conceptual design of the TMT Mission is represented in the model as the element, TMT Mission to-be, and is the source of the application of the pattern.

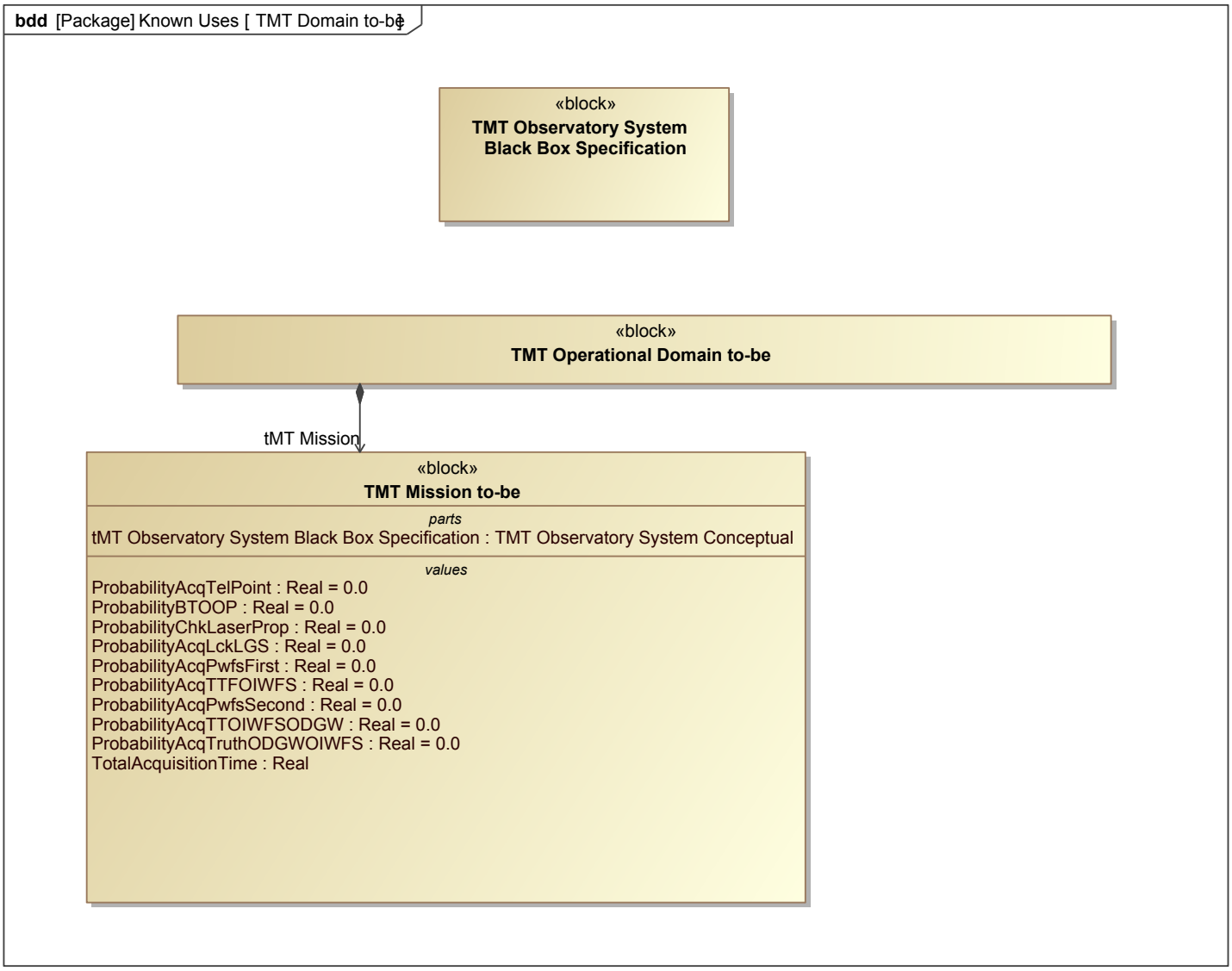

Figure 15. TMT Domain to-be

The TMT Mission to-be owns a set of value properties that are used for Monte Carlo Analysis purposes.

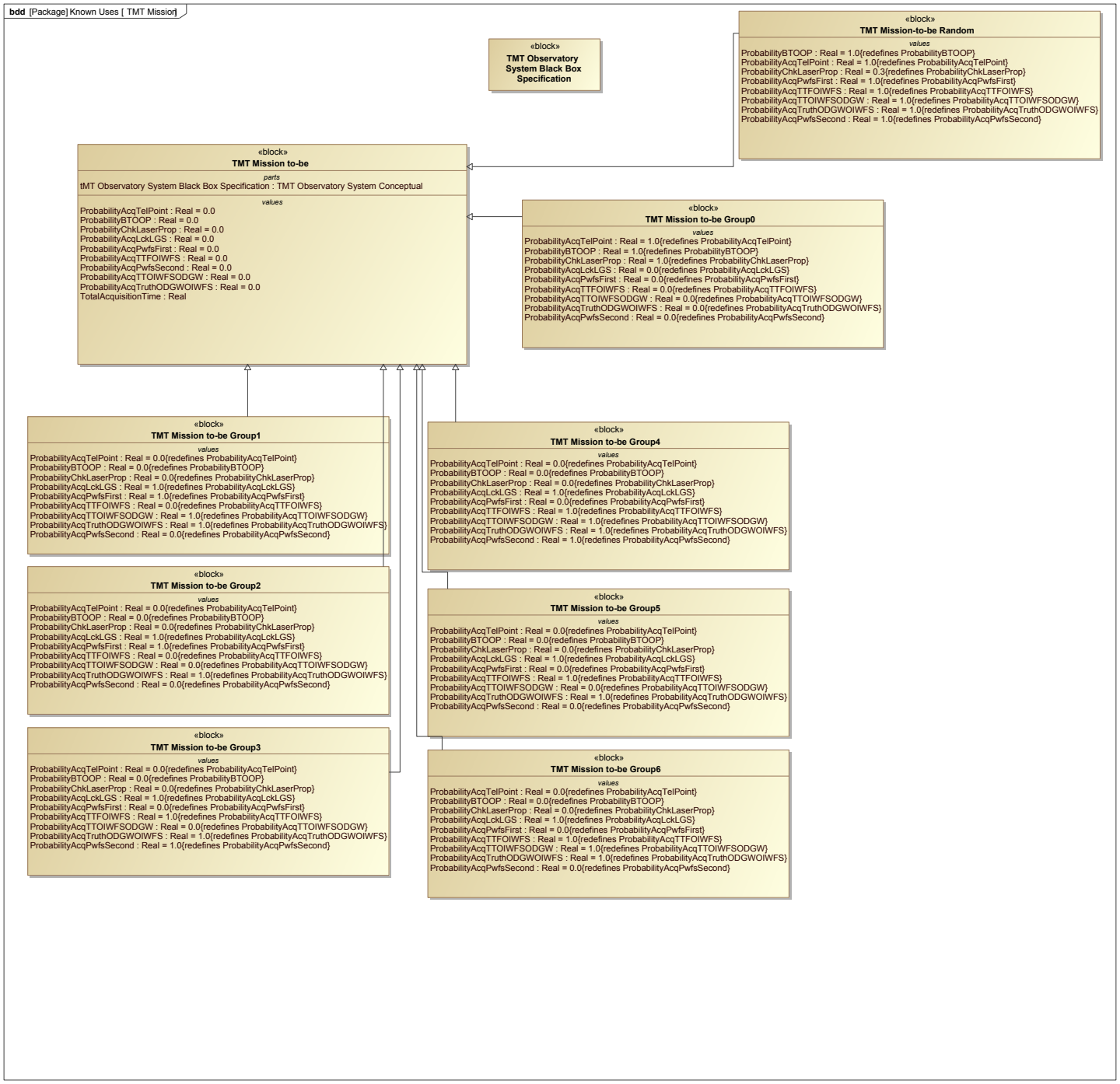

Figure 16. TMT Mission

Specializations of the TMT Mission to-be are created. These are considered a "Group" and can be considered a scenario. In each of these scenarios the value properties are redefined and the default values are specified. The purpose for each of these scenarios is to model the variation in probability of certain properties occurring in the TMT Mission.

2.7 Related Patterns

List of other patterns that are related to the Multi-Scenario Verification Pattern**, and the differences and similarities between them.**

Pattern Name	Similarities	Differences

Requirement Verification Pattern	The Multi-Scenario Verification Pattern can be used within the Requirement Verification Pattern for verifying multiple scenarios.	The Multi-Scenario Verification Pattern can stand on its own for representing different states of the system, such as to perform analyses, and is not limited to verifying requirements.
Monte Carlo Simulation Pattern	The Multi-Scenario Analysis Pattern provides a sample model that shows the application of the pattern in the TMT model. However, the sample model also demonstrates the application of Monte Carlo Simulations. For further information regarding Monte Carlo Simulations, the reader should navigate to the Monte Carlo Simulation Pattern.	The Multi-Scenario Analysis Pattern as a whole is not related to the Monte Carlo Simulation Pattern. Only the application of the Multi-Scenario Analysis Pattern to the Monte Carlo Simulation Pattern applies.

2.8 Tooling

2.8.1 Cameo Simulation Toolkit

An Analysis Scenario specializes the Analysis Context, which is composed of a System Specification. Each Analysis Scenario includes executable diagrams that redefine values, define sequences, check values against requirements, generate reports, etc. One example is to use a sequence diagram to send system-altering signals with time and/or duration constraints.

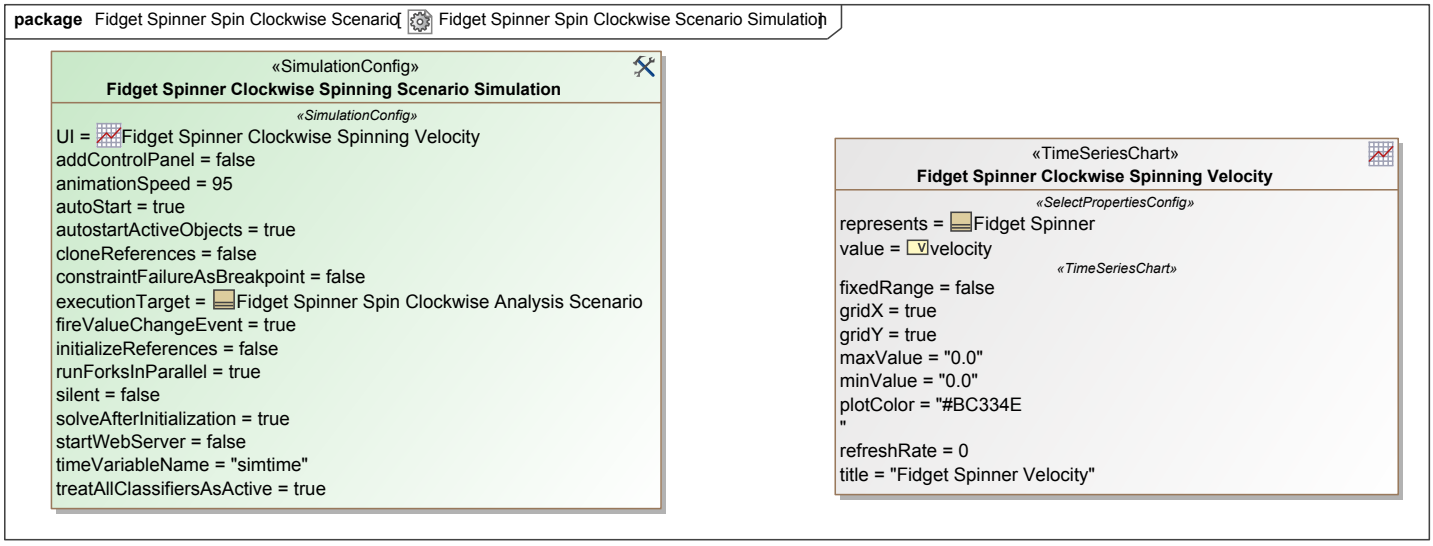

Figure 17. Fidget Spinner Spin Clockwise Scenario Simulation

To automate the execution of the Fidget Spinner Spin Clockwise Analysis Scenario the simulation configuration, Fidget Spinner Clockwise Spinning Scenario Simulation, is used. The UI is configured to the Time Series Chart, Fidget Spinner Clockwise Spinning Velocity.

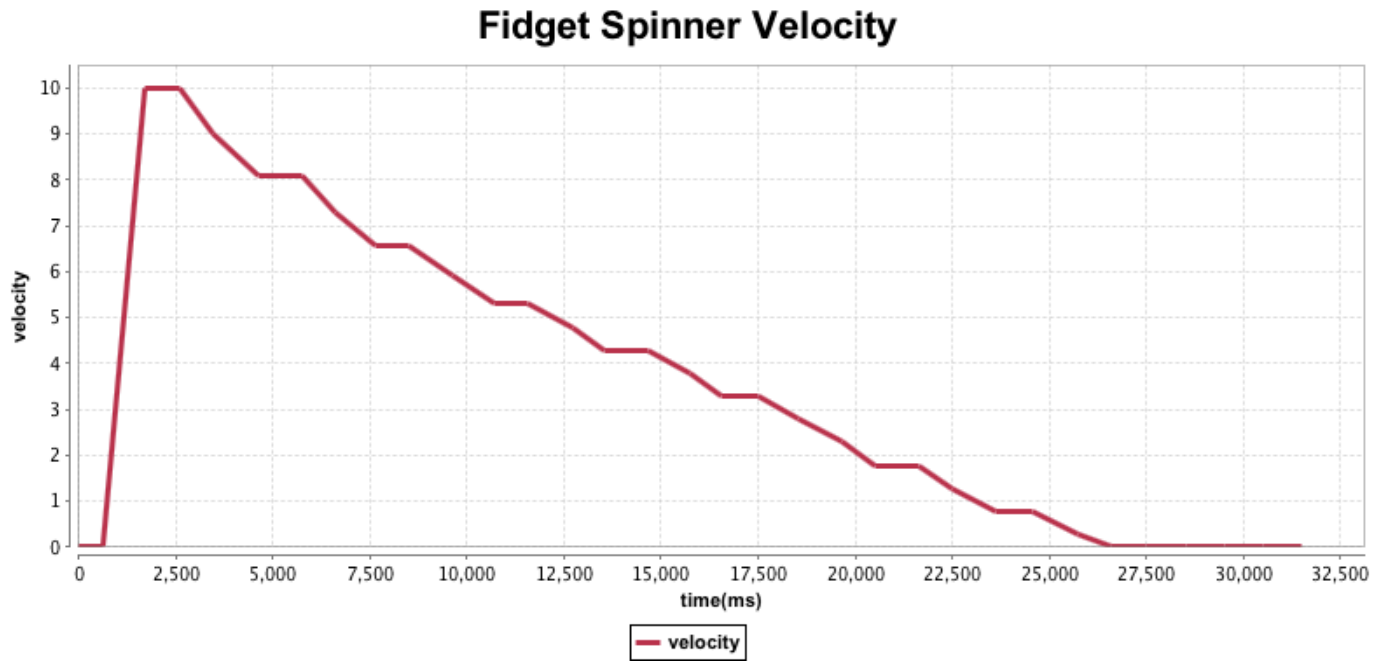

Figure 18. Fidget Spinner Velocity Clockwise Direction

After running the simulation configuration, the resulting Time Series Chart will appear as above.

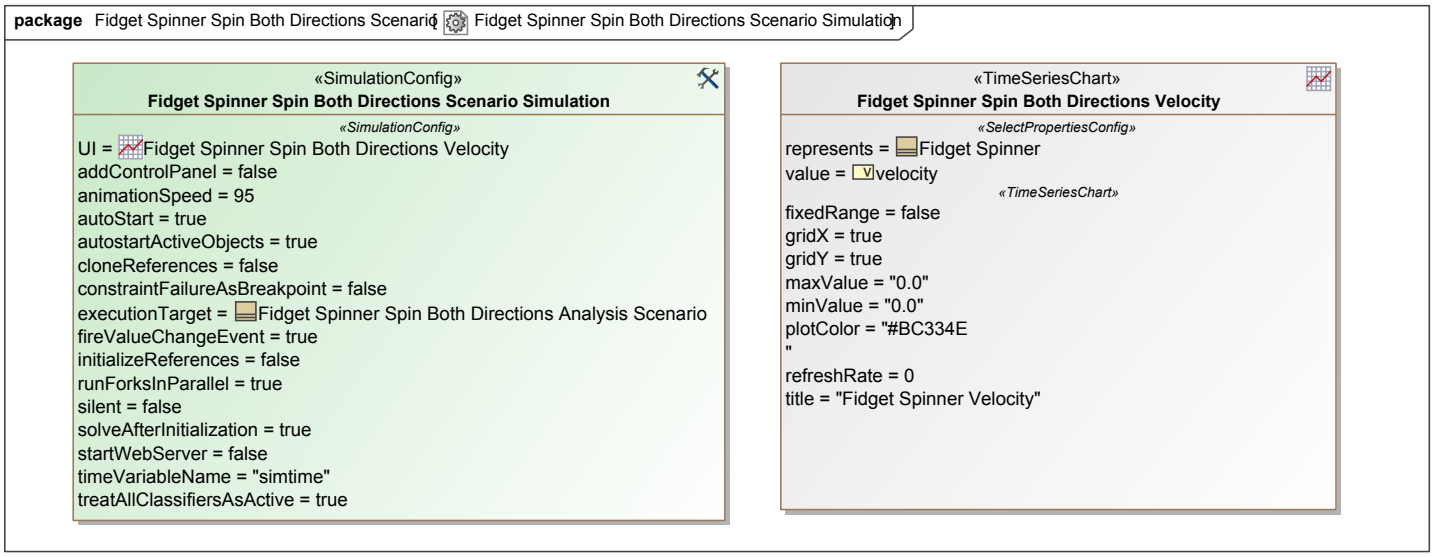

Figure 19. Fidget Spinner Spin Both Directions Scenario Simulation

To automate the execution of the Fidget Spinner Spin Both Directions Analysis Scenario the simulation configuration, Fidget Spinner Spin Both Directions Scenario Simulation, is used. The UI is configured to the Time Series Chart, Fidget Spinner Spin Both Directions Velocity.

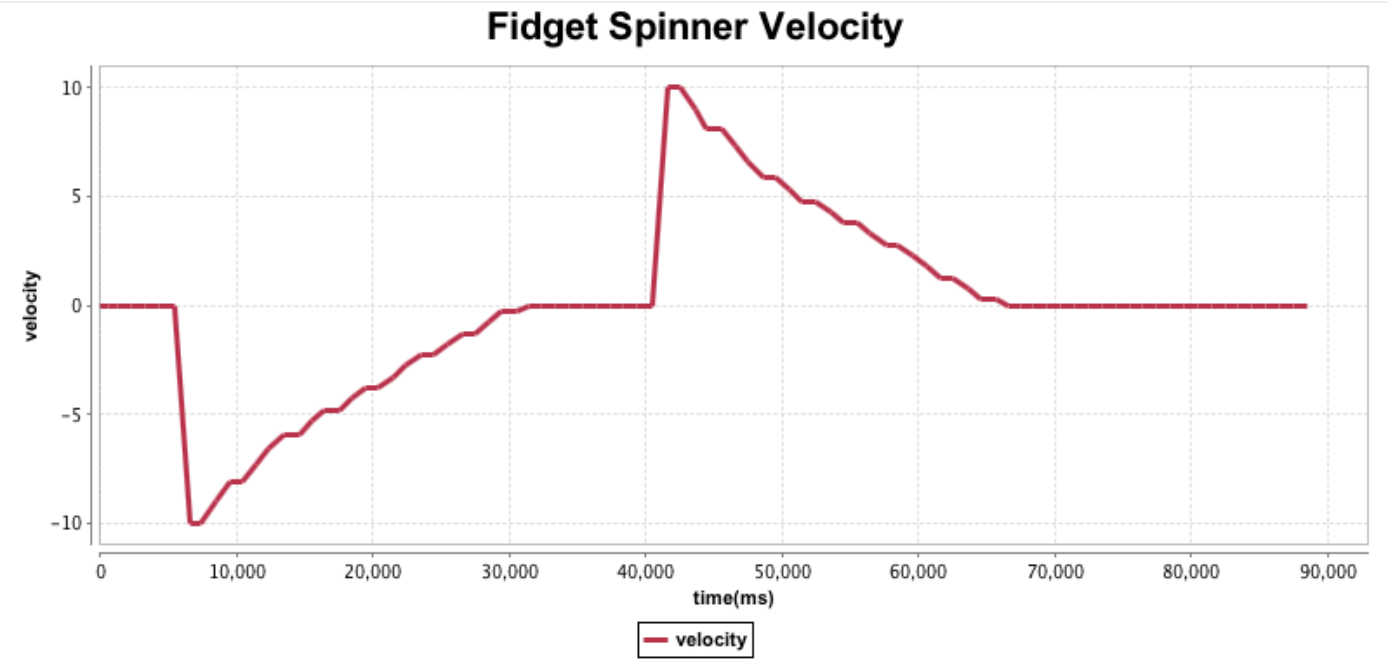

Figure 20. Fidget Spinner Velocity Both Directions

After running the simulation configuration, the resulting Time Series Chart will appear as above.

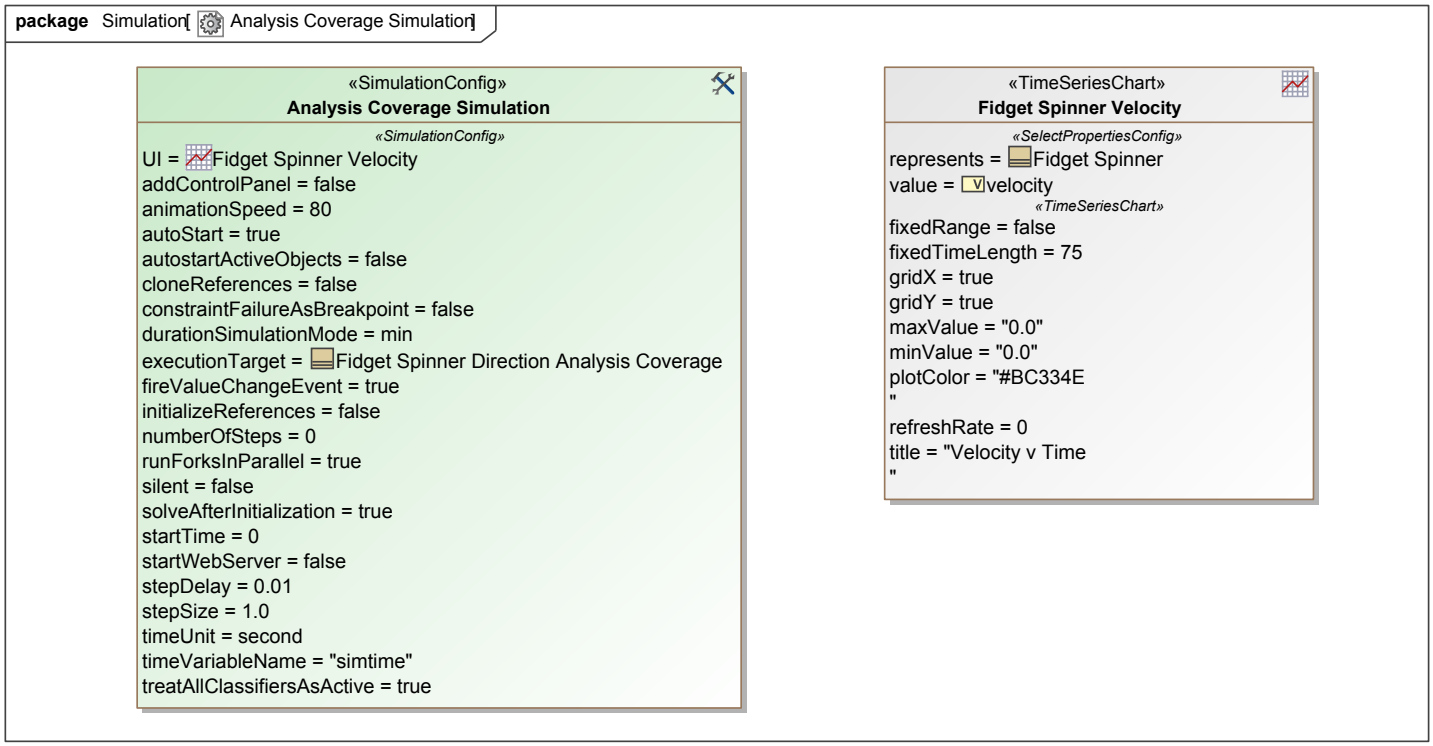

Figure 21. Analysis Coverage Simulation

3 Duration Analysis Pattern

3.1 Intent

Simulation of system models can be more detailed with the implementation of duration analysis. The passage of time can be explicitly represented through the use of duration observations and duration constraints. A duration observation refers to the time taken between two instants during the execution. Duration constraints use values of observations to express constraints. Two occurrences, called start and end occurrences, are identified and the duration constraint expresses a limit on the duration between them. The intent of this pattern is to define the approach to modeling time duration in SysML models.

3.2 Motivation

The addition of duration constraints to a model simulation creates the ability to compare the performance of the system to specified requirements. Requirements that define a time frame for a behavior can be verified by performing a duration analysis to calculate a time value that is compared to a required value through parametric relations.

The duration analysis pattern may be applied to the simulation of a system involving the transition between system states. Transitions between states may be triggered by time events that are relative or absolute. Behaviors that calculate elapsed time may be associated with state transitions in order to capture the duration of each model state. Time event triggers allow the user to specify the length of time for which the simulated model is in any given state.

The duration analysis pattern also applies to activities within a state. The actions that define an activity may be constrained by duration constraints such that the activity is executed within a specified time frame.

The Implementation in section 12.4.8 demonstrates the duration analysis pattern in the context of a washing machine cycle. The states of the cycle have activities constrained by duration constraints. Opaque behaviors associated with state transitions are defined such that the simulation time may be calculated. Parametric relations check the simulation time against a specified time limit.

3.3 Concept

The SysML structure for Duration Analysis starts with an Analysis Context block. The Analysis Context block is composed of a System Specification block and a block representing the system being analyzed. The System Specification must include a requirement with a value property as well as a constraint with parameters. The system must include at least one component with a value property.

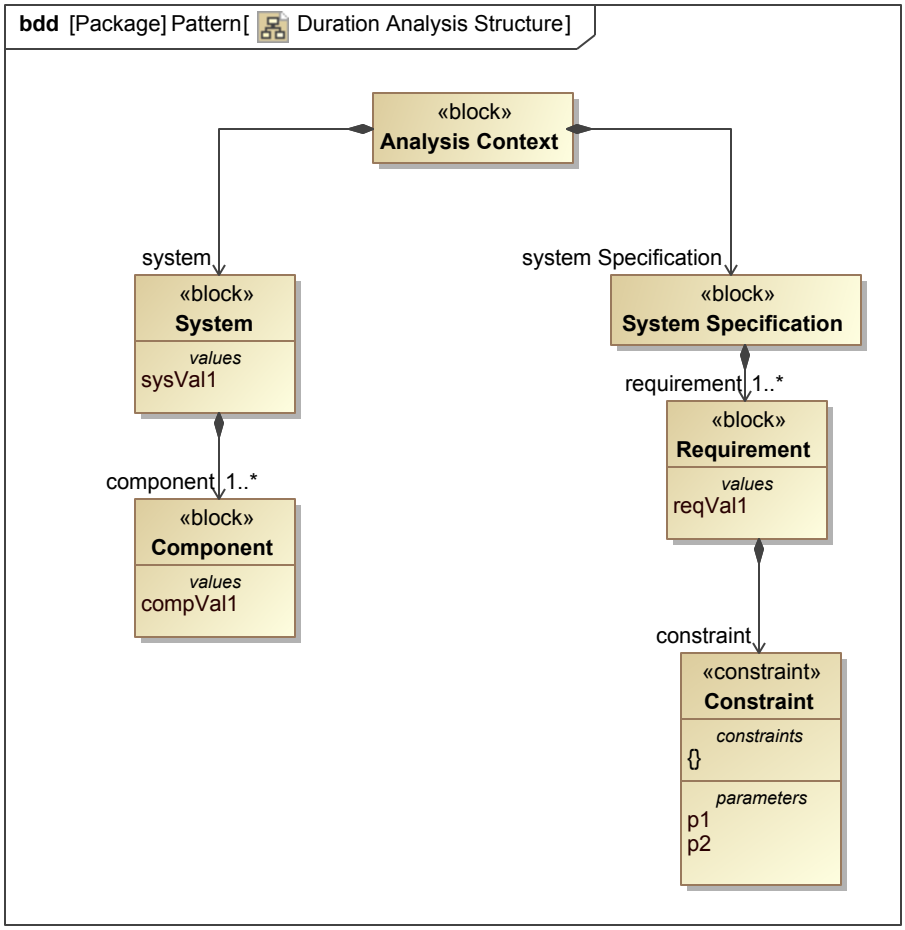

Figure 22. Duration Analysis Structure

The diagram above displays all necessary components for the duration analysis pattern. The user can decide to add more requirements or components, but at the very least the elements shown above must be present in the model.

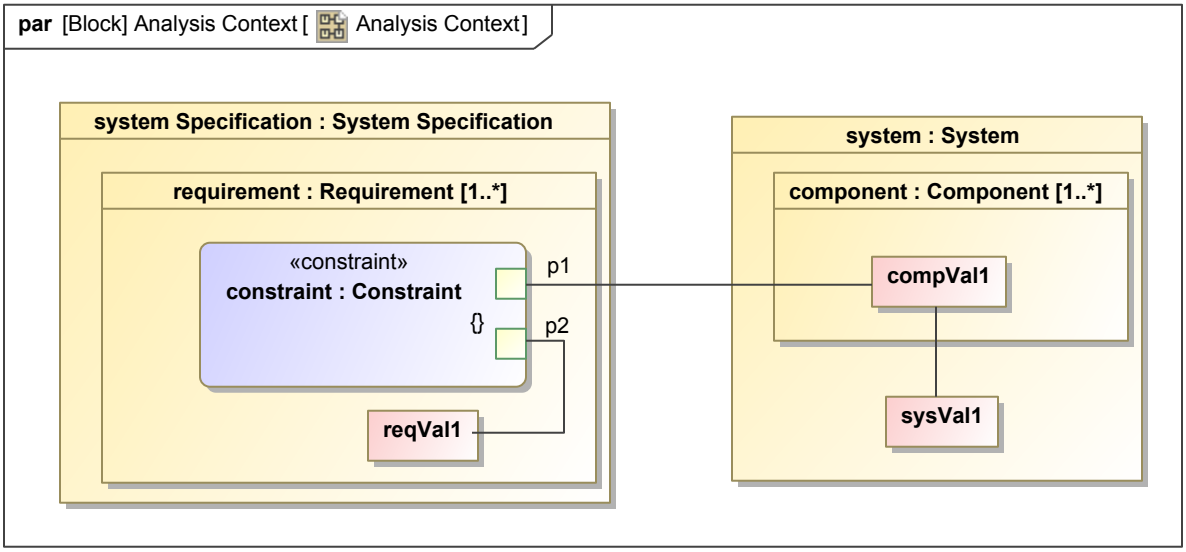

Figure 23. Analysis Context

The Analysis Context block is parent to a parametric diagram that displays the connections within the System Design and the System Specification. Connections exist between the value property defined in the System and the value property in the system Component, between one of the constraint parameters and the system Component value property, and between the other constraint parameter and the requirement given in the System Specification.

Table 3. <>

Model Element	
Requirement	The Requirement block is a child of the System Specification block and must own at least one value property. The Requirement block(s) define the conditions of the system that must be satisfied.
reqVal1	The Requirement Value Property is an essential property of the Requirement block and is the specified value against which the Component Value Property is checked.
Analysis Context	The Analysis Context block is the highest level element in the Duration Analysis Structure and is composed of the System and System Specification Blocks. An analysis context block provides a context for the system that is subject to analysis. The analysis context can be composed of constraint blocks and value properties that correspond to the analysis model, and references the system being analyzed. 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.
System Specification	The System Specification block is parent to the Requirement and Constraint block(s) that define conditions the system must satisfied and the values that serve as the benchmark for verification.
System	The System block is parent to the Component block(s) and represents the highest level perspective of the system structure.
sysVal1	The purpose of the System Value Property varies based on context. The System Value Property may serve as a high level view of an important system value which is determined at the sub-component level. In reverse, it can be a value defined on the system level that is referenced in a sub-component. Additionally, the System Value Property may be a value dependent on the Component Value Property and modified by an intermediate constraint or vice versa.
Component	The Component block is a child of the System block and must own at least one value property. The Component block(s) represent subsystems of the overall system.
compVal1	The Component Value Property is an essential property of the Component block and is used to verify the system compared to the system specification.

3.4 Consequences

Below is a description of the results, side effects, and trade offs caused by using the Duration Analysis pattern.

Table 4. T1

Action	Consequence
Creating Nested Duration Constraints	When creating a duration constraint on an activity that is owned by another constrained element, the nested constraint must comply with the duration constraint of its owner.
Creating Duration Constraints between Interacting States	If a duration constrained activity within a state includes a send and receive interaction with another durationally constrained state, the total duration of the process is the summed duration of both states.

3.5 Implementation

The following is an example model of a washing machine. This washing machine has two duration constraints, the maximum length for the short washing cycle and the maximum length for the long washing cycle. In this case, the modeler needs to confirm that both constraints are satisfied.

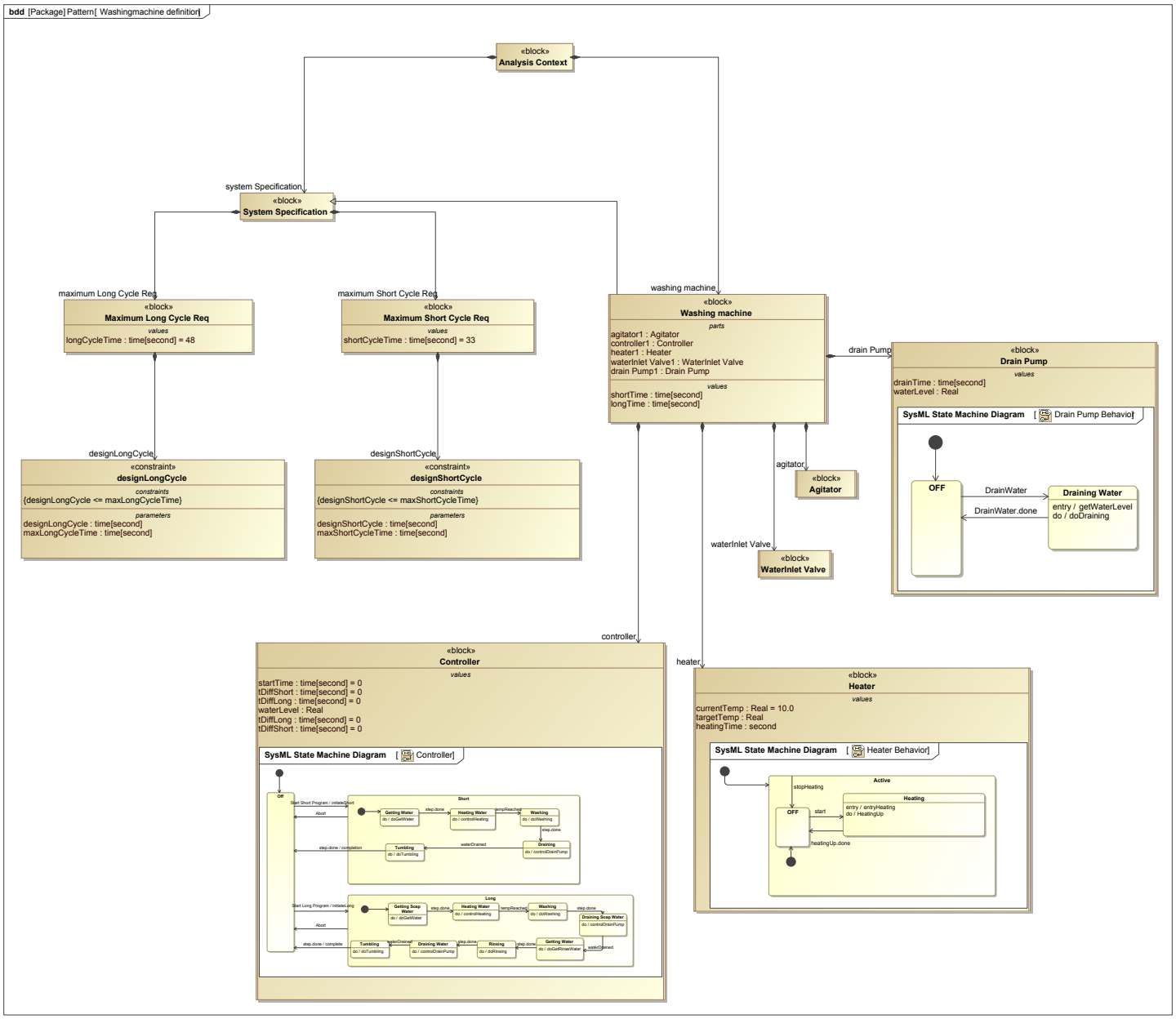

Figure 24. Washingmachine definition

Above is the structure of the washing machine analysis. The machine is composed of five parts, the controller, heater, agitator, water inlet valve, and drain pump. The behavior of each is these is captured in state machines and activities; however, the behavior of the agitator and the water inlet valve have been omitted to simplify the model. The controller, heater and drain pump are sufficient to show the interaction between the behaviors of the different parts of the washing machine. The System Specification is composed of blocks that define the length of the washing cycles and constraint blocks that evaluate the simulated value compared to the maximum allowable value.

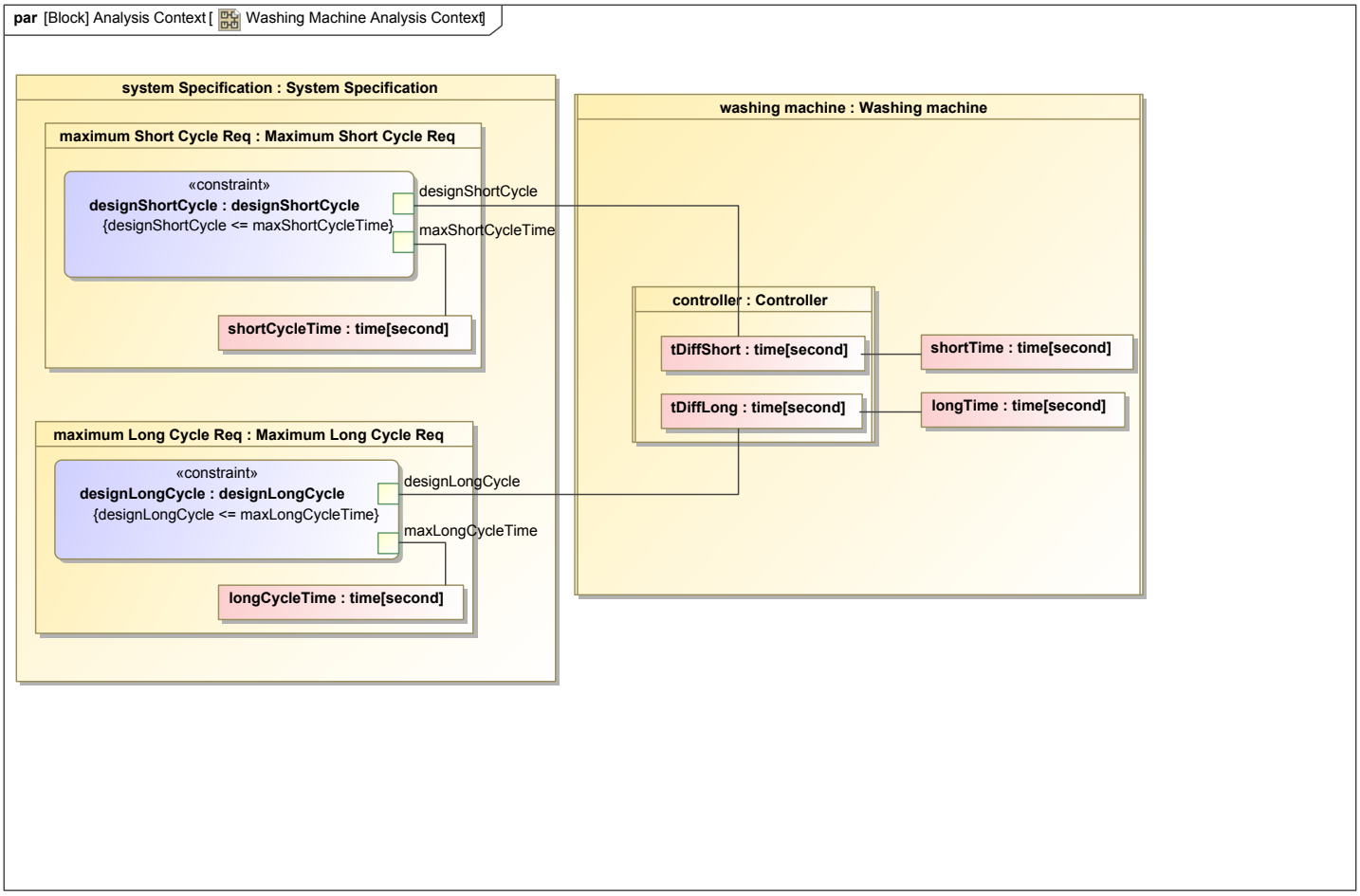

Figure 25. Washing Machine Analysis Context

This parametric diagram of the Analysis Context block displays how to model constraints and evaluate them. Two system requirements are parents to constraint properties, typed by the constraint blocks designShortCycle and designLongCycle. Both of the constraints are connected via binding connectors to value properties on the Controller. When the model is executed, value properties tDiffShort and tDiffLong return the length of the short cycle and the length of the long cycle, respectively. A second constraint parameter on the constraint blocks is bound to a value property on each of the requirements. The simulation engine evaluates the constraint by comparing value properties and notifies the user if the constraint is satisfied.

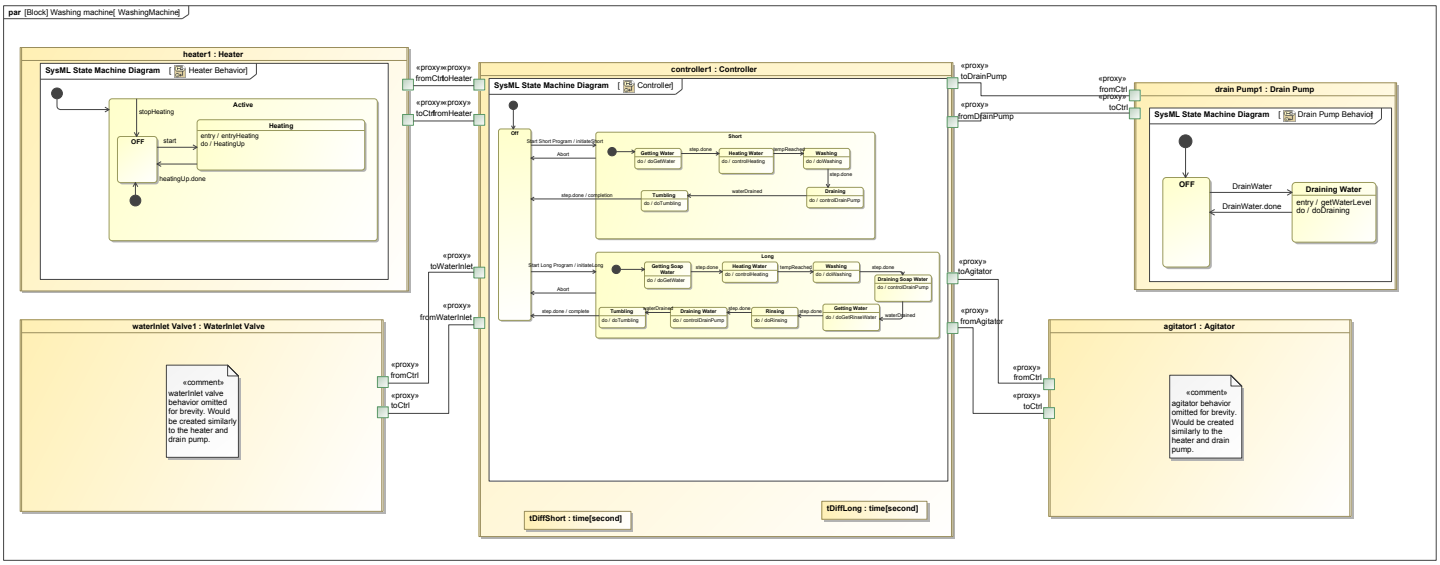

Figure 26. WashingMachine

The parametric diagram above demonstrates the interactions between washing machine components. State machines define the behavior of each of the components. Activities within the state machines have duration constraints on their processes and send signals

to other activities in order to progress through states of the washing cycle. Communication among machine components is achieved via proxy ports on the washing machine components through which signals are sent. Below is the state machine that defines the behavior of the Controller.

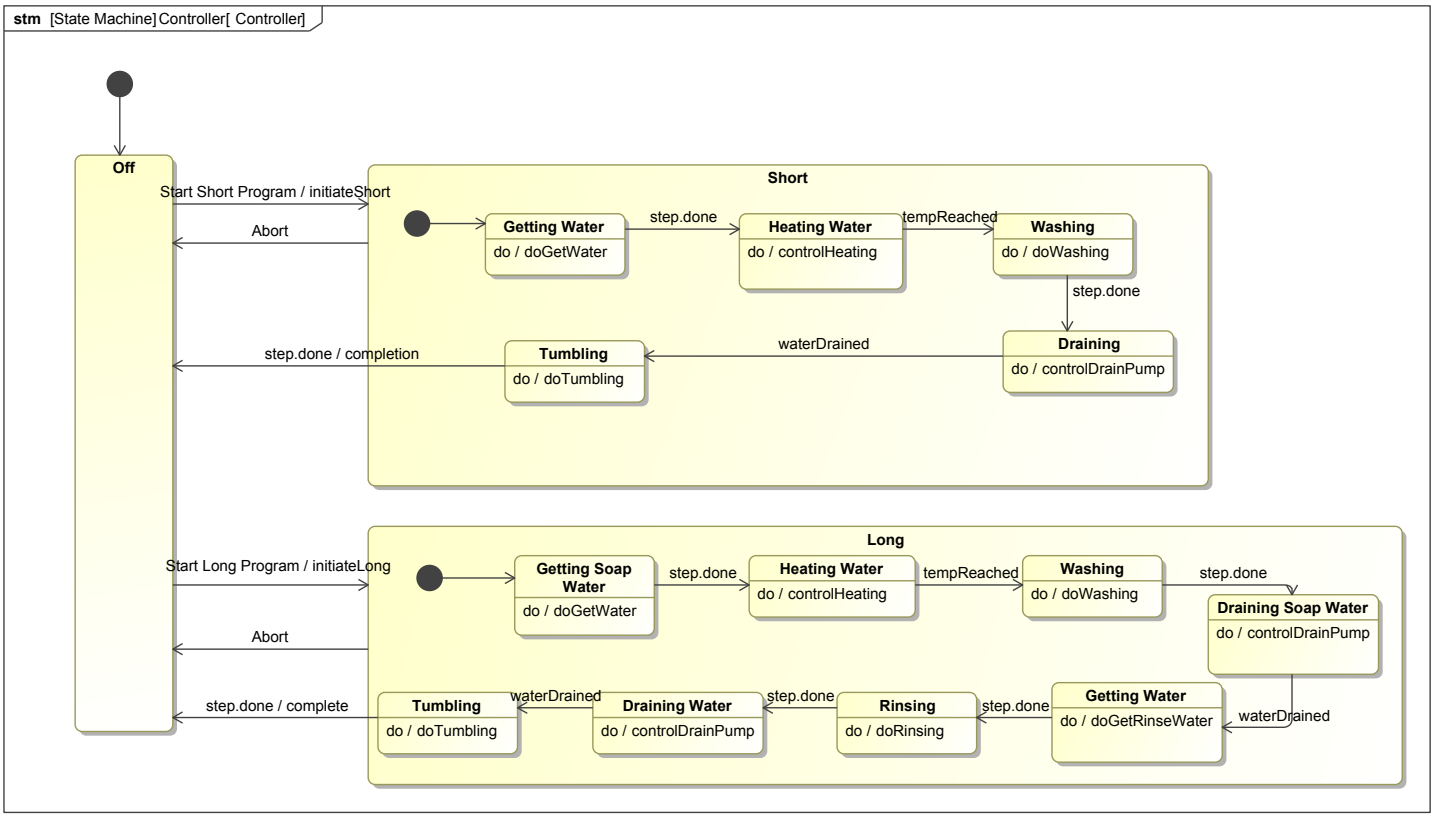

Figure 27. Controller

The Controller enters either the 'Short' or 'Long' state after receiving a signal from the user. The simulation engine simulates the progression between states with duration constraints on activities defining the simulation time spent in each state. Below is the activity diagram associated with the 'Washing' state.

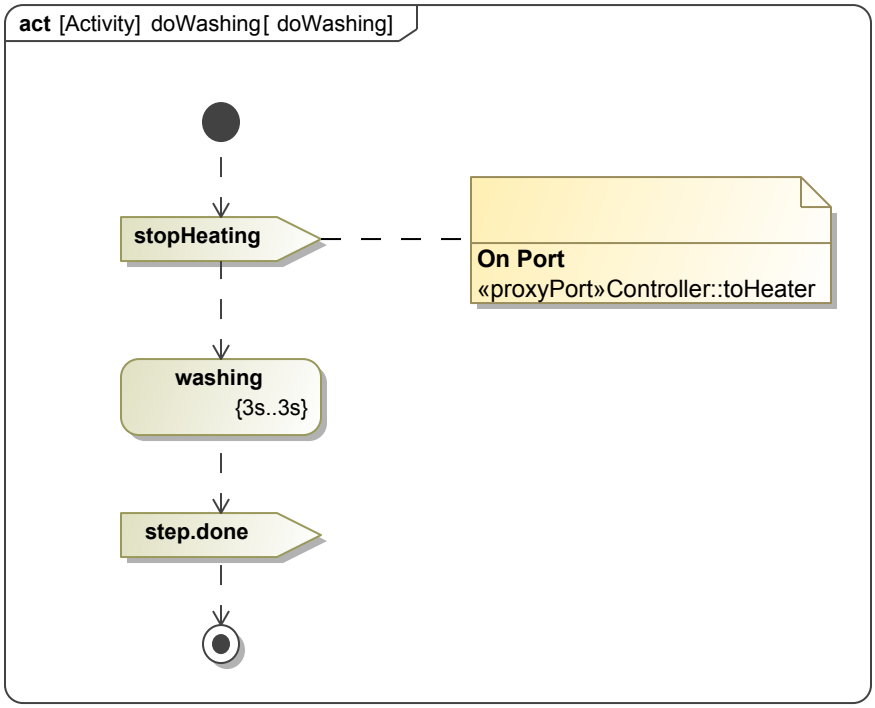

Figure 28. doWashing

When the 'Washing' state is reached, a signal is sent to Heater component to enter the 'OFF' state. The washing behavior is then performed for 3 seconds as defined by a duration constraint. After 3 seconds of simulation time have passed, a signal is sent to proceed to the next state in the cycle.

In this pattern, the time calculations are done in opaque actions defined by either Jython or JavaScript. Elapsed time is tracked while the cycle is running by recording the time upon entry and exit and calculating the difference. For example, the opaque behavior 'initiateShort' records the initial time and the opaque behavior 'completion' calculates the time passed for the short cycle. These opaque behaviors occur on the transitions to and from the 'Short' state and do the following operations:

1. The 'initiateShort' behavior sets the startTime value property of Controller to simtime which is the current time on the simulation engine clock.
2. The 'completion' behavior then calculates the difference between the simtime at the time of exit and startTime. The result is set as the value of tDiffShort, the value property on Controller that is bound to its respective constraint. The value of tDiffShort is also printed in the simulation engine log.

The specifications of opaque behaviors are shown below. The body of the 'initiateShort' behavior sets the startTime variable to the current simulation time. The 'completion' behavior calculates the duration of the activity by subtracting the starting time from the current simulation time and prints the result.

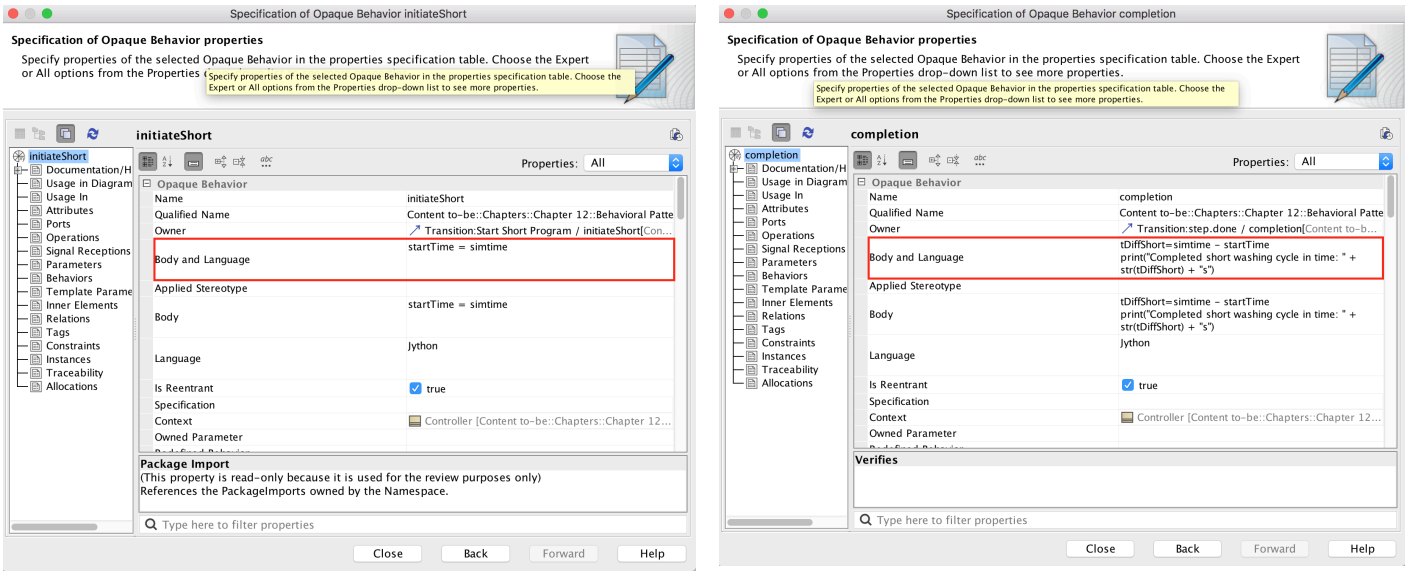

Figure 29. initiateShort and completion Specifications

3.6 Analysis

The Duration Analysis Pattern helps to analyze a system design by relating the system to its specification through the analysis context. The owned relationships of the Analysis Context block define the structure of the analysis and where system components, requirements, and constraints are located within the model.

The implementation of duration constraints on the internal processes of the system allow the modeler to partition the total length of the cycle and allocate time to specific states within the cycle. This pattern gives the user the ability to design a system based on specified time requirements and reallocate time to different states and activities as necessary.

3.7 Related Patterns

Table 6. T1

Pattern	Similarities	Differences
Time Modeling Specification	Duration Analysis is similar to the concepts in the Time Modeling Specification in that duration constraints are used to regulate the progression of system states based on the simulation time.	Duration Analysis differs from the concepts in the Time Modeling Specification in that the resulting duration of the simulation is a constraint parameter that is evaluated against a specified value. Rather than simply controlling the changes of state, the Duration Analysis returns a verdict that tells the user if a requirement is satisfied.

3.8 Conflicting Usages

3.9 Tooling

This section explains the tools involved in implementing the Duration Analysis Pattern.

3.9.1 Cameo Simulation Toolkit

NoMagic's Cameo Simulation Toolkit (CST) is implemented as the simulation engine for this duration analysis pattern. CST simulates the progression between states, executes actions within activities, and evaluates parametric constraints to produce the simulation results.

The parameters of each simulation are defined in a simulation configuration as seen below. Among other things, the simulation configuration defines the target of the simulation, launches a GUI that obtains user input, and defines the characteristics of the simulation clock. CST is launched by right-clicking a simulation configuration and selecting the 'run' option under the 'simulation' tab.

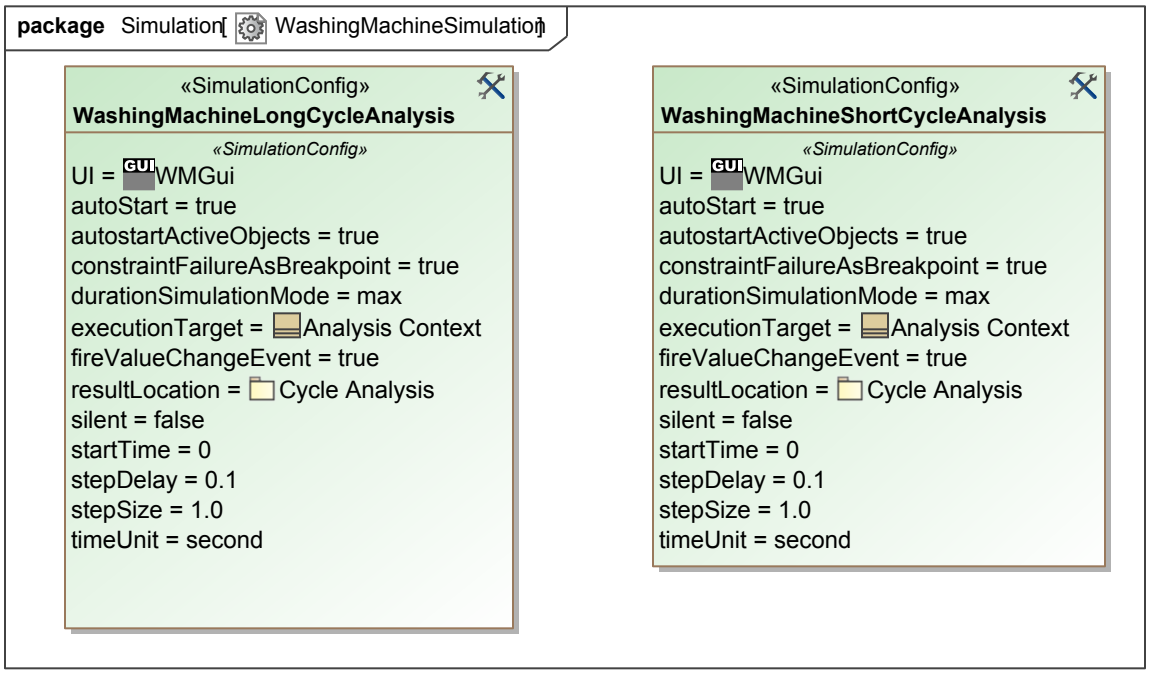

Figure 30. WashingMachineSimulation

Below are the results of a simulation of the short cycle washing process. The current system design results in a 34 second cycle length as is seen in the simulation console and designShortCycle constraint parameter in the variables window. Because the maximum duration of the cycle was specified to be 33 seconds, the simulation shows that the current design does not satisfy the requirement.

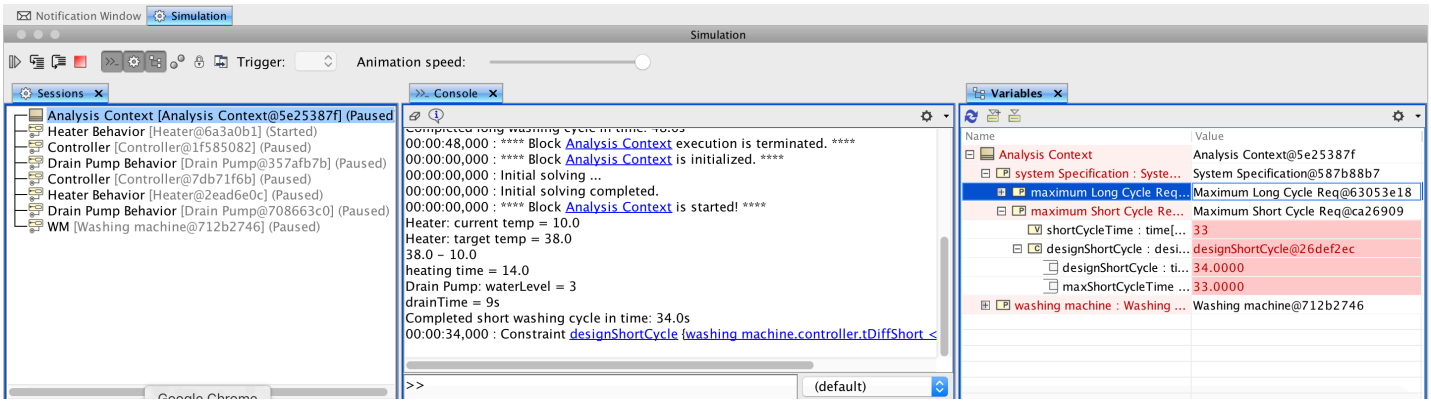

Figure 31. Cameo Simulation Results

After each simulation execution, the results are stored as an instance of the Analysis Context block. The instances of Analysis Contexts contain instances of its part components with constraint parameters as slot values. An example of an Analysis Context block is seen below. Additionally, multiple Analysis Context instances corresponding to simulation trials can be added to an instance table.

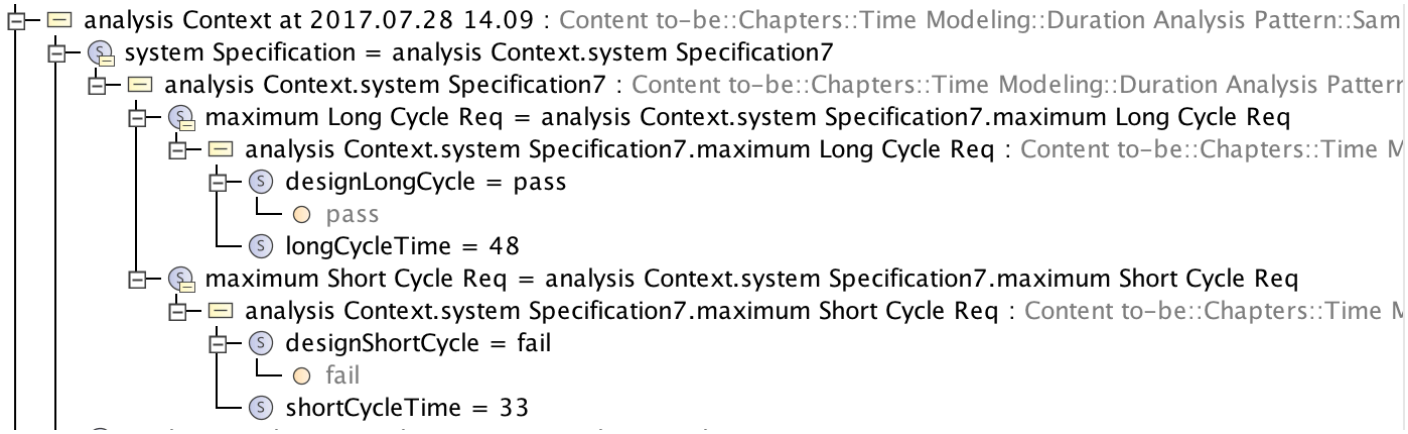

Figure 32. Cycle Analysis

The instance table below shows the Analysis Context blocks created during multiple simulations.

Table 7. Analysis Context Instances

Name
analysis Context at 2017.07.20 11.39
analysis Context at 2017.07.20 14.15
analysis Context at 2017.07.20 14.56
analysis Context at 2017.07.20 14.59
analysis Context at 2017.07.26 13.05
analysis Context at 2017.07.28 13.45
analysis Context at 2017.07.28 13.47
analysis Context at 2017.07.28 14.09
analysis Context at 2017.07.31 10.28

3.10 Known Uses

The TMT SysML model applies the concept of duration analysis. Below is a state machine from the model where the 'Take Exposure' state is entered after receiving a signal.

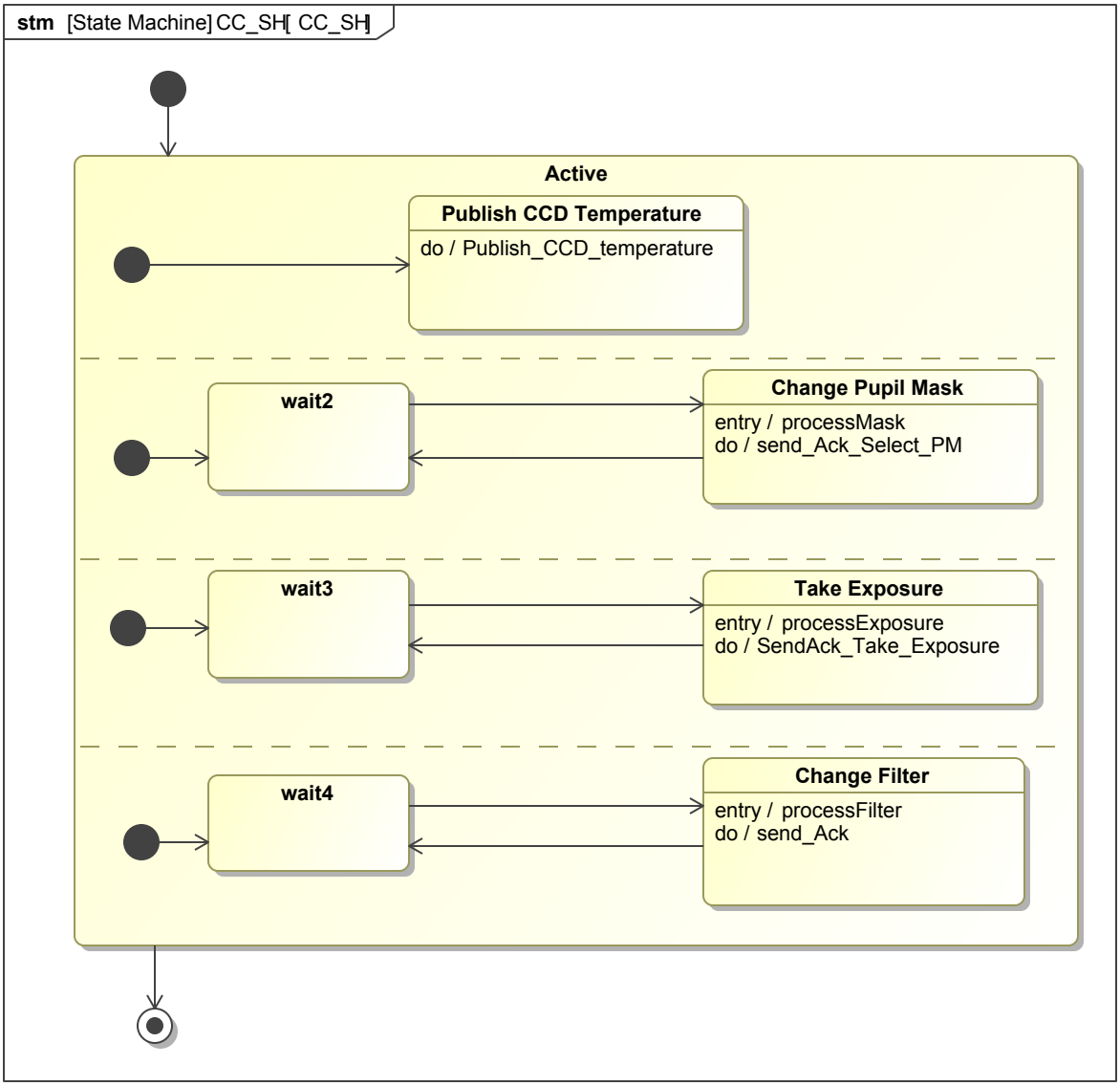

Figure 33. CC_SH

The 'Take Exposure' state has a do-behavior defined by the activity diagram seen below. The action 'Read Out Behavior' within the activity has a duration constraint of 10 seconds.

Figure 34. SendAck_Take_Exposure

The following parametric diagram compares the simulated value of 'tFinal' to a defined maintenance alignment maximum value. The constraint enforces the relation that 'tFinal' should be less than or equal to the maximum allowable time. After the simulation is run, the user receives feedback about the compliance of the design with the specified requirements.

Figure 35. Maintenance Alignment Duration Analysis

4 Black Box Specification with Property Based Requirements Pattern

4.1 Intent

The intent of the Black Box Specification with Property Based Requirements Pattern is to document the purpose of a "black-box specification" of a system according to the Object Oriented Systems Engineering Method. A black-box specification captures a set of features- functions, properties, and interfaces. Capturing the system functions as part of the black-box specification can be used to flow-down the system's functional requirements to system component functions. In this pattern, the black-box specification realizes one or more physical architectures. Additionally, the application of property based requirements to a black-box specification will be discussed. Property based requirements allow for text based requirements and more precise statements of requirements, property based requirements. Primary function, performance, and interface requirements are identified from formal requirements and can be represented as operations, properties with values (and types), and ports on the block. Through the application of property-based requirements, engineers have the capability to model requirements as formal expressions that constrain property values.

4.2 Motivation

Object-Oriented Systems Engineering Method (OOSEM) is a scenario-driven process that uses SysML to support the analysis, specification, design, and verification of systems. OOSEM leverages object-oriented concepts and other modeling techniques to help architect flexible and extensible systems that can accommodate changing requirements. In OOSEM, the system model is a primary output of the system specification and design process. The model artifacts present different views of the system such as structure, properties, and traceability to its requirements. OOSEM helps ensure views provide a consistent representation of the system. The OOSEM process includes fundamental systems engineering activities such as stakeholder needs analysis, requirements analysis, architecture design, trade studies, and verification. OOSEM includes the following modeling techniques: analysis, black box and white box descriptions, logical decomposition, partitioning criteria, node distribution, variant design, to address a variety of system concerns. Applying the process at a system-of systems level, results in the specification and verification of system elements.

The Black Box Specification with Property Based Requirements Pattern can be applied to a modeler's system to specify system requirements as a black-box specification in terms of input and output responses needed. The requirements of the system can be captured as property based requirements to specify a black-box specification. The traceability of requirements are maintained from the overall system level down to the component level. The black box specification defines the system's externally observable behavior and physical characteristics. However, a black box specification doesn't specify how the system achieves the externally observable behavior, which is defined by the system design. Through the integration of system elements and/or components in the black-box specification allows for the verification that the integrated design satisfies its requirements.

4.3 Concept

From the Object Oriented Systems Engineering Method (OOSEM), the OOSEM Black Box Specification supports the analysis, specification, design, and verification of systems. The OOSEM Black Box Specification defines a system's externally observable behavior and physical characteristics. The black-box specification doesn't specify how the system achieves the externally observable behavior, which is defined by the system design. A black-box specification can be applied at any level of design, including system, element, and component levels. The System Specification inherits the properties, functions, and attributes of the OOSEM Black Box Specification block. Therefore, the System Specification for all purposes is a type of black-box specification.

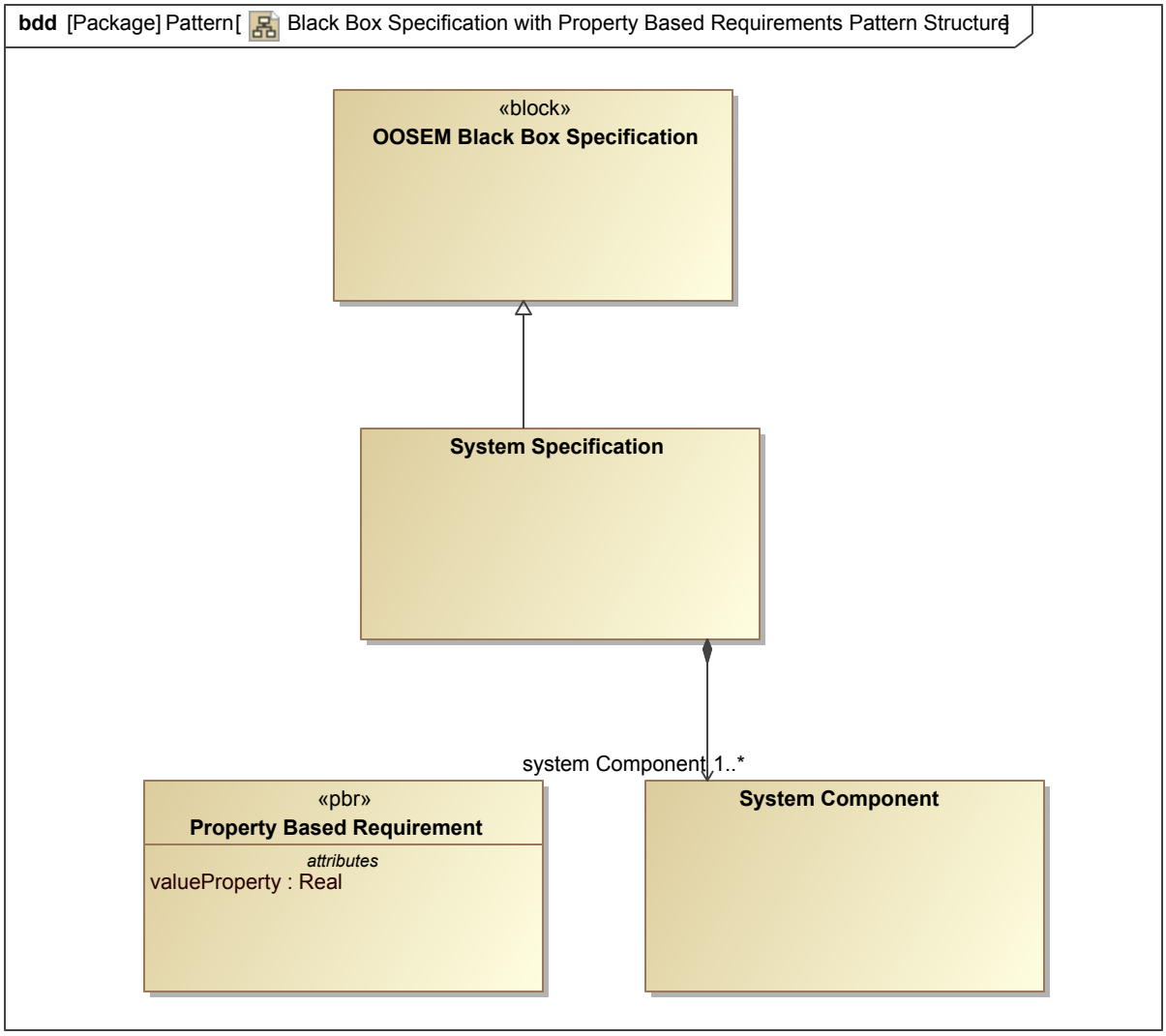

Figure 36. Black Box Specification with Property Based Requirements Pattern Structure

The System Specification represents the functional specification of a system design. The properties, attributes, behaviors, and functions are inherited from the OOSEM Black Box Specification. The System Specification can own part properties that are typed by a Property Based Requirement depending on the flow down of property based requirements. Depending on the system design, which the System Specification inherits, the System Specification can be composed of components. These components are realization elements of the system. The Property Based Requirement allows engineers to model a more formal expression of requirements which retains the properties of a block and a requirement. The user has the ability to specify value, constraint, part, and reference properties to reflect refined textual requirements.

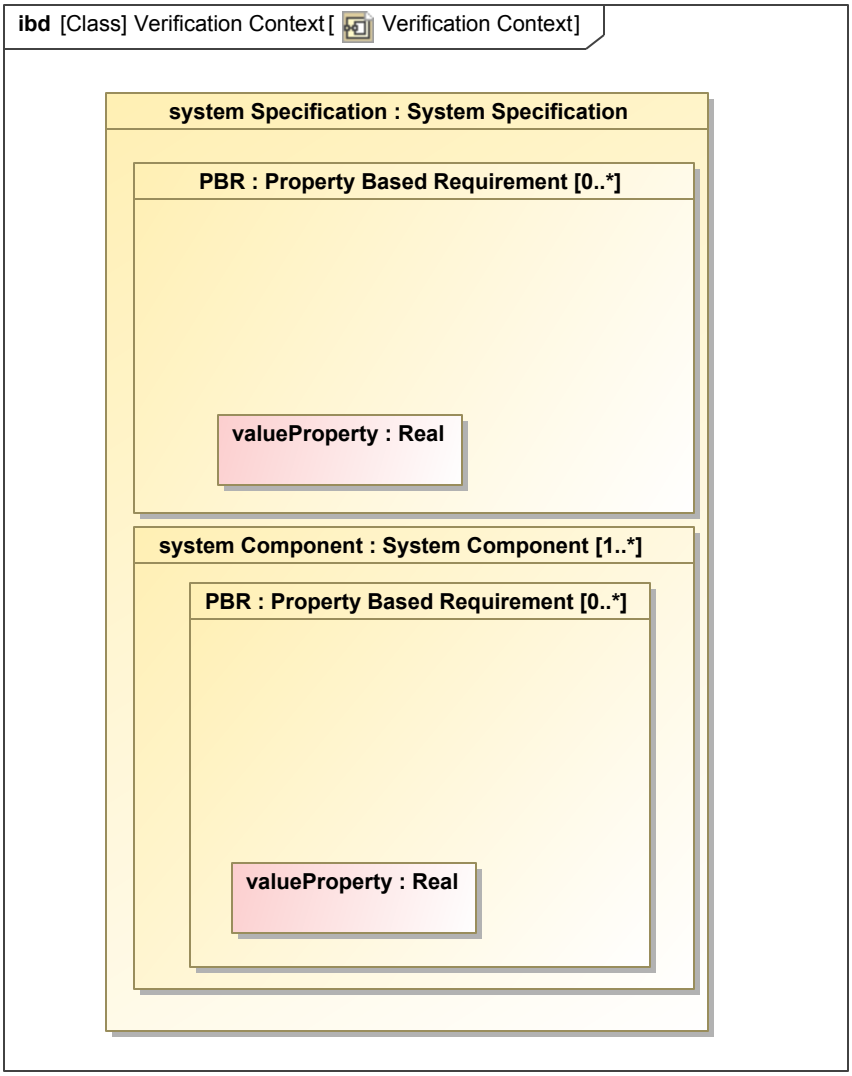

Figure 37. Verification Context

An overview of the internal structure of the System Specification is shown above in an internal block diagram. Notice both the system specification and system components can be specified by property based requirements.

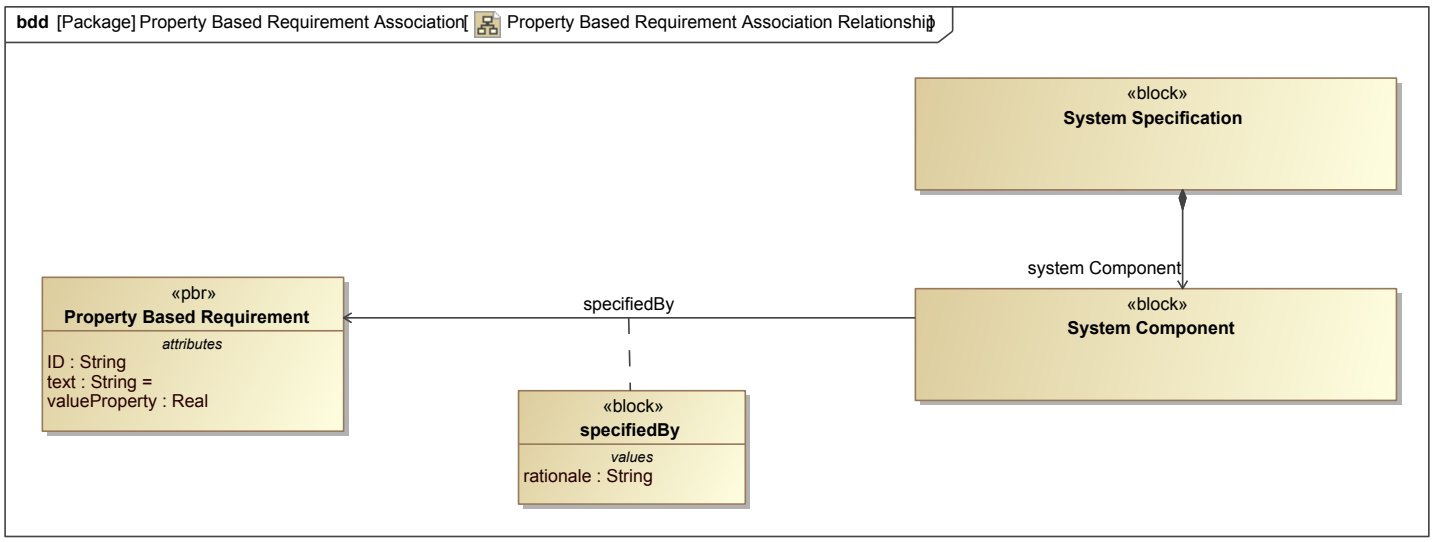

Figure 38. Property Based Requirement Association Relationship

Table 8. <>

Model Element	
OOSEM Black Box Specification	The OOSEM Black Box Specification defines a system's externally observable behavior and physical characteristics. The black-box specification doesn't specify how the system achieves the externally observable behavior, which is defined by the system design. The black-box specification may be augmented by constraints to constrain how the black-box requirements are implemented. The OOSEM Black Box Specification is modeled as a SysML block and can contain any of the following features: the required functions that are to be performed and the associated inputs and outputs, external interfaces that allow the specification to interact with other external systems, interfaces that are specified by ports, performance and quality metrics that impact the functions of the system, the initial value properties that are typed by value types, initial conditionals or input events can be specified to determine when functions are performed, and required items that the system should store.
Property Based Requirement	The Property Based Requirement element was generated using the System Reasoner tool's "Specialize Structure" action on the cf name([cf:Generic Property Based Requirement .name]) does not exist element. The 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 library:propertyBasedRequirement stereotype. The owned attributes of the property based requirement are a unique ID and text. The generated element allows engineers to model a more formal expression of requirements which retains the properties of a block and a requirement. Therefore, the user is able to specify value, constraint, part, and reference properties to reflect derived textual requirements.
valueProperty	

4.4 Consequences

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

Table 9. Consequences Table

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.
The user wants to create a composition relationship between a System Specification or System Component to a property based requirement.	In the SysML 1.4 specification, property based requirement inherit the attributes of a requirement. Due to this relationship, the user is unable to create a directed composition relationship between a property based requirement and other blocks. To replicate this relationship, the user should create a part property of a System Component and set the type to the property based requirement.
The user wants to create an association relationship between a property based requirement and system component, and specify an association block.	The modeler would create an association block in replace of the typed part property of the System Component. The modeler would be able to create an association block between the Property Based Requirement and the System Component in the relation section of the specification. As previously mentioned, a property based requirement doesn't entirely behave as a block would. However, it is important to note the differences in the process of creating relationships between blocks and property based requirements.

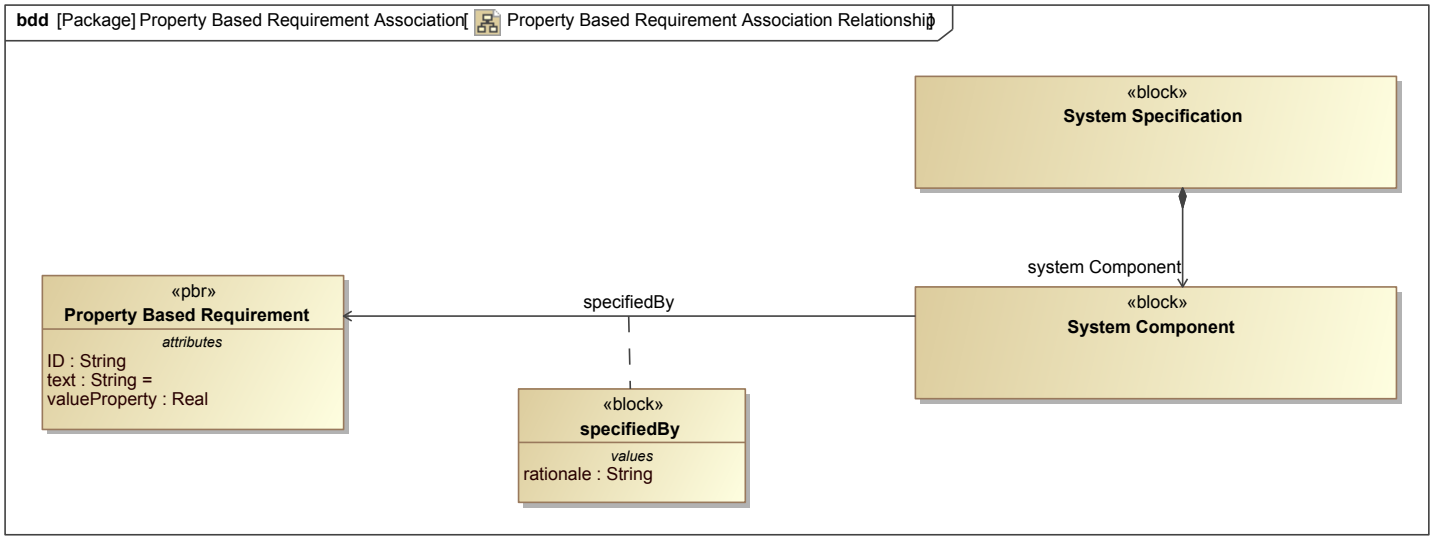

Figure 39. Property Based Requirement Association Relationship

Listing of possible conflicts that can occur while applying or using the Black Box Specification with Property Based Requirements Pattern **.**

Usage of the Pattern	**Explanation of Conflict**
A user can apply property based requirements to a System Component and to a System Specification.	The relationship between a System Component or a System Specification is a directed composition relationship. However, based on the current structure of a Property Based Requirement, the modeler is unable to create a composition relationship between a System Component or Specification, which are both blocks. Currently, the Property Based requirement inherits the attributes of a requirement, abstract element, and a block. Due to the generalization relationship between the property based requirement and the requirement, the user is unable to create a composition relationship. The SysML 1.5 spec addresses this issue, and restructures the relations of a property based requirement. In addition to the change of the <<library:propertyBasedRequirement>> to <<PBR>>, the property based requirement will no longer have a relationship to a requirement. This modification will allow users to create composition relationships from blocks to property based requirements.
A user can create their own domain specific property based requirements, by specializing the structure of the Generic Property Based Requirement element.	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.
A user can attribute their own domain specific property based requirements that were specialized by theGeneric Property Based Requirement element with value properties.	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 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 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.

A user can attribute their own domain specific property based requirements that were specialized by the Generic Property Based Requirement element with constraints.	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 Property Based Requirement element 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 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.

4.5 Implementation

In this sample model, we will provide a model-based system specification and design method that users can adapt to their domain specific needs and applications. The focus of the example is to specify a system specification that inherits the qualities of a OOSEM Black Box specification through the use of property based requirements to capture critical system properties and constraints. The decomposition of system components and requirements for a Drone are modeled.

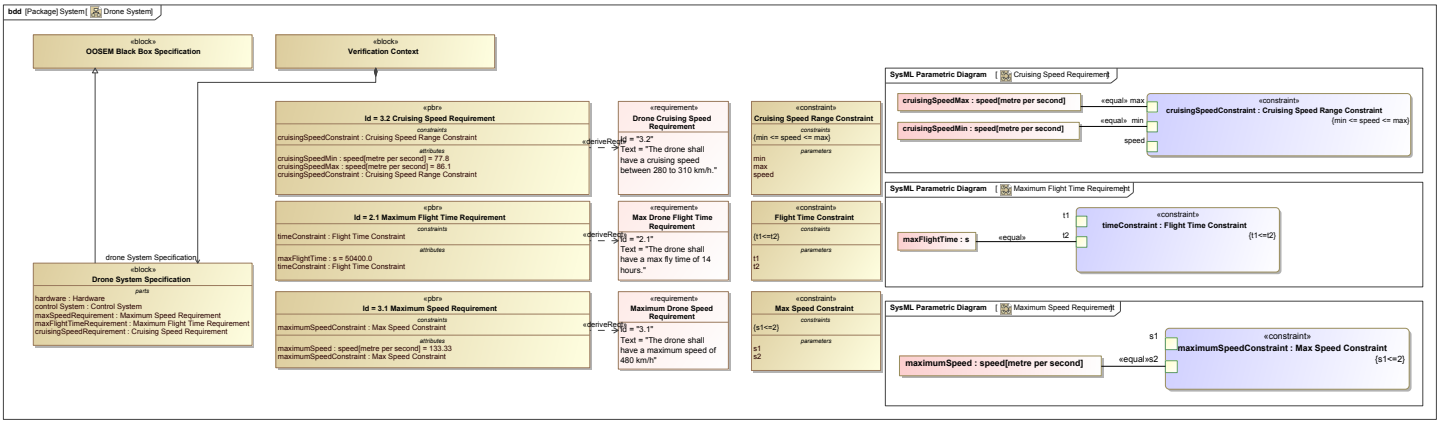

Figure 40. Drone System

As specified previously, the structure of the pattern consists of the relationships between the OOSEM Black Box Specification, a Drone System Specification, and the property based requirements of the system components. The Drone System Specification is composed of component parts (Hardware and Control System), and the property based requirements Maximum Speed Requirement,Maximum Flight Time Requirement, and Cruising Speed Requirement. The requirements that the property based requirements are shown to display the transformation between the two. If the modeler was only given textual requirements, Drone Cruising Speed Requirement, Max Drone Flight Time Requirement, andMaximum Drone Speed Requirement, the modeler would extract the value properties and other physical attributes. For example, the Cruising Speed Requirement owns the value properties cruisingSpeedMin and cruisingSpeedMax and constraint property Cruising Speed Range Constraint to reflect the Drone Cruising Speed Requirement. The constraint,Cruising Speed Range Constraint, is used to constrain the minimum and maximum values for the drone's speed through the constraint expression cruisingSpeed. The parametric models allow for the verification and analysis of the system's components.

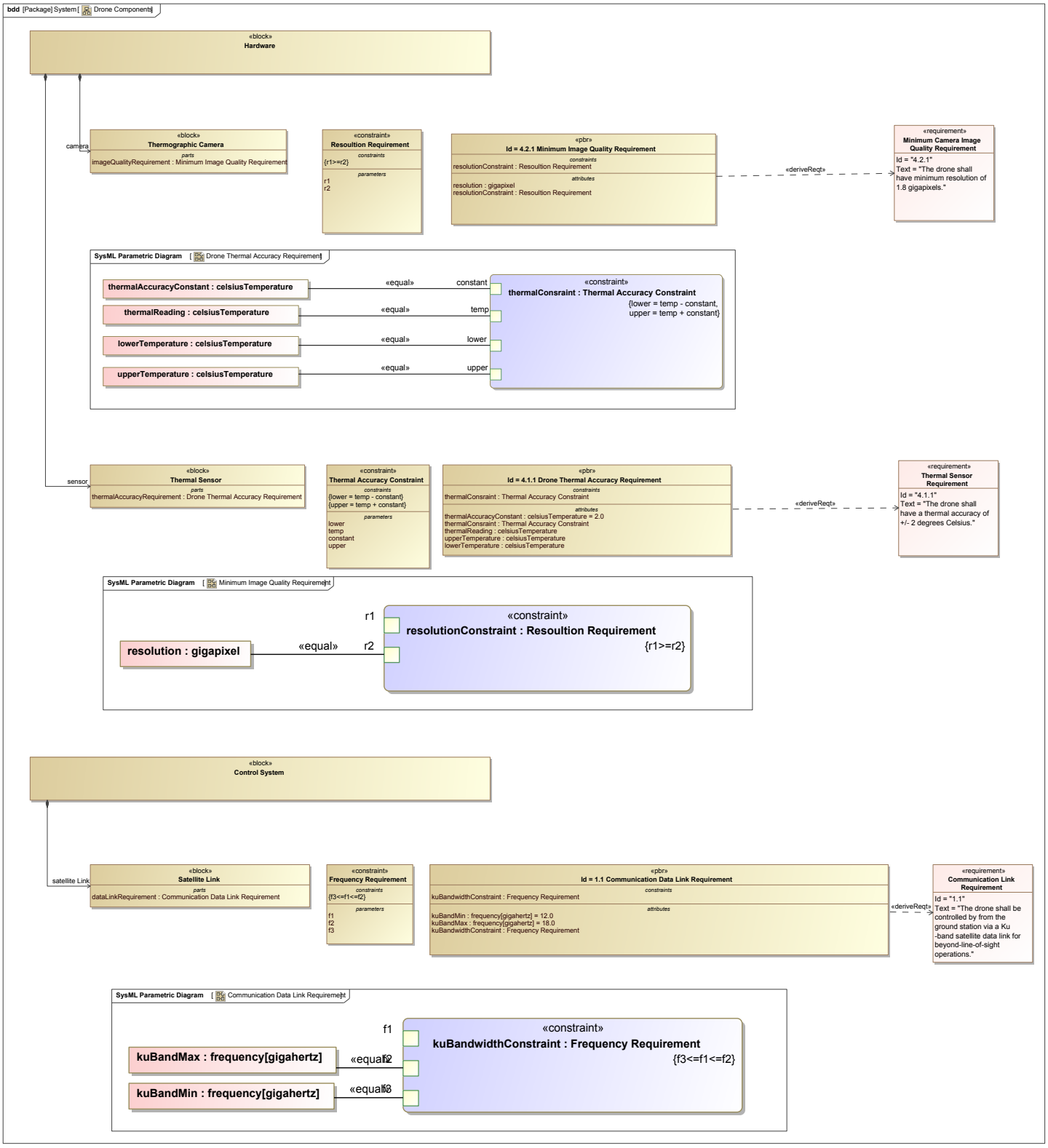

Figure 41. Drone Components

The Drone Specification is composed of Hardware and Control System blocks which are the system components of the Drone. The Hardware component decomposes into a Thermographic Camera and a Thermal Sensor. The Thermographic Camera owns a part property typed by the property based constraintMinimum Image Quality Requirement which refines theMinimum Camera Image Quality Requirement. The PBR, Minimum Image Quality Requirement, owns the value property resolution, and the constraint property resolutionConstraint which is typed by Resoultion Requirement. The Thermal Sensor owns a part property typed by the property based constraint Drone Thermal Accuracy Requirement. The PBR, Drone Thermal Accuracy Requirement, owns the value properties thermalAccuracyConstant, thermalReading, upperTemperature, and lowerTemperature.

The Control System component is composed of a Satellite Link owns a part property typed by the property based constraint Communication Data Link Requirement which refines Communication Link Requirement. The PBR, Communication Data Link Requirement, owns the value propertieskuBandMin and kuBandMax, and the constraint property kuBandwidthConstraint which is typed by Frequency Requirement.

The parametric models are owned by the property based requirements, and represents the operations and functions of the textual requirement. The models allow the engineer to perform verification analysis of the drone system.

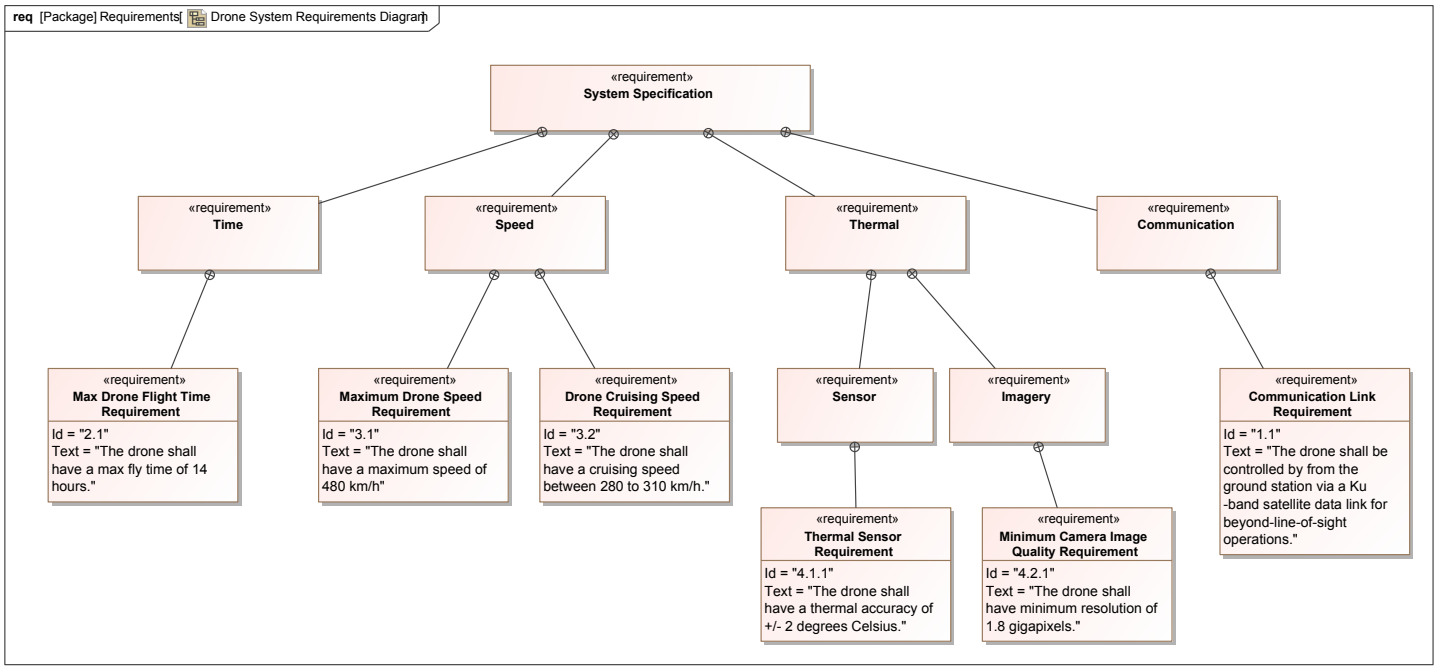

Figure 42. Drone System Requirements Diagram

An example of requirements traceability and flow down from a system requirement to a component requirement can be seen above. The requirements that are contained within the System Specification is shown. The System Specification has the nested classifiers of all the requirements that are contained below.

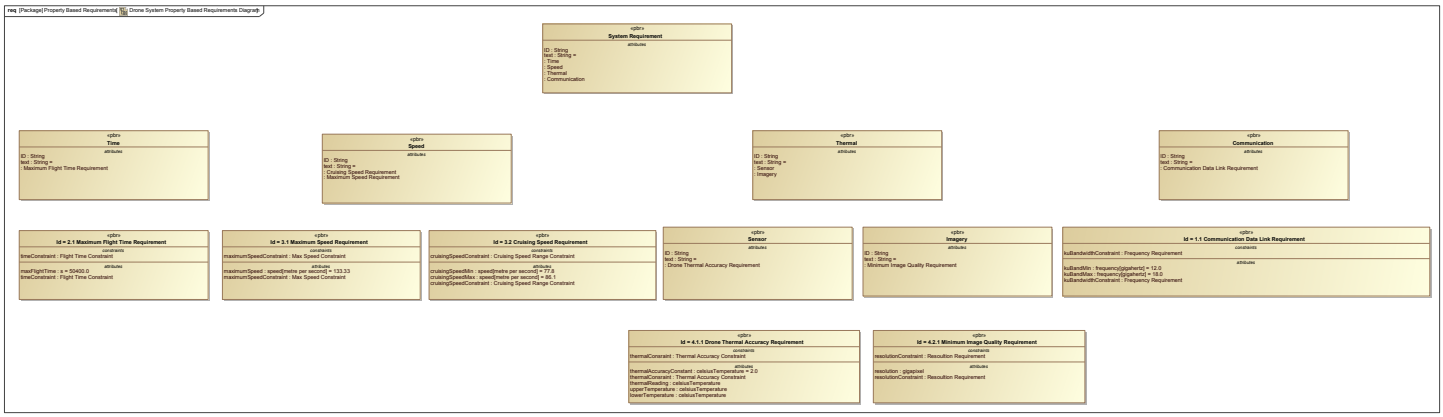

Figure 43. Drone System Property Based Requirements Diagram

Since, property based requirements have both the properties and operations of a block and requirement we are able to create a system requirement decomposition. The property based requirements traceability and flow down from a system property based requirement to a component property based requirement is shown above. The method of representing the flow down of the property based requirements is different from the formal requirement hierarchy. To express the requirement flow down of the system specification, the part properties are typed by the next nested level of property based requirements. This process is done until all property based requirements of the hierarchy have been typed or applied as a type. The resulting hierarchy shares the same traceability capabilities as the formal requirement hierarchy as seen in Drone System Requirements Diagram.

4.6 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.

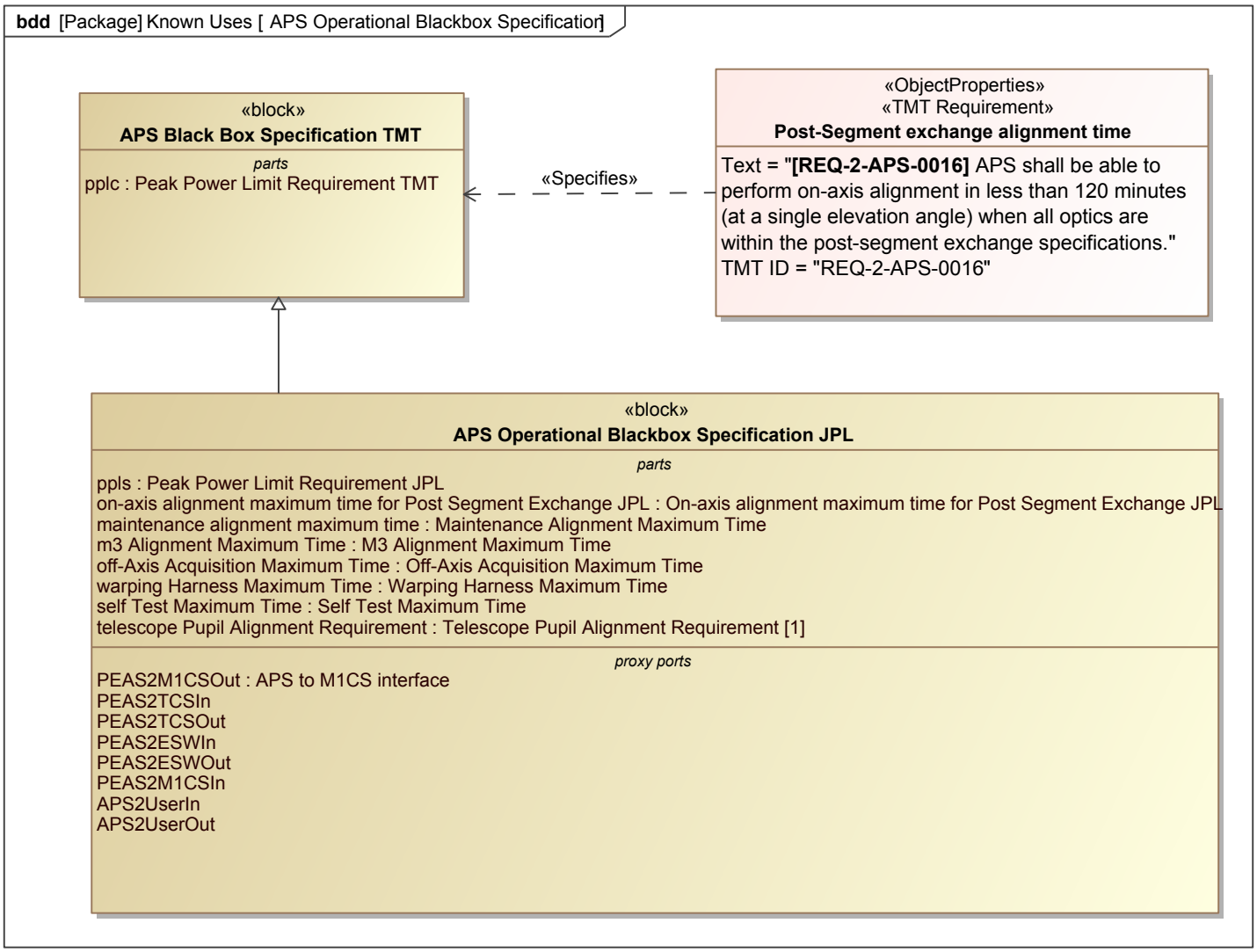

Figure 44. APS Operational Blackbox Specification

The APS Black Box Specification TMT represents the system specification for the Alignment and Phasing System that is defined by the TMT. Although, the APS Black Box Specification JPL may not have the same system specification due to analysis purposes. Therefore a generalization relationship between the two denotes that the APS Black Box Specification JPL will inherit the properties of the TMT specification. In the TMT and JPL specification, property based requirements are modeled as the types of part properties. For example, the Black Box Specification is specified by the formal requirement Post-Segment exchange alignment time. In the property based requirement, Peak Power Limit Requirement TMT, the physical properties of the formal requirement are modeled as value properties. In the APS Operational Blackbox Specification JPL, the part properties- ppls, on-axis alignment maximum time for Post Segment Exchange JPL, maintenance alignment maximum time, m3 Alignment Maximum Time, off-Axis Acquisition Maximum Time, warping Harness Maximum time, and self Test Maximum Time are typed by property based requirements.

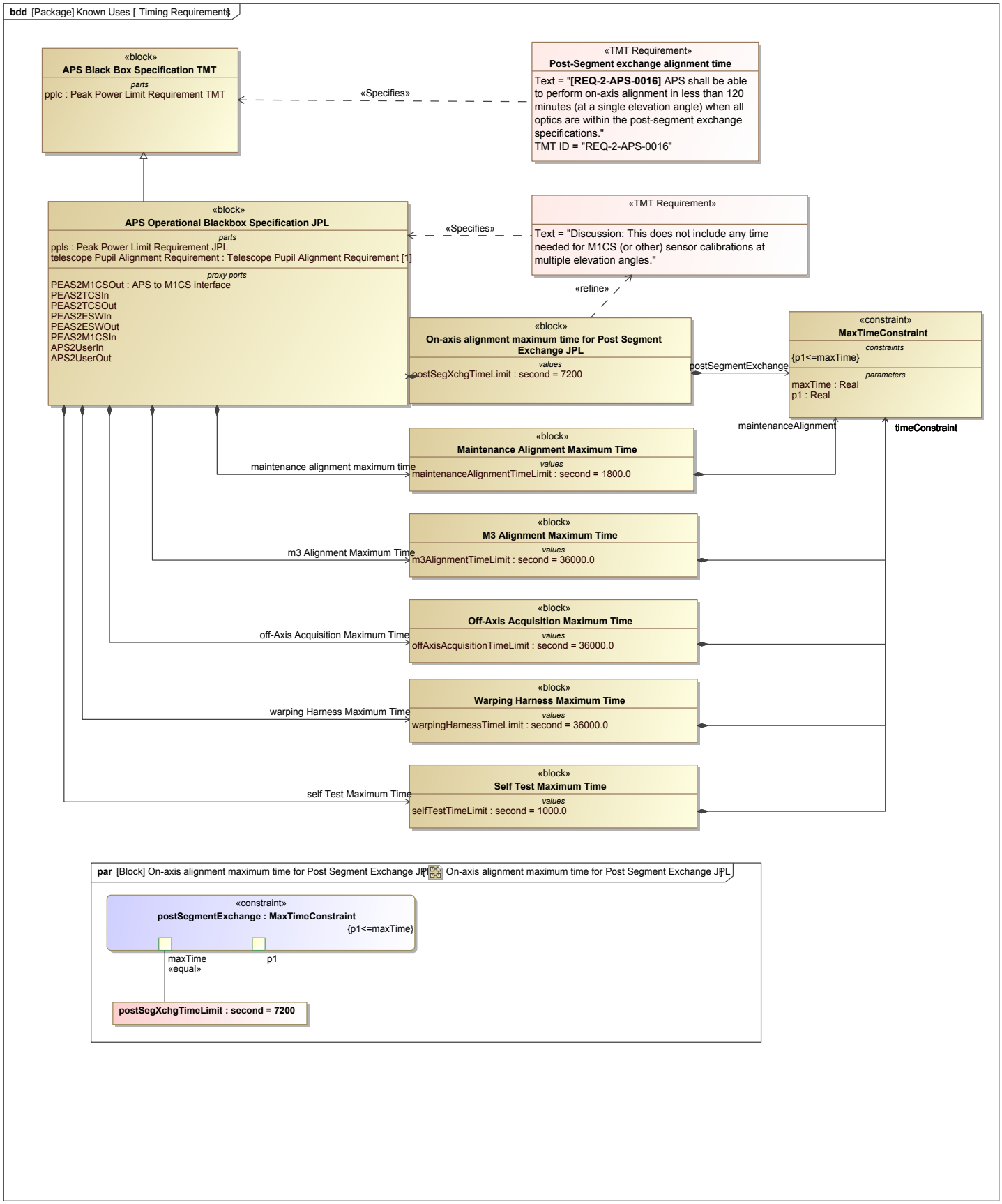

Figure 45. Timing Requirements

4.7 Tooling

4.7.1 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 Black Box Specification with 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 Specialize Structure Action documentation for more details.

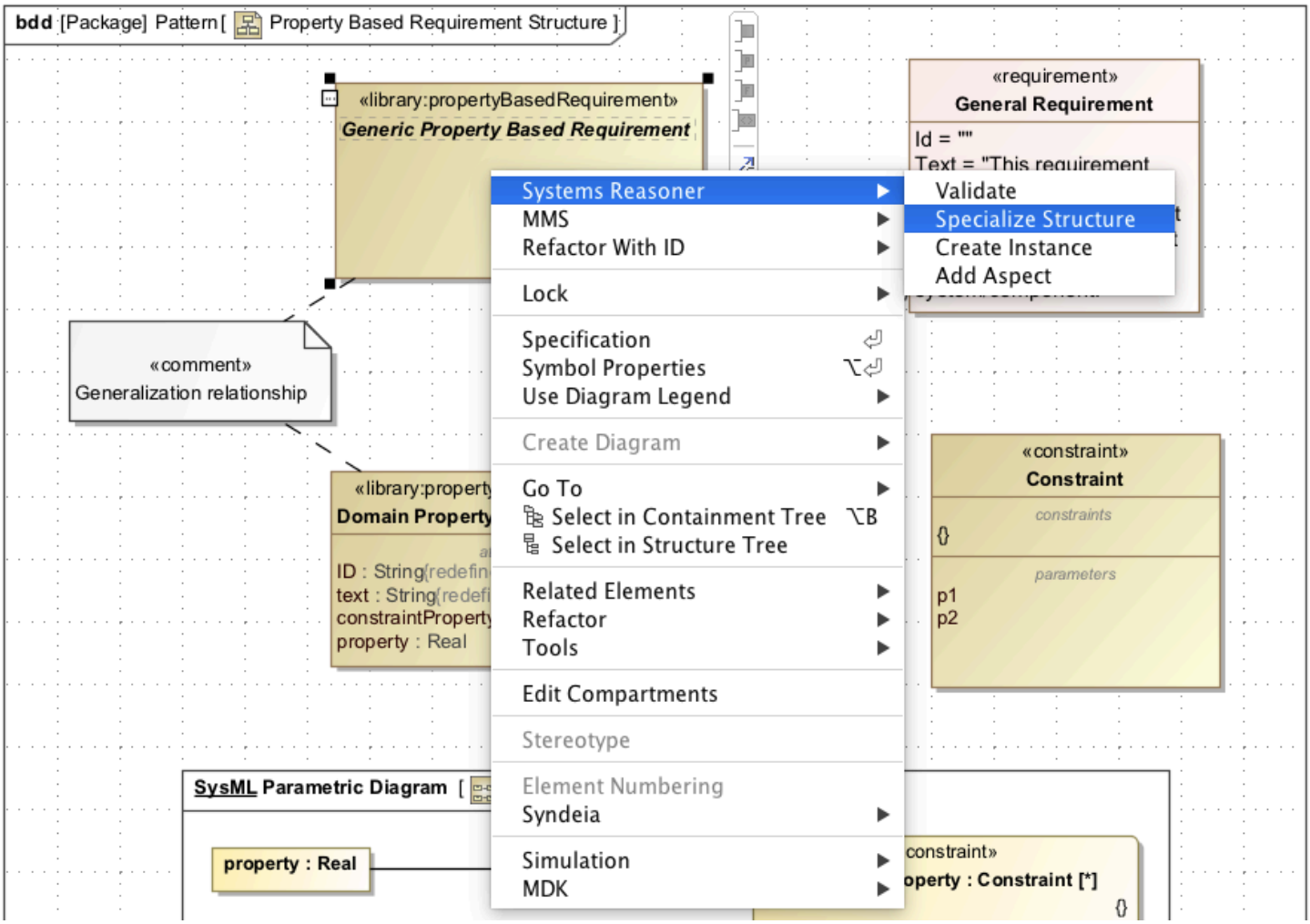

Figure 46. Model Construction View

4.8 Related Patterns

List of other patterns that are related to the Black Box Specification with Property Based Requirements 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.
Customer/ Supplier Pattern	The System Components of the Customer/Supplier Pattern may not be divided into functional and realization components in the pattern, but they certainly can be depending on the system design. Similar to the Customer/ Supplier Pattern, the System components can specialize another system component, if needed. In these cases, the property based requirements can also specialize another property based requirement depending on the system.	In the Customer/Supplier Pattern, the black box specification of the customer or the Customer System Specification is generalized by the Supplier System Specification. The System Components of the Customer/Supplier Pattern are divided into Functional and Realization components. The primary difference between both patterns is that the Customer/ Supplier Pattern doesn't address the transformation of requirements to qualitative properties of a component.

5 Monte Carlo Simulation Pattern

5.1 Intent

A Monte Carlo simulation is a statistical technique used to model probabilistic systems and establish the probability for different outcomes. The simulation is stochastic which means that the resulting output is at random and will vary from run to run. Monte Carlo simulations use random inputs to model a system and produce probable outcomes. Based on a range of values for each variable, the Monte Carlo simulation will select a value within the range at random, and repeat the process for a defined set of iterations. The number of iterations and the input provided will determine whether realistic simulation results are produced. The primary benefit of using Monte Carlo simulations is to understand the impact of uncertainty on a system, and to develop plans to mitigate and/or cope with risk. The Monte Carlo Simulation Patternprovides a modeling pattern for Monte Carlo simulations, demonstrates how to perform a Monte Carlo simulation of a system, and a method to analyze the results of the simulation.

5.2 Motivation

Systems engineers can estimate the characteristics and probability of a particular behavior using modeling anchored through data. The dynamic performance of a system can be given as a probability distribution that includes the variation of attributes. Considering the uncertainties, Monte Carlo simulations can be conducted to numerically investigate operational scenarios. Through Monte Carlo Simulations in SysML, systems engineers are given a mechanism to predict acceptable dynamic performance of complicated systems. In SysML, system engineers are able to specify these uncertain values through the probability attribute of control flows between actions. By specifying the probability of a control flow, the system engineer indicates the likelihood that a value leaving the decision node or object node will traverse an edge.

An operational scenario is a set of actions or functions representing the dynamic of exchanges between the functions allowing the system to achieve a mission or a service. Monte Carlo simulations can be applied to evaluate a variable of the system (i.e. time) to perform a set of actions for a defined number of runs. Requirements are critical parameters in the design space of systems and can influence the system design. If the system design must satisfy system requirements, the system engineer can verify the results of the computational analysis through an automated fashion. To evaluate the resulting SysML instance specifications from the Monte Carlo simulations, the systems engineer must devise a system of equations through SysML parametric models to constrain the results of the Monte Carlo simulation to achieve a desired output. From the results of the Monte Carlo simulations, additional computational analysis can be performed such as determining the average or RMS. After analyzing and manipulating the variables, the systems engineer could visualize the results through instance tables or histograms.

5.3 Concept

The necessary model elements that are required to perform a Monte Carlo simulation consists of an Analysis Context, System, and Simulation Configuration. If the engineer wants to perform additional analysis on the results of the Monte Carlo Simulation and additional Analysis Context should be created. If the system needs to satisfy a requirement, the requirement should be modeled as a Property Based Requirement. The combination of the elements required for the Monte Carlo Simulation, and the instance of the Analysis Context allows for computational analysis of the Monte Carlo results.

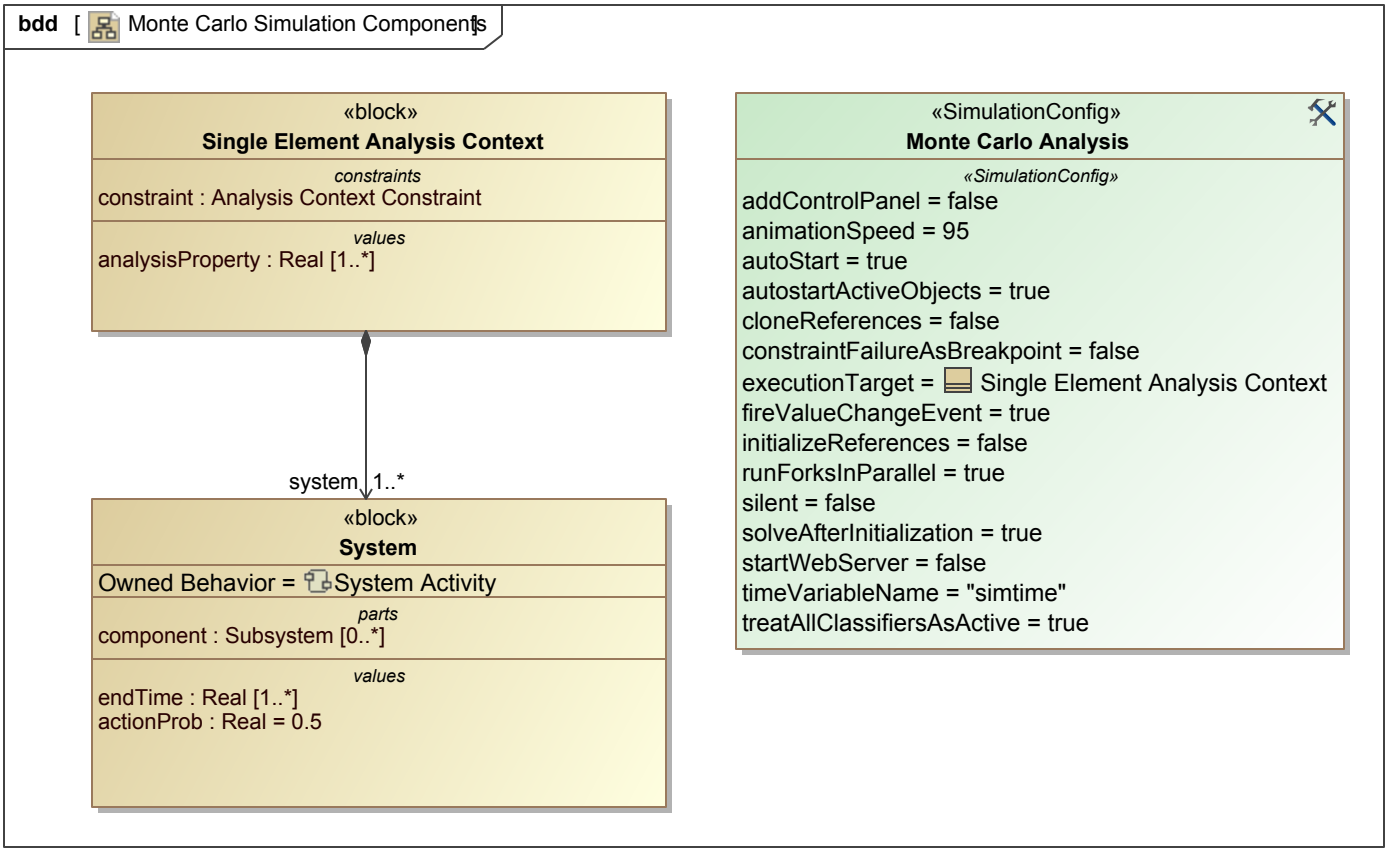

Figure 47. Monte Carlo Simulation Components

The following model elements are required for performing a Monte Carlo Simulation: an Analysis Context block, a System for analysis, and a simulation configuration. The System block represents the system for analysis, and can be composed of any subsystems/subcomponents if needed. The behavior of the System block is modeled as an activity. The System can be composed of any subsystems/subcomponents if needed. The value of the endTime attribute is equal to the simulation time at the end of the System Activity. The element is applied to specify the structure, to store test results, and to avoid any impact to the system.

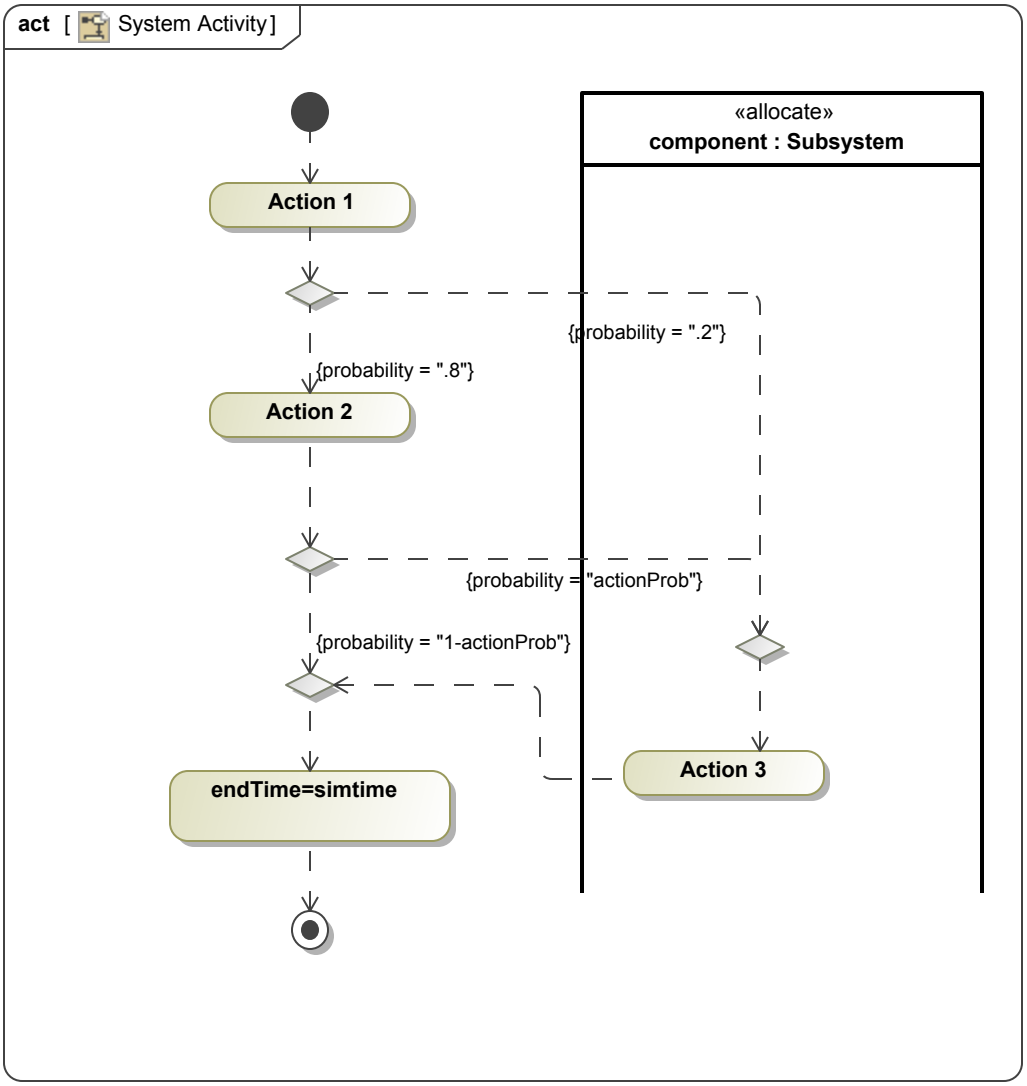

Figure 48. System Activity

The System Activity is the classifier behavior of the owning block, System. The swimlane represents the Subsystem Block, and indicates that any behaviors invoked by call actions in the partition are the responsibility of the block. A flow can be tagged with a probability to specify the likelihood that a given token will traverse a specific flow among available alternative flows, this is called a probabilistic flow. Each token can only traverse the edge with the specified probability. All alternative flows must have a probability and the sum of the probabilities of all flows must equal 1. If all the control flows aren't specified with a probability a warning error will occur in the notification window during simulation. The error will signify that the total probability is not equal to 1, and the the probability of all outgoing activity edges of the decision node will be scaled to make the total probability equal to 1.0. The user can set the value of the probability by modifying the probability attribute of the control flow. The probability can be changed by directly inputing a value or through the usage of a value property. To use a value property to specify the probability, a value property should be created in the owning element (actionProb) and the default value should be specified. In the probability attribute of the control flow, the name of the value property can then be entered. Both options for modeling probabilities are shown above. Note, the names of the actions and the values of the probabilities are only for example purposes.

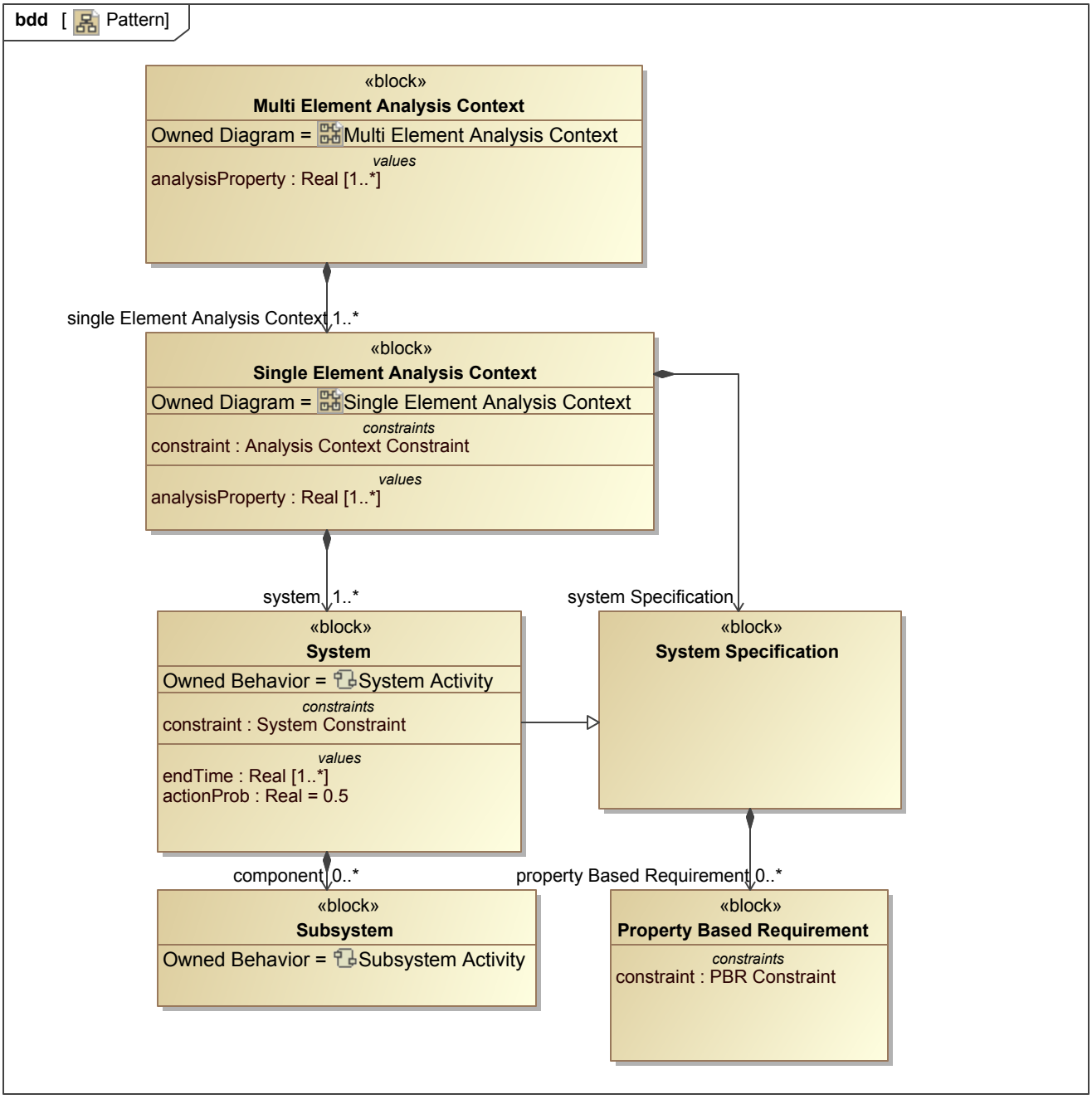

Figure 49. Pattern

If the modeler wants to perform additional analysis on the results of the Monte Carlo Simulation, it is suggested they create additional analysis contexts. The modeler can calculate the standard deviation, average, etc by binding the output (analysisProperty) of the Single Element Analysis Context to the input parameter of the Multi Element Analysis Context. The Multi Element Analysis Context consists of a constraint Analysis Context Constraint, the value property analysisProperty, and a parametric model. The Analysis Context Constraint represents a general constraint that can be used to specify a network of constraints representing mathematical expressions which constrain the physical properties of a system. The modeler can calculate the standard deviation, average, etc by constraining the output (analysisProperty) of the Single Element Analysis Context to the Multi Element Analysis Context. From the Object Oriented Systems Engineering Method (OOSEM), the System Specification supports the analysis, specification, design, and verification of systems. The System Specification may contain Property Based Requirements. If the engineer wants to verify if their system satisfies a requirement they can constrain the results of the Multi Element Analysis Context to a parameter of the PBR Constraint. The System specializes the System Specification, and will inherit its attributes. Note the Single Element Analysis Context must have a composition relationship to both the System and the System Specification.

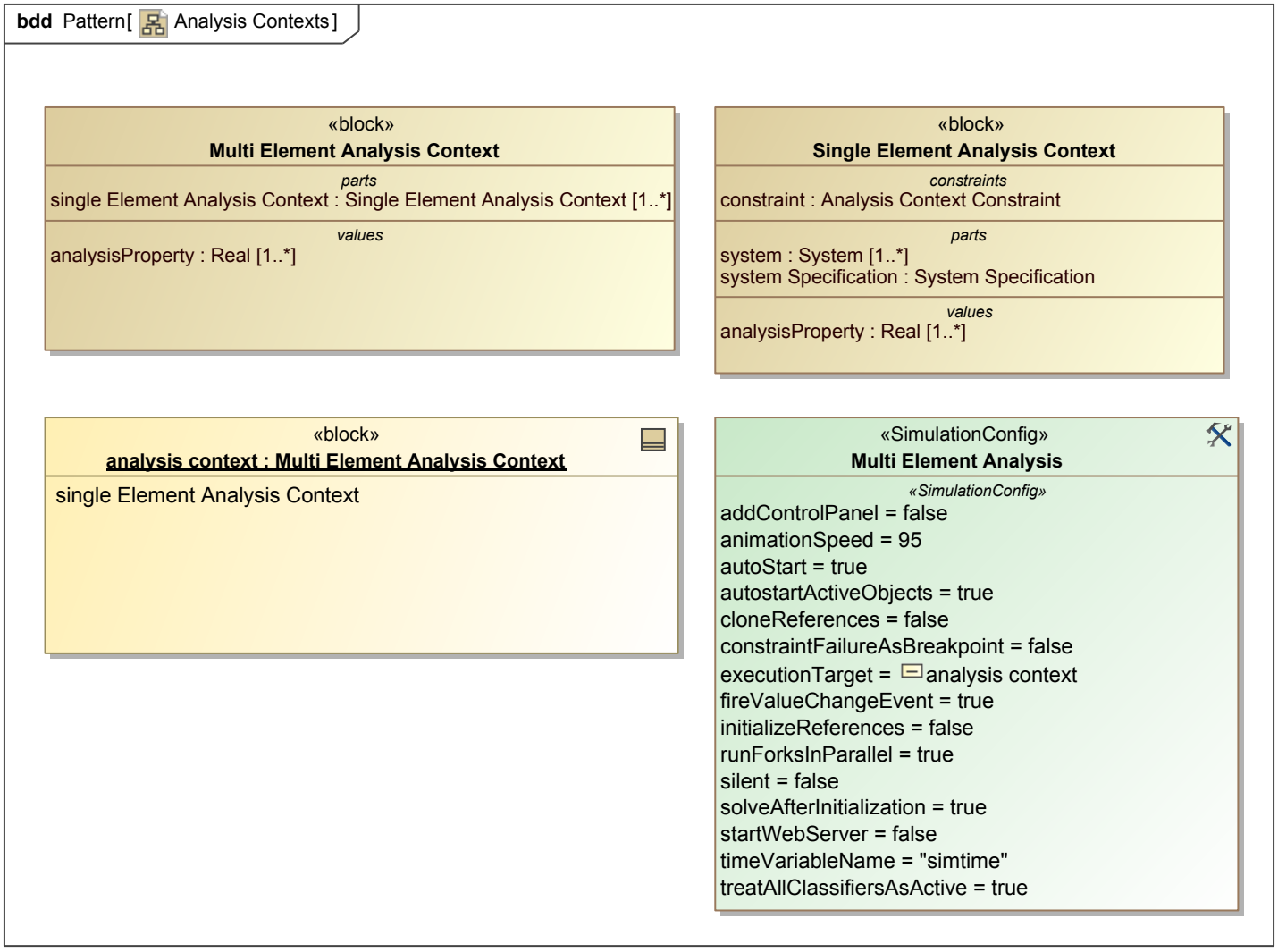

Figure 50. Analysis Contexts

The Multi Element Analysis Context is applied to create a context for performing analysis on the results of the Monte Carlo Analysis simulation. Through the usage of the Multi Element Analysis Context, modelers can bind the instance values of the Monte Carlo Analysis in a parametric model to calculate the average, standard deviation, etc. In the parametric model the modeler would constrain the analysisProperty of the Single Element Analysis Context as the input of the Multi Element Analysis Context. Through a system of constraints and parameters the final value for analysis would be the Multi Element Analysis Context's value property, analysisProperty. A mechanism to select the instance results of the Monte Carlo Analysis is to create an instance of the Multi Element Analysis Context and to specify the single Element Analysis Context slot. A slot specifies that an entity modeled by an instance specification has a value or values for a specific structural feature. The slot value of the single Element Analysis Context can be set equal to the instance results of a particular Monte Carlo run. After the slot value has been specified, the modeler can then run a simulation of the instance. The execution target of the simulation is the Multi Element Analysis Context instance specification.

Table 10. <>

Model Element	
Single Element Analysis Context	The Single Element Analysis Context block provides a context for the system that is subject to analysis. The analysis context is composed of the constraint blocks that correspond to the analysis model and references the system being analyzed. The Single Element Analysis Context parametric diagram could be used to bind the relevant properties of the block (i.e. valueProperty), and the parameters of the analysis model. The only analysis the Single Element Analysis Context should perform is to constrain the endTime value property of the system to its own analysisProperty through a constraint block (Analysis Context Constraint). The Single Element Analysis Context must equal the execution target of the simulation in order to perform a Monte Carlo simulation of the System.
analysisProperty	

Model Element	
Multi Element Analysis Context	The Multi Element Analysis Context is modeled as a block with a composition relationship to the subject of analysis (the System) and any other constraint blocks. The Multi Element Analysis Context can be applied to create a context for performing computational analysis of the Single Element Analysis Context results. The Multi Element Analysis Context could be applied to bind the relevant properties of the block (i.e. analysisProperty) and the parameters of the analysis model. The systems engineer can add different constraints that are relevant to a particular analyses and requirements such as calculating the average or standard deviation of the Single Element Analysis Context instance results. The Analysis Context Constraint represents a general constraint that could be used for computation depending on a specific analysis.
analysisProperty	
Subsystem	The Subsystem element represents a component whose behavior could interact with the System and the Monte Carlo simulation. The probability of a behavior being performed can be specified in its classifying activity.
Property Based Requirement	The requirements of the system can be captured as property based requirements to specify the specification. Through the integration of system elements and/or components, the system specification allows systems engineers to verify that the integrated system design and its realization satisfy its requirements. Property Based Requirements allows systems engineers to model a more formal expression of requirements that retain the properties of a block and a requirement. Therefore, the user is able to specify value, constraint, part, and reference properties to reflect derived textual requirements. Note Property Based Requirements are not required to perform a Monte Carlo simulation. However, they can be used to
System Specification	The System Specification is modeled as a SysML block, and represents the functional specification of a system design. The application of Object Oriented Systems Engineering Method (OOSEM) results in the specification of the system based on the scenario analysis and other engineering analyses performed. The System Specification is specialized by a system. Additionally, it can be composed of Property Based Requirements as it is common practice to group requirements for a system into a specification.
System	The System block represents a top level hierarchical component where the value properties are defined. The properties of the System include a value property endTime which represents the final time to perform a run of a Monte Carlo simulation. The value of the property is set during the owning activity of the System block. The final opaque action of the Activity is specified by the expression endTime = simtime which returns the time from the simulation clock and sets the value equal to the value property. These properties are inherited by any component that specialize the System block. In the specification of the Monte Carlo simulation, the execution target should equal the System block.
endTime	
actionProb	
PBR Constraint	
Analysis Context Constraint	
System Constraint	

5.4 Consequences

There are several trade-offs to consider when applying the Monte Carlo Simulation Pattern to the modeler's system.

Table 11. Consequences Table

Action	Consequence

Usage of multiple analysis contexts	An analysis context block provides a context for the system that is subject to analysis. The analysis context can be composed of constraint blocks and value properties that correspond to the analysis model, and references the system being analyzed. 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. In the context of the Monte Carlo Simulation Pattern, the modeler can decide to apply one global analysis context or multiple analysis contexts for each analysis. The primary difference between the options is level of granularity. If multiple analysis contexts are applied to the system, the system will have a greater granularity because the results of the simulation are encapsulated and provide a higher level view of the system.

5.5 Analysis

Analysis is performed during simulation of the system's owning behavior. The probabilistic flows specified in the activity determine the probability of different outcomes. In the activity, each action is specified with duration constraints. Therefore, when simulated the final time of the simulation is the result of the probabilistic flows and the duration of each action. During each run of the system an instance specification will be produced. The resulting output of the Monte Carlo simulation can be computationally analyzed through a system of equations in a parametric model. The standard deviation or average could be calculated, and compared against a system requirement. Verification ensures that the system design satisfies a corresponding requirement for every defined specified scenario.

5.6 Known Uses

In the new era of Extremely Large Telescopes (ELTs) performance requirements are not the only critical parameters in the design space. Others requirements such as acquisition times and operational behavior of systems can influence the design significantly. Cameo Simulation Toolkit in NoMagic MagicDraw is used to verify timing requirements in the early life-cycle phase through system-level simulation for the LGS MCAO and IRIS acquisition mode. This system modeling approach is effective because integrated model for performing a number of analyses and the diagrammatic language allows for quick trades to be performed: e.g. effect of parallelization of tasks on timing, checks interfaces and specify new requirements)

Operational modes and behavior are modeled using activity and state machine diagrams with Cameo Simulation Toolkit in NoMagic MagicDraw. Scenarios are captured primarily using sequence and activity diagrams.

Verifiable requirements are formally captured using constraints on properties. The TMT identify requirement(s) which are used to design model and perform analysis. The modeler can identify for each mode all plausible scenarios where the requirement(s) will apply, and group them by communalities. Model the scenarios and groups with activity and sequence diagrams. This type of modeling can prove to be particularly useful when wanting to investigate the effect of parallelizing or re-ordering sequence acquisition tasks.

After the requirements have been specified, the modeler can run different scenarios in a Monte-Carlo statistical sense. The systems engineer can compare data obtained from simulations with Requirement(s) and reevaluate model assumptions. This process can be done until there is a converge of data or have reached the desired outcome.

New requirements can be created based on the scenarios simulation, and will be applied to subsystem components (e.g. probe arm speed, close loop settle time, parallelizing activities for sequences).

act [Acquire a target with IRIS and NFIRAOS - Conceptual Actual]

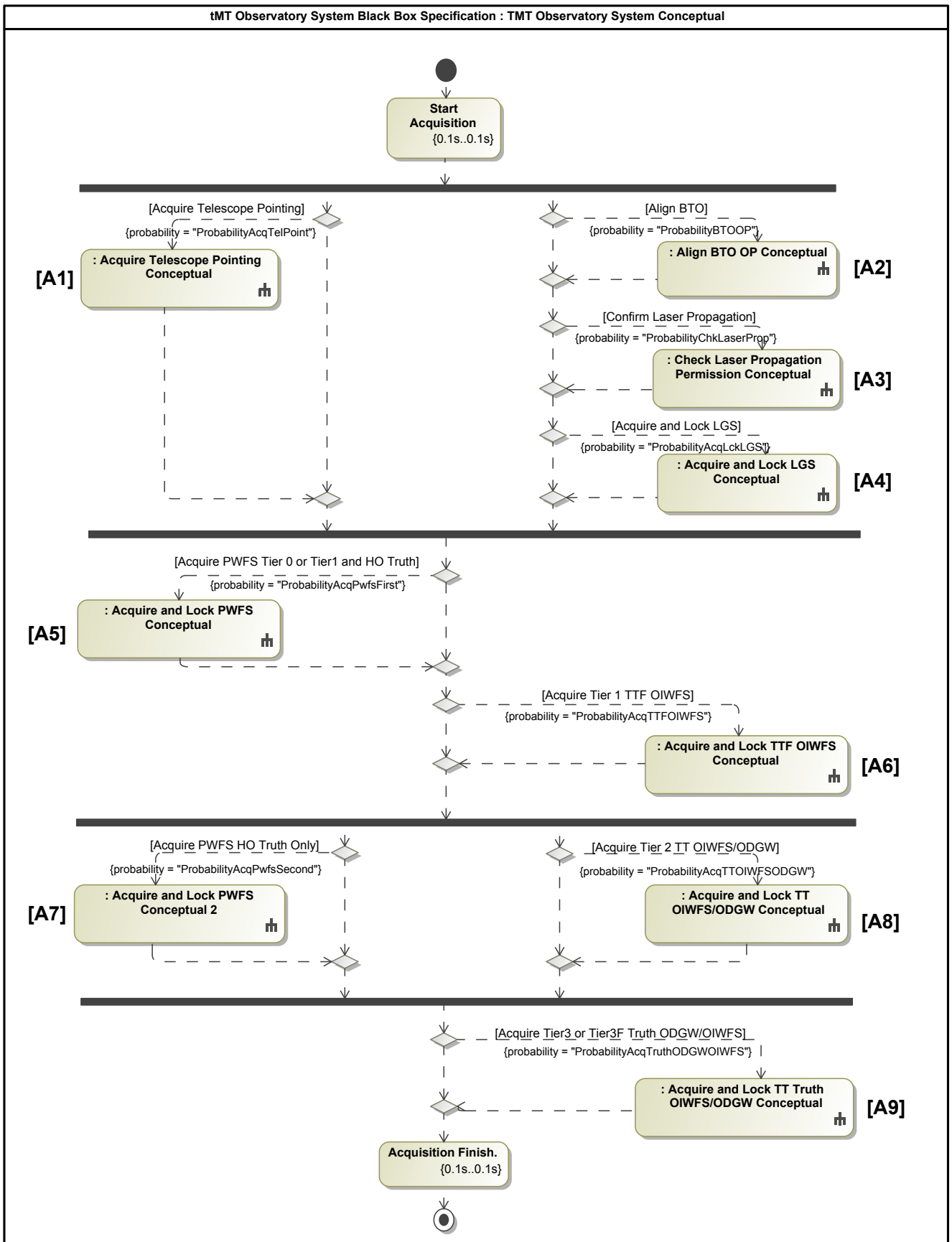

Figure 51. Acquire a target with IRIS and NFIRAOS - Conceptual Actual

To display the application of the probability value properties, one of the owned activities, Acquire a Target with IRIS and NFIRAOS , is shown above. The probability of a transition to an activity is determined by the value specified in each of the Mission to-be Groups. The TotalAcquisitionTime is determined by the probability of an action occurring.

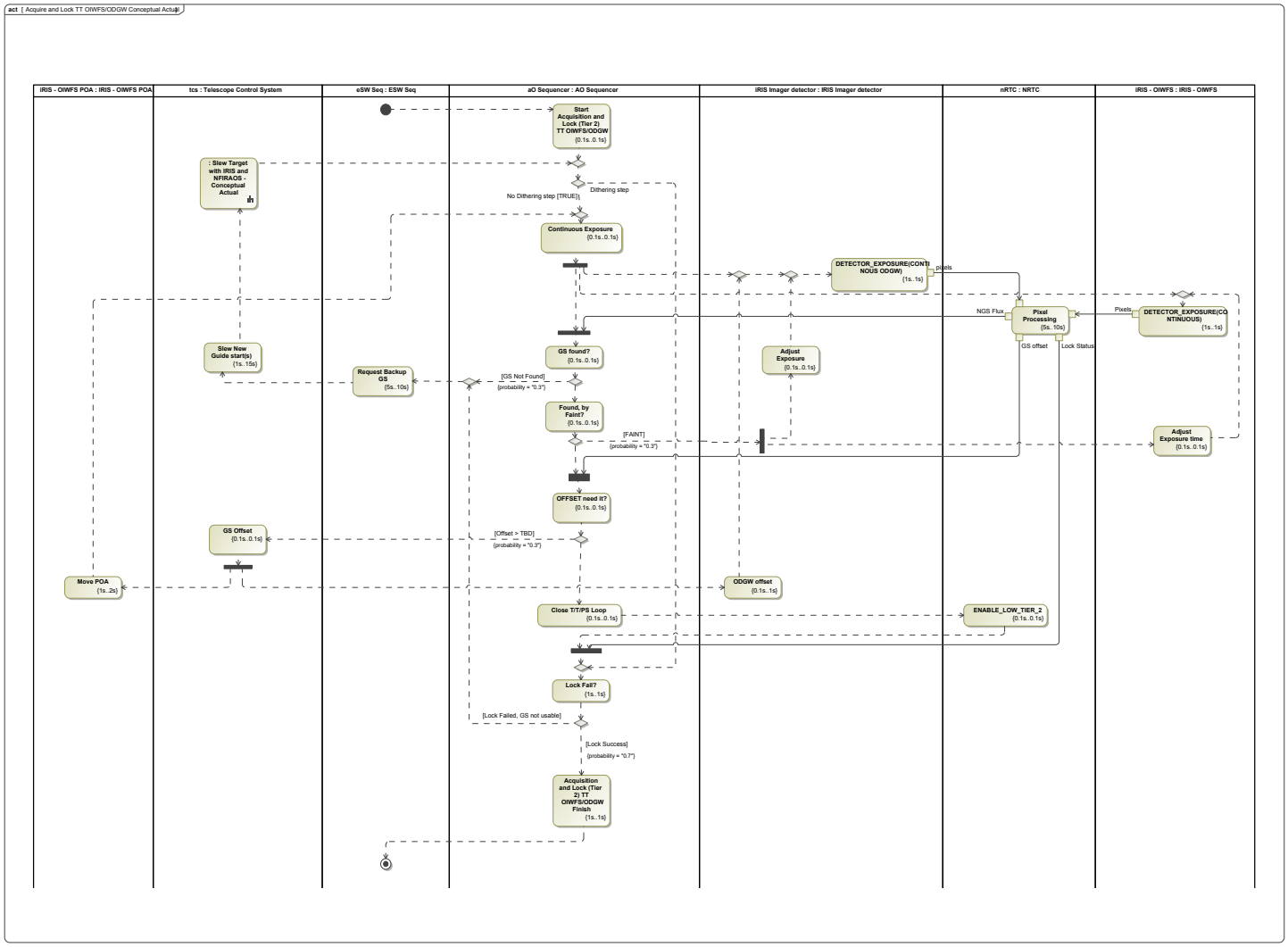

Figure 52. Acquire and Lock TT OIWFS/ODGW Conceptual Actual

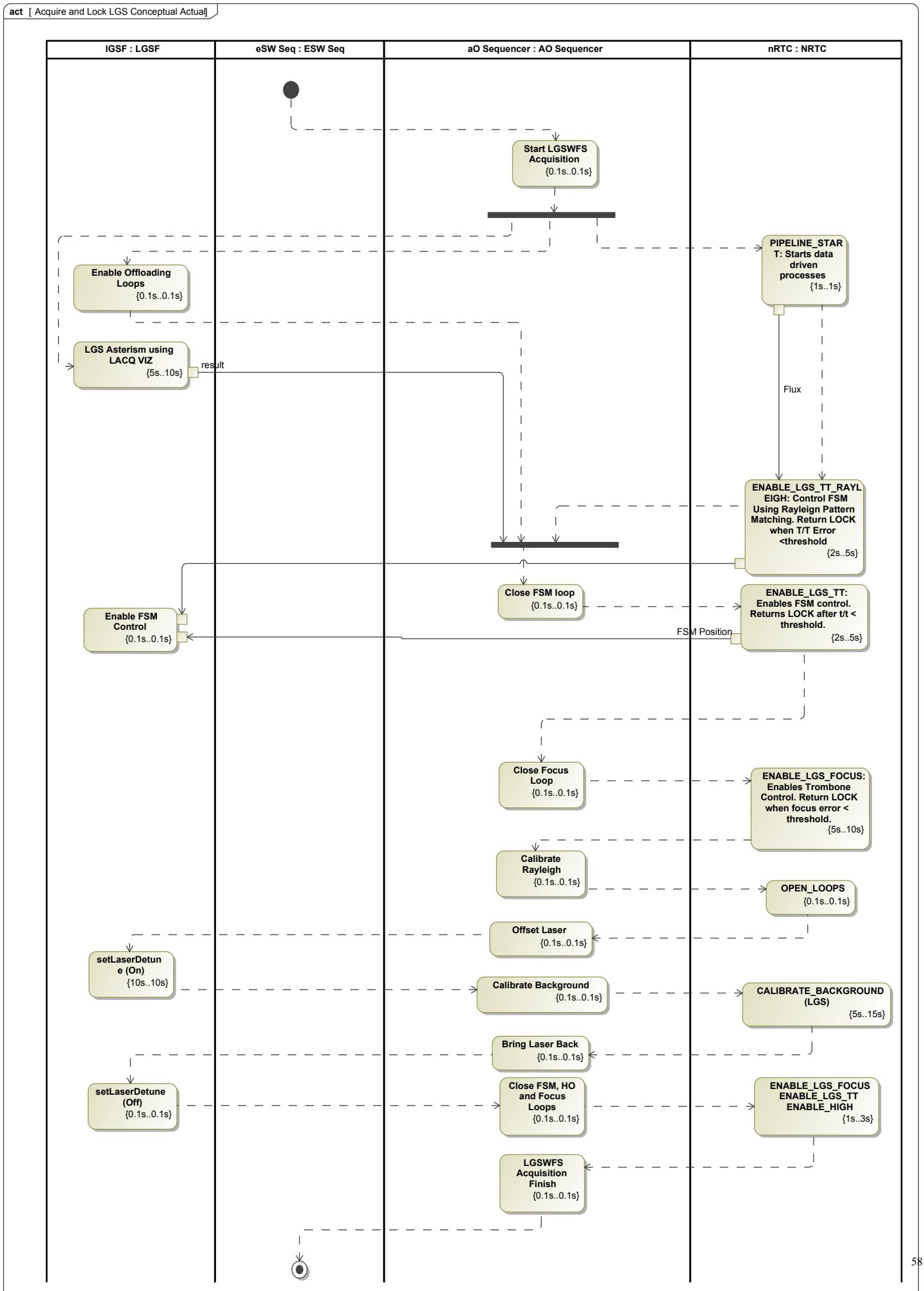
act [Acquire and Lock LGS Conceptual Actual]
IGSF : LGSF
eSW Seq : ESW Seq
aO Sequencer : AO Sequencer
nRTC : NRTC
Start LGSWFS Acquisition {0.1s..0.1s}
PIPELINE_START: Starts data driven processes {1s..1s}
Enable Offloading Loops {0.1s..0.1s}
LGS Asterism using LACQ VIZ {5s..10s}
result
Flux
ENABLE_LGS_TT_RAYLEIGH: Control FSM Using Rayleign Pattern Matching. Return LOCK when T/T Error <threshold {2s..5s}
Close FSM loop {0.1s..0.1s}
ENABLE_LGS_TT: Enables FSM control. Returns LOCK after t/t < threshold. {2s..5s}
Enable FSM Control {0.1s..0.1s}
FSM Position
Close Focus Loop {0.1s..0.1s}
ENABLE_LGS_FOCUS: Enables Trombone Control. Return LOCK when focus error < threshold. {5s..10s}
Calibrate Rayleigh {0.1s..0.1s}
OPEN_LOOPS {0.1s..0.1s}
Offset Laser {0.1s..0.1s}
setLaserDetune (On) {10s..10s}
Calibrate Background {0.1s..0.1s}
CALIBRATE_BACKGROUND (LGS) {5s..15s}
Bring Laser Back {0.1s..0.1s}
setLaserDetune (Off) {0.1s..0.1s}
Close FSM, HO and Focus Loops {0.1s..0.1s}
ENABLE_LGS_FOCUS ENABLE_LGS_TT ENABLE_HIGH {1s..3s}
LGSWFS Acquisition Finish {0.1s..0.1s}

Figure 53. Acquire and Lock LGS Conceptual Actual

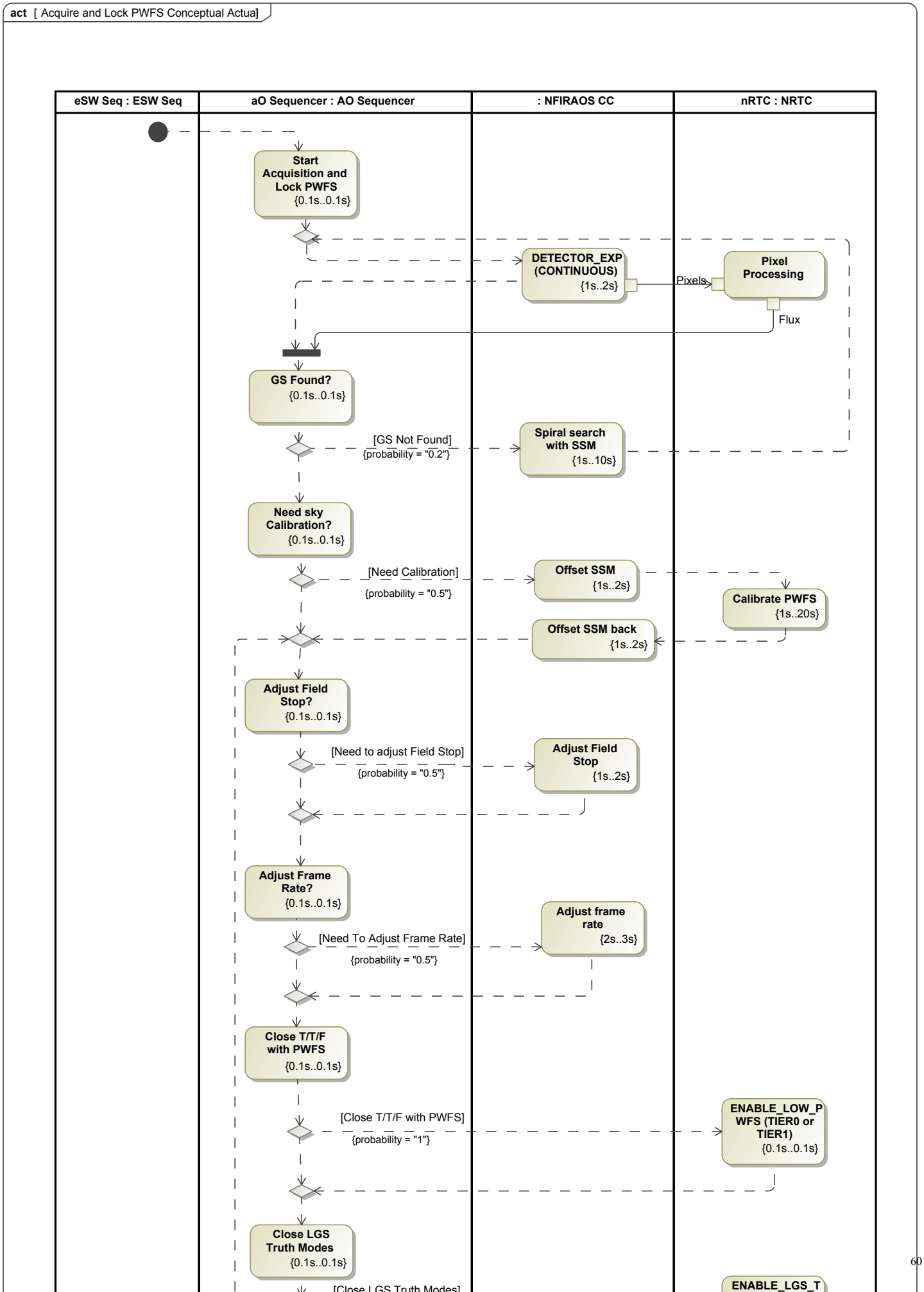
act [Acquire and Lock PWFS Conceptual Actua]
eSW Seq : ESW Seq
aO Sequencer : AO Sequencer
: NFIRAOS CC
nRTC : NRTC
Start Acquisition and Lock PWFS {0.1s..0.1s}
DETECTOR_EXP (CONTINUOUS) {1s..2s}
Pixels
Pixel Processing
Flux
GS Found? {0.1s..0.1s}
[GS Not Found] {probability = "0.2"}
Spiral search with SSM {1s..10s}
Need sky Calibration? {0.1s..0.1s}
[Need Calibration] {probability = "0.5"}
Offset SSM {1s..2s}
Calibrate PWFS {1s..20s}
Offset SSM back {1s..2s}
Adjust Field Stop? {0.1s..0.1s}
[Need to adjust Field Stop] {probability = "0.5"}
Adjust Field Stop {1s..2s}
Adjust Frame Rate? {0.1s..0.1s}
[Need To Adjust Frame Rate] {probability = "0.5"}
Adjust frame rate {2s..3s}
Close T/T/F with PWFS {0.1s..0.1s}
[Close T/T/F with PWFS] {probability = "1"}
ENABLE_LOW_PWFS (TIER0 or TIER1) {0.1s..0.1s}
Close LGS Truth Modes {0.1s..0.1s}
[Close LGS Truth Modes]
ENABLE_LGS_T

Figure 54. Acquire and Lock PWFS Conceptual Actual

Figure 55. Acquisition Simulation

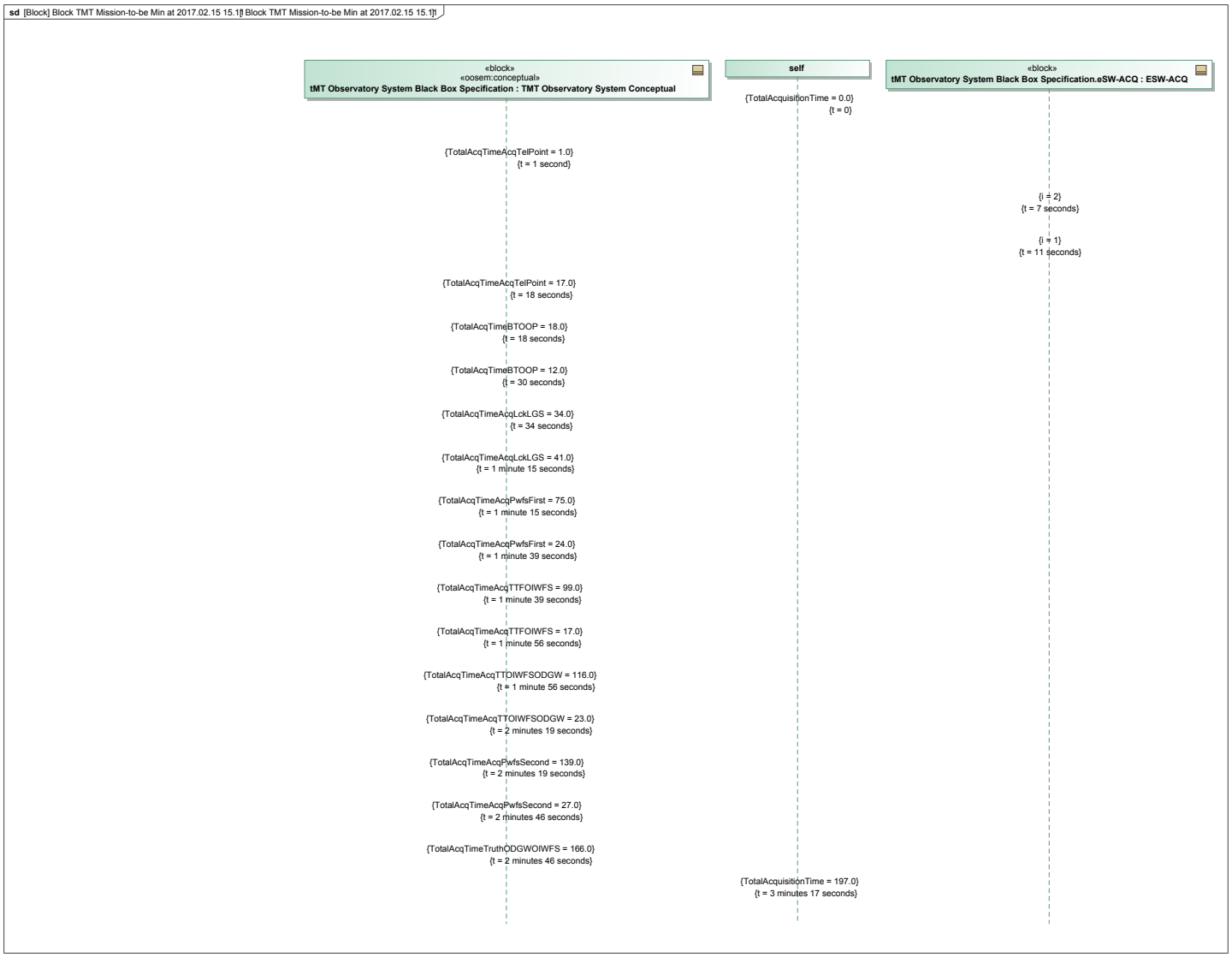

Figure 56. Block TMT Mission-to-be Min at 2017.02.15 15.11

#	Name	ProbabilityAcqTe	ProbabilityBTOO	ProbabilityChkLa	ProbabilityAcqL	ProbabilityAcqP	ProbabilityAcqT	ProbabilityAcqP	ProbabilityAcqT
1	tmt mission to-be grou	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0
2	tmt mission to-be grou	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0

Figure 57. Probability Instance Table

5.7 Related Patterns

List of other patterns that are related to the Monte Carlo Simulation Pattern **, and the differences and similarities between them.**

Pattern Name	Similarities	Differences
Dynamic Constraint Pattern	In the Monte Carlo Simulation Pattern constraints can be applied in the Analysis Context block in order to perform analysis of the Monte Carlo Simulation. The Dynamic Constraint Pattern can be applied to specialize the constraint that is a part of the Analysis Context.	The purpose of the Dynamic Constraint Pattern is to provide a method for how to model component specific behavior of a component through the dynamic usage of constraints by redefining the behavior of a component to satisfy a system requirement. In primary purpose of the Monte Carlo Simulation Pattern is to show how the decision making tool can be applied in statistical analysis of SysML models.

Requirement Verification Pattern	In the sample model of theMonte Carlo Simulation Pattern the system under analysis is verified against a system requirement. The method for requirement verification is described in the Requirement Verification Pattern. Additionally, the relationship between the system specification and system of the Monte Carlo Simulation Pattern are modeled identically as described in the Black Box Specification with Property Based Requirements Pattern. The System and Subsystem components are specializations of the System Specification.	The primary purpose of the Requirement Verification Pattern is to evaluate a system based on the probability of uncertain actions and values being performed. Requirement verification is a method of analyzing the system after running a Monte Carlo simulation.
Black Box Specification with Property Based Requirements Pattern	A System Specification is applied in the structure of theMonte Carlo Simulation Pattern. The relationship between the system specification and property based requirements is further described in theBlack Box Specification with Property Based Requirements Pattern.	The Black Box Specification with Property Based Requirements Pattern doesn't provide a context to analyze and execute the system unlike the Monte Carlo Simulation Pattern.

5.8 Implementation

The purpose of the sample model is to demonstrate how users can perform a Monte Carlo Simulation of their system based on a timing operational scenario. The sample model uses the Monte Carlo plugin that is described in Monte Carlo Plugin. By running a Monte Carlo simulation of the system, the transitions of the activities of the system will be determined based on probability.

5.8.1 Example 1

The purpose of the sample model is to demonstrate how users can perform a Monte Carlo Simulation of their system based on a timing operational scenario. The sample model uses the Monte Carlo plugin that is described in Monte Carlo Plugin. By running a Monte Carlo simulation of the system, the transitions of the activities of the system will be determined based on probability.

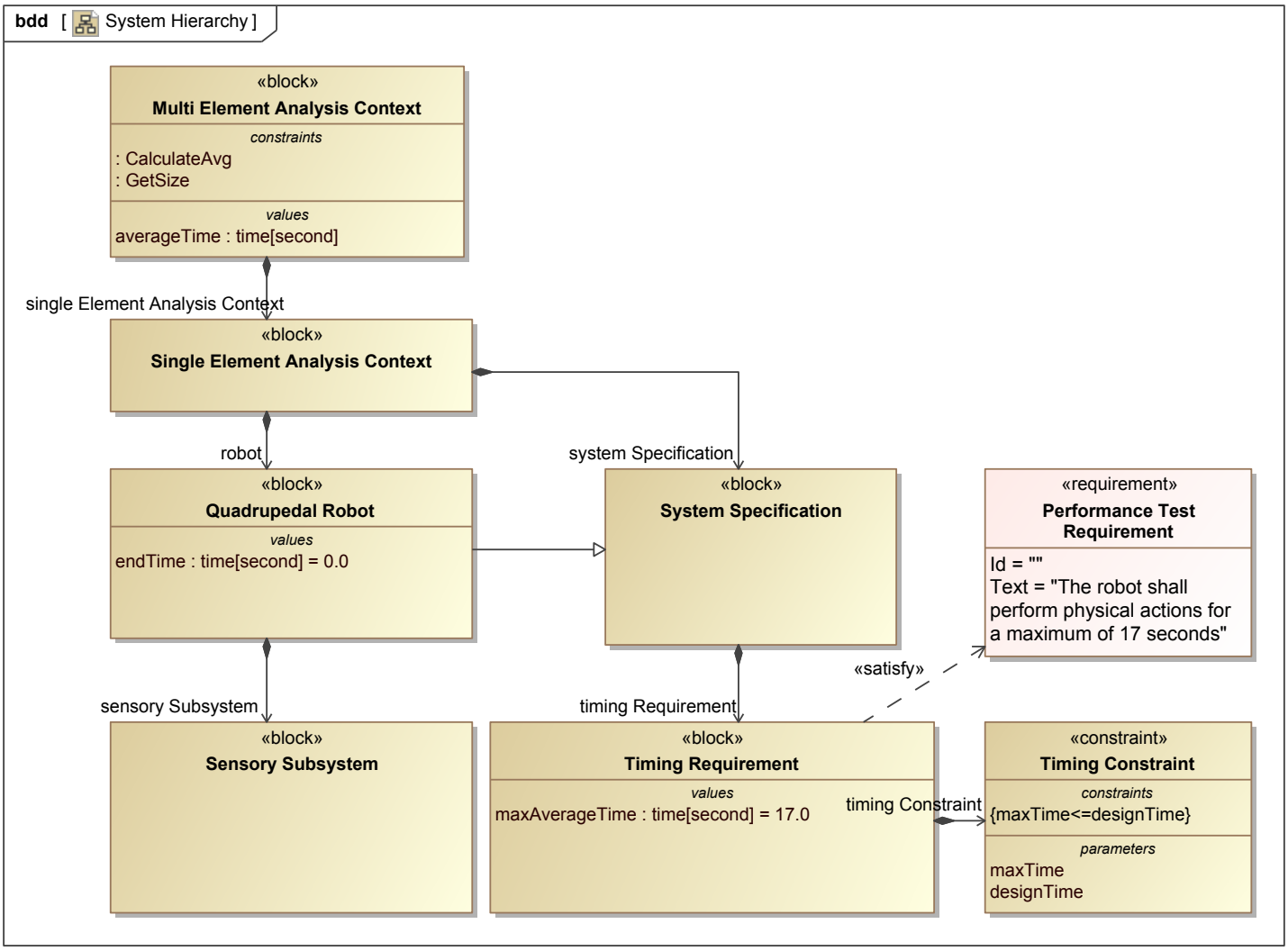

Figure 58. System Hierarchy

The system for analysis is a Quadrupedal Robot that is composed of Sensory Subsystem, and owns the value property endTime. The value of the endTime is initialized to zero, and represents the final time of the system. The Quadrupedal Robot generalizes the System Specification, and will inherit the Timing Requirement. The System Specification represents the functional specification of a system design. The Quadrupedal Robot must satisfy the Timing Requirement. The Timing Requirement is modeled as a property based requirement, and is derived from the textual requirementPerformance Test Requirement. The Timing Constraint is required to evaluate whether the endTime property of the Quadrupedal Robot meets the maxAverageTime property of the Timing Requirement.

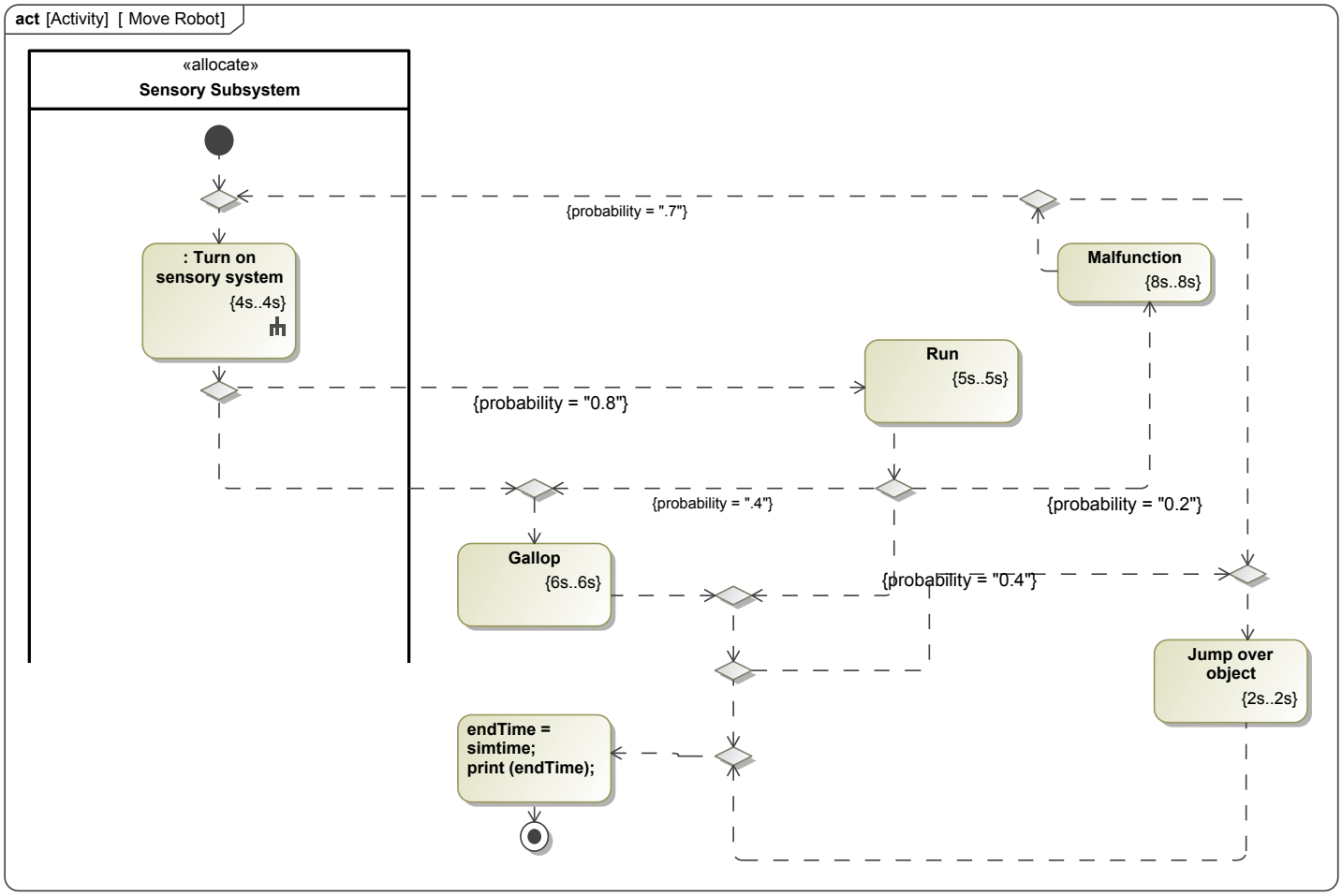

Figure 59. Move Robot

The classifier behavior of the Quadrupedal Robot is specified in the Move Robot activity. The swimlane represents the Sensory Subsystem, and indicates that any behaviors invoked are the responsibility of the Sensory Subsystem. Each action owns a duration constraint which expresses a constraint on the duration of the action. During simulation, the paths will be determined based on the probability of each control flow. The flows are tagged with a probability to specify the likelihood that a given token will traverse a particular flow among available alternative flows.

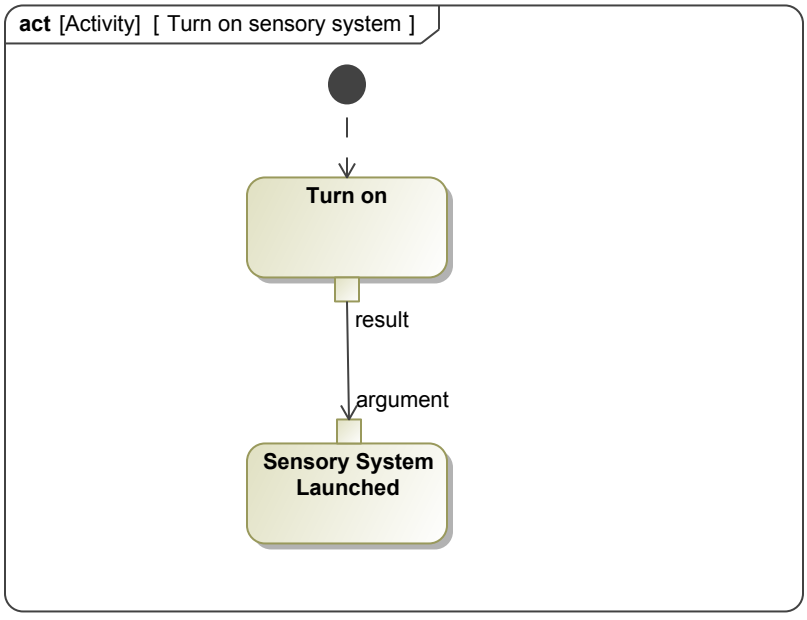

Figure 60. Turn on sensory system

An additional classifier behavior of the Quadrupedal Robot is the Turn on sensory system activity. In the activity, the call behavior actions Turn on and Sensory System Launched will be executed. The Turn on sensory system activity is executed during the Move Robot activity.

Table 12. Timing Results

Name	endTime : time[second]
robot at 2017.07.20 17.38	10.0
robot at 2017.07.20 17.38	12.0
robot at 2017.07.20 17.38	15.0
robot at 2017.07.20 17.38	12.0
robot at 2017.07.20 17.38	17.0

The results of the Timing Analysis can be displayed in an instance table. Since five runs were specified, there were five instance specifications produced. The values of the endTime were determined by the probabilistic flows of the Move Robot activity. These instance specifications will be the target of further analysis as described in the Analysis Context Instance.

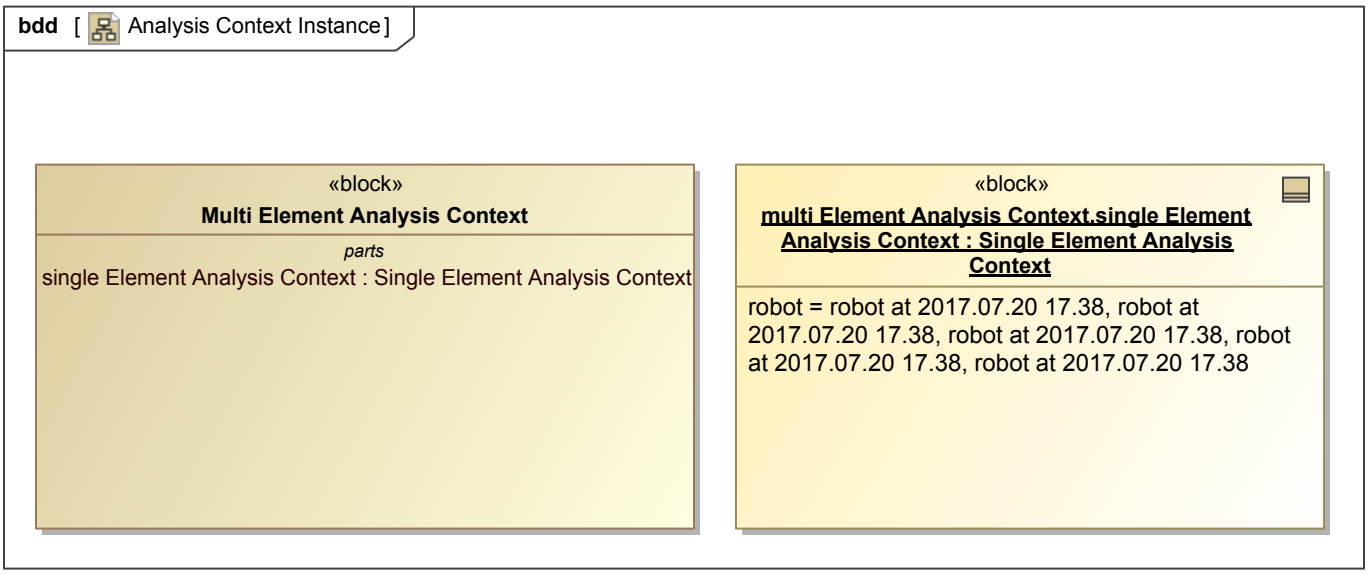

Figure 61. Analysis Context Instance

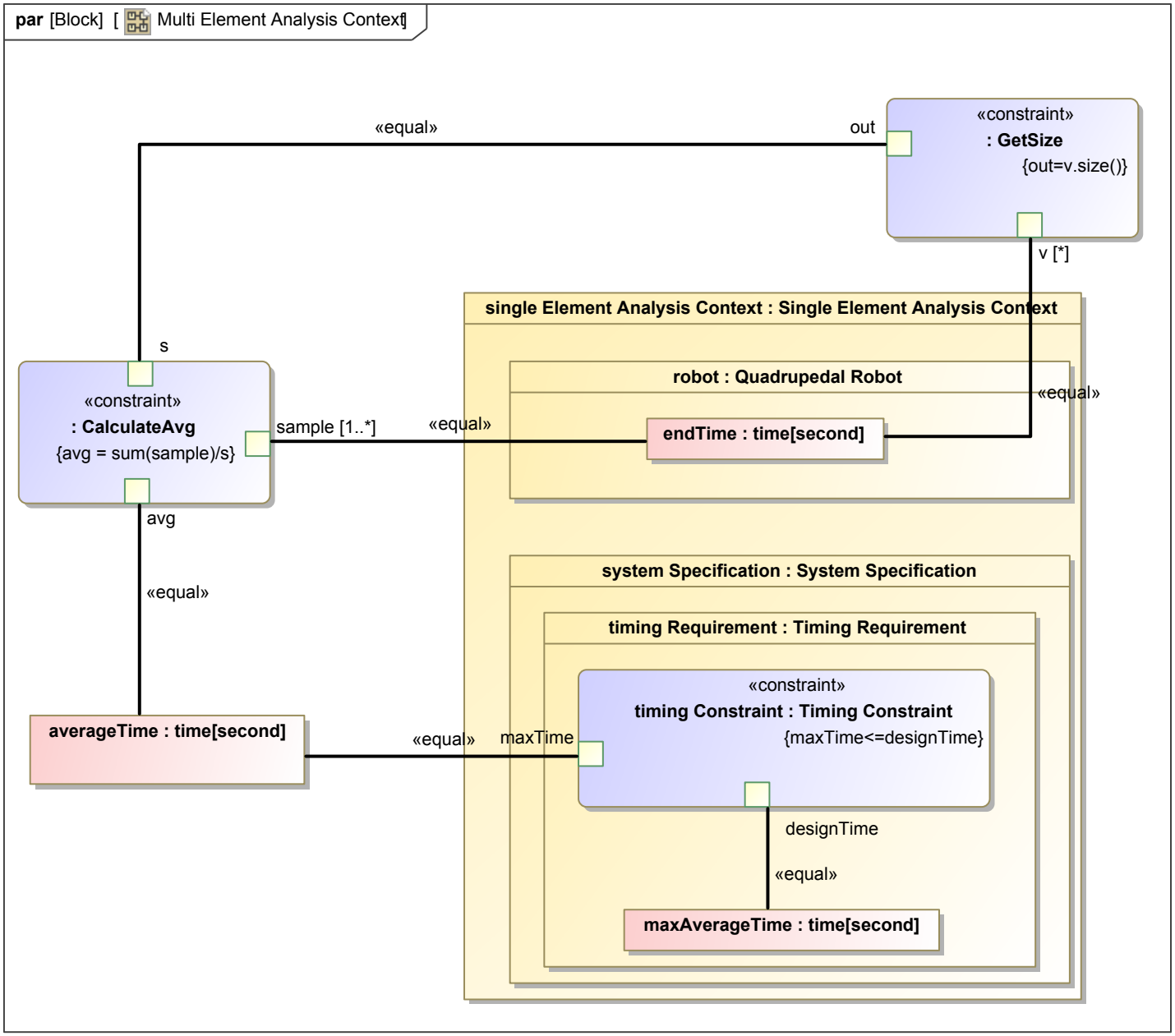

Figure 62. Multi Element Analysis Context

The parametric diagram of the Multi Element Analysis Context displays the system of equations to constrain the averageTime property. The systems of equations are required to calculate the average time from the instance specifications of the Monte Carlo simulation. The constraint GetSize is required to obtain the number of endTime value properties of the Quadrupedal Robot. The endTime property is binded to the parameter v. In the constraint expression, the java method will return the number of elements. The output parameter of the constraint, out, binds the total number of endTime elements to the s parameter of the CalculateAvg constraint. The CalculateAvg is required to calculate the average time of the results of the Monte Carlo instance specifications. The constraint expression will calculate the average time by summing the values of all endTime properties (equal to the parameter sample) and dividing by the total number of properties (equal to the parameter s). The average of the endTime properties is binded to the averageTime property, and allows the system engineer to perform requirement verification. The design of the Quadrupedal Robot can then be verified against the Timing Requirement.

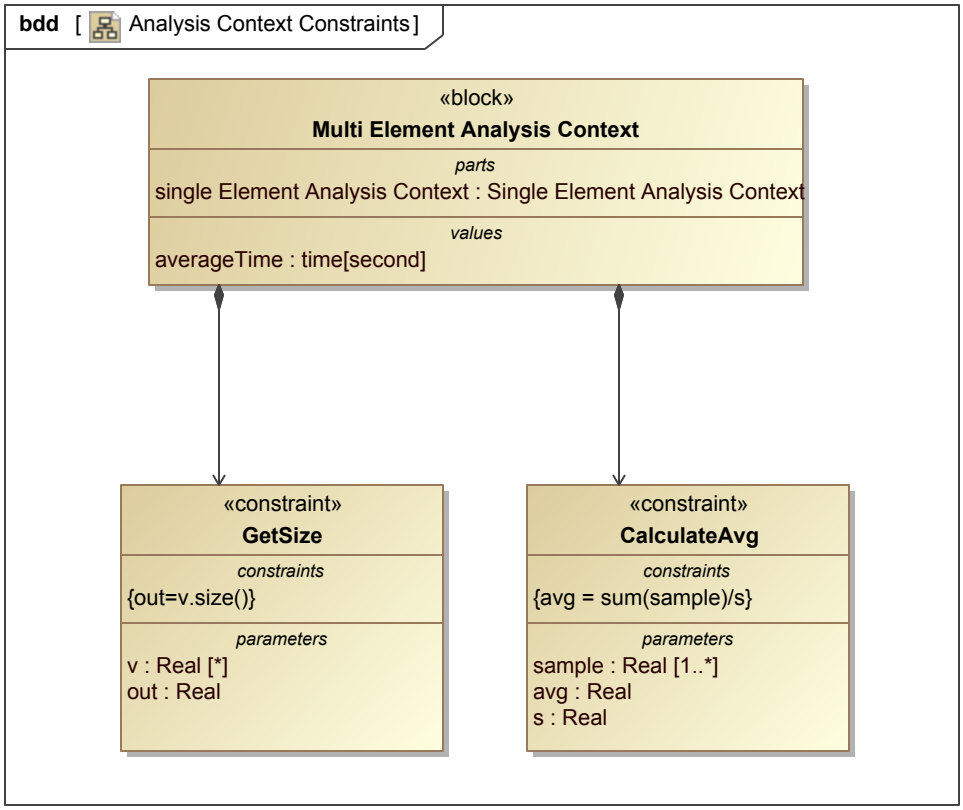

Figure 63. Analysis Context Constraints

The purpose of the Multi Element Analysis Context is to provide a context to perform computational analysis of the Quadrupedal Robot. The analysis context is composed of the constraints GetSize and CalculateAvg. The GetSize is applied to obtain the number of elements in the robot slot of theMulti Element Analysis Context instance specification. After obtaining the total number of elements, the average can be calculated through the application of the CalculateAvg constraint.

Table 13. Calculation Results

Name	system Specification.timing Requirement.timing Constraint : Timing Constraint	robot.endTime : time[second]
single Element Analysis Context at 2017.07.28 11.32	pass	

The results of the Timing Analysis can be displayed in an instance table. Since five runs were specified, there were five instance specifications produced. The values of the endTime were determined by the probabilistic flows of the Move Robot activity. These instance specifications will be the target of further analysis as described in the Analysis Context Instance.

Verification ensures that the system design satisfies a corresponding requirement for every defined specified scenario. The resulting instance specifications of the Timing Calculations simulation can be displayed in an instance table. To display whether the average time satisfies the Timing Requirement, a column for the Timing Constraint is created through the "Select Nested Columns" option of the tab. After selecting the constraint, the value of the timing Constraint slot is displayed. The value of the average time satisfies the property based requirement, Timing Requirement. The value of the avg is less than or equal to the value of the maxAverageTime.

5.8.2 Example 2

Here we have a system that takes measurements and requirements for a Turkey Dinner and evaluates if there will be No Dinner, a Fine Dinner, or a Great Dinner. For this example, the dinner option is chosen by a probability. The purpose of this example is to demonstrate how to a Monte Carlo simulation can be used to investigate this type of system. This example model uses the Monte Carlo plugin described here: Monte Carlo simulation.

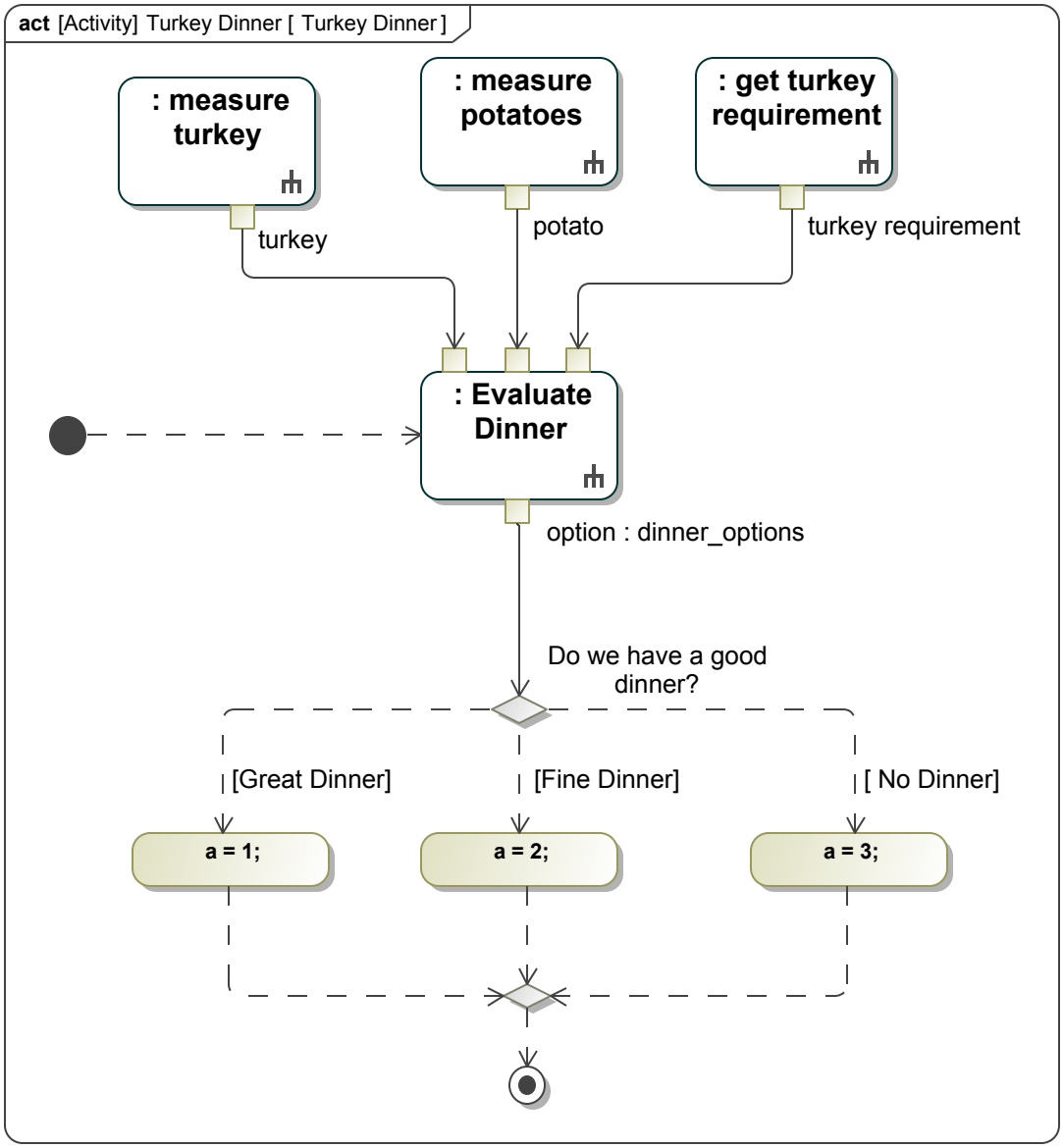

Figure 64. Turkey Dinner

This example model is answering the question "Do we have a good dinner?" based on requirements and measurements of turkey and potatoes. The options for dinner are Great Dinner, Fine Dinner, and No Dinner. There are no values set for the measurements or requirements so a Monte Carlo Simulation has been utilized to execute the scenarios and investigate the outcomes. The value "a" is being set to 1, 2, or 3 so the statistics that are built into Monte Carlo have numerical values that align with the three options.

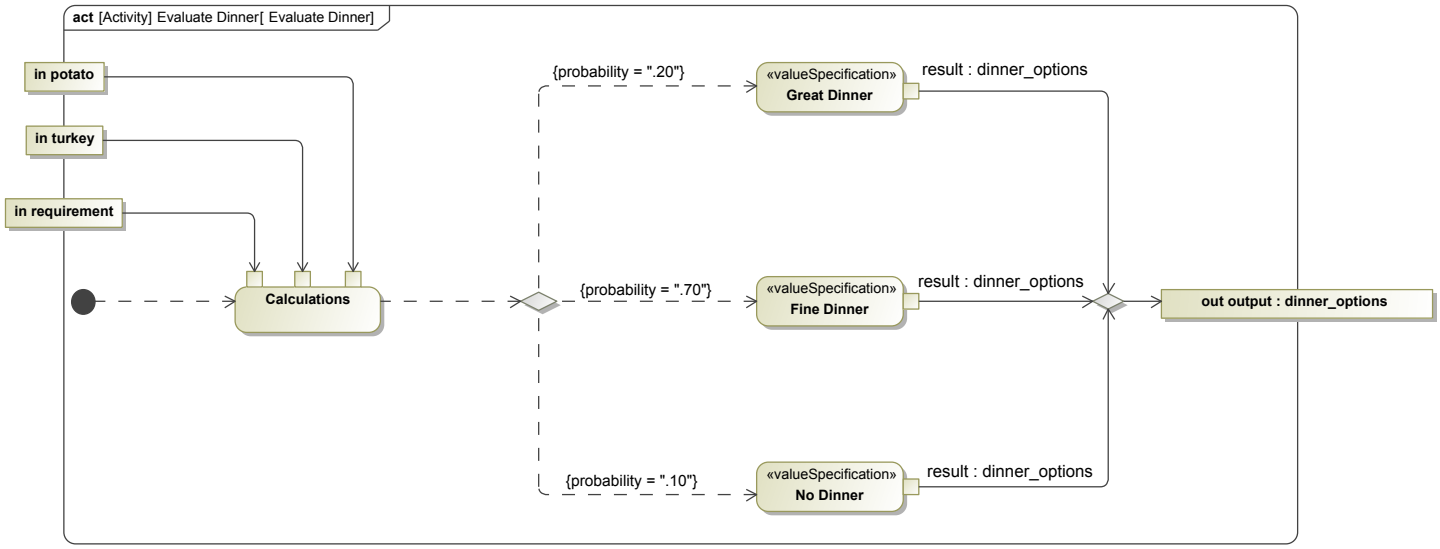

Figure 65. Evaluate Dinner

In Evaluate Dinner the activity Calculations is a place holder for some mathematical calculation based on the three inputs given, potato, turkey and requirement. The three inputs shown here do not have actual values associated with them in this example, but they could. Each option has a probability as a guard on the transition, this probability is what controls how often each option is chosen. The output is the selection of the dinner options, Great Dinner, Fine Dinner, or No Dinner.

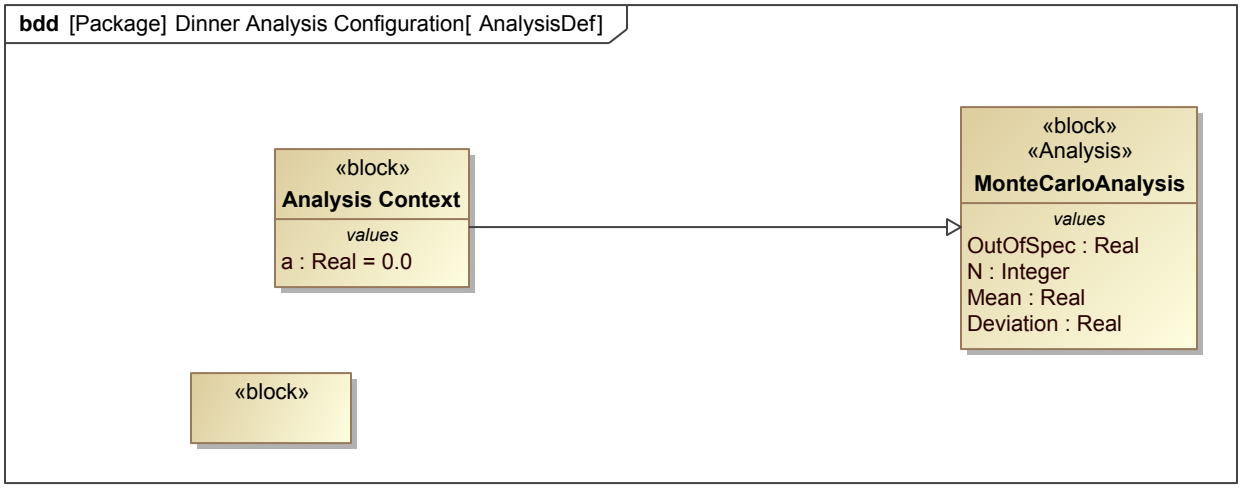

Figure 66. AnalysisDef

This BDD is showing the Analysis Context for this example is generalized by the MonteCarloAnalysis Block which can be found in the MD Customizations for SysML model. This will enable the simulation configuration to have a set number of runs and automatically save mean and standard deviation from Monte Carlo.

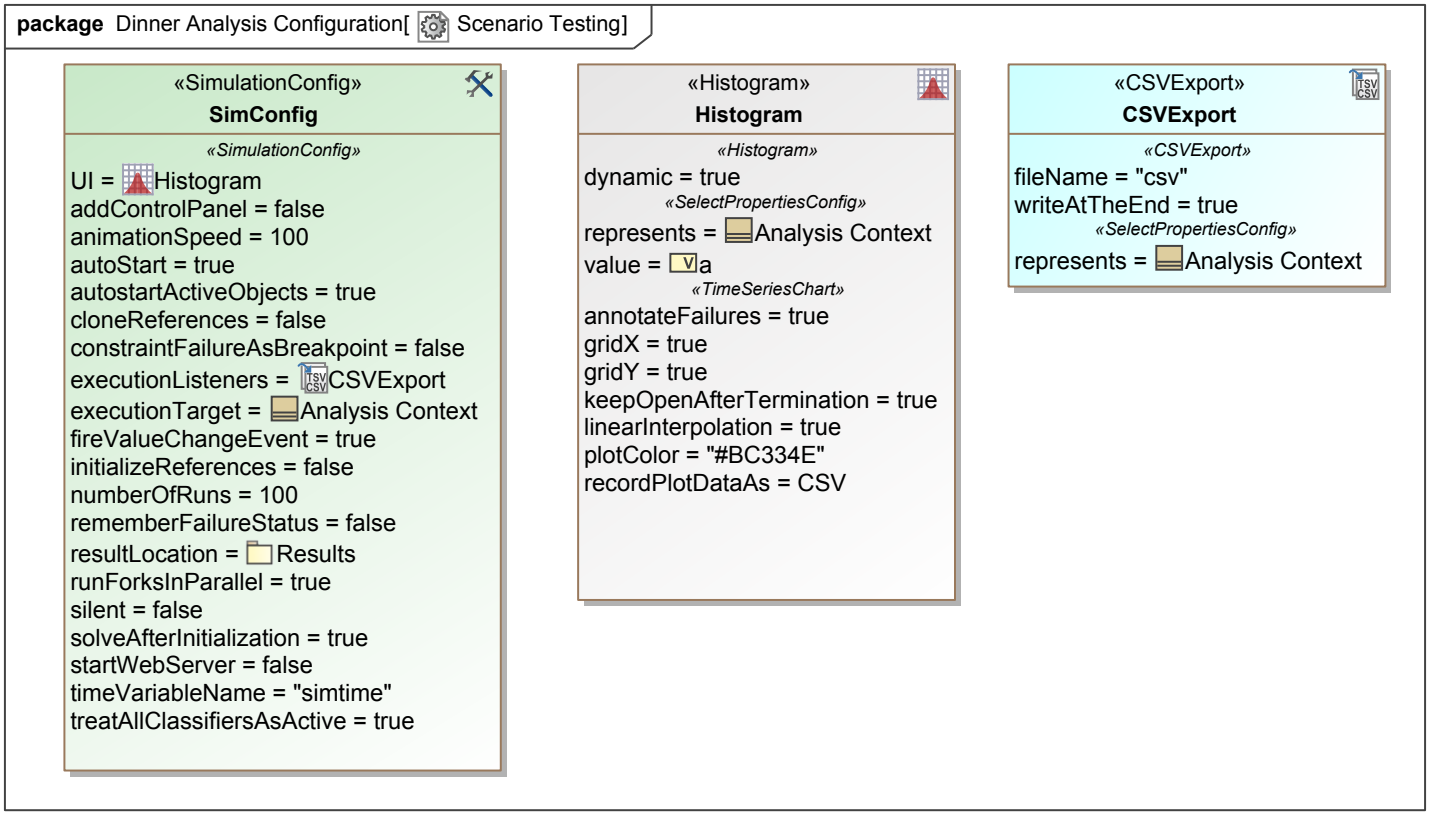

Figure 67. Scenario Testing

Shown here is the simulation configuration of the system. For the Monte Carlo simulation to work properly you need to set the UI as a Histogram, executionTarget as the analysis context block, numberOfRuns, resultLocation, and you can include an executionListener. The Histogram configuration has the value "a" and represents Analysis Context. The CSVExport configuration sets up an exported CSV file with the results of each run.

Table 14. results

Name	Classifier	a : Real	N : Integer	Mean : Real	Deviation : Real
analysis Context at 2020.01.09 09.38	Analysis Context	1.86	100	1.86	0.5508487482370669
analysis Context at 2020.01.09 10.03	Analysis Context	1.81	100	1.81	0.5064233865696236
analysis Context at 2020.01.09 12.46	Analysis Context	1.92	200	1.92	0.514708289340261
analysis Context at 2020.01.09 12.53	Analysis Context	1.9016	10000	1.9016	0.5429059906571939
analysis Context at 2020.01.14 12.35	Analysis Context	1.8987341772151898	158	1.8987341772151898	0.5071447916682155
analysis Context at 2020.01.14 13.09	Analysis Context	0.0	100	0.0	0.0

The results of the simulation are shown in an instance table. The table above includes the name of the instance, the classifier, the last value of "a", the number of runs "N", the mean value of the runs and the deviation. The instance table can be set to display different columns if desired.

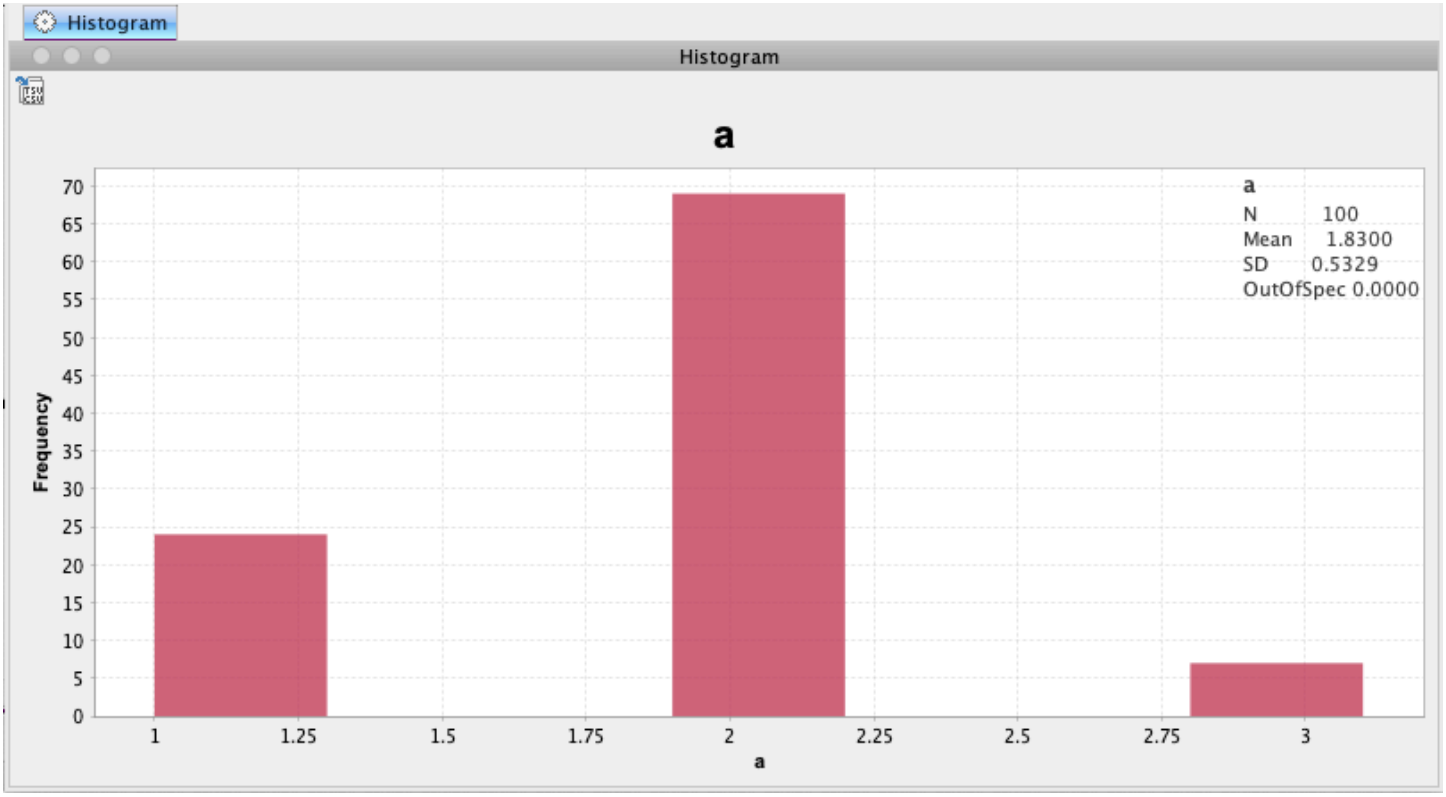

Figure 68. Histogram

Shown here is a histogram generated from 100 runs of the Turkey Dinner simulation. The red bars represent how many times the simulation chose each of the dinner options. The legend in the upper right corner shows the calculated statistics from Monte Carlo. The first bar is "Great Dinner", the second is "Okay dinner", and the third is "No Dinner". You can see that the probabilities defined in Evaluate Dinner are reflected here.

5.9 Tooling

The following tooling supports the modeler in the implementation of the Monte Carlo Simulation Pattern in MagicDraw.

5.9.1 Cameo Simulation Toolkit

The user can produce the same results as the Monte Carlo plugin through Cameo Simulation Toolkit. The following example model displays how to perform a Monte Carlo simulation using CST to evaluate the final time of a system. See [md-cst-montecarlo] in the OpenMBEE repository to download the sample model.

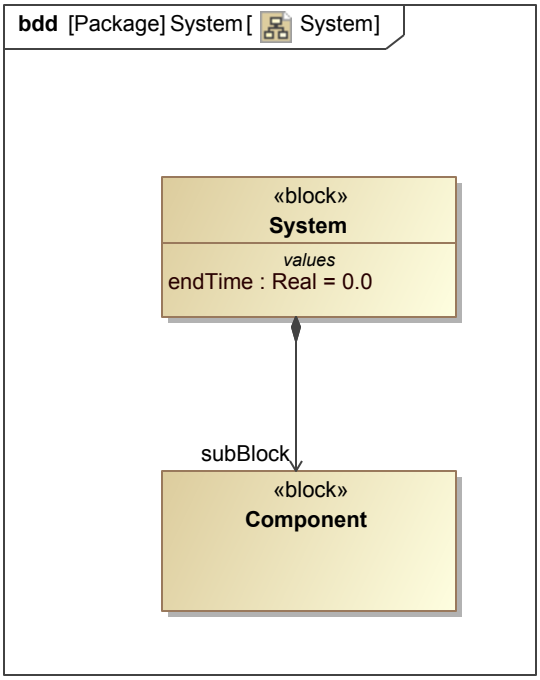

Figure 69. System

The System owns a value property (endTime) that is the output of the Monte Carlo simulation. Additionally, the System can can be composed of a component if needed.

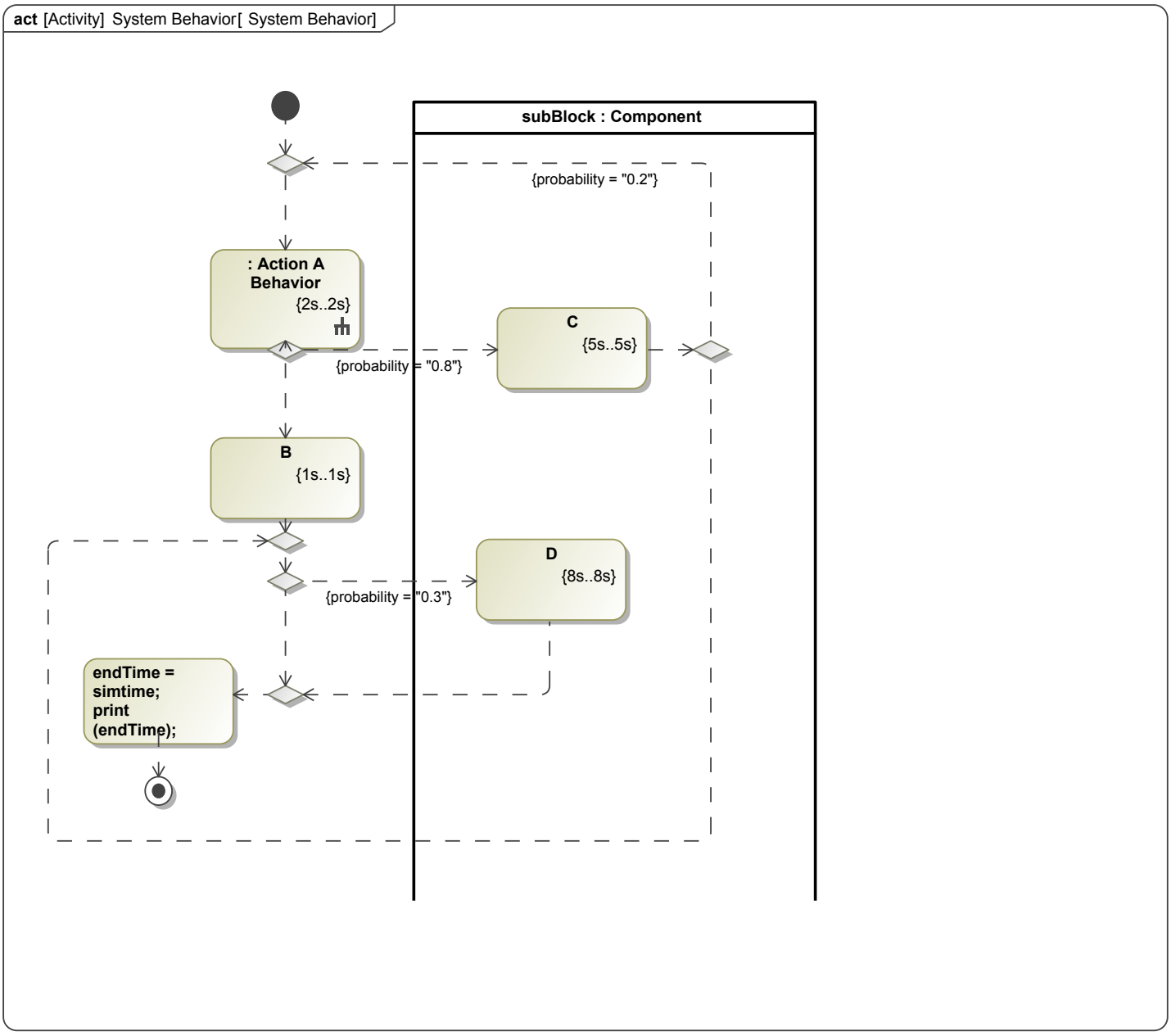

Figure 70. System Behavior

In the owning behavior of the System, the probability of certain actions to be performed are modeled with following SysML model elements- actions, opaque actions, control flows, decision nodes, merge nodes, and swimlanes. The output of the simulation is the result of the specified probability values.

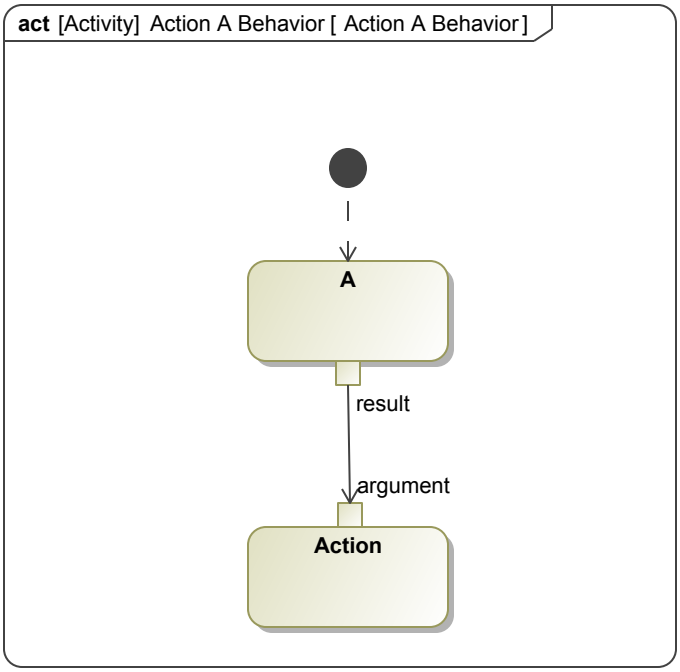

Figure 71. Action A Behavior

During the System Behavior activity, the nested Action A Behavior activity will be performed.

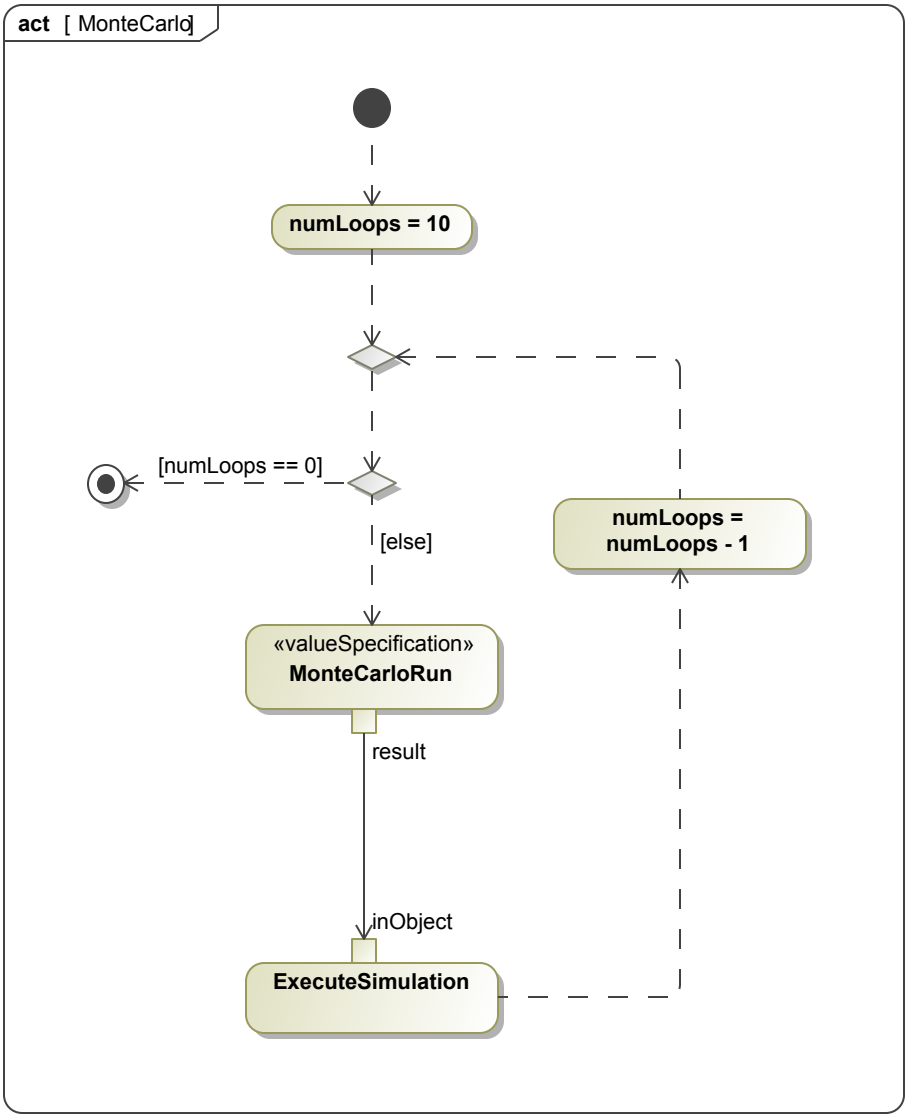

Figure 72. MonteCarlo

The MonteCarlo activity functions similarly to the Monte Carlo plugin. The user can specify the number of runs, specify the target simulation configuration, and execute the simulation until all runs have been completed.

Body and Language	runConfig = ALH.createObject(inObject);
	session =
	com.nomagic.magicdraw.simulation.SimulationManager.execute(runConfig.getTypes().get(0), true, true);
	print("Simulation running...");
	while(!session.isClosed()) {
	  print("Simulation not yet done...");
	  java.lang.Thread.sleep(1000);
	}

Figure 73. Body and Language

The body and language of the opaque action, ExecuteSimulation, is specified with the above content.

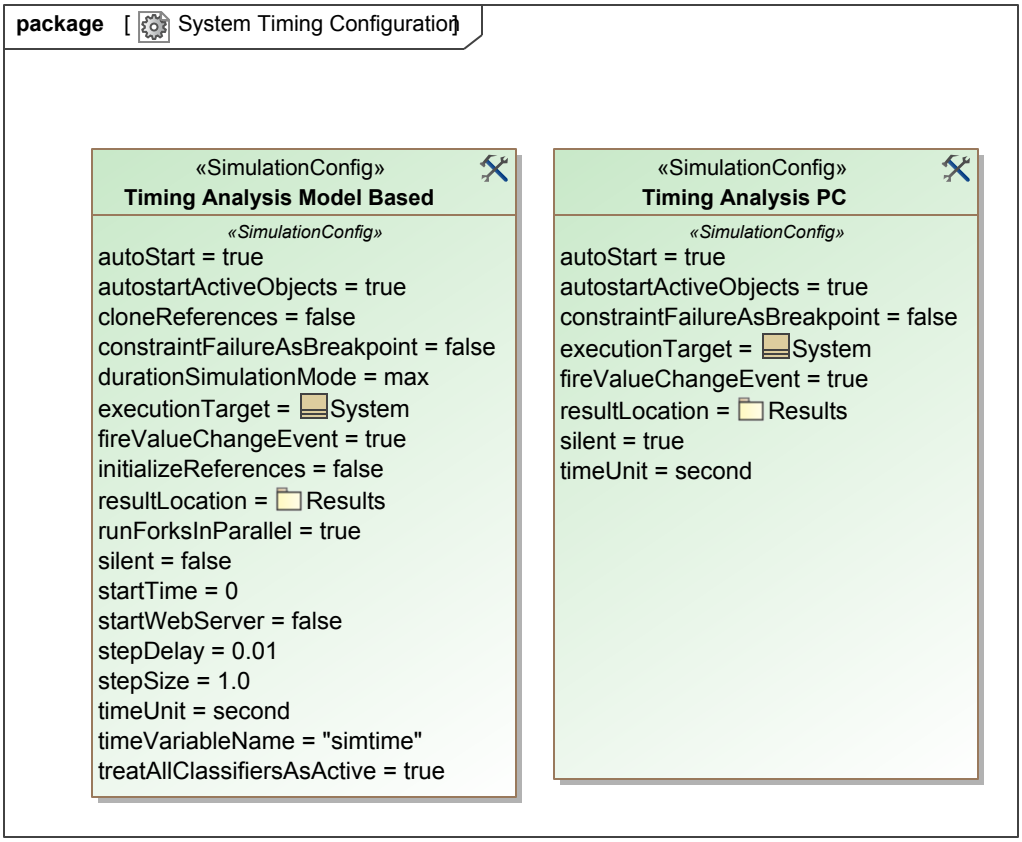

Figure 74. System Timing Configuration

Simulation configurations are required to execute a Monte Carlo simulation of the System to determine the final time of a specified number of runs.The modeler can perform a simulation of the system using either the model based clock or the PC clock.

Table 15. Timing Analysis Results

Name	endTime : Real	subBlock : Component
system at 2017.07.18 18.19	3.0	system.component7
system at 2017.07.18 18.19	7.0	system.component8
system at 2017.07.18 18.20	7.0	system.component10
system at 2017.07.18 18.20	15.0	system.component9
system at 2017.07.18 18.21	15.0	system.component11
system at 2017.07.20 14.18	0.127	system.component15
system at 2017.07.20 14.18	0.124	system.component16
system at 2017.07.20 14.18	0.046	system.component12
system at 2017.07.20 14.18	0.085	system.component13
system at 2017.07.20 14.18	0.128	system.component14
system at 2017.07.20 15.06	0.124	system.component19
system at 2017.07.20 15.06	0.123	system.component20
system at 2017.07.20 15.06	0.122	system.component21
system at 2017.07.20 15.06	0.045	system.component17
system at 2017.07.20 15.06	0.061	system.component18

A simulation of the system will result in instance specifications of the System for the specified number of runs. The endTime value property is determined by the probability of each control flow.

5.9.1.1 Implementation Simulation Configurations

5.9.2 Monte Carlo Plugin

The Monte Carlo Plugin is for MagicDraw's Cameo Simulation Toolkit, and is compatible with MagicDraw 18.0 and 18.5. The two features of the plugin are the options to run a Monte Carlo simulation and to CallActions into CallOperationActions. See [md-cst-montecarlo] in the OpenMBEE repository to download the Monte Carlo plugin.

In the MagicDraw 19.0 release, the Monte Carlo Plugin will be integrated into the Cameo Simulation Toolkit. In the value property numberOfRuns, the user can specify the number of times the system is simulated. An instance specification of the executionTarget will be created for each run. The primary difference between the Monte Carlo plugin and the MD 19.0 Cameo Simulation Toolkit is in the way the user can specify the number of runs.

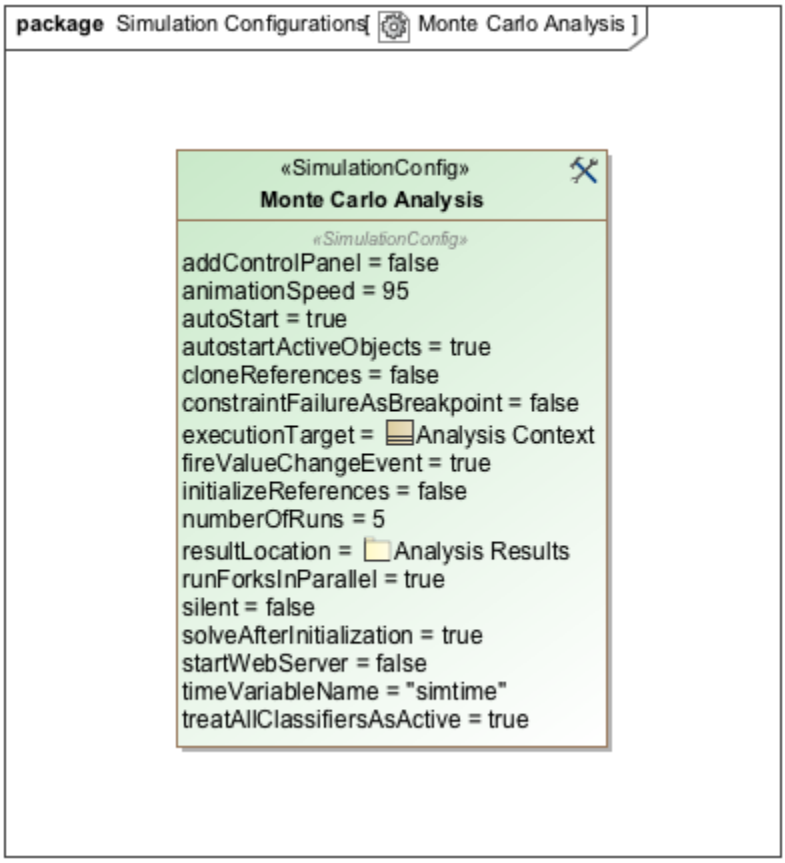

Figure 75. MD 19.0 Simulation Configuration

To run a Monte Carlo Simulation using the Monte Carlo plugin, the user must have a Simulation Configuration model element configured to their system. As a general practice, the executionTarget attribute should be equal to an analysis context.

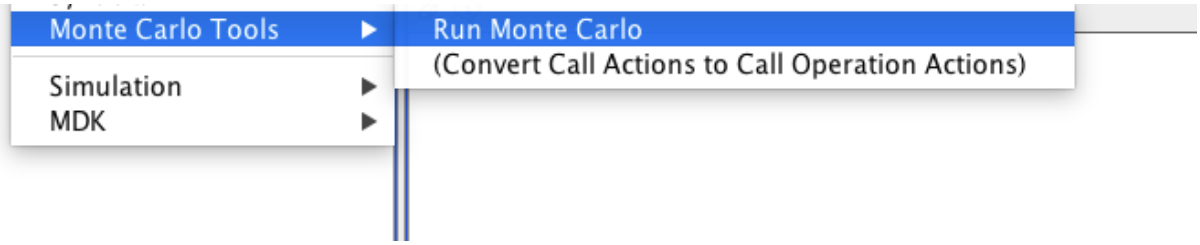

Figure 76. Run Monte Carlo

To run a Monte Carlo Simulation using the Monte Carlo plugin, the user must have a Simulation Configuration model element configured to their system. The user can select the simulation configuration in the Containment Tree, right click "Monte Carlo Tools", and select the option "Run Monte Carlo".

6 Scenario Modeling Pattern

6.1 Pattern Name and Classification

Pattern Name	Scenario Modeling Pattern
Classification	System Specification, Validation, and Testing

6.2 Intent

The intent of the Scenario Modeling Pattern is to simulate a system of interest navigating through a pre-defined process separate from the system. Property-based requirements are utilized in this pattern to validate the system of interest's performance as it goes through the scenario.

6.3 Motivation

The motivation for this pattern is to capture a simulation and its actors for multiple scenarios and evaluate their outputs. A variant of this pattern, the Augmented Scenario Pattern is able to support multiple scenarios that inherit key structural properties.

The Scenario Modeling Pattern and its variant apply to not only the components and activities of the system of interest, but also to activities of the scenario runner.

6.4 Concept

The SysML structure for this pattern begins with defining the context in the form of a block definition diagram. The context is directly composed of the system of interest and that system of interest's behavior. The Scenario Block represents a discreet script of signals that are injected into the system off interest. The system of interest has behavior that responds to the injected signals.

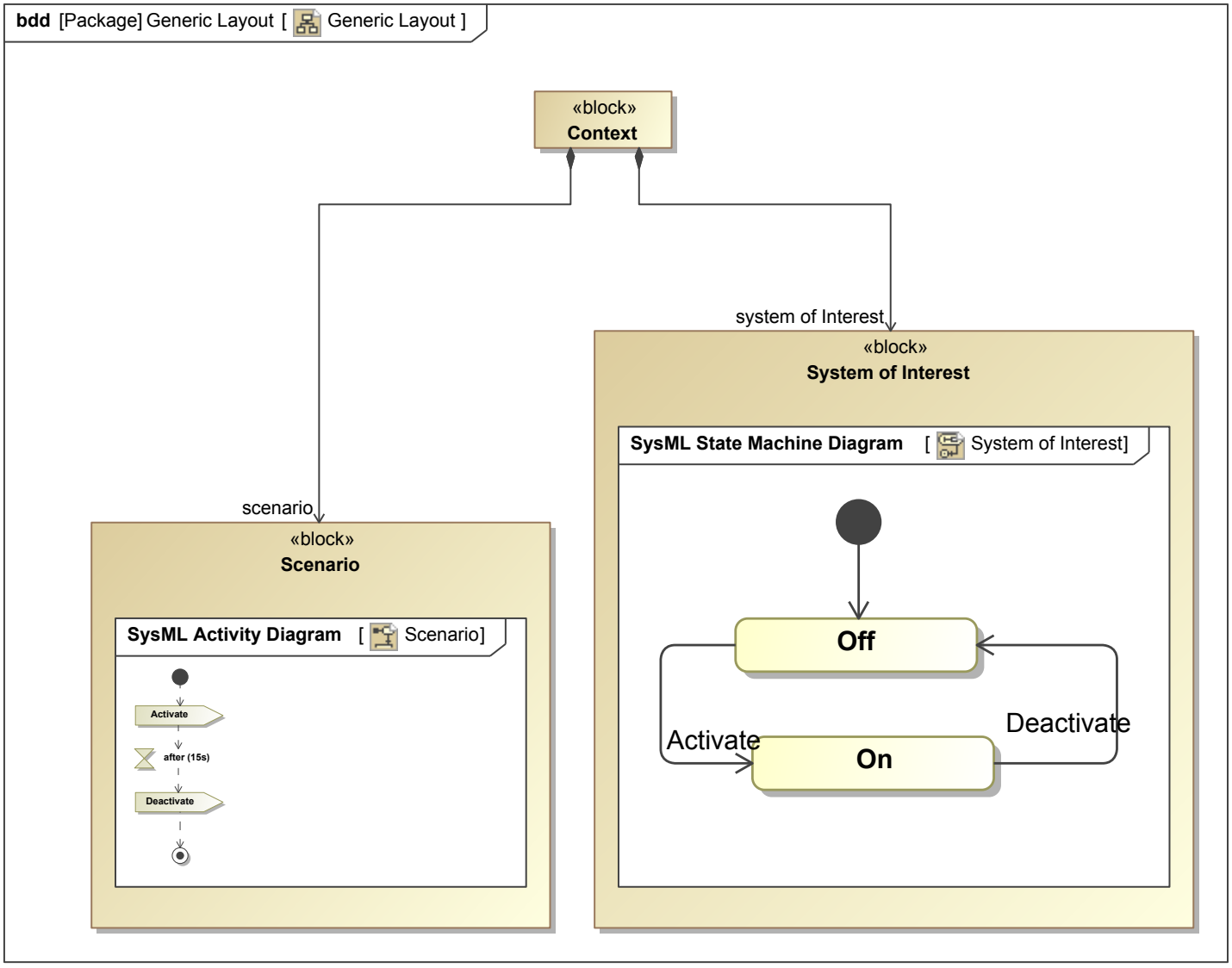

Figure 77. Generic Layout

The Block Definition Diagram shown above illustrates a generic layout for a scenario to be modeled, namely the direct composition relationships between the context of a scenario, a behavior, and a system of interest.

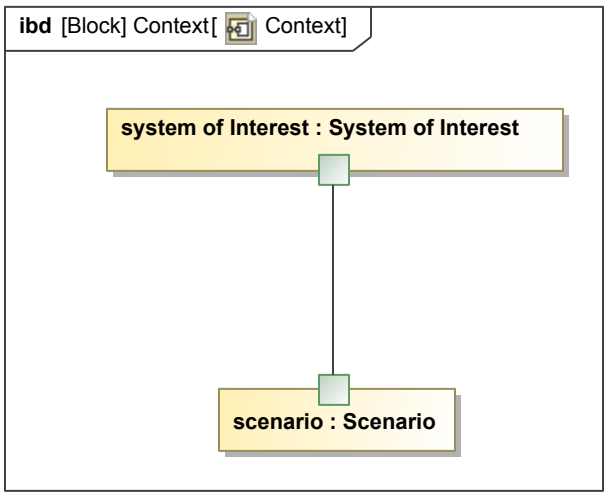

Figure 78. Context

The Internal Block Diagram of the Context shows the relationship between the System of Interest and the Scenario that it is associated with. The ports and connector show the logical path for these two entities use to communicate to each other.

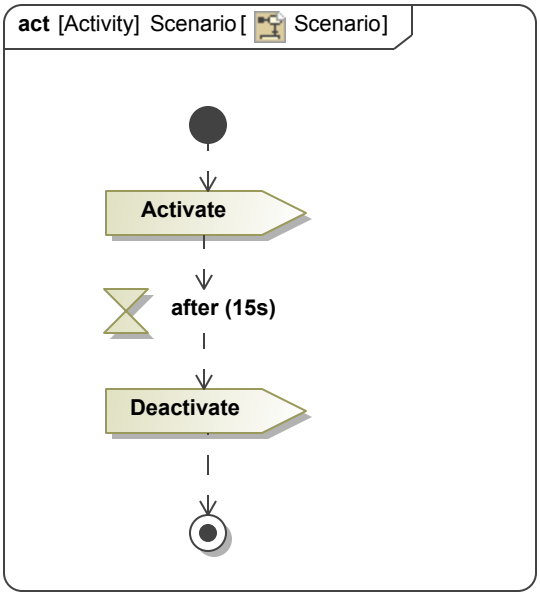

Figure 79. Scenario

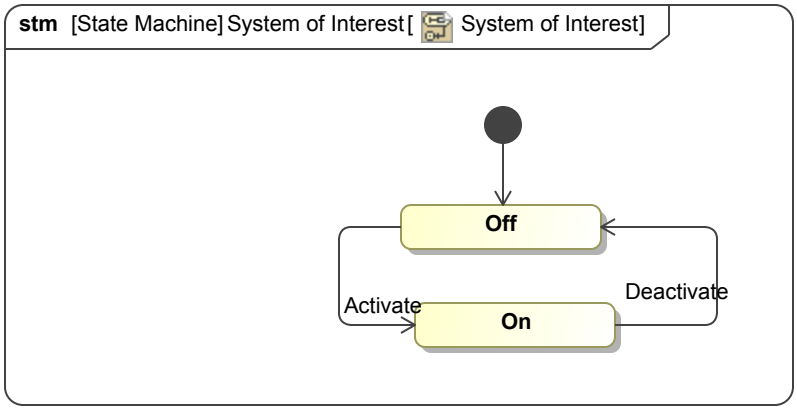

Figure 80. System of Interest

6.5 Consequences

Action	Consequence

6.6 Sample Model

These diagrams show a sample model and all of its components.

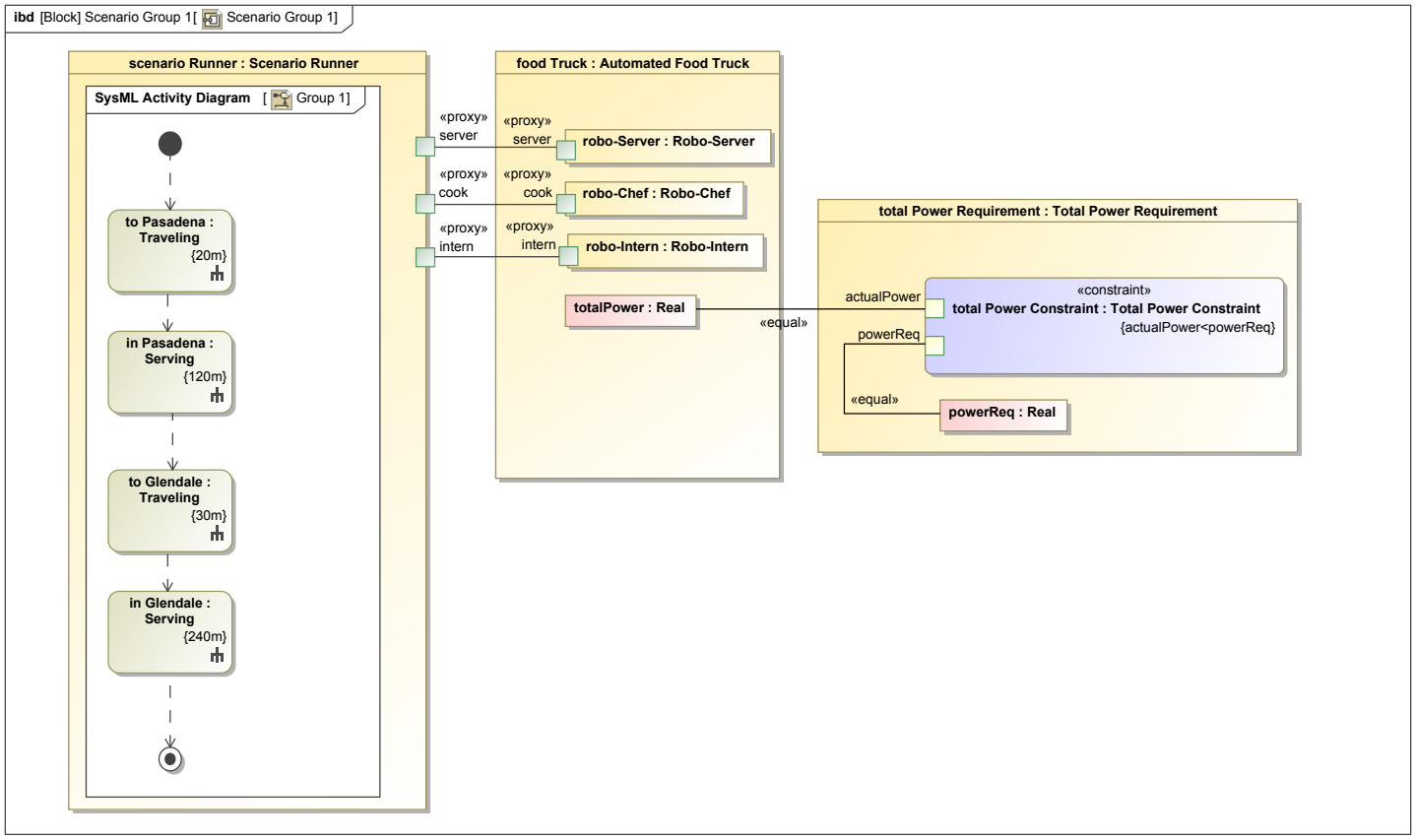

Figure 81. Scenario Group 1

In the diagram above, the scenario runner is responsible of initiating the illustrated behavior, which directs the operations performed by the system of interest. The value properties of the system of interest are connected to the appropriate parameters and restraints. The diagram below shows the power consumption value properties associated with each component in the system of interest.

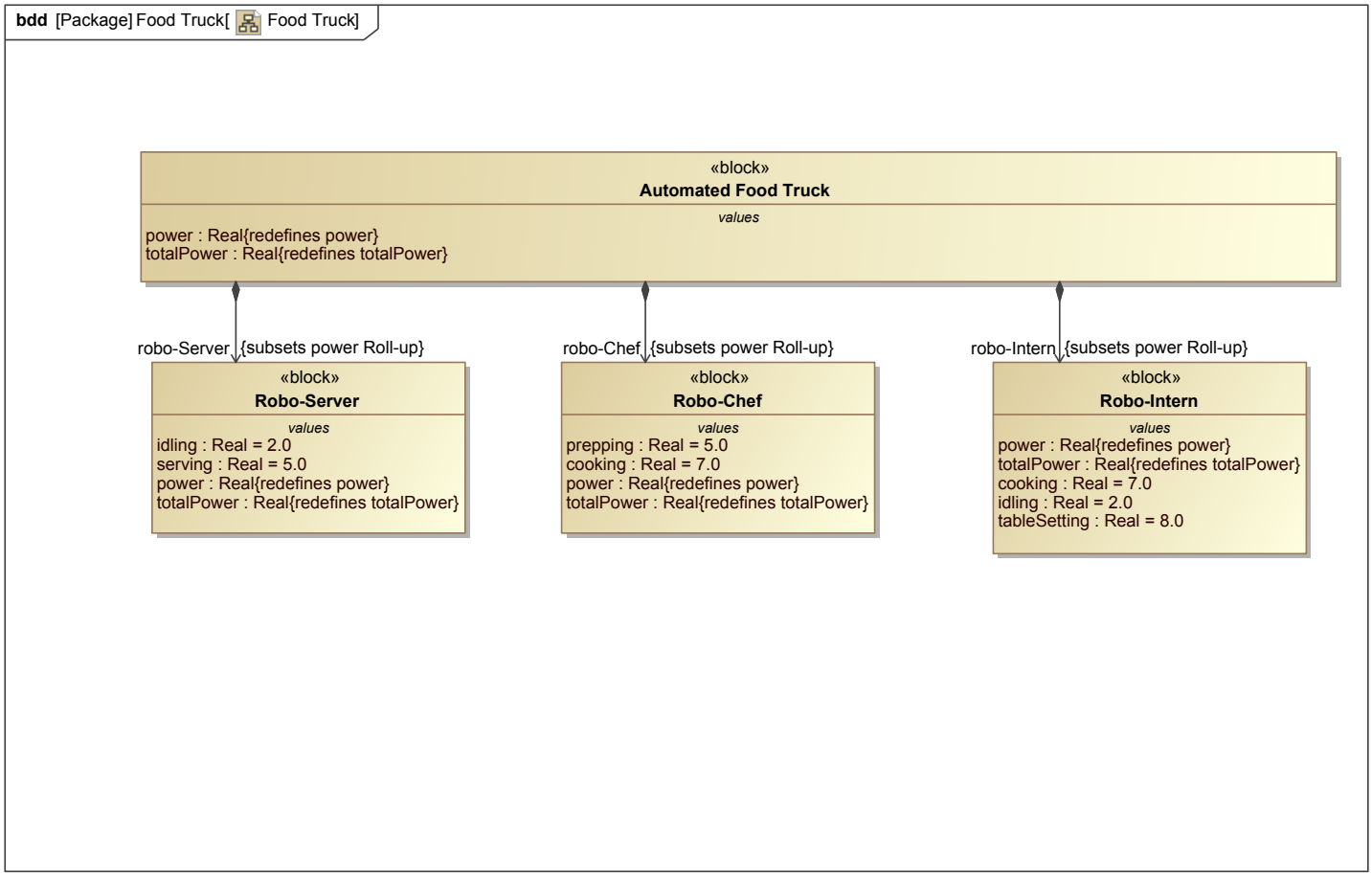

Figure 82. Food Truck

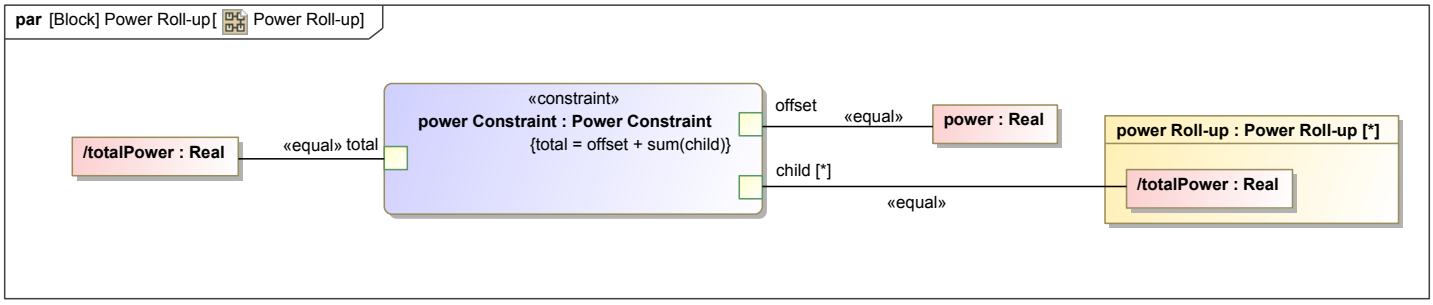

Figure 83. Power Roll-up

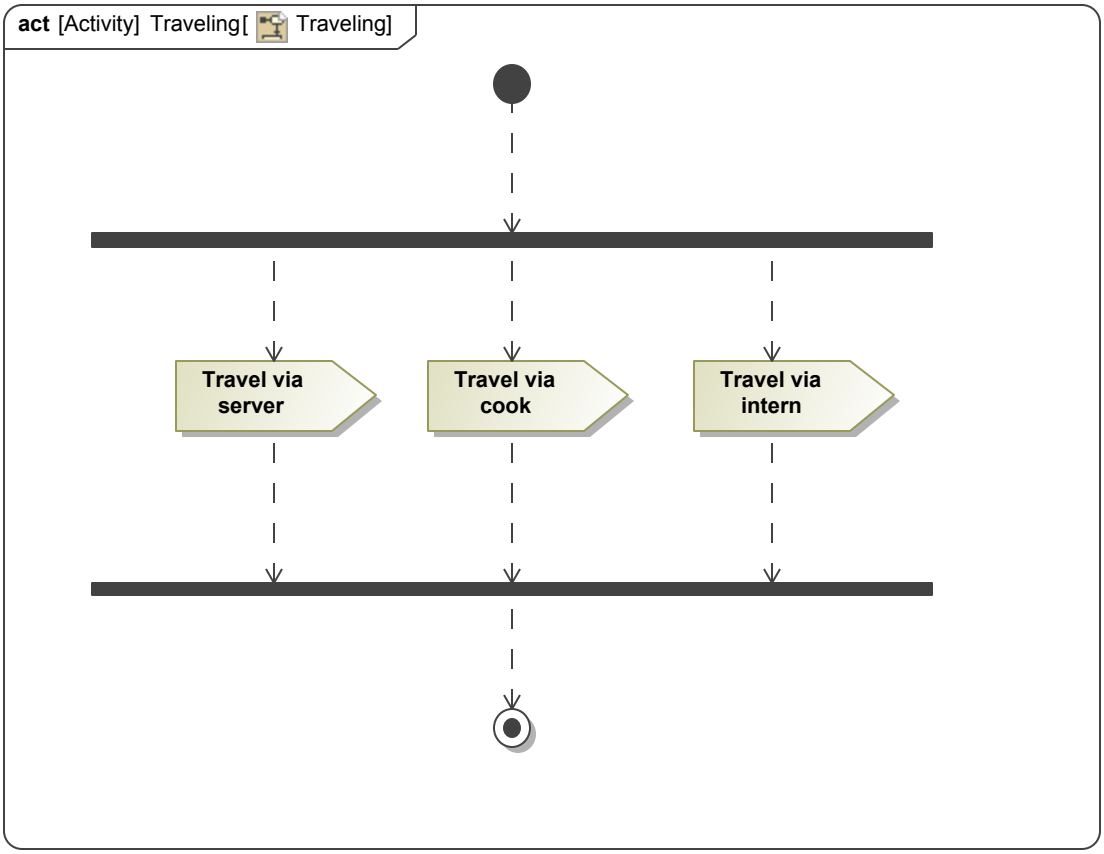

Figure 84. Traveling

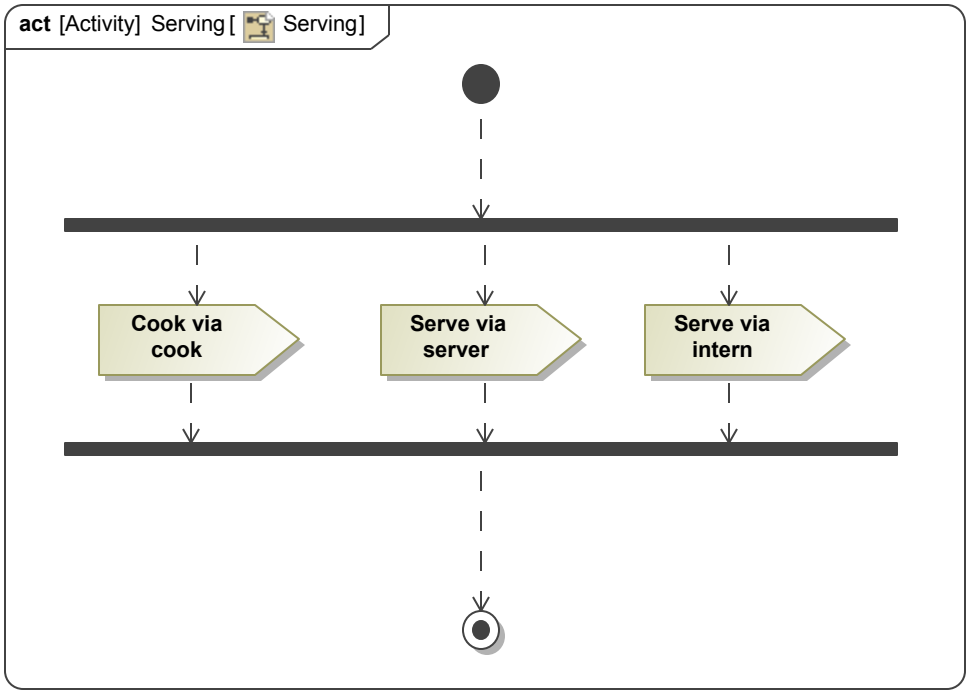

Figure 85. Serving

These state machines (Figs. 82-83) show the two possible modes of operation the system of interest can be in, whether it is in transit to or serving its destination.

6.7 Known Uses

The pattern is configured by setting up a specific or general context model with Block Definition Diagrams and Internal Block Diagrams. The next step in configuration is to establish parameters and create a roll-up pattern with the parameters. With this roll-up

in place, it is then possible to configure a simulation to monitor the behavior of the system of interest and its parameters for a defined period of time.

6.8 Related Patterns

A variant of this process is the Augmented Scenario Pattern (see 6.13).

Section 4

6.9 Tooling

6.9.1 MagicDraw

To execute simulation between on scenario and the corresponding system of interest, ensure that the Cameo Simulation Toolkit plugin for MagicDraw is installed and is the latest version.

6.10 Augmented Scenario Pattern

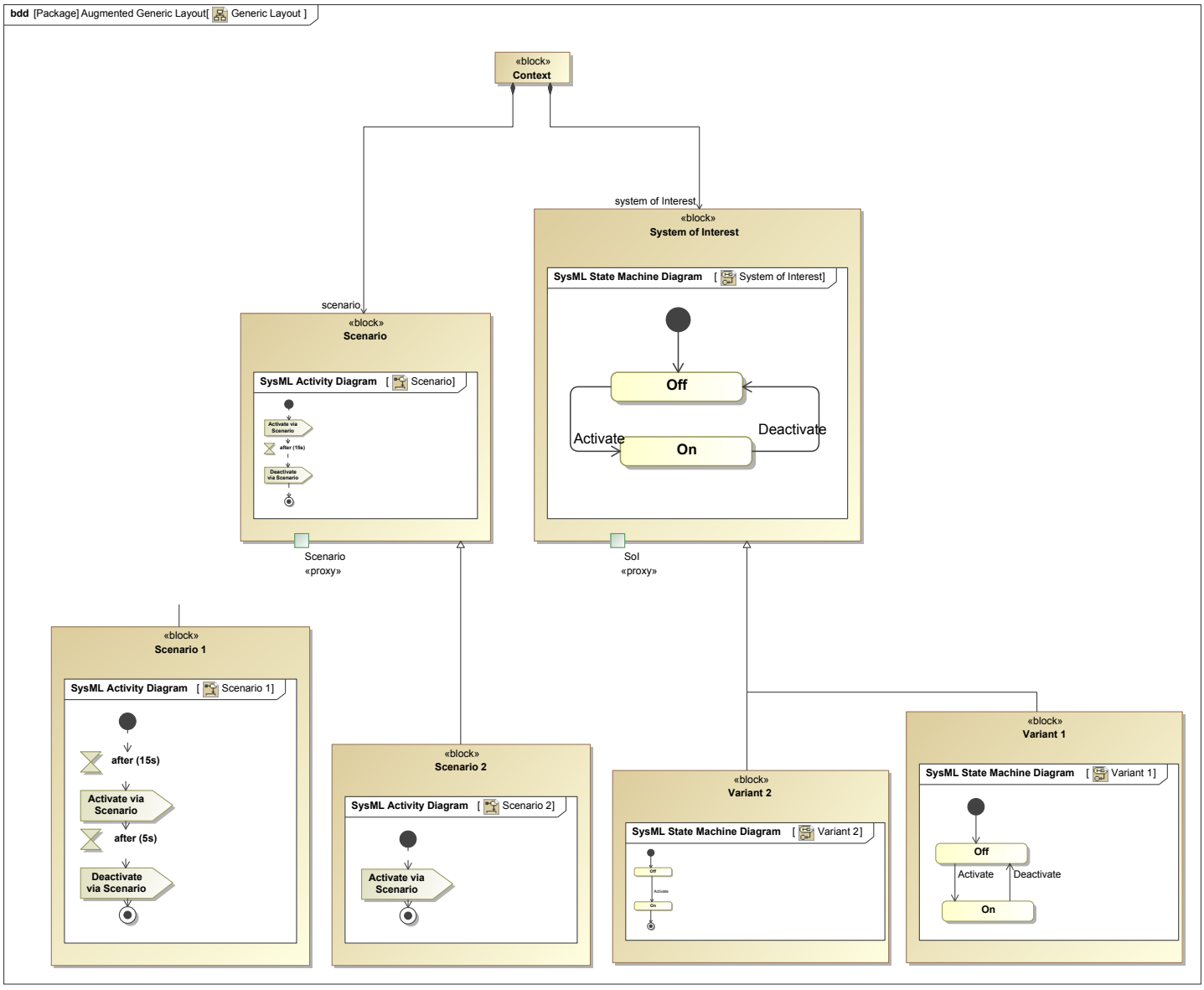

Figure 86. Generic Layout

The augmentation of the pattern allows for different variants to run with different activities. Variant 2 can be run with Scenario 1. To do this, remove default values in variable window of the simulation.

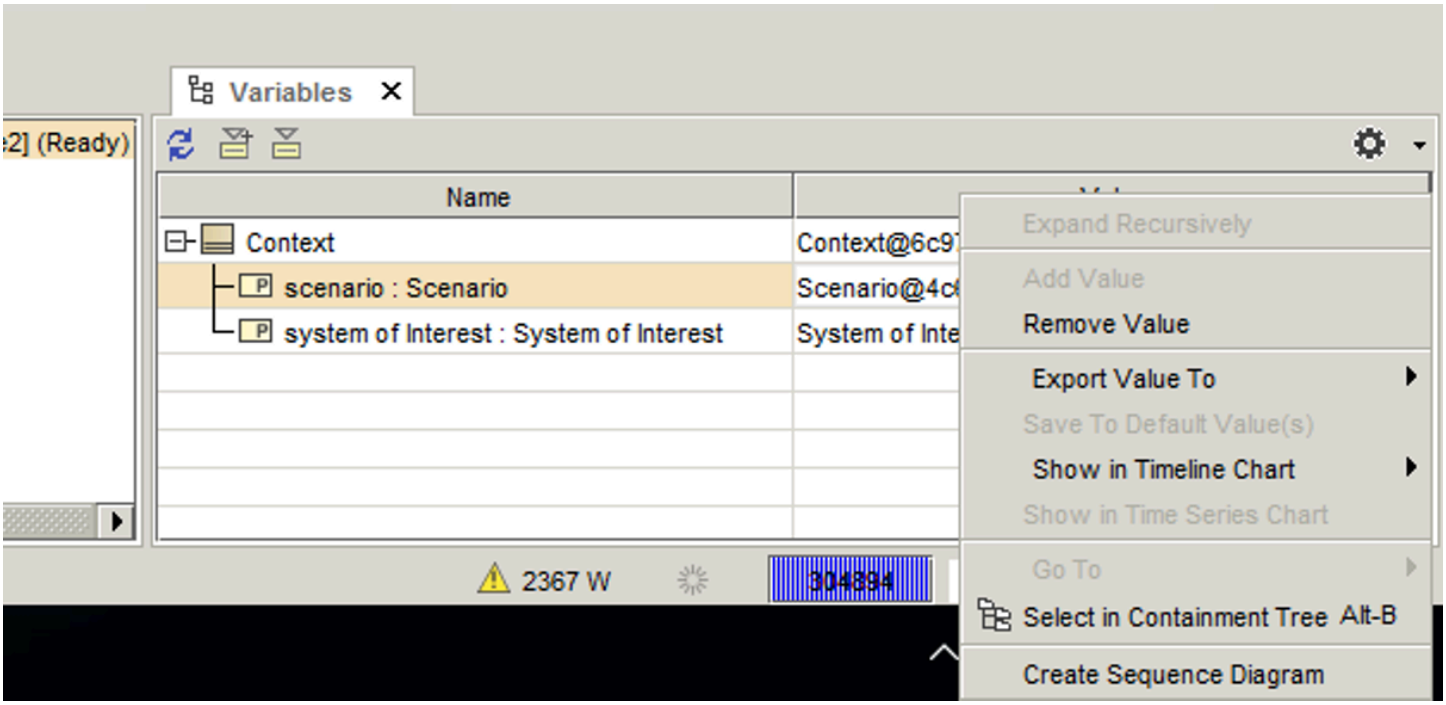

Figure 87. Variable Remove

Add values of desired variables.

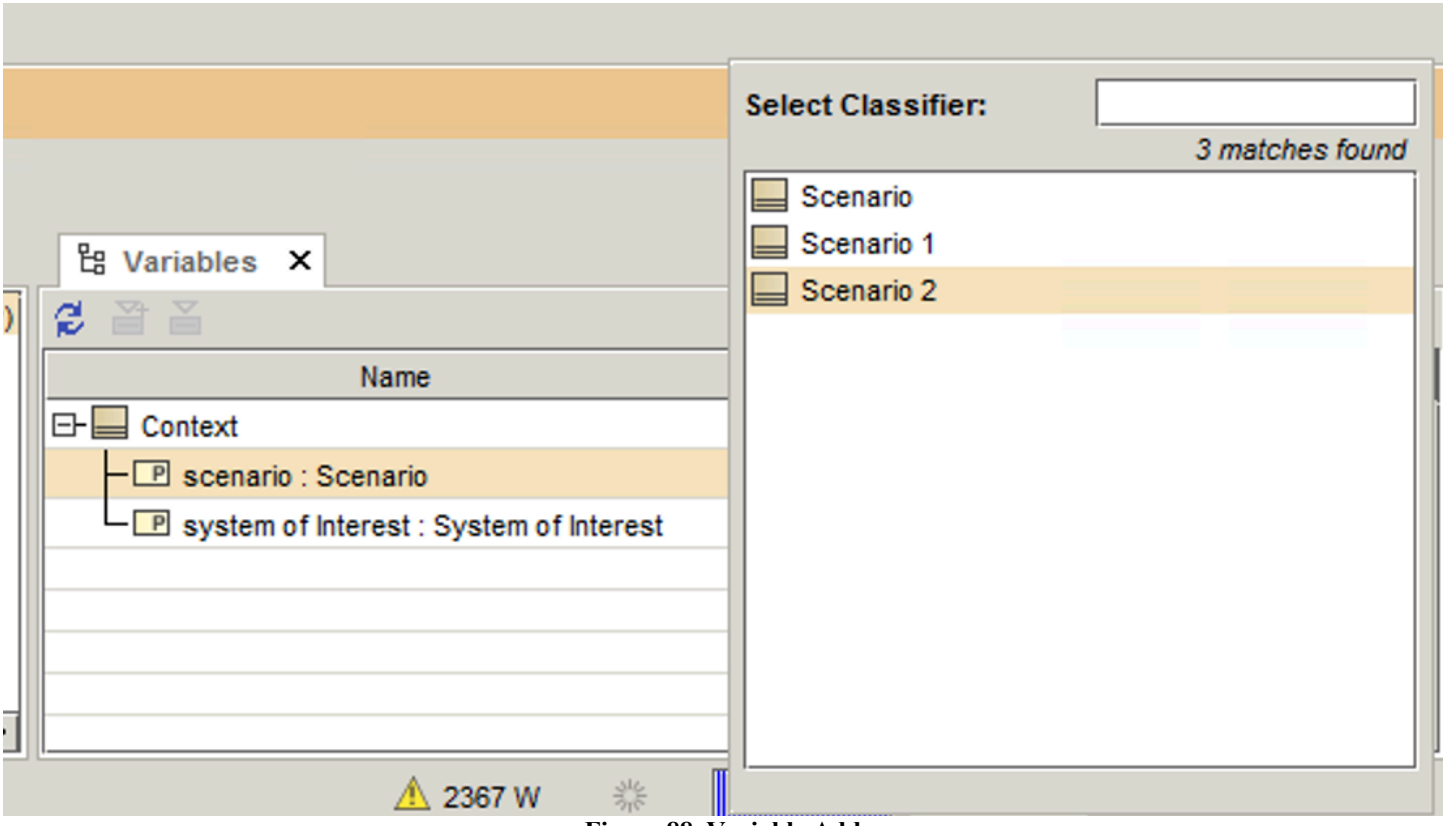

Figure 88. Variable Add

The simulation will then be able to run according to the newly defined Variant and Scenario. the ports on the System of Interest and Scenario are inherited by the Variants and Scenarios.

6.11 Augmented Scenario Pattern 2

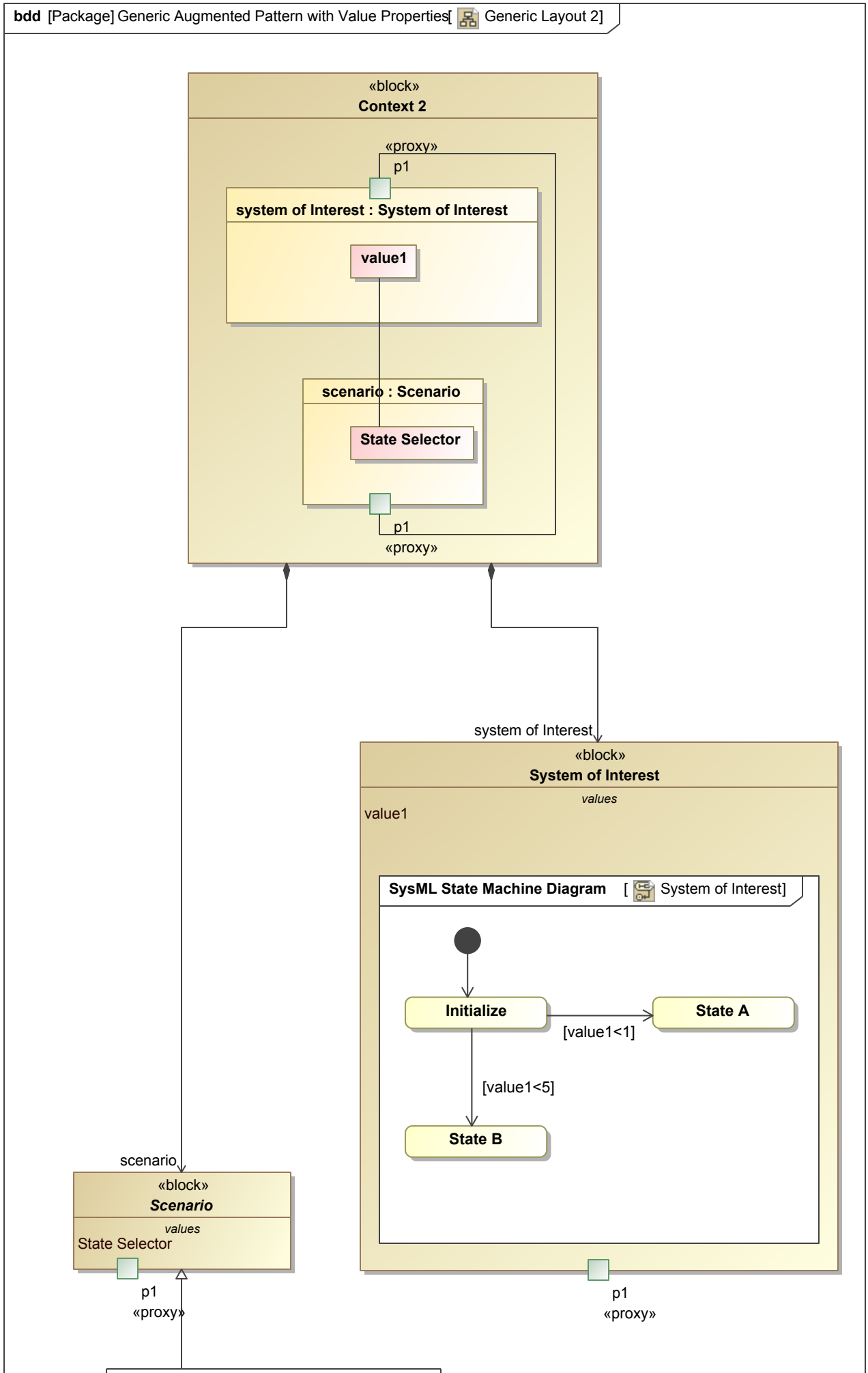
bdd [Package] Generic Augmented Pattern with Value Properties [Generic Layout 2]
«block»
Context 2
«proxy»
p1
system of Interest : System of Interest
value1
scenario : Scenario
State Selector
p1
«proxy»
system of Interest
«block»
System of Interest
values
value1
SysML State Machine Diagram [System of Interest]
Initialize
State A
[value1<1]
[value1<5]
State B
scenario
«block»
Scenario
values
State Selector
p1
«proxy»
p1
«proxy»

Figure 89. Generic Layout 2

In this Augmentation, the Scenario Block is italicized, signifying that it is abstract. This allows the scenario to run any valid variant in Cameo Simulation Toolkit. To make a block abstract, open the Specification window and make "Is Abstract" "True."

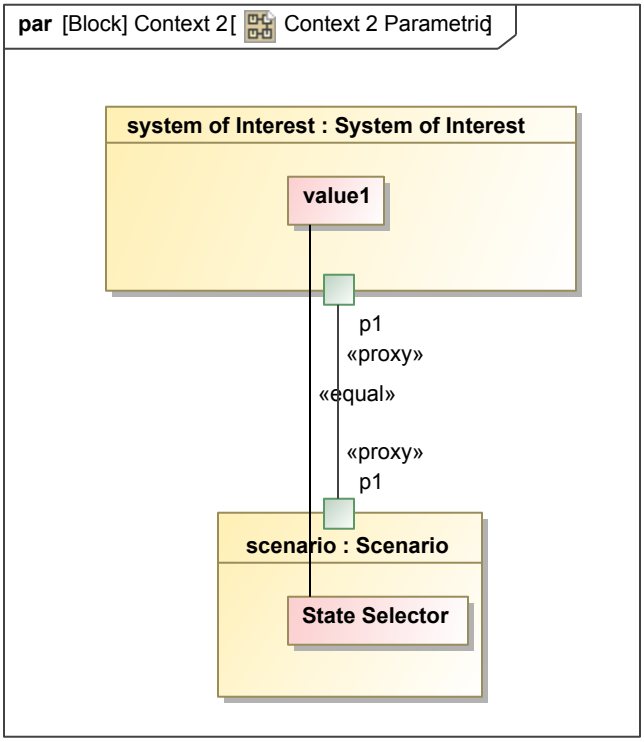

Figure 90. Context 2 Parametric

This parametric diagram informs guards in the state machine which activity to execute.