08 Systems Resource Management

Table of Contents

List of Tables

List of Figures

1 Introduction

2 Quantities Units Dimensions and Values

2.1 ISO80000

2.1.1 Example of Derived Definitions

Definitions illustrated in this figure		Count
1.5	derived quantity	3
1.11	derived unit	1
1.22	quantity equation	3
1.23	unit equation	2

In SysML 1.4, ISO 80000 has definitions for:

- 325 quantity kinds
- 191 non-prefixed units; 2564 prefixed units

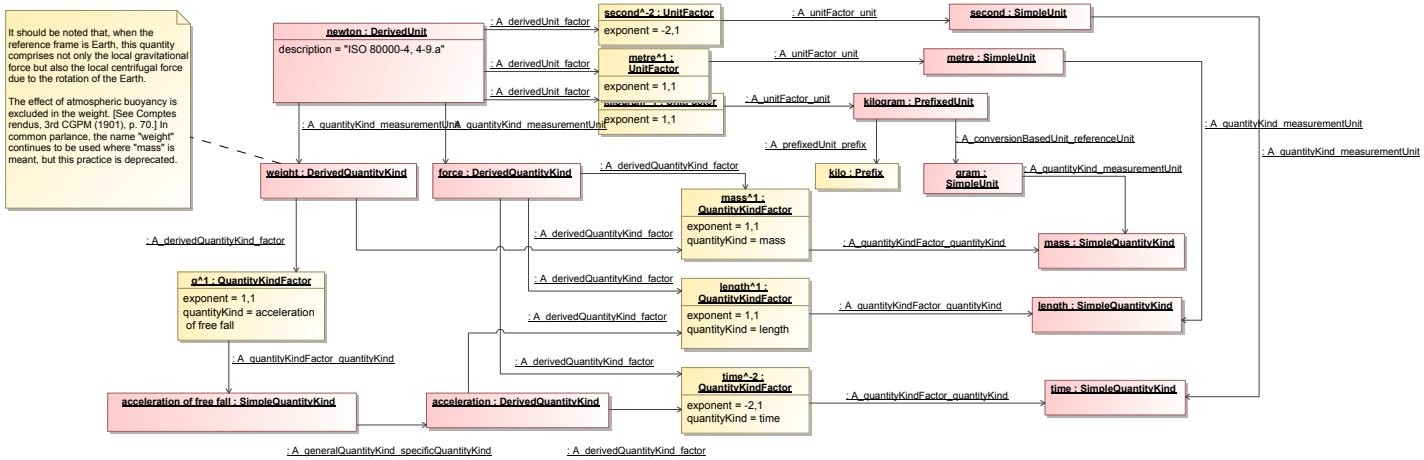


Figure 1. Example of Derived Definitions

2.1.2 What is ISO/IEC 80000

The standard consists of several parts. Each document part is identified by:

- The publication date (e.g., ISO 80000-1:2009)
- An edition number (e.g., 1st edition)
- Technical Corrigendum (as applicable)

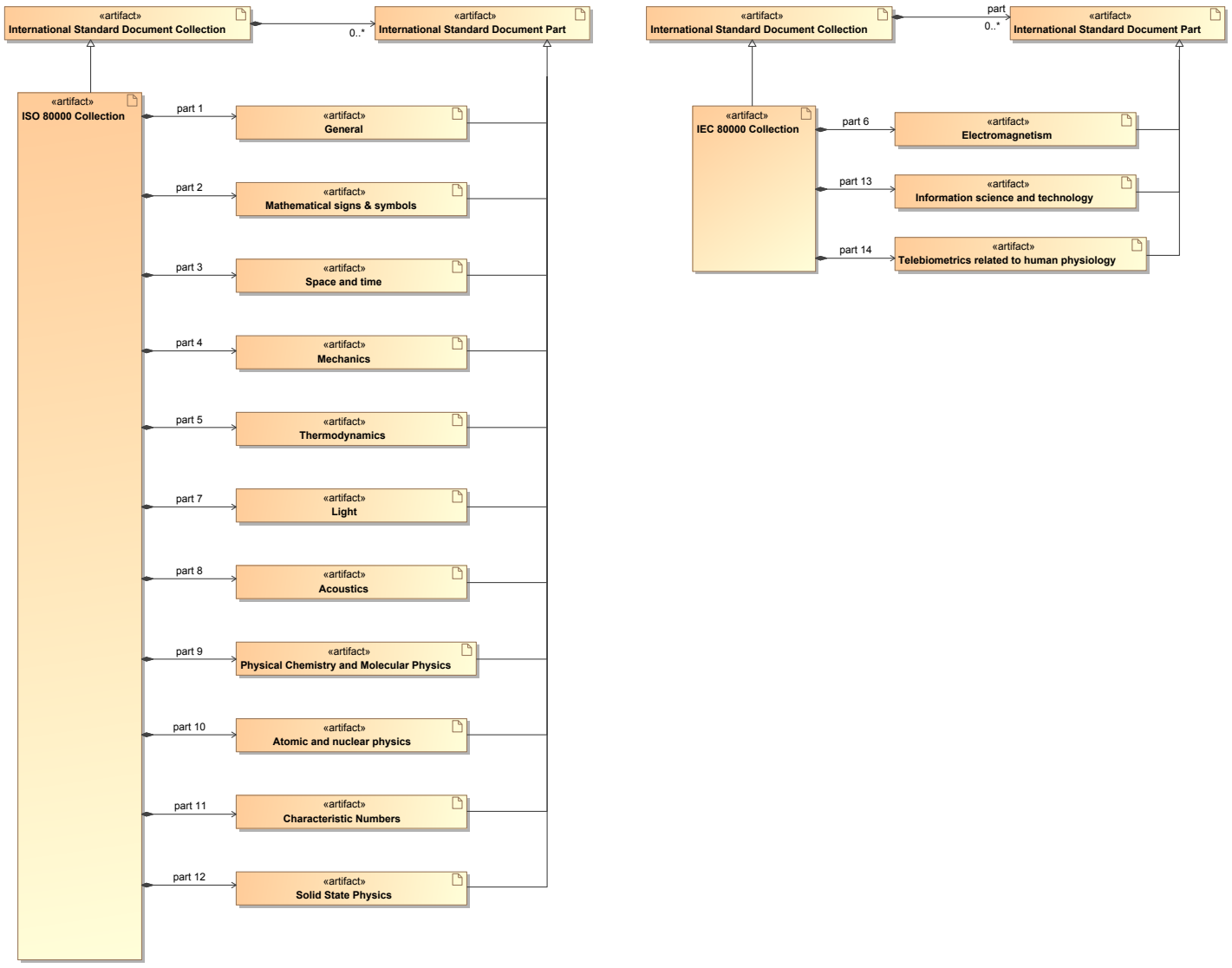


Figure 2. What is ISO/IEC 80000

2.1.3 Coverage of ISO/IEC 80000 in SysML 1.4

The ISO/IEC 80000 standard is covered in the **non-normative** Annex E.6 of the SysML specification.

No coverage in SysML 1.4 for:

- Part 8 Acoustics
- Part 11 Characteristic Numbers
- Part 12 Solid-state physics
- Part 14 Telebiometrics related to human physiology

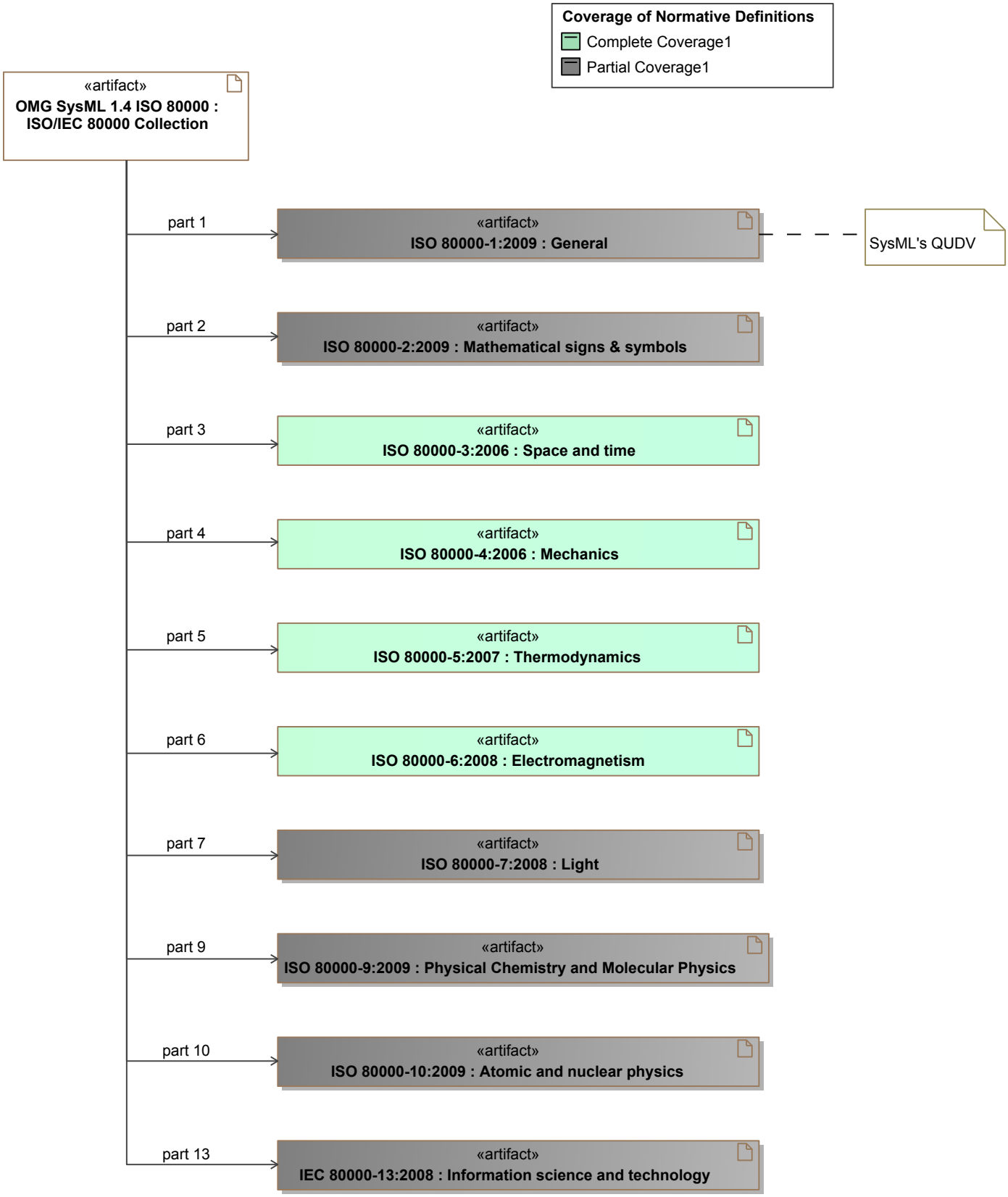


Figure 3. Coverage of ISO/IEC 80000 in SysML 1.4

2.1.4 Organization of the ISO 80000 library

Practical Perspective:

1. Using the library

2. Extending the library
3. Dimensional Analysis

Notes:

OMG SysML 1.4 ISO 80000 uses <<import>> with
a stronger modularity semantics than defined in UML

In OMG SysML 1.4 ISO 80000, [X] <<import>> [Y] means:

1. Some definition in [X] depends on some definition in [Y]
2. No definition in the <<import>> closure of [Y] depends on any definition in [X]
3. In most cases: ISO/IEC document part [X] has a normative reference to ISO/IEC document part [Y]

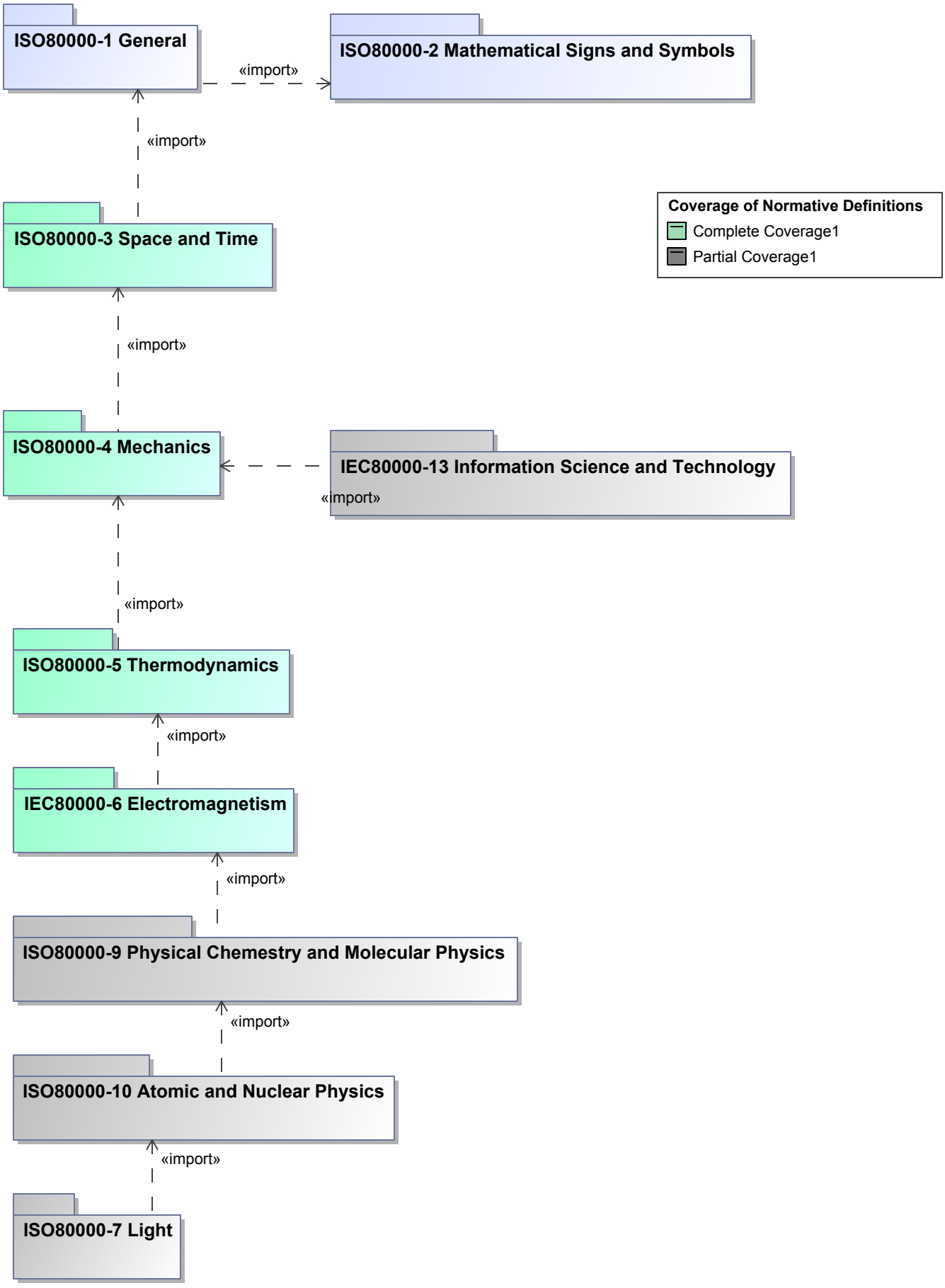
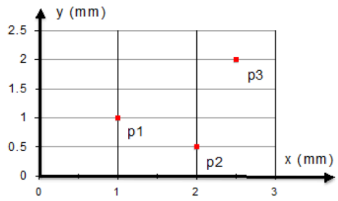


Figure 4. Organization of the ISO 80000 library

2.1.4.1 Using the ISO 80000 library

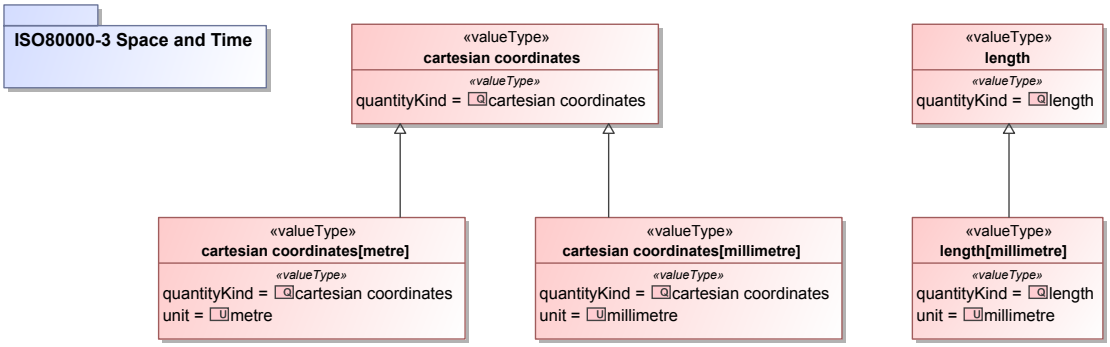
2.1.4.1.1 Example - Modeling the position of a point on a 2D Cartesian plane

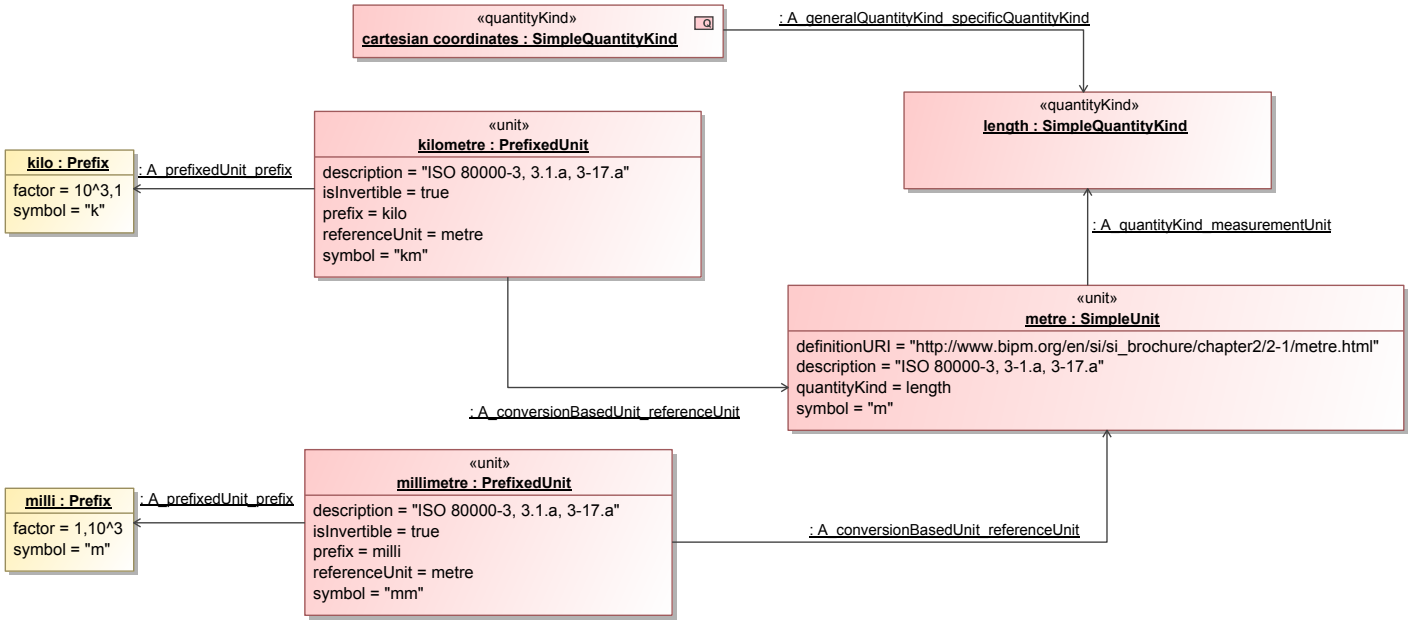


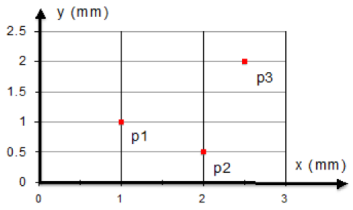
Goal: Represent in SysML the following points in the 2D plane:

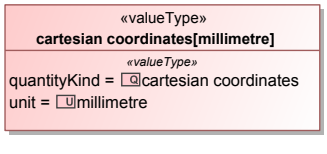
- Origin (x=0mm, y=0mm)
- p1 (x=1 mm, y=1 mm)
- p2 (x=2 mm, y=0.5 mm)
- p3 (x=2.5 mm, y=2 mm)

Modularity: Import ISO 80000-3 Space & Time (his part suffices for this application)









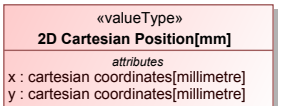


Figure 5. Example - Modeling the postition of a point on a 2D Cartesian plane

2.1.4.2 Modeling Values

2.1.4.2.1 Modeling Values of Scalar ValueTypes

IMPORTANT CLARIFICATION !
In SysML, values do not have a ‘unit’ per se (e.g., “42mm” is not a value in SysML!)
In SysML, values have an optional ‘unit’ through their type (i.e., a SysML ValueType)

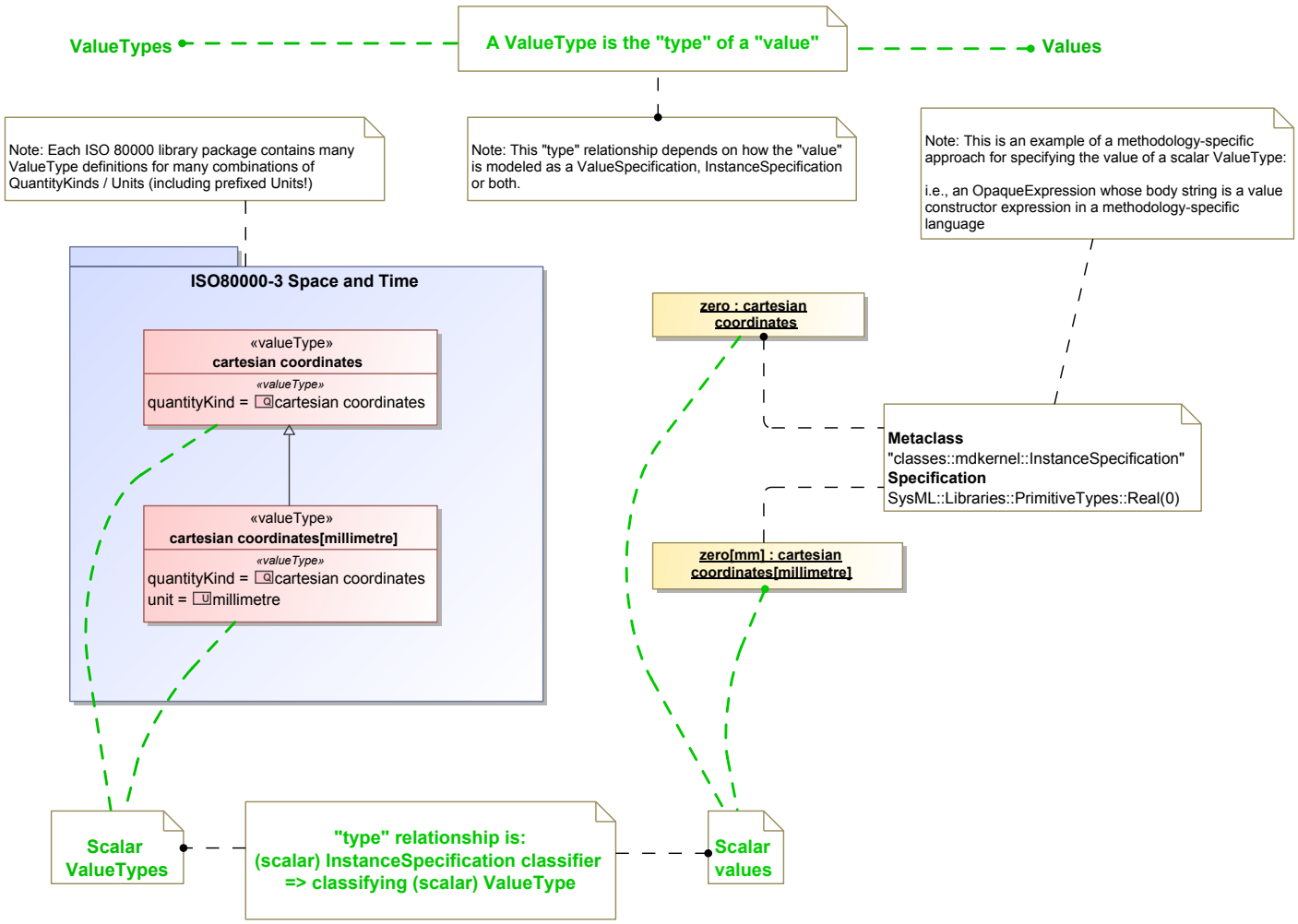


Figure 6. Modeling Values of Scalar ValueTypes

2.1.4.2.2 Modeling Values of Structured ValueTypes

IMPORTANT CLARIFICATION !
Although SysML values do not have a ‘unit’ per se, Value / ValueType Conformance is essential (otherwise a value has no meaning in SysML)

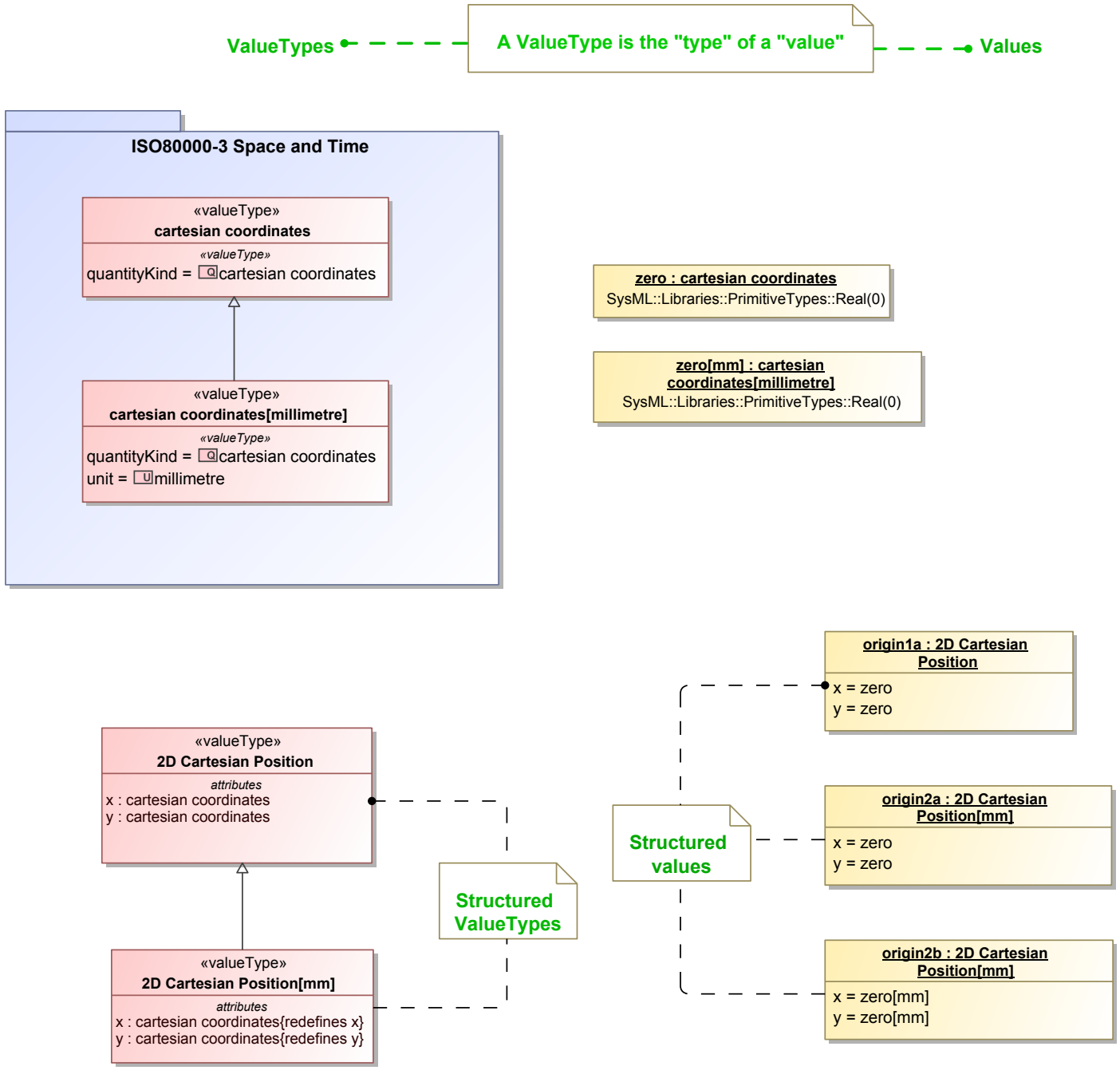


Figure 7. Modeling Values of Structured ValueTypes

2.1.4.2.3 Type Checking: Value / ValueType Conformance

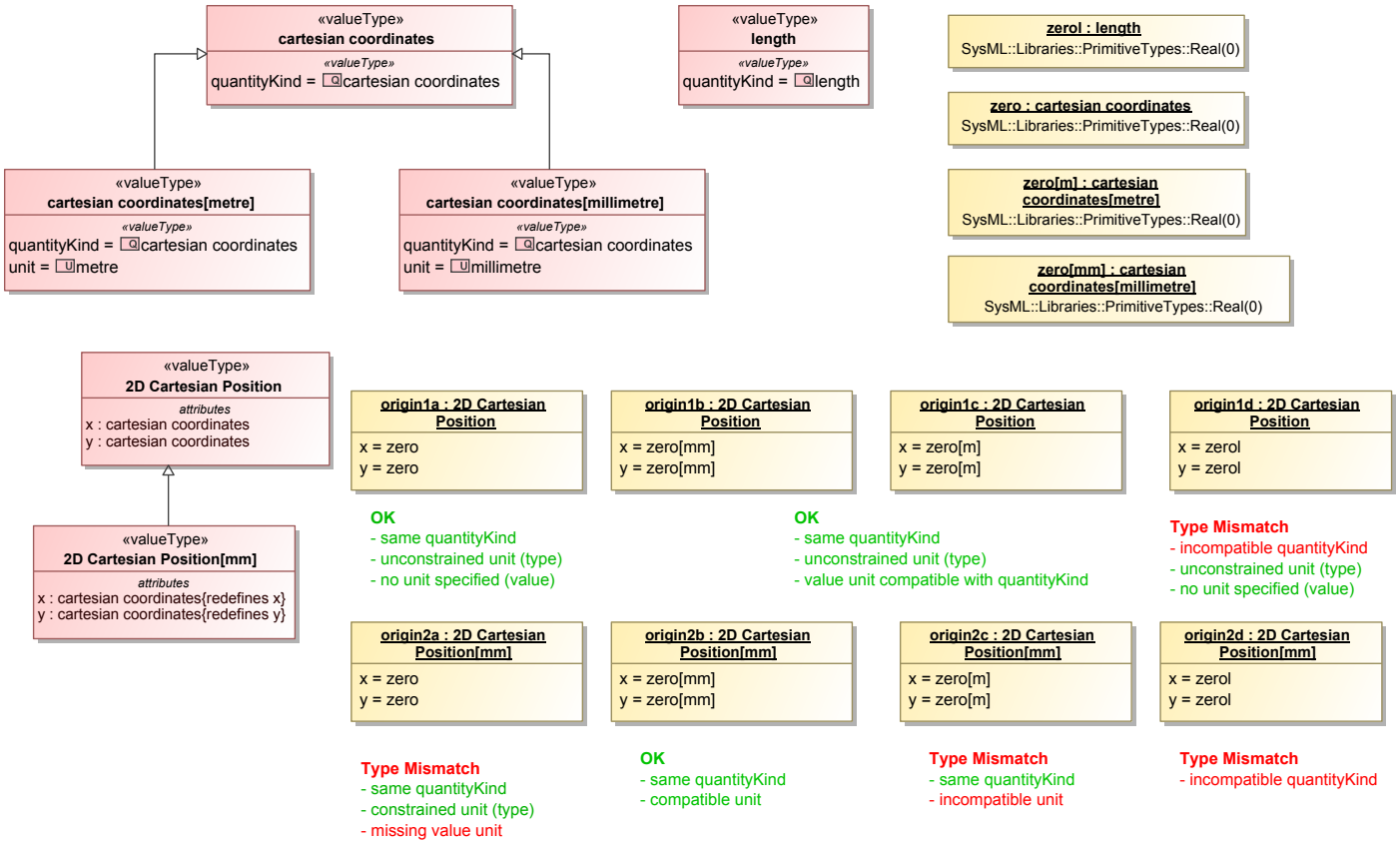


Figure 8. Type Checking: Value / ValueType Conformance

2.1.4.3 Extending the ISO-80000 Library

2.1.4.3.1 Example - Extending the ISO 80000 library for imperial units

Goal: Define 'yard (US survey)'
Identify the relevant ISO 80000 Library: Extend ISO 80000-3 (Space & Time)

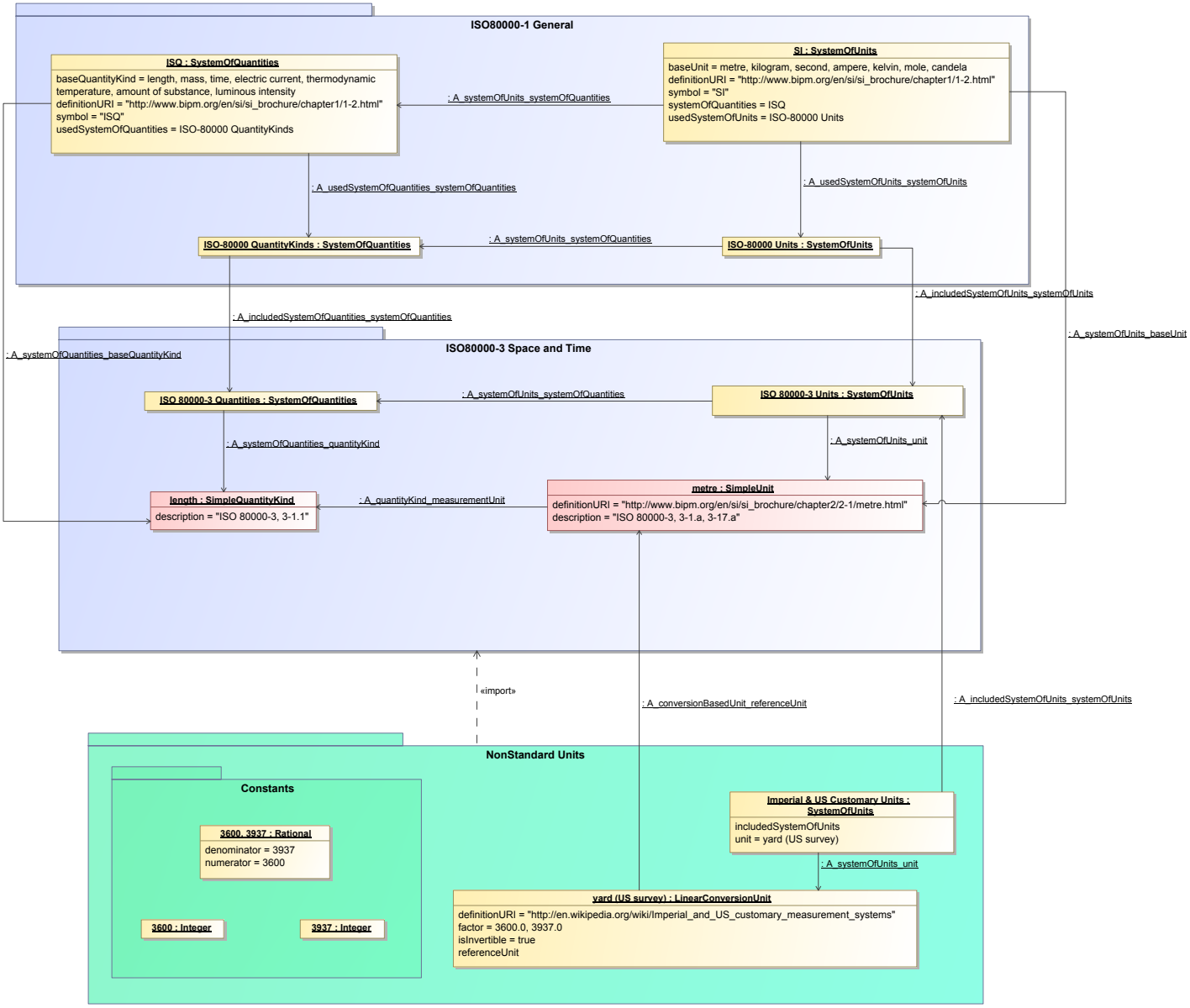


Figure 9. Example - Extending the ISO 80000 library for imperial units

2.1.5 Base quantity kinds (ISQ) and base units (SI)

1	Quantities and units (30 definitions)	Normative SysML		Non-normative QUDV & ISO-80000		
		language concept	modeling pattern	language concept	modeling pattern	ISO-80000-1 Library
1.2	kind of quantity			✓		
1.4	base quantity			✓		
1.9	measurement unit	✓				
1.10	base unit			✓		
1.17	multiple of a unit			✓		

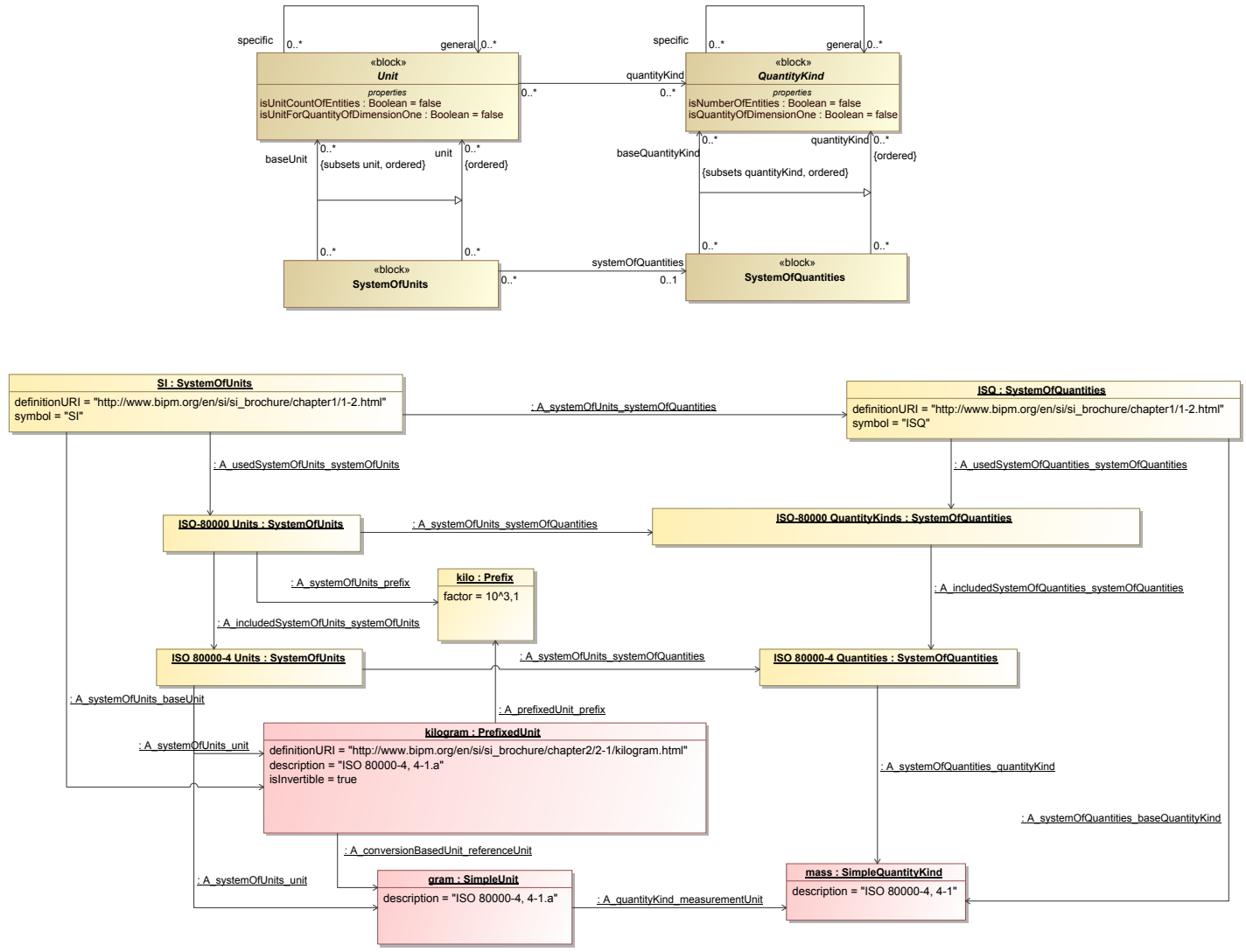


Figure 10. Base quantity kinds (ISQ) and base units (SI)

2.1.6 Introduction to QUDV

QUDV is covered in the non-normative Annex E.5 of the SysML specification.

Why QUDV?

- SysML has the concept of ‘quantity’ [VIM 3-1.1]: “property of a phenomenon, body or substance, where the property has a magnitude that can be expressed as a number and a reference”
- SysML 1.1 had limited vocabulary: Unit & Dimension
- SysML lacked the concept of ‘kind of quantity’ [VIM 3-1.2]
- QUDV first introduced in SysML 1.2 & subsequently revised

SysML 1.4 & QUDV:

- Improved alignment with VIM 3rd edition, JCGM 200:2012
- Adequate for modeling a significant scope of ISO/IEC 80000
- Semantics allow for computer-based Dimensional Analysis

1	Quantities and units (30 definitions)	Normative SysML: language concept	Normative SysML: modeling pattern	Non-normative QUDV & ISO-80000: language concept	Non-normative QUDV & ISO-80000: modeling pattern	Non-normative QUDV & ISO-80000: ISO-80000-1 Library
1.1	quantity	✓				
1.2	kind of quantity			✓		
1.3	system of quantities			✓		
1.4	base quantity			✓		
1.5	derived quantity			✓		
1.6	International System of Quantities (ISQ)					✓
1.7	quantity dimension			✓		
1.8	quantity of dimension one			✓		
1.9	measurement unit	✓				
1.10	base unit			✓		
1.11	derived unit			✓		
1.12	coherent derived unit				✓	
1.13	system of units			✓		
1.14	coherent system of units				✓	
1.15	off-system measurement unit				✓	
1.16	International System of Units (SI)					✓
1.17	multiple of a unit			✓		
1.18	submultiple of a unit			✓		
1.19	quantity value		✓			
1.20	numerical quantity value		✓			
1.21	quantity calculus				✓	
1.22	quantity equation			✓		
1.23	unit equation			✓		
1.24	conversion factor between units			✓		
1.25	numerical value equation		✓			
1.26	ordinal quantity		✓			
1.27	quantity-value scale					
1.28	ordinal quantity-value scale		✓		✓	
1.29	conventional reference scale		✓		✓	
1.30	nominal property				✓	

Def.	Quantities and units (30 definitions)	Normative SysML: language concept	Normative SysML: modeling pattern	Non-normative QUDV & ISO-80000: language concept	Non-normative QUDV & ISO-80000: modeling pattern	Non-normative QUDV & ISO-80000: ISO-80000-1 Library
1.3	system of quantities			✓		
1.6	International System of Quantities (ISQ)					✓
1.13	system of units			✓		
1.16	International System of Units (SI)					✓

	No Coverage in QUDV or SysML	
2	Measurement	(53 definitions)
3	Devices for measurement	(12 definitions)
4	Properties of measuring devices	(31 definitions)
5	Measurement standards (etalons)	(18 definitions)

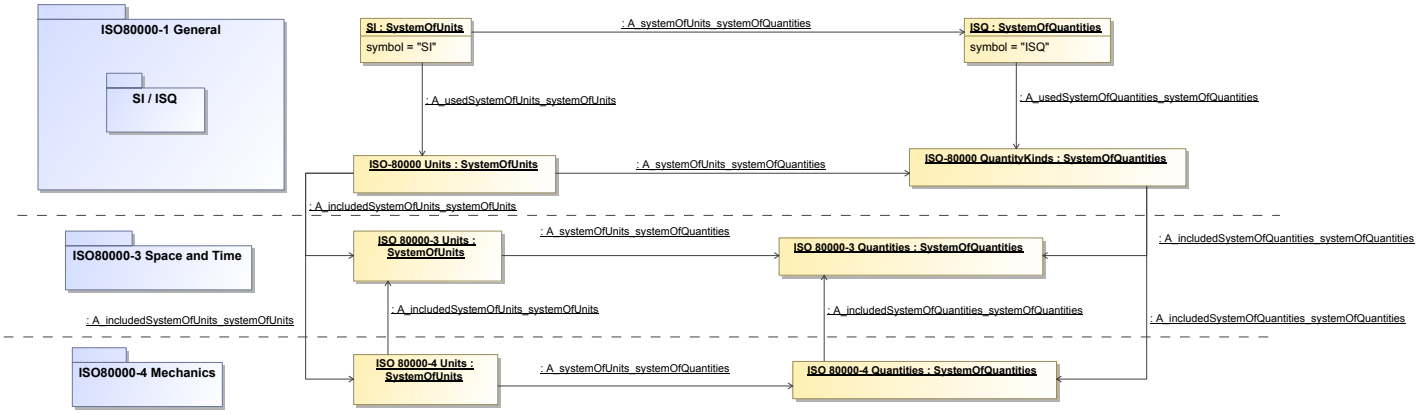


Figure 11. Introduction to QUDV

2.2 References about Metrology

1. Vocabulary of International Metrology (VIM)
   SysML 1.4 QUDV is aligned with VIM 3rd edition
   http://www.bipm.org/utils/common/documents/jcgm/JCGM_200_2012.pdf
2. The Rationale for VIM 3 (scientific papers)
   http://www.bipm.org/en/publications/guides/rationale_vim3.html
3. The Rationale for SysML 1.4 QUDV (subset of above)
   ISO terminological analysis of the VIM3 concepts ‘quantity’ and ‘kind-of-quantity’, Dybkaer R., Metrologia, 47(3), 2010, 127-143 http://dx.doi.org/10.1088/0026-1394/47/3/003

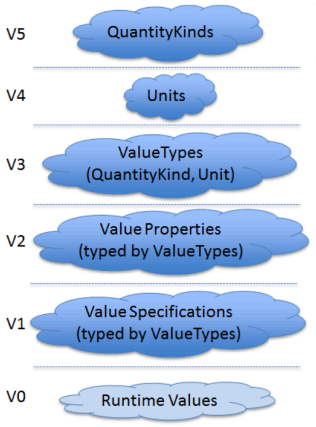
This paper is a small excerpt from Dybkaer's thesis:
An ontology on Property for Physical, Chemical and Biological Systems, Dybkaer R., Copenhagen University Hospital, Denmark, 2009
http://ontology.iupac.org/ontology.pdf

2.3 Tool Support

2.3.1 Magic Draw

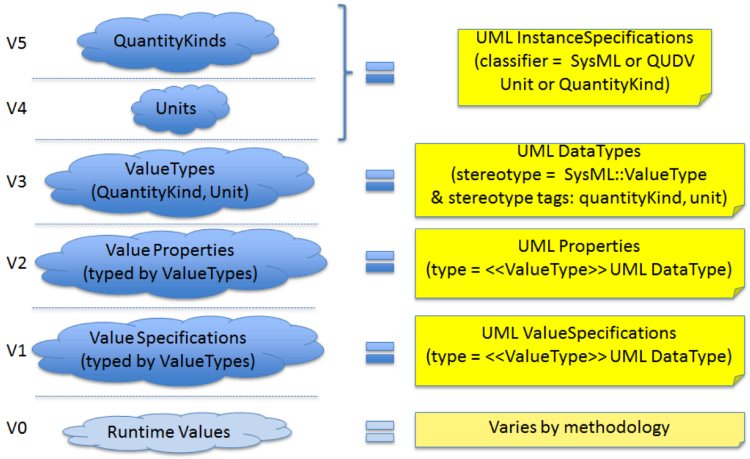
2.4 Practical considerations for modeling Values in SysML

#1: Recognize the ***distinct levels of modeling*** involved

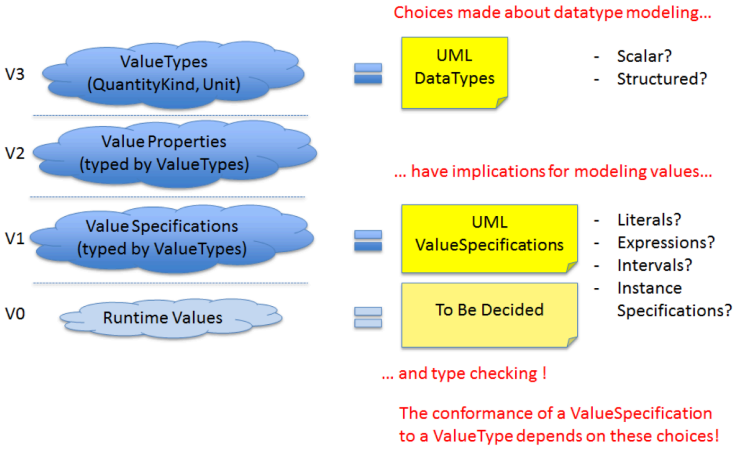


- A Level depends on those above
  i < j => Vi depends on Vj
- V4 < V5
  See VIM3 & ISO 80000-1 foreword:
  > This first edition of ISO 80000-1 cancels and replaces ISO 31-0:1992 and ISO 1000:1992. The major technical changes from the previous standard are the following:
  > ⍰ the structure has been changed to emphasize that quantities come first and units then follow;
- V3 < V4, V5
  Per SysML ValueType
- V2 < V3
  Per UML TypedElement/Type relationship
- V1 < V2
  Per UML ValueSpecification/Slot/ StructuralFeature & defaultValue relationships

#2: Recognize the ***alignment with UML***



#3: Recognize the ***implications of DataType modeling in UML***



Don't use a unit as the name of a ValueType!

Use a quantity kind and optionally a unit to name a ValueType

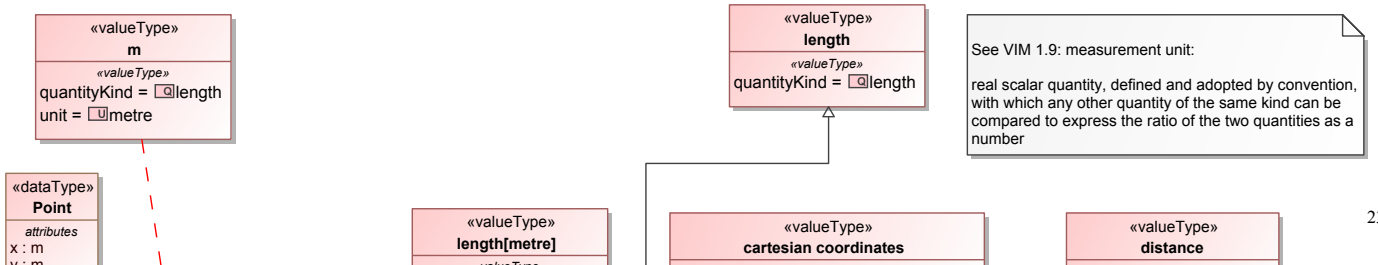
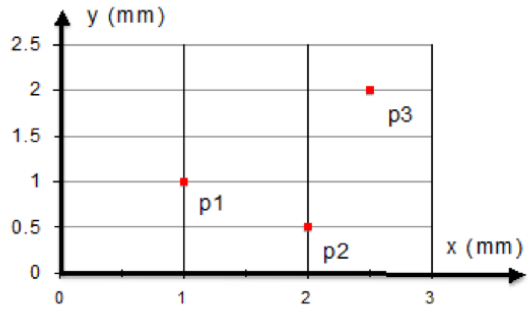


Figure 12. Practical Considerations for Modeling Values in SysML

2.5 Example of Structured vs. Scalar DataType Modeling

2D Cartesian Geometry Example (Alternative: Monadic ValueType): http://en.wikipedia.org/wiki/Monad_(functional_programming)

The semantics of UML DataTypes are well-suited for “functional modeling”. Comparing types & values/instances is important for assessing the pros/cons of a particular modeling methodology and for adopting one.



«valueType»
cartesian coordinates[millimetre]
«valueType»
quantityKind = cartesian coordinates
unit = millimetre

«Monad»
2D Cartesian Position[mm]
/x : cartesian coordinates[millimetre]{readOnly}
/y : cartesian coordinates[picometre]{readOnly}
x() : cartesian coordinates[millimetre]{query}
y() : cartesian coordinates[millimetre]{query}

Parameters:
* in serialization: String
* return result: 2D Cartesian Position[mm]

«Monad Constructor»
2D Cartesian Position[mm]1

«Metaclass»
DataType

«stereotype»
ValueType

«stereotype»
Monad

«Metaclass»
FunctionBehavior

«stereotype»
Monad Constructor

Each InstanceSpecification has an OpaqueExpression:
- language = "2D Cartesian Position[mm]"
- body = (see left)
- Behavior = 2D Cartesian Position[mm]

<<Monad Constructor>>
(a possible methodology-specific extension of UML)

p1 : 2D Cartesian Position[mm]
x = 1.0
y = 1.0

p2 : 2D Cartesian Position[mm]
x = 2.0
y = 0.5

p3 : 2D Cartesian Position[mm]
x = 2.5
y = 2.0

«valueType»
2D Cartesian Position[mm]
attributes
x : cartesian coordinates[millimetre]
y : cartesian coordinates[millimetre]

«valueType»
cartesian coordinates[millimetre]
«valueType»
quantityKind = cartesian coordinates
unit = millimetre

+ Simple

«Monad»
2D Cartesian Position[mm]
/x : cartesian coordinates[millimetre]{readOnly}
/y : cartesian coordinates[picometre]{readOnly}
x() : cartesian coordinates[millimetre]{query}
y() : cartesian coordinates[millimetre]{query}

«Monad Constructor»
2D Cartesian Position[mm]1

- Monadic ValueType modeling is uncommon
- Need to develop tool support extension

Comparing ValueTypes

0.5 : cartesian coordinates[millimetre]
Owned Element = SysML::Libraries::PrimitiveTypes::Real(0.5)

1 : cartesian coordinates[millimetre]
Owned Element = SysML::Libraries::PrimitiveTypes::Real(1)

2 : cartesian coordinates[millimetre]
Owned Element = SysML::Libraries::PrimitiveTypes::Real(2)

2.5 : cartesian coordinates[millimetre]
Owned Element = SysML::Libraries::PrimitiveTypes::Real(2.5)

p1 : 2D Cartesian Position[mm]
x = 1.0
y = 1.0

p2 : 2D Cartesian Position[mm]
x = 2.0
y = 0.5

p3 : 2D Cartesian Position[mm]
x = 2.5
y = 2.0

- Fastidious

Comparing

p1 : 2D Cartesian Position[mm]
x = 1.0
y = 1.0

p2 : 2D Cartesian Position[mm]
x = 2.0
y = 0.5

p3 : 2D Cartesian Position[mm]
x = 2.5
y = 2.0

+ Simple

Figure 13. Example of Structured vs. Scalar DataType Modeling

3 Time Modeling Specification

Creating time models with real time scenarios can be challenging in SysML. A specification of time modeling offers how to model time correctly and analyze the model using simulation. These patterns will help give a better understanding how to model time in your system and simulate models using real time.

3.1 Intent

The intent of providing a specification is to document and analyze how to model time constraints in system models. The time modeling specification is used to give another level of understanding to the functionality and behavior of a modeled system. Time based analysis allows for the verification of time based requirements and for the propagation of use case scenarios. The expected result of the time modeling specification is to provide a better understanding of how to model time in behavioral models and execute simulations based on real time. In order to have executable simulations, the semantics of the time usage in behavioral models needs to be well defined.

3.2 Motivation

Time constraints specify the overall system behavior in a system decomposition where there are concurrently active elements. A motivation is to capture time constraints of component and system level behavior of different scenarios, like operational scenarios. The pattern provides a collection of examples and rules how to specify time and duration constraints, using SysML activities, state machines, and interactions. The timing specification enables systems engineers to evaluate their system design and verify against duration requirements. A motivation is to take a time budget and allocate it to desired behaviors.

Applicability of time modeling is found in situations in which this pattern is usable; the context for the pattern.

Time modeling specification is applicable in system models with time requirements. Time constraints can be modeled and analyzed to verify the requirements.

The applicability of Time Modeling Simulation is found in system models like the TMT model and APE model. The addition of meaningful real-time based simulations allows the models to be executable. Executable means that the
The Cameo Simulation Toolkit contains four main engines which allow for the simulations of Activity, State Machine, Sequence, and Parametric Diagrams. Time modeling is used actively to describe time and its use in the same types of diagrams.

3.3 Concept

The main structure used to model time comes in the form of association relationships which allow for simulations to be ran together in parallel. State machines, Lifelines, and Activities can all be ran simultaneously using the parent block as the Target in the Simulation Configuration. The directed composition relationship between blocks that are the owners of the different diagrams allow for multiple simulations to be connected and ran together.

The three types of behavioral diagrams are State Machine, Sequence, and Activity Diagrams. Within an analysis context each of the diagrams have a block owner that is used to define the overall structure. Under each of the behavioral diagrams there are the different ways to represent how a system is designed to work.

In a state machine the main feature are the states for which a block can be in during the simulation. The way to connect the states is through transitions. Now transitions have a lot of features that determine when a state is entered/exited. A null transition is a transition that does not have any deterministic qualities, it simply transitions from one state to another. Transitions also have triggers, guards, and effects. Triggers would be an event that cause the transition to happen, commonly in the examples a trigger is a signal that is sent from an activity or user to initiate the transition between states. A guard is a condition that must be met in order to transition states. A guard would be more like a boolean statement that if true allows for the transition to happen. An effect on a transition would represent an activity that needs to occur before the states can transition.

Also common in state machines are nested states. The nested states represent the multiple states that a single element can be in at one time. The nested states are useful to create more organization between states. For example, there can be operating and idle states which then have nested states to better describe the behavior of when the system is operating or idling.

Sequence diagrams are composed of lifelines which are graphical representations of part properties that have types. On a lifeline you can add in the different states that are owned by the block, these are called state invariants. The state invariants represent the states that

will be executed in the state machine if they are simulated in the same analysis context. Sequence diagrams can have multiple lifelines to represent all of the different participants in a simulation. In order to communicate between lifelines, there are messages and calls that can be sent between lifelines, and even sent to self. There are async and sync calls, where async calls do not wait for a response while sync calls must wait for a response from the receiver. Messages are an effective way to send signals to self to represents a triggered transition on a state machine.

The activity diagrams are important because they can be run independently or dependently of the state machine and sequence diagrams. The activity diagrams are filled with different types of actions and constraints. If the activity is dependent there can be signals sent in between the activity diagrams to control the flow. One activity diagram can send a signal to another through ports, and then the signal can initiate an activity, where at the end another signal can be sent back to the first activity and the activity can continue.

In conjunction with the behavioral diagrams, definition diagrams are used to full represent the elements involved in a simulation. BDD's are useful to show the overall analysis context that is created by the association of blocks. IBD's can be used to show to ports and connectors between blocks that will be used to send signals to each other. These definitions are essential to understanding how the elements work with each other to create a complete time model.

3.3.1 Context of Analysis

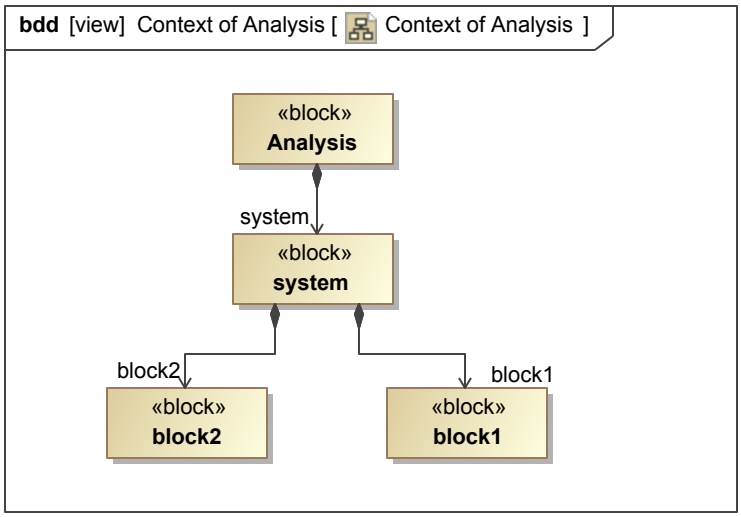


Figure 14. Context of Analysis

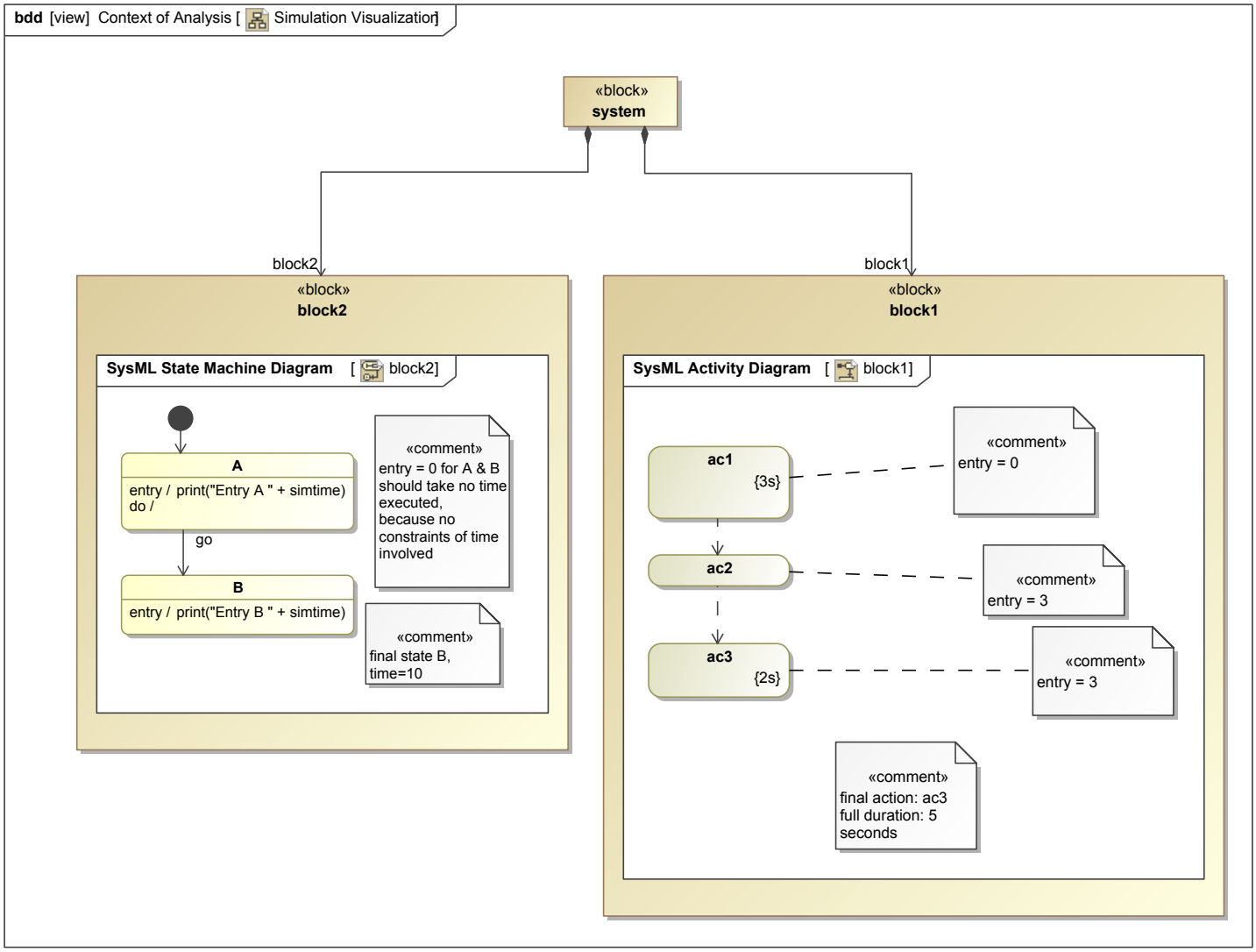


Figure 15. Simulation Visualization

Simulation Visualization shows the two behavior diagrams, the state machine under block 1 and an activity diagram under block 2. When running a simulation, the simulations both have the same simulation clock, but do not depend on each other. If the state machine cycles through all the states first, the activity diagram continues its processes until itself has completed all of the activities.

The comments are provided to explain how the states and activities should run, and their expected related times.

Another example would be to have both the sequence and state machine diagrams on the same BDD to show the states concurrently. This is advantageous because the state machines and sequence diagrams should match each other and have identical run times. This is the best way to verify the accuracy of the time model. There are time constraints added to both the state machines and lifelines which should correspond to each other in order to accurately describe the behavior of the model.

3.3.1.1 State Machines

State machine diagrams are used to represent the state of the model at different times during execution. For more information on the specific uses of state machines when modeling time, please see the Sample Model section.

The overall structure of state machines are important in order to understand the best way to model time. State machines are composed of states and transitions. In the scope of time modeling, it is most important to look at the states and their time constraints.

As shown in the State Machine diagram states have three main behaviors that affect the flow of the state machine: entry, do, and exit behaviors. State machines can also be nested within each other. This means that the state machine is in multiple states at one instance. For example, the state of the model is in both initial and A at 0 seconds into the execution. When modeling concurrently with sequence diagrams, there is not a way to specify that an element is in two states at one time.

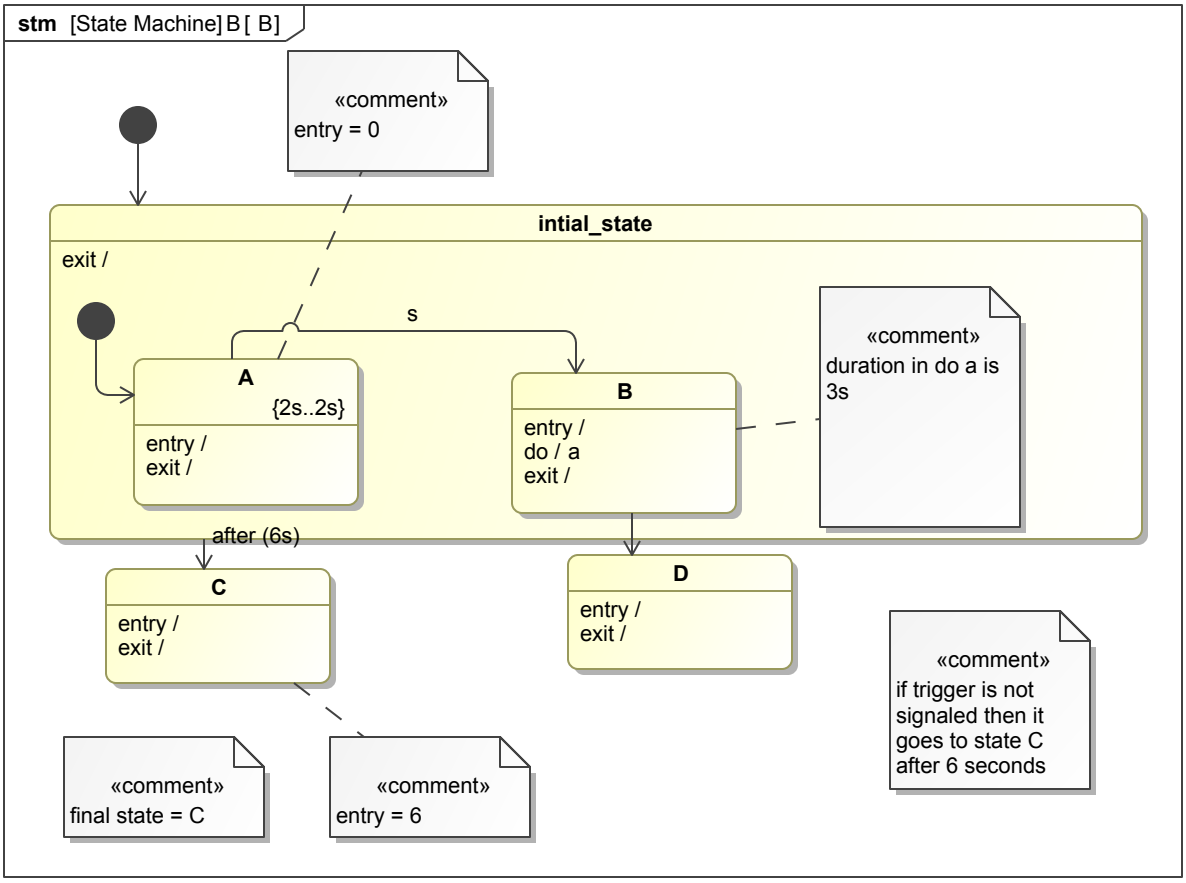


Figure 16. B

3.3.1.2 Interactions

Sequence diagrams are used to represent the states of an element at a specific instance in time. Sequence diagram structure is based on what is called a lifeline. Lifelines represent a block or other element that owns specific states.

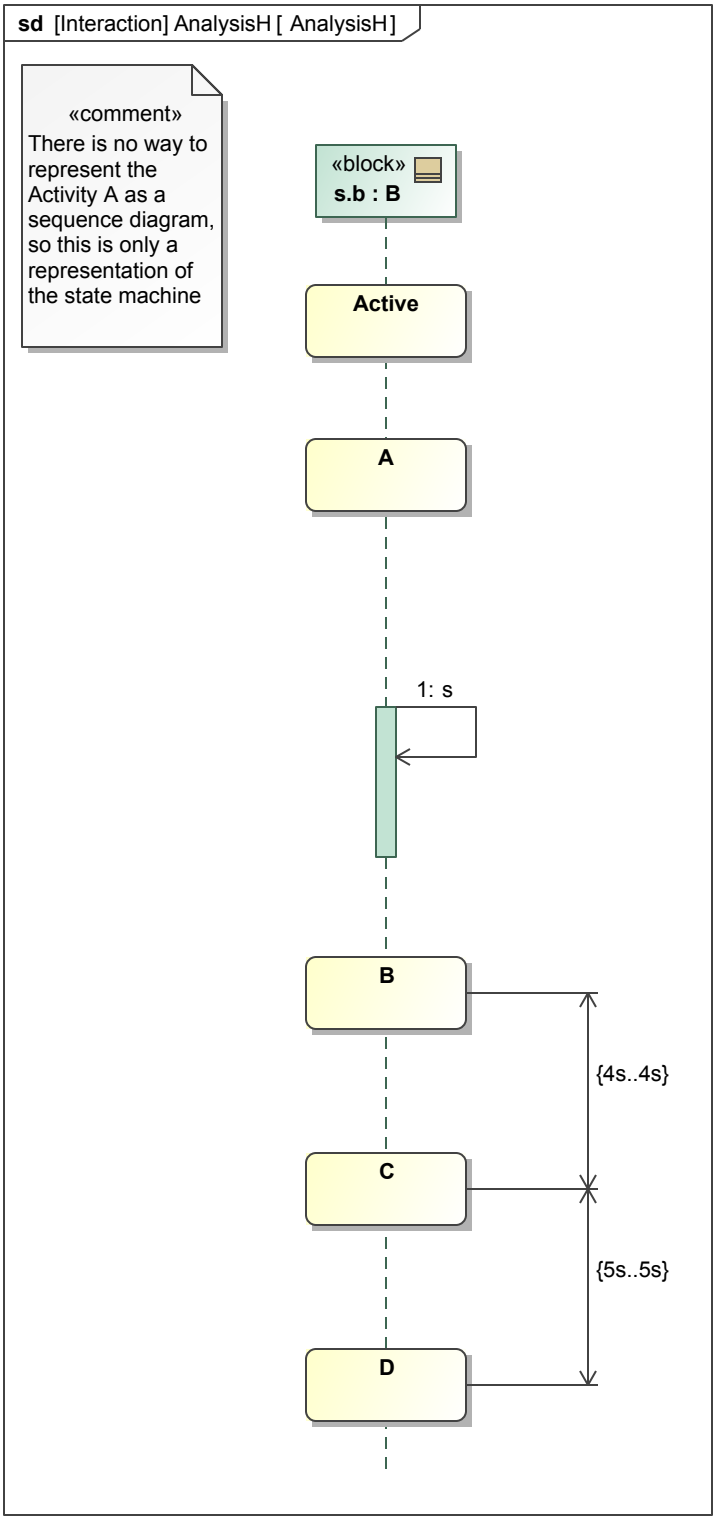


Figure 17. AnalysisH

Interactions are composed of lifelines which are graphical representations of part properties (or other elements). On a lifeline you can add in different state invariants that are owned by the element. The state invariants represent the states that will be executed in the state machine if they are simulated in the same analysis context. Sequence diagrams can have multiple lifelines to represent all of the different participants in a simulation. In order to communicate between lifelines, there are messages and calls that can be sent between lifelines, and even sent to self. There are async and sync calls, where async calls do not wait for a response while sync calls must wait for a response from the receiver. Messages are an effective way to send signals to self to represents a triggered transition on a state machine.

In interactions there is no way to represent the nested states in state machines.

3.3.1.3 Activities

3.4 Simulation

The purpose of a simulation is to gain system understanding, explore and validate desirable or undesirable behaviors of a system without manipulating the real system, which may not have yet been defined or available, or because it cannot be exercised directly due to cost, time, resources or risk constraints. Simulation specifies the behavior and interactions of a system on a component and system level.

Time constraints and duration constraints are the interactions used in time modeling. "A time constraint and a duration constraint can use observations to express constraints involving the values of those observations. A time constraint identifies a constraint involving the values of those observations. A duration constraint identifies two occurrences, called start and end occurrences and expresses a constraint on the duration between them." (p. 258)

3.5 Consequences

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

For time simulation modeling, there are lots of results that can stem from them. The main results that come from the simulations models are the outputs of time. Such as how long a state took, how long an action took, how long it took to enter into a specific state, or how long the whole simulation took. The end result of this pattern is that the time represents what a system would do in a certain amount of time.

The side effects of using the simulation is that the simulation, may not be exactly the correct time. There have been issues with the simulations not executing correctly. Know that when creating duration constraint or time constraint they must be a duration or time constraint, not a constraint or it will not run. The side effects come also with the simulations not always being exactly specified before so there will come lots of questions.

Some trade offs to the simulation is that it is simulated time. When is comes to real time, some things could end up a lot differently. When it comes to simulating a system some things may be effected in different ways then what the simulation shows, therefore effecting the time.

3.6 Implementation

In each of the sample models, there are sequences and state machines being simulated. The sequences represent a general understanding for how the state machines should run with the simulation. The life line represents an instance's interactions as an ordered list of occurrence specifications describing what can happen to the instance during an execution of the interaction (Friedenthal, Moore, Steiner p.251). The Sequence Diagram will be discussed more in-depth later. "State machines typically execute in the context of a block, and events experienced by the block instance may cause state transitions" (Friedenthal, Moore, Steiner p.275).

In each of these examples, there are state machines that consist of states. "States represent some significant condition in life of a block, typically because it represents some change in how the block responds to events and what behaviors it performs"(Friedenthal, Moore, Steiner p.276). To simulate time in the state machines, there are duration constraints and time constraints. "A time constraint and a duration constraint can use observations to express constraints involving the values of those observations. A time constraint identifies a constraint that applies to a single occurrence..." (Friedenthal, Moore, Steiner p.258). "A duration constraint identifies two occurrences, called start and end occurrences, and expresses a constraint on the duration between them" (Friedenthal, Moore, Steiner p.258). On the states in the examples there is also entry behaviors and do behaviors. Entry behaviors are executions that occur whenever the state is enter (p.276). There is also exit behaviors, which are not used in these examples, but occur whenever exiting the state (p.276). Do behaviors occur after the entry behaviors are completed (p.276). Each of the entry behaviors, in these examples, are print statements that will appear in the console of the simulation tab. The print statement state the time it took to enter into the state. The do behaviors have activities that should be executed.

The sequence diagrams that are shown in the examples contain duration constraints and time constraints as well; they represent the same thing as state machines. In the sequence diagrams there are state invariants that represent the states in the state machines. "A state invariant on a lifeline is used to add a constraint on the required state of a lifeline at a given point in a sequence of event occurrences" (Friedenthal, Moore, Steiner p.572). In these examples, there are basic interaction operators. "An interaction operator defines the logic used to time-order the execution of the operands" (p.260). An operand is complex control logic for the interactions. For these examples, the interaction operators used are just alternatives (Alt). Alternative is "an operator in which exactly one of its

operands will be selected based on the value of the guard" (p. 261). A guard contains a constraint expression that needs to be validated to execute (p.260). There are other interaction operators that can be viewed in "The Practical Guide to SysML". Note that these sequence diagrams in the examples could be used to recreate the state machine as shown in the examples.

3.6.1 Example A

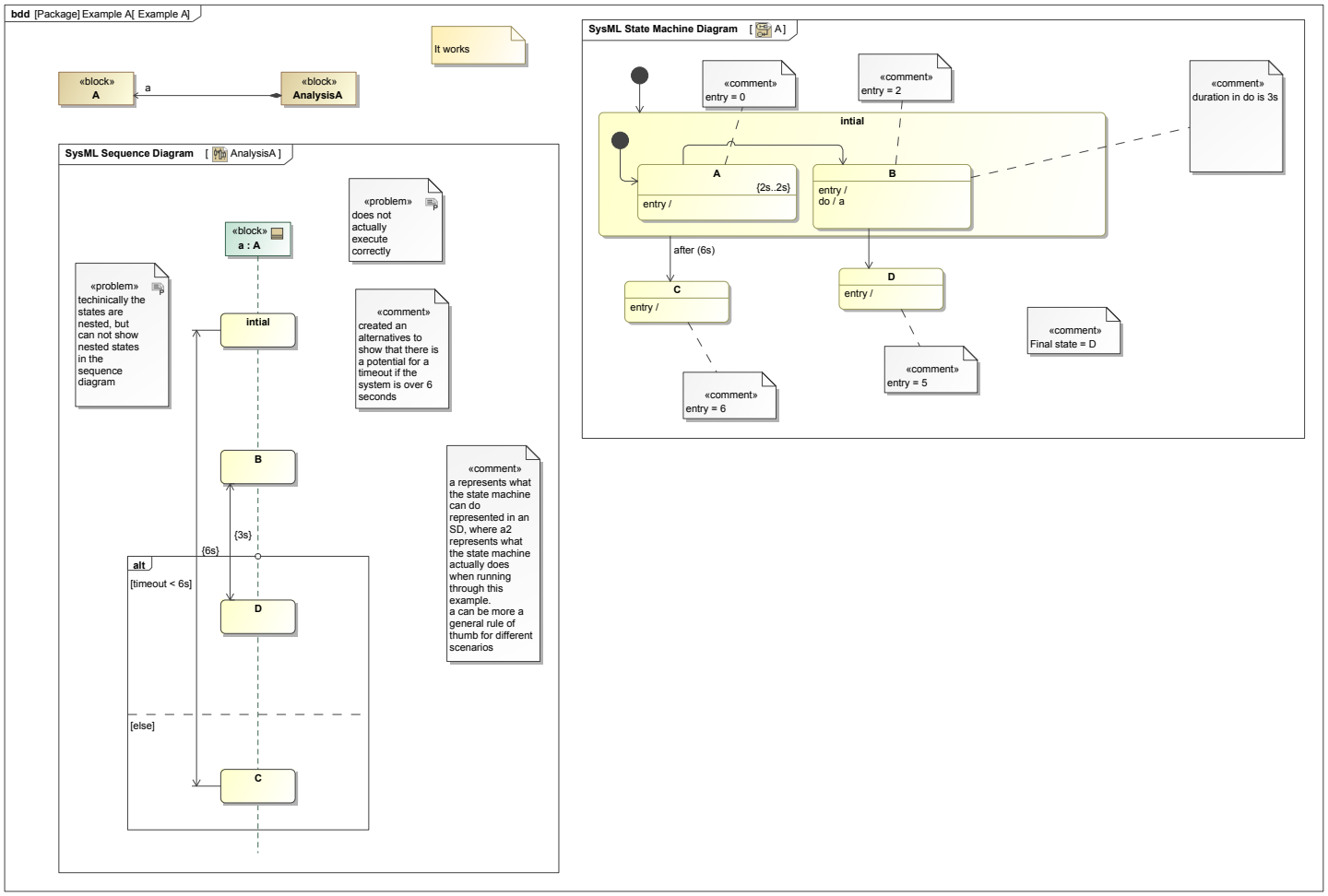


Figure 18. Example A

The simulation of the state machine runs as such:

1. The simulation enters nested states *initial and A*.
2. A two second duration constraint in state *A* is satisfied before entering state *B*.
3. A do-behavior is executed in state *B* with a duration of three seconds.
4. State *D* is entered and the total simulation time is five seconds.

The final state of the simulation will always be State *D*. State *A* has a duration of two seconds and state *B* has a duration of three seconds which sums to a total time duration of five seconds. The transition to state *C* is specified to occur at a simulation time of six seconds and therefore will never be reached.

As stated before, the sequence diagrams represent a general overview of the state machines and how they should execute. The sequence diagram runs just as the state machine runs, except through the state invariants. The alt interaction is specified for the time out situation, where if some how the transitions and do behaviors between state A and B were longer then 6 seconds then the final state would be C. However, this should not happen due to the specific duration constraints and the final state should be D.

3.6.2 Example B

In Example B, the goal is to navigate through the state machine and either end up in state C or state D. In this case there is a signal that needs to be sent in order for the state machine to eventually transition into state D. The Execution Target in the simulation configuration is based on Analysis B. The sequence diagram is owned by Analysis B and the state machine is owned by block B.

When running the simulation using Cameo Simulation Toolkit, the timer on both the sequence and state machine are started at the same time.

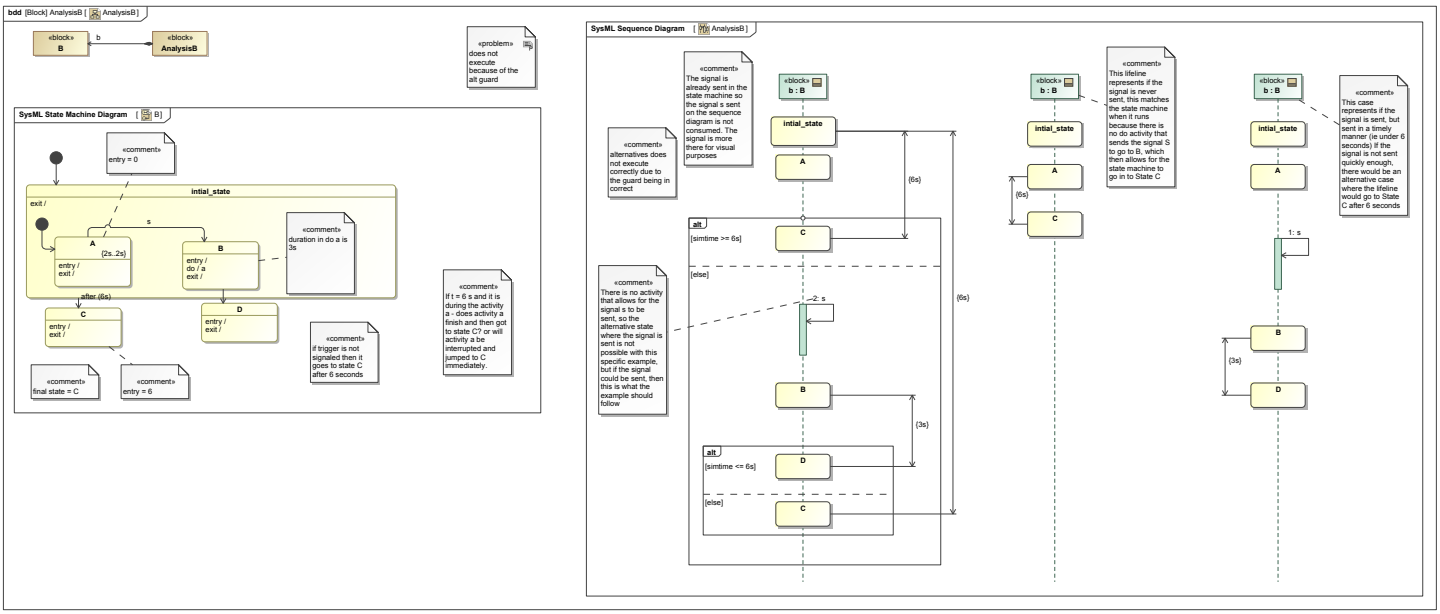


Figure 19. AnalysisB

The simulation runs as follows:

1. The states *initial_state* and *A* are entered at a simulation time of zero seconds.
2. The simulation remains in state *A* for two seconds.
3. If signal *s* is manually injected withing two seconds, state *B* will be entered.
4. If the signal is not injected, an additional four seconds will pass before the transition into state *C*.
5. If state *B* is entered, a do-behavior is executed with a duration of three seconds before entering state *D*.

Despite a constraint of two seconds, the simulation will remain in state *A* for six seconds before entering state *C* if the signal *s* is not injected.

If the do-behavior executed in state *B* had a duration time of four seconds, there would be two possible state outcomes for the transition from B. State C could be entered since the simulation time has reached six seconds, or state *D* could be entered since *D* is the subsequent state of *B* as defined by the transition. In practice, the final state is C even though the final state could be D.

If the signal *s* is added to a do-behavior, the time constraint of two seconds is ignored state *A* and will go straight into state *B*.

The Sequence Diagram has the lifeline symbol that represents Block B and all of the owned states that B could be in. The states are represented as state invariants on the lifeline. The sequence diagram displayed shows some of the alternate cases that the model could follow when executed. Based on the way the state machine is built, the lifeline should only go from initial state to A to C, where there is a six second duration constraint in between the state A and C. However, if multiple situations are considered, it allows for a plethora of additional operational scenarios for the sequence diagram to follow. Unfortunately the operands of the alternate frame does not work as expected, so the multi-case scenarios were unable to be successfully tested.

3.6.3 Example C

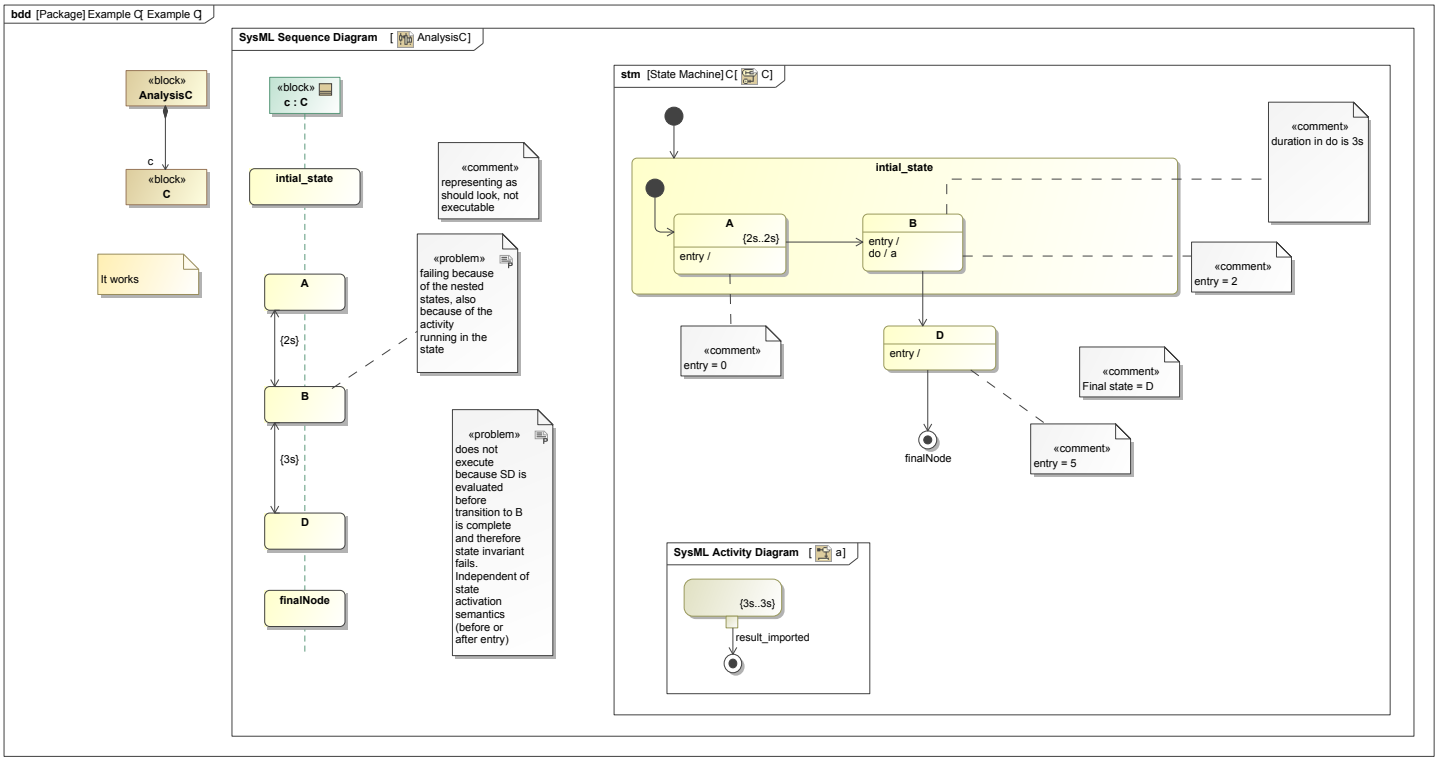


Figure 20. Example C

The state machine runs as such:

1. The simulation enters states *initial_state* and *A*.
2. State *A* remains for two seconds before entering into state *B*.
3. A do-behavior in state *B* is executed for three seconds.
4. State *D* is entered and the total simulation time is five seconds.

The sequence diagram says the same thing as the state machine. Notice that entering into a do behavior with an activity diagram should not take any longer then if it was just a do behavior saying to wait three seconds. The duration will still end at five seconds and end in state D.

3.6.4 Example D

In Example D, the goal is to navigate through the state machine to states *C* or *D*. The Execution Target in the simulation configuration is *Analysis D*. The sequence diagram is owned by *Analysis D* and the state machine is owned by block *D*. When running the simulation using Cameo Simulation Toolkit, the timer on both the sequence and state machine are started at the same time.

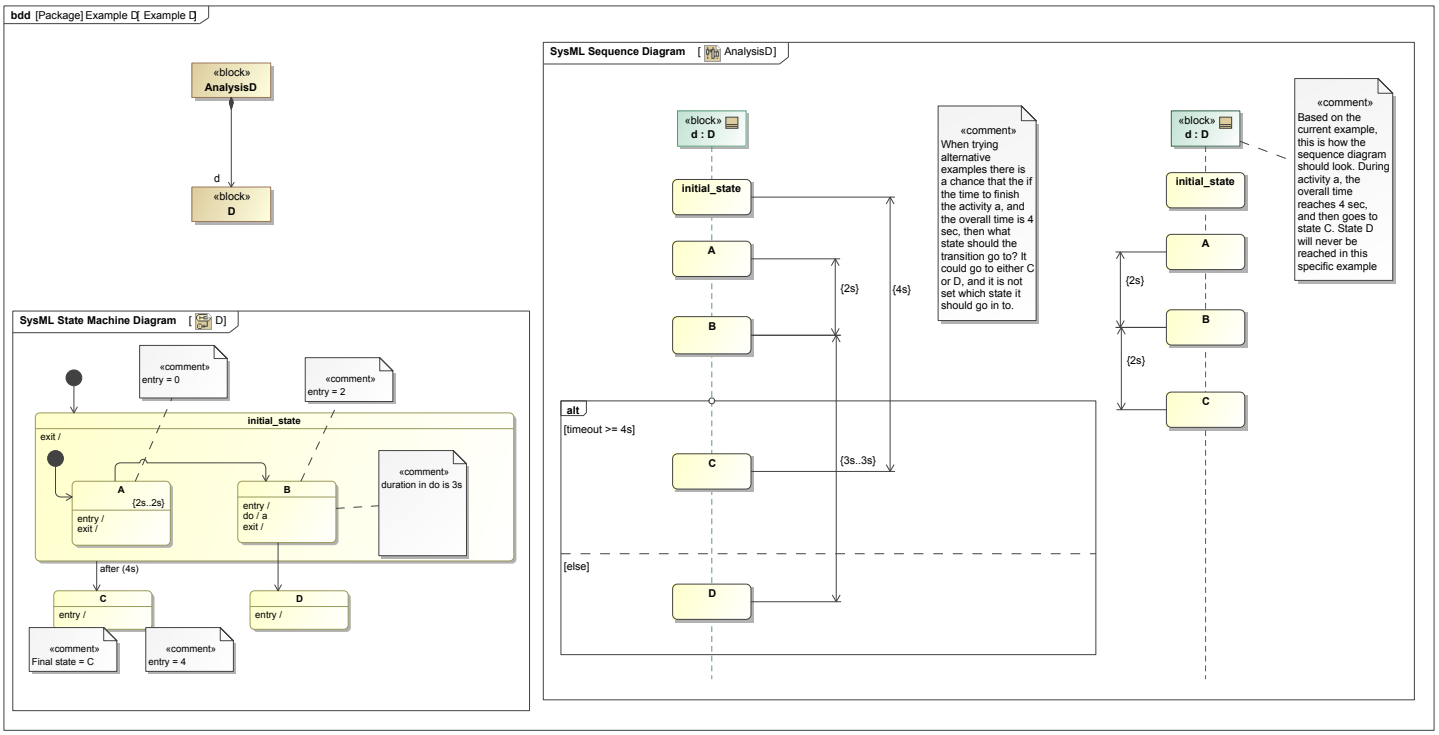


Figure 21. Example D

The simulation runs as follows:

1. States *initial_state* and *A* are entered at time zero.
2. State *A* is entered and exited all while the simulation time is zero.
3. State *B* is entered at two seconds and do-behavior *a* is executed with a duration constraint of three seconds.
4. The four second constraint on the transition from *initial_state* to *C* prevents the completion of state *B*.
5. State *C* is entered at four seconds. State *D* is never reached.

The time constraint applied in state *A* allows for state *A* to be entered and exited when time is zero, but the state machine waits two seconds until entering state *B*. The time constraint does not apply directly to state *A*, but to the transition from state *A* to state *B*. Also, the do-behavior of State *B* is never finished and never officially "exited".

The Sequence Diagram has the lifeline symbol that represents Block D and all of the owned states that D could be in. The states are represented as state invariants on the lifeline. The sequence diagram displayed shows some of the alternate cases that the model could follow when executed. Based on the way the state machine is built, the lifeline should only go from initial state, state A, state B, and finally state C. The duration constraints represents that from the initial state, it will take 4 seconds to reach the final state C. There is no way to represent that there is an activity executed on a lifeline. A duration constraint can only be applied to B in order the accurately describe the time behaviors of the model. The two lifelines represent the same block, just different alternative scenarios are being considered for each lifeline. The alternative is attempting to represent the equivalent of an if statement, but the if statement cannot be properly modeled on a lifeline.

3.6.5 Example E

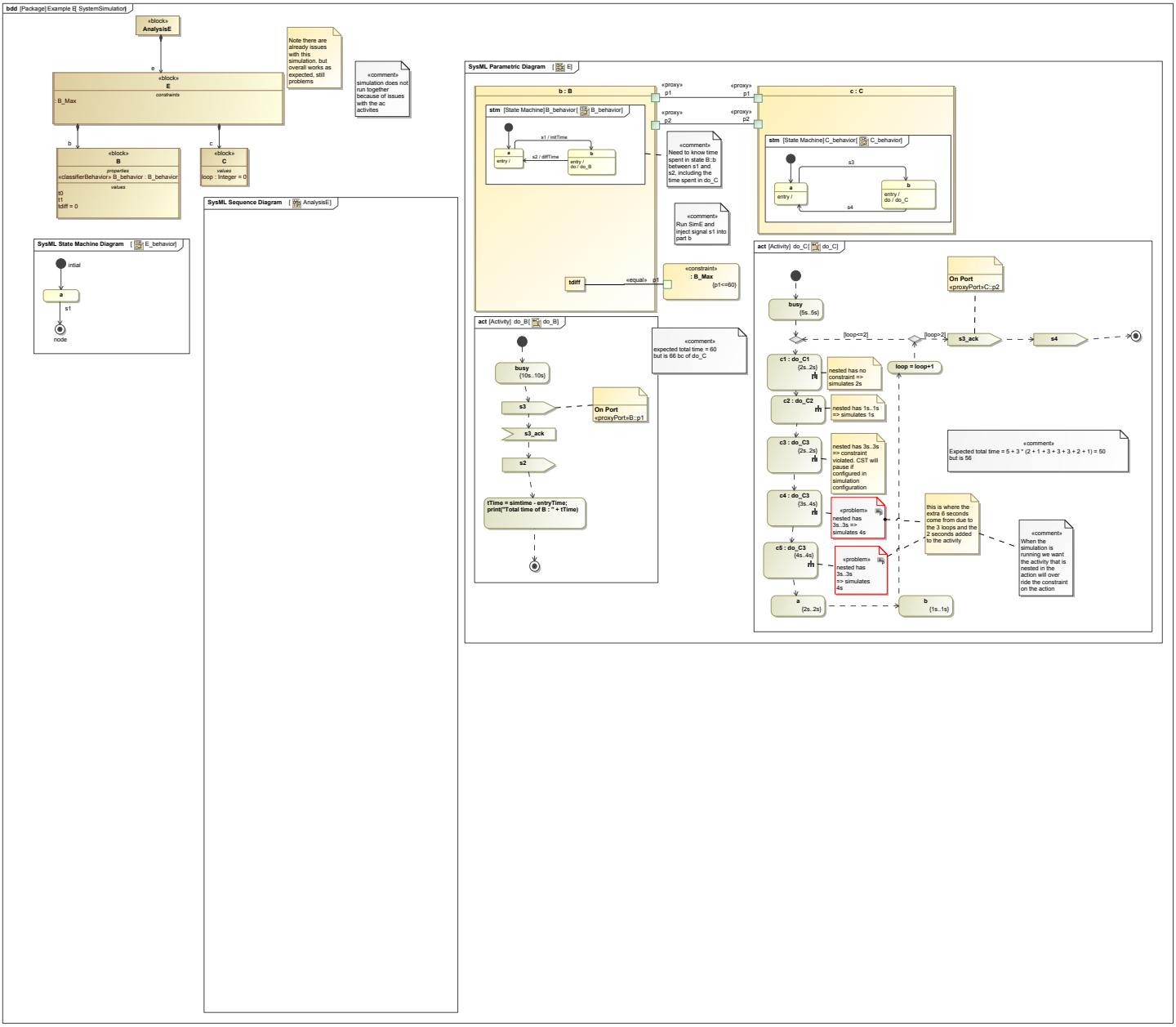


Figure 22. SystemSimulation

In this example, block *E* has two subsystems, block *B* and block *C*. The simulation should run as such:

1. State machines associated with blocks *B* and *C* run concurrently with state *a* being the initial state.
2. Signal *s1* transitions the state machine of block *B* to state *b*.
3. The do-behavior in state *b* is executed which enters into activity *do_B*.
4. The action *busy* within *do_B* has a duration of ten seconds before sending signal *s3*.
5. Signal *s3* initiates the transition to state *b* in block *C* where the do-behavior *do_C* is executed.
6. The action *busy* within *do_C* has a five second duration before continuing to action *c1*.
7. Action *c1* executes for two seconds with a nested activity.
8. Action *c2* executes with a nested activity activity for one second.
9. Action *c3* executes for two seconds with a nested activity lasting three seconds, causing a violation. The simulation will execute the three second nested activity, but pause to report the violation.
10. Action *c4* executes with a minimum duration of three seconds and a maximum duration of four seconds. A nested activity lasts three seconds; however, the simulation lasts four seconds.
11. Action *c5* executes for four seconds despite a nested activity of three seconds.
12. Action *a* executes with a duration of two seconds followed by action *b* with a duration of one second.
13. The variable *loop* is incremented by one, and the cycle continues until *loop* has a value greater than two. The duration for each cycle should be 50 seconds, although in practice the duration is 56 seconds.

14. After the loop, signal *s3_ack* is sent and received by the activity *do_B*.
15. Signal *s2* is sent which transitions from state *b* to state *a* in block *B*.
16. Signal *s4* is sent from the *do_C* activity and initiates the transition from state *b* to state *a* in block *C*.
17. Overall, the simulation lasts for 66 seconds, although it is expected to be 60 seconds.

During this simulation, the importance of nested constraints is shown. Nested constraints will always take precedence and override the duration constraint of the parent element.

There is also a SD in this example, that shows how the general simulation will run. The SD runs in parallel with the state machine. Starting in state invariants a for all the life lines, then the main block E sending a signal s1 to block B, transition into state invariant b. Then waiting ten seconds before sending the signal s3 for the transition into state invariant c for the block C. Then having three loops on the state invariant c in block C, where the first is 22 seconds long, then the last two are 17 seconds long. Then sending a signal s3_ack back to block B and the sending s2 signal to itself to go to state invariant a again. Then the block E sends a signal s4 for the transition block C state invariant to go back to a. Then block E signaling s1 to itself to finish of the simulation at the final node.

3.6.6 Example F

Example F is the most robust example, for it has interactions between state machines and activity diagrams and the sequence diagrams have many lifelines to represent the different elements involved. Below is a BDD which displays all of the state machines involved (3 of them), all the activity diagrams involved (3 of them), and the sequence diagram which represents the three different blocks and their owned states.

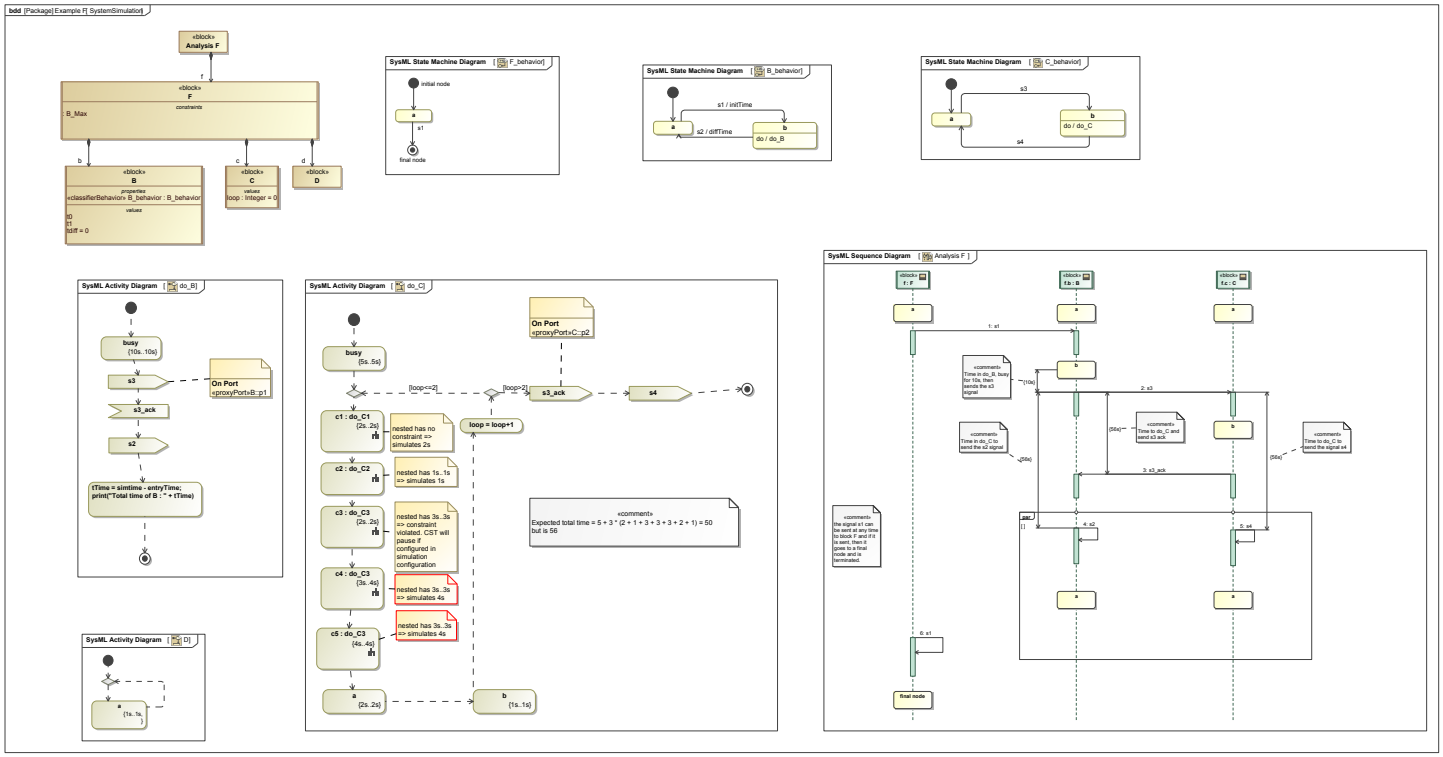


Figure 23. SystemSimulation

In this example, there are many possible outcomes based on the signals sent at what time. The initial state of the state machine is a in F_behavior. Now from there a signal s1 can be sent back to itself and the whole simulation is over. There state of the analysis is in a from B_behavior and a in C_behavior. Until a signal is sent from the user there simulation is waiting patiently. The block in question can be changed in the variables section of the Simulation toolbar. In order to send the right signal in the right state machine, the main block needs to be changed. In order to send the right signal, change the block in question to B and then send the trigger signal s1 to transition from state a to b in the B_behavior state machine.

Once in state b in B_behavior, the do activity do_B starts. In the do_B activity there is a 10 second wait and then another signal is sent automatically to the C_behavior. The s4 signal is sent to C_behavior which transitions the state from a to b, which then starts the do_C activity. Now, the do_B activity is waiting for a signal, s3_Ack, which will not be sent to the do_B until the loops of the do_C activity are completed. In the do_C activity, there is a major loop consisting of activities with their own time constraints. The comments in the activity outline the expected outcome of the duration constraints.

Table 1. Duration Constraints

	Duration Constraint	Nested Constraint	Expected Result	Actual Result
c1	Yes (2s)	Yes (2s)	2s	2s
c2	No	Yes (1s)	1s	1s
c3	Yes (2s)	Yes (3s)	3s	3s
c4	Yes (3s...4s)	Yes (3s)	3s	4s
c5	Yes (4s)	Yes (3s)	3s	4s

So the conclusion of the different combinations of the duration constraints, nested or not, is that the overall longer duration constraint is the overall time envelope that is followed. In Example F, the difference in run time is multiplied from the three times through the loop, so what starts out as a 2 second time difference is propagated in the loops and throughout the rest of the simulation.

Once the loop has been executed three times (total loop count = 2), then the s3_Ack is sent back to do_B and s4 is sent to C_behavior. In C_behavior, the transition is made from b to a, where the state waits for the next signal. In do_B after receiving s3_Ack, signal s2 is sent to B_behavior which sends B_behavior from state b to a.

The signal s1 can be sent in the scope of F which will end the simulation at any time. Additionally, there is the Activity D which acts as an external clock to time keep during the overall simulation. There is no relation to the other behavioral diagrams from Activity D, but because they are in the same analysis context they will have the same starting time. Activity should not increase the time envelope and should be running in parallel for the duration of the simulation.

The sequence diagram represents the three main blocks that have states in the context analysis: F, B, and C. Since there is no time constraint on when certain signals are sent, the sequence diagram is more of a guideline of a possible flow of events. It is also not possible to depict the activities that are being completed internally (waiting for the signals).

3.6.7 Example G

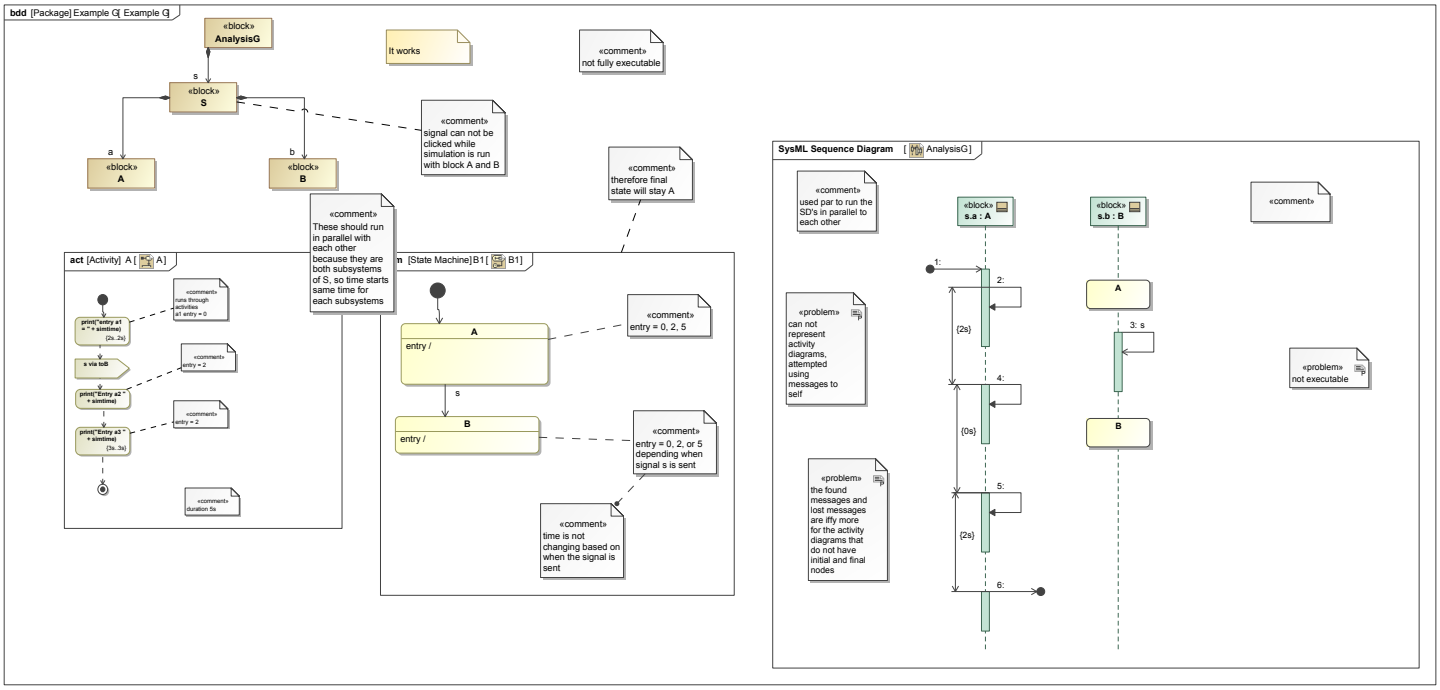


Figure 24. Example G

In this simulation, subsystem blocks A and B compose the system block S. Block B owns a state machine diagram, while block A owns an activity diagram. Simulation of each block runs in parallel. The simulation runs as such:

1. The simulation starts with block S as the target of execution. Blocks A and B simulate concurrently.
2. State A is entered and activity a1 executes with a duration constraint of two seconds.
3. A send signal action, sends signal s to the state machine of block B.

4. Activity a2 and State B are entered at a simulation time of two seconds.
5. Activity a3 is also entered at a simulation time of two seconds and has a duration constraint of three seconds.
6. Total simulation time is five seconds.

In a previous version of the model, the send signal action was not included between actions a1 and a2. In such a case, simulation time at the entry of State B is dependent on user injection of signal s into the simulation. If the signal is injected before any action has executed, then State B will be entered at zero seconds. If the signal is injected after the a2 action, then the simulation time at entry is two seconds. If injected after the last action, the simulation time at entry to State B will be five seconds.

In all cases, total simulation time will always be five seconds due to the duration constraints specified in activity A which are the basis for the global clock. The state machine has no duration constraints and therefore cannot define the simulation length.

In the sequence diagram, there is a lifeline representing block B. This represents how the state machine will run in a general sense. The simulation will run into state A, as it will also run into the state invariants state A on the sequence diagram. Then after the signal is sent the simulation will go into the invariants state B, such as the state machine goes into state B. A sequence diagram can not represent an activity diagram, therefore the activity diagram is not represented in the sequence diagram.

3.6.8 Example H

In Example H, the goal is to navigate through the state machine end in state D. The Execution Target in the simulation configuration is based on Analysis H. The sequence diagram is owned by Analysis H, the state machine is owned by block B, and the activity is owned by block A. When running the simulation using Cameo Simulation Toolkit, the timer on both the sequence, state machine, and activity are started at the same time. The activity diagram in this example runs completely separate from the sequence and state machine diagrams. Their only connection is that they are in the same analysis context and then have the same start time.

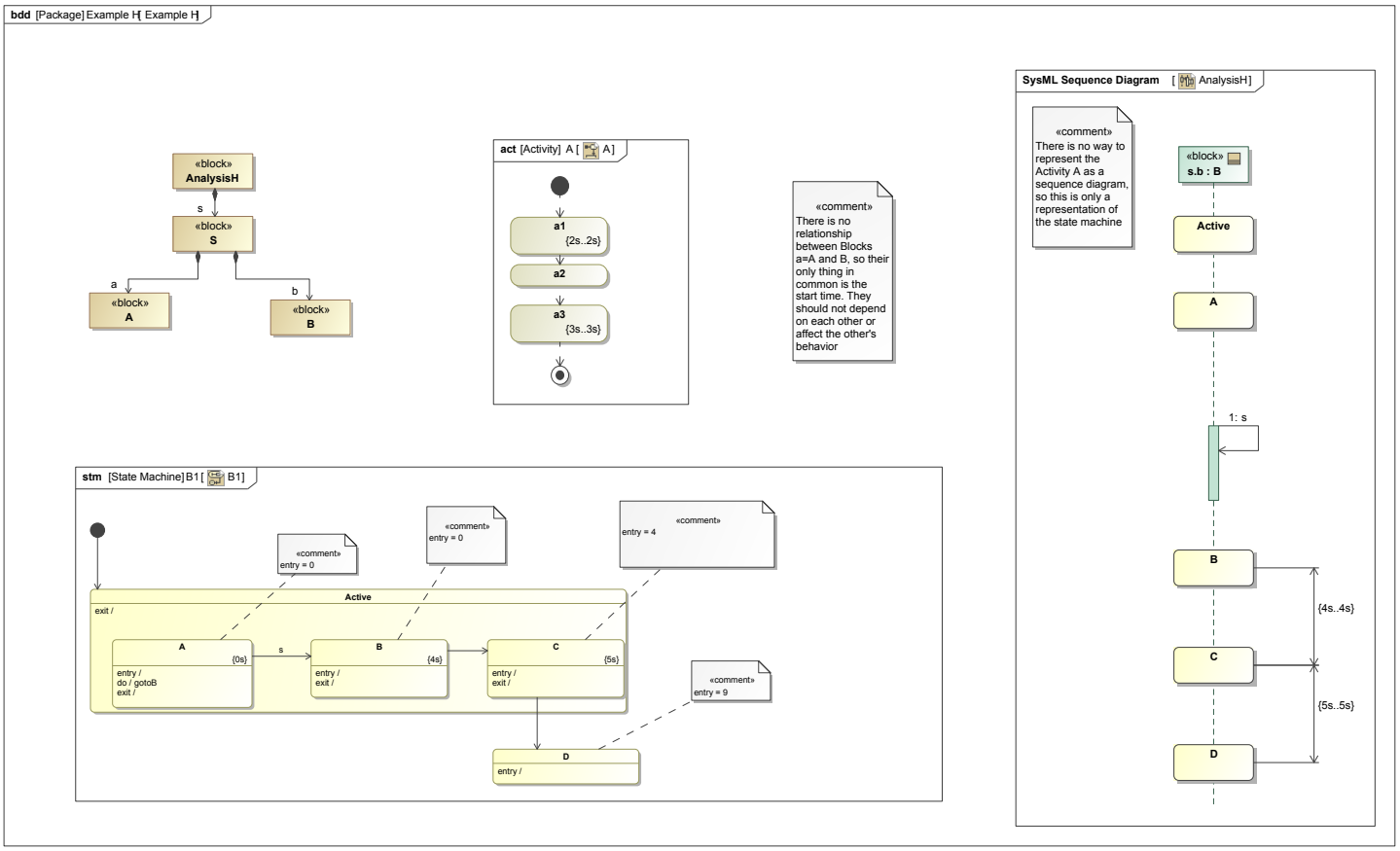


Figure 25. Example H

The State Machine runs as follows:

The state machine at time 0 is in both the Active state and in state A, then while the time is still equal to zero, signal s is sent in the gotoB and the transition is triggered to state B, which is then exited all while the simulation time is 0. Next there is a four second duration constraint in state B, then entry into state C after the duration constraint has been met. Block B enters into state C when the

simulation time is 4 and exits state C when the simulation time is 4. After 5 seconds, the duration constraint of state C is met and block B goes into the final state D, and is exited from the Active State at a simulation time of 9 seconds.

Again, it is the interesting feature of when the state is entered and exited in the same simulation time and the duration constraint does not hold the analysis block in the same state with the duration constraint.

The Sequence Diagram has the lifeline symbol that represents Block B and all of the owned states that B could be in. The states are represented as state invariants on the lifeline. Based on the way the state machine is built, the lifeline should go through all of the states with their specified duration constraints. On the sequence diagram the duration constraints are modeled differently from their corresponding constraints in the sequence diagrams. The duration constraints in the sequence diagrams say that it takes 4 seconds to go from state B to state C. The implication is that you stay in state B for 4 seconds and then transition to state C immediately after. If there are no time constraints, then the lifeline does everything in the same simulation time. In this case, from the first active state all the way to entering state B is with a simulation time of 0, which matches what is expected from the state machine.

With the state machines and sequence diagrams being run concurrently, in the case of a signal being sent, or modeled there is a discrepancy of which is the actual signal. In this example, the signal s is sent twice, once in the state machine, and once by the lifeline. In the simulation toolbar, there is a warning describing how the signal s was sent, but not consumed. Additionally, there becomes a discrepancy in the times of the states. When the behavior diagrams are simulated, a visual comparison can be made of the states of the analysis block in question. In this case, the diagrams should match up exactly, but in simulation, the sequence diagram runs significantly faster which causes the simulation to report errors that the block is in a different state than it actually is.

The activity diagram is run alongside the other diagrams but has no influence on the other diagrams in this example. It simply starts at simulation time t = 0 and moves from action a1 after 2 seconds to a2, and then onto a3 for a final 3 seconds. The total simulation time is 5 seconds. There is no relation to the other diagrams, the simulation of the activity is over and waits for the entire context of the simulation to be over. In other examples, signals are sent between the state machines and the activity diagrams. In that case, there would be a time dependence coming from the activity diagram.

3.6.9 Example I

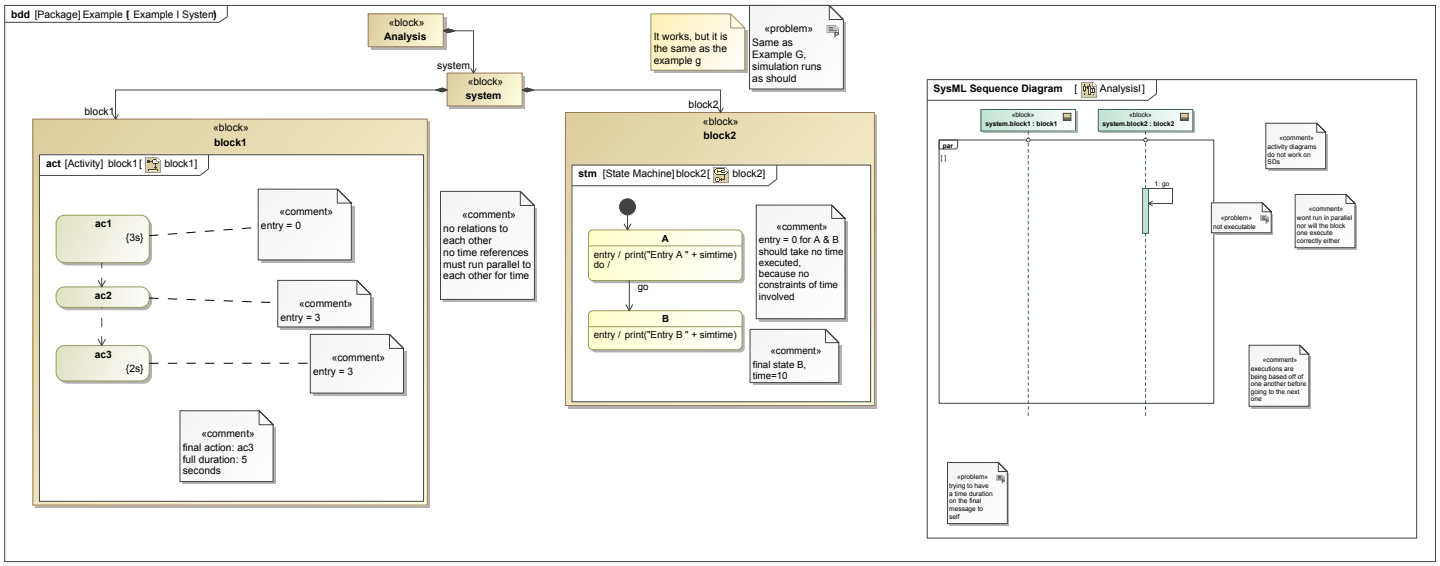


Figure 26. Example I System

This is another example very similar to Example G. Except in this example, we have a nested time constraint on the do behavior in block2 on state A for a duration of ten seconds. There is a main block system, the whole system, with two blocks, known as the subsystems, block1 and block2. Block2 owns a state machine diagram, while block1 owns an activity diagram. The simulation can run as such:

The simulation starts in the system, block system, it continues to the subsystems running them in parallel. Entering into the first state A and the first activity, ac1. Activity ac1 has a duration constraint of three seconds, then entering into activity ac2, and then to ac3 with a duration constraint of two seconds. The state machine and the activity diagram are running in parallel with each other, so as the activity diagram runs the state machine is also running starting in state A. Entering the do behavior with a ten second constraint, then transitioning into state B.

The global clock, or the time simulation clock, will never be more then ten seconds. The system has two different simulations going, but both should be in unison so the time they run through should be the same amount of time. The time it takes to get through the state machine diagram will take longer due to the duration constraints of ten seconds. After the state machine is done so should the activity diagram be because the activities ran through their amount of time as the states did.

The sequence diagram represents the state machine in a different format. The lifeline that is on the diagram represents the block that owns the state machine. The block goes through states, which can be depicted as state invariants on the lifeline. The duration constraint between the two state invariants represent the do behavior that contains the duration constraint of ten seconds on the state machine.

3.6.10 Example J

In Example J, the goal is to navigate through the state machine end in state D. The Execution Target in the simulation configuration is based on Analysis J. The sequence diagram is owned by Analysis J, the state machine is owned by block3 , and the activity is owned by block1. When running the simulation using Cameo Simulation Toolkit, the timer on both the sequence, state machine, and activity are started at the same time. The activity diagram in this example runs completely separate from the sequence and state machine diagrams. Their only connection is that they are in the same analysis context and then have the same start time.

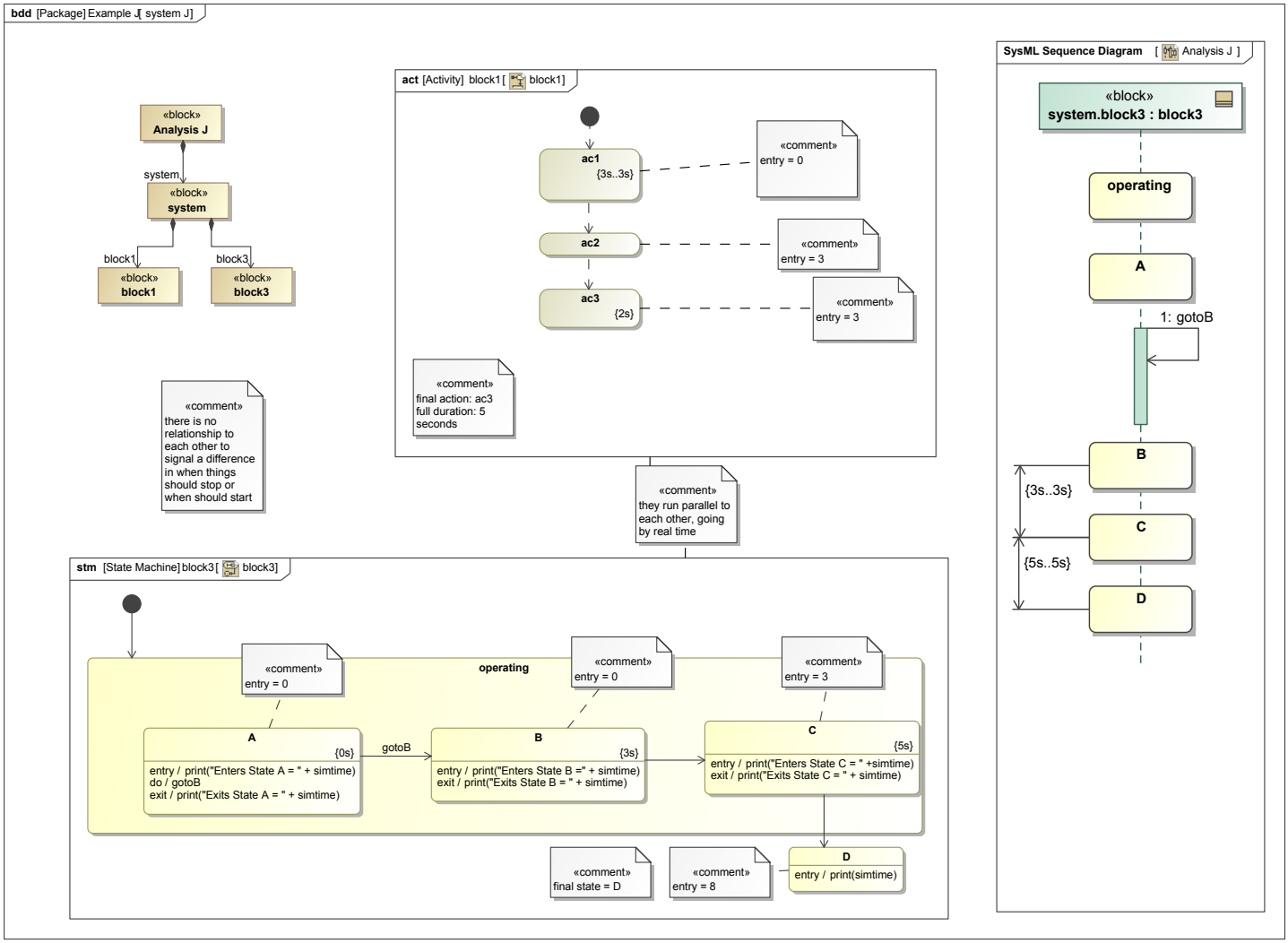


Figure 27. system J

The State Machine runs as follows:

The state machine at time 0 is in both the operating and state A, then a signal is sent from a do activity in state a that allows for the transition to state B to occur. The state machine exits state B at time = 0 and waits for 3 seconds to transition to state C. Also at 3 seconds, the state machine exits state C. After five more seconds (the total simulation time is 8 seconds) the state machine enters state D, where it will end the simulation. After states B and C are exited, the state machine remains in the operating state until the time duration constraints are met.

The Sequence Diagram has the lifeline symbol that represents block3, where the state machine is held. The flow from the state machine represented on the lifeline on the sequence diagram is represented, where the first state would be operating, then to state A, where a signal would be sent to allow the lifeline to enter state B. This all happens when the simulation time is at 0 seconds. The message to self in the lifeline is the best way on a sequence diagram to represent the activity in A. The duration constraints on the sequence diagram are synonymous with the ones in the state machine. The duration constraints allow for the delay into state C (enters at 3 seconds) and then another delay into state D (delay of 5 seconds, total time 8 seconds).

In the activity diagram, at time 0, the activity diagram is operating on activity ac1, then enters ac2 after 3 seconds, then to ac3 also at 3 seconds. After two more seconds the activity diagram is done running. Since there is no relation to the state machine or sequence diagram, once the time constraint on ac3 is met (3 seconds) then the activity diagram is done being simulated.

With the current the configuration, there is a discrepancy between the state machine and sequence diagram. When run concurrently, even though the simulation time is at zero, the state machine runs "slower" than the sequence diagram, and there is an error at state C. This comes from the signal being sent earlier in the sequence diagram, which allows the overall progression to be faster, while the state machine runs sightly slower. So even though the simulation time is still zero, the simulation visuals do not match up and an error is created.

3.6.10.1 Related Graphs

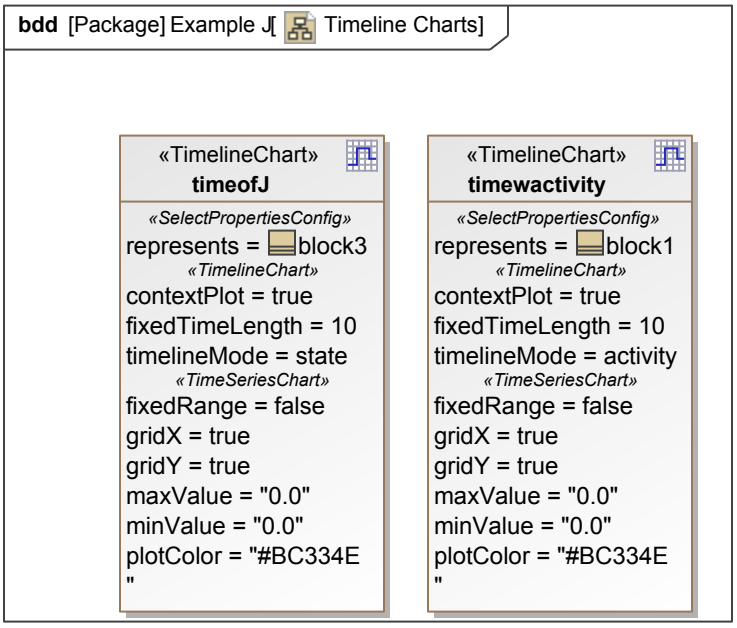


Figure 28. Timeline Charts

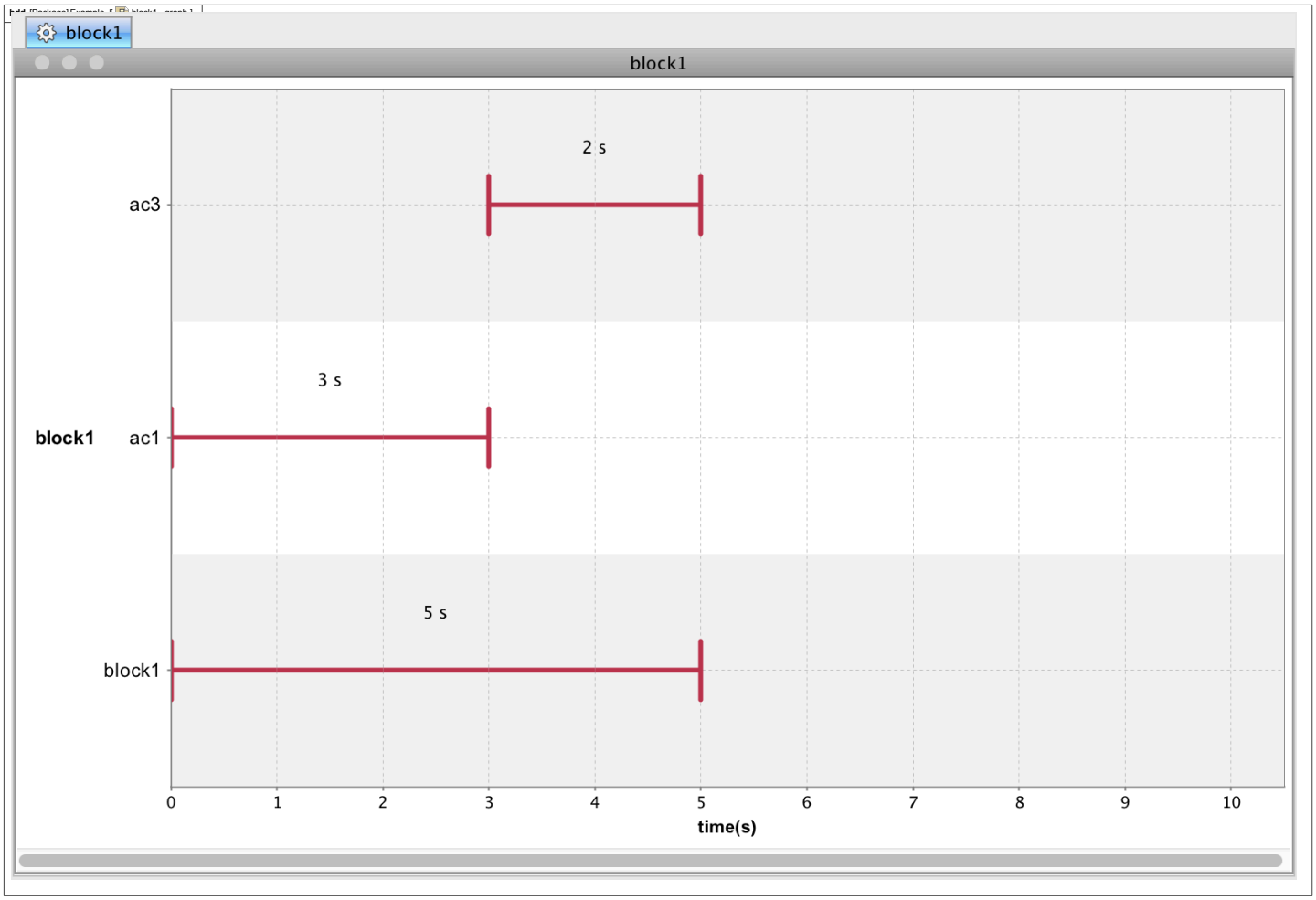


Figure 29. block1 - graph

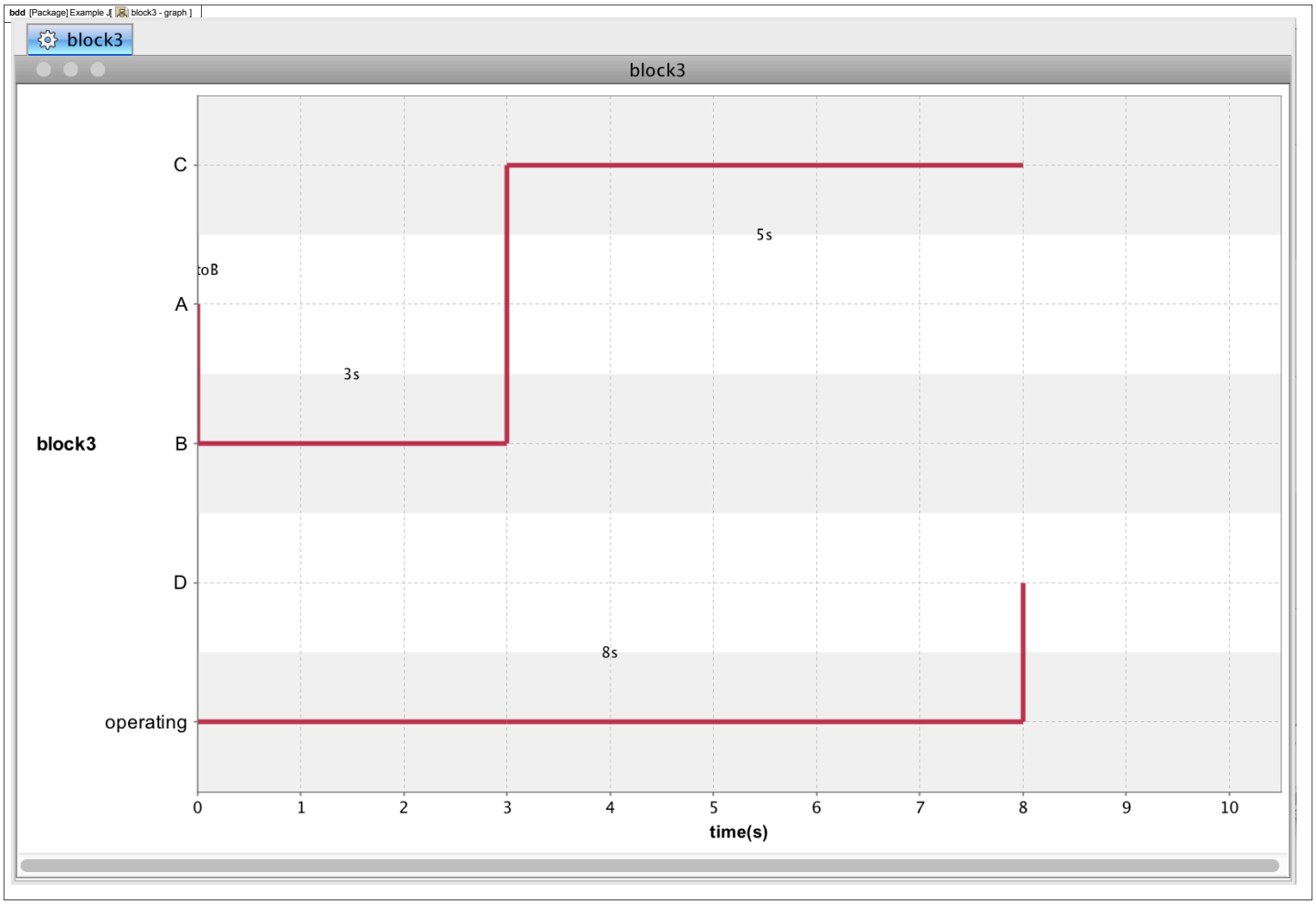


Figure 30. block3 - graph

3.6.11 Example K

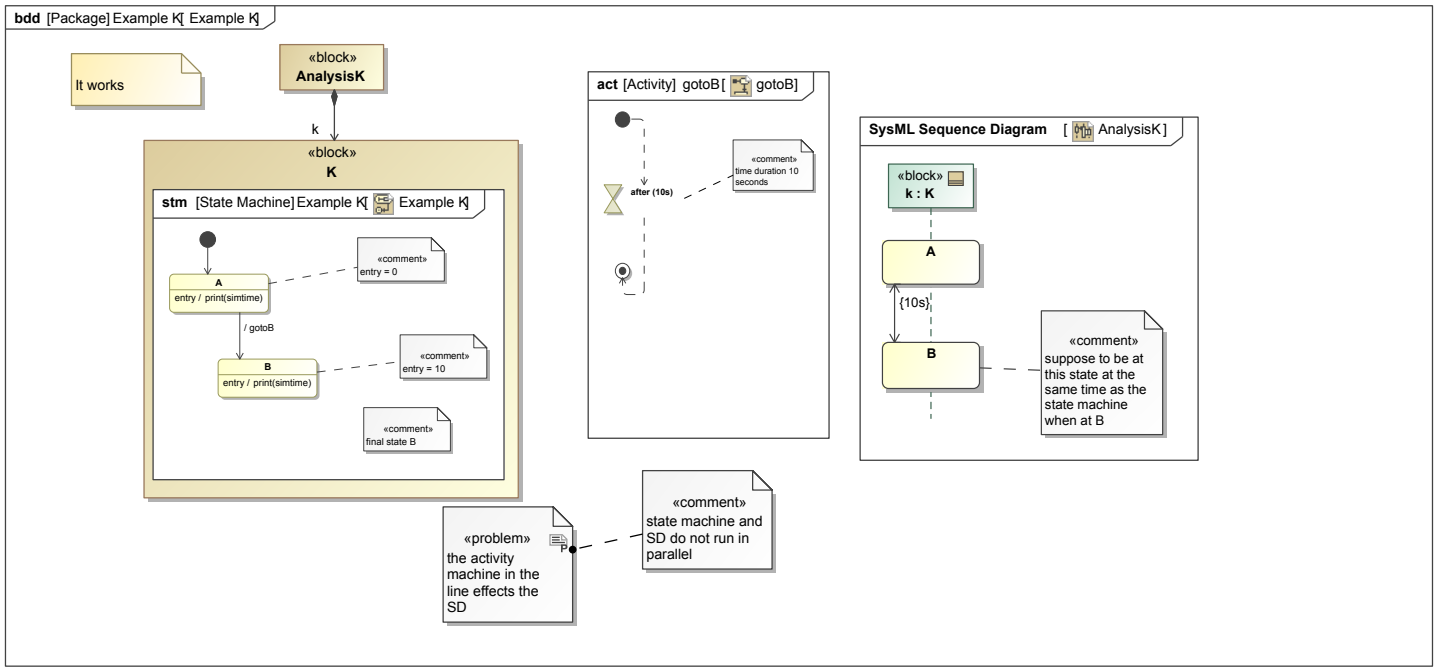


Figure 31. Example K

In this diagram, there is a block that owns a state machine. The simulation runs as such:

Enters into state A, then into the transition to go to state B. But the transition needs to go through a do behavior before continuing to state B. The do behavior has a ten second duration constraint. Then the final state will be B.

The lifeline that represents the block K shows the state machines general simulation. The sequence diagram goes from the state invariant A then to B after a ten second duration.

3.6.12 Example L

In Example L, the goal is to navigate through the state machine end in state B. The Execution Target in the simulation configuration is based on Analysis L. The sequence diagram is owned by Analysis L, the state machine is owned by block B, and the activity is owned by block A. When running the simulation using Cameo Simulation Toolkit, the timer on both the sequence, state machine, and activity are started at the same time. The activity diagram in this example runs completely separate from the sequence and state machine diagrams. Their only connection is that they are in the same analysis context and then have the same start time.

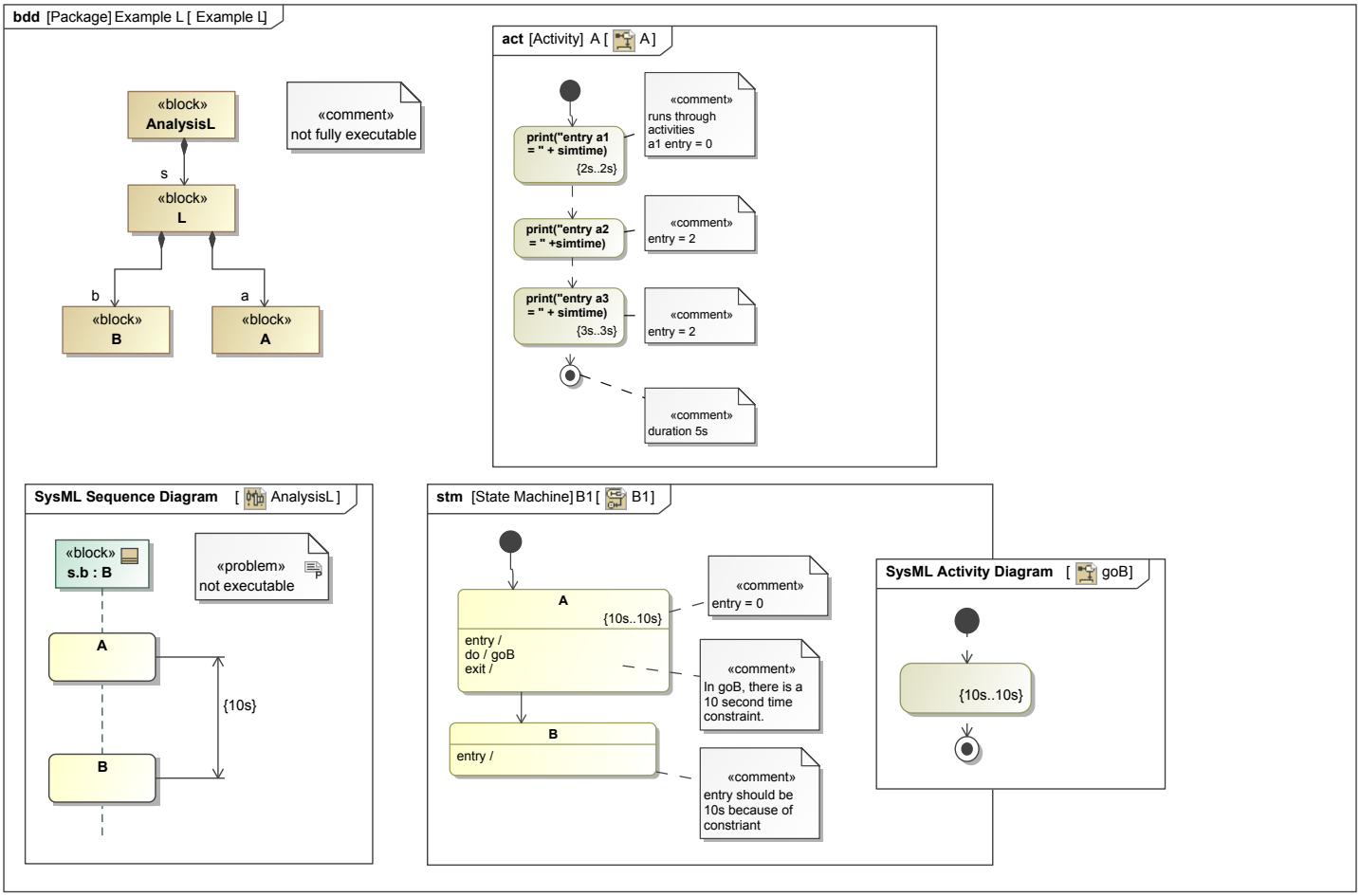


Figure 32. Example L

The state machine starts in state A, and state A has a 10 second constraint on it, and a do activity. The do activity has another 10 second constraint on it, which will start at t = 0 and run for ten seconds, then once the activity is done the state machine will go from state A to state B. Once in state B, that is the end of the simulation.

It is important to note that the duration constraint on the state will be met if the do activity has a less than or equal to duration constraint (or total time to run the activity). If the activity runs longer than 10 seconds then the do activity "wins" and will run for the entire duration of the activity. This order of operations is important to know when modeling time because it will dictate that if the activity is longer than the duration constraint on the state, the activity will prevail and the duration constraint on the state is ignored.

The activity diagram in this example starts at t = 0 and spends 2 seconds in a1, then enters into activity a2 at t = 0 seconds, and immediately into ac3. Once in ac3, the activity will obey the 3 seconds time constraint and exit ac3 with a final time of 5 seconds.

The sequence diagram should start in state invariant A and then move down the lifeline to state B after 10 seconds in unison with the state machine. When run separately from the state machine, the lifeline follows the expected behavior, but in the same analysis context

of the state machine, there is an error due to the extra steps involved in the state machine (specifically the activity in the state A). The result can be seen in the image below.

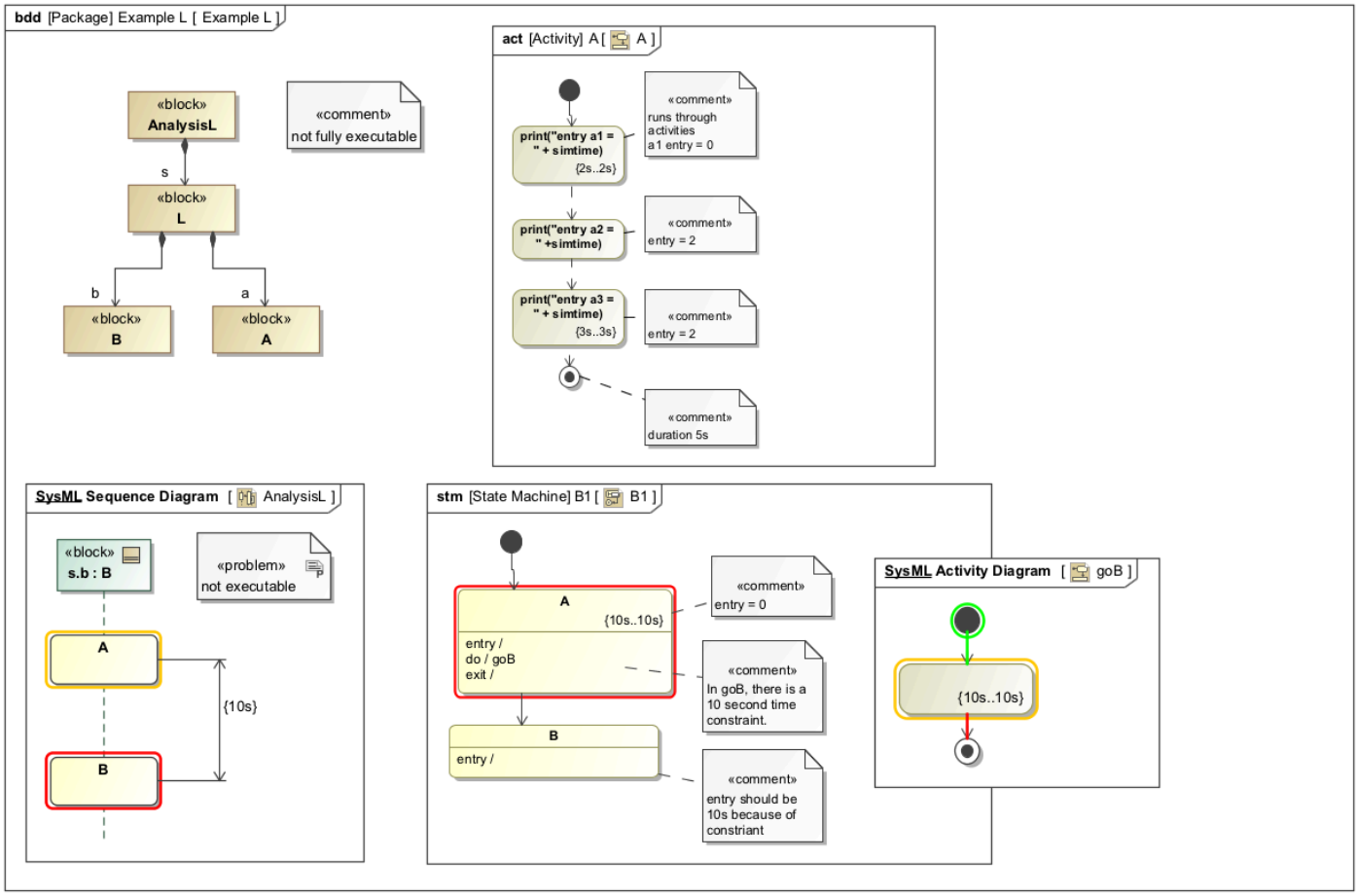


Figure 33. Example L - error in Sim

As shown in the image above the state machine is at 10 seconds in simulation time. The duration constraint on the sequence diagram has been met and the do activity in state A has also been completed. However, there is an error because the nested activity in the state sets off the parallel nature of the sequence and state machine diagrams. Since there are more steps involved in the activity portion, there is a disconnect between the sequence and state machine diagrams. As shown, the lifeline on the sequence diagram is already in state B, but the activity is has not made it to the final node yet. This discrepancy creates an error in the Cameo Simulation Toolkit Console, saying that the actual state is state A, but the state invariant says the state should be state B. When the behavioral diagrams are run not in the same context analysis, there is no error and they would run as expected. The activity does not have any impact on the other behavior diagrams.

3.6.13 Example M

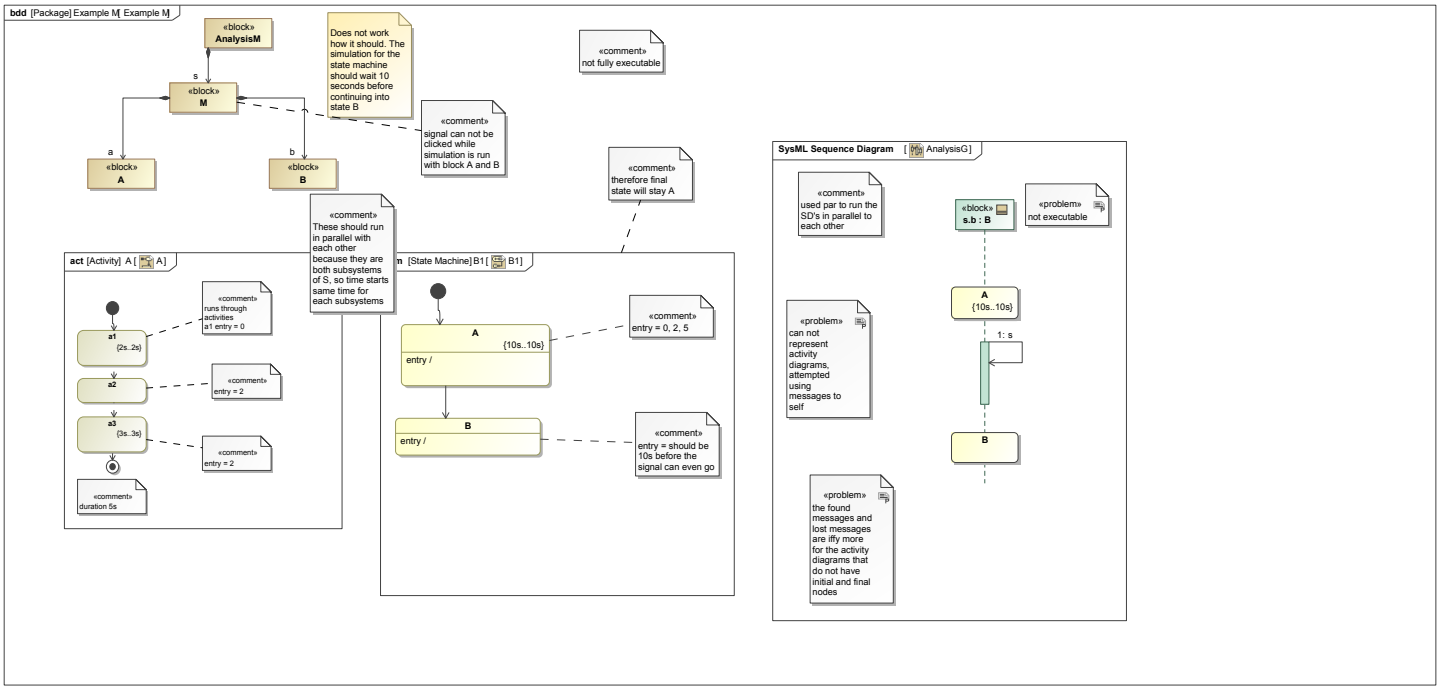


Figure 34. Example M

In this example, there is a main system, block M, and two subsystems, block A and block B. In block A, there is an activity diagram owned by it. In block B, there is a state machine owned by B. When running the simulation the two blocks must run in parallel to each other. The simulation will run as such:

When the simulation starts, the system is the first element that then executes to the subsystem. The activity and the state machine start at the same time entering into the activity, a1, and state A. Activity, a1, has a duration constraint of two seconds before continuing into a2, which then enters into a3. The activity a3 has a three second duration constraint. The state A has a ten second duration constraint before going into state B. Therefore, after ten seconds the execution of the transition will happen and go into state B. This is happening as the activity diagram is running. Therefore, the simulation time at the end should be ten seconds, the state machine should end in the state B and the activity diagram should have already run.

The SD does not support an activity diagram, therefore in the sequence diagram all we see is the state machine. The state machine runs from A to B with a duration constraint of ten seconds. It waits in A for ten seconds before sending the signal to B.

3.6.14 WatchDog Timer Examples

In the following examples, there are four different situations of watchdog timers. The watchdog timers are timers that automatically trigger the system to reset if the system does not periodically use it. The timer is an automatic reset for a system when the system fails inside a certain function. If a function takes longer then the watch dog timer, the system will be reset. In the following executable examples, there are different variations of a watch dog timer that can be created in a model.

3.6.14.1 Example A

Watchdog timers are extremely useful in modeling time, because a watchdog timer allows for the system to fail if the timing goes too long for a system. In this example, the goal is to make it to state E before the timer runs out. If the simulation time, which is counted in the initial states do behavior, is greater than 240 seconds, then the whole state machine will fail.

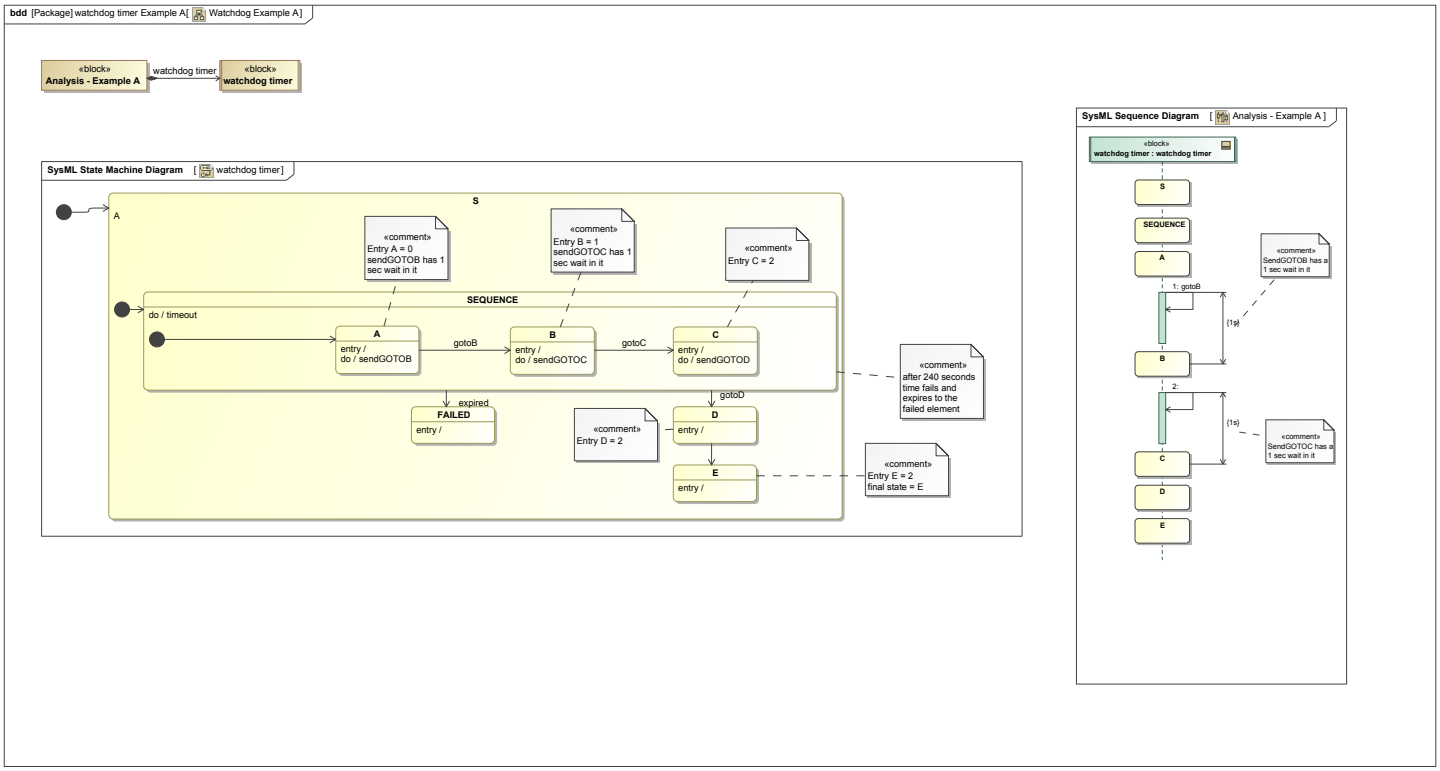


Figure 35. Watchdog Example A

The state machine in this example starts in state S, Sequence, and A all while the simulation time = 0, then in state A the do activity is to wait one second and then send the signal sendGOTOB. When the simulation time is 1 second, the state machine enters state B, and the do behavior begins, which is to wait another second and then send the signalGOTOC. The sent signals match up with the triggers on the transitions that allow the state machine to change states. From state C, the state machine should transfer from state C, to state D, and then to state E all in the same simulation time.

If the overall runtime was to be greater then the time set on the watchdog, the signal Expired would be sent to Sequence, which sends the state machine into a failed state. In this case, the watchdog timer is not used, but is a good option to have in order to verify timing requirements.

The sequence diagram in this example shows the flow of state machine. Since there is no duration constraint on the first three state invariants, it is assumed that they are all entered in order but in the same simulation time. Then, the signal sendGOTOB is sent, to self, which allows for the transition to state C. The same process is then repeated for C to D. Then state E is entered immediately after state D.

3.6.14.2 Example B

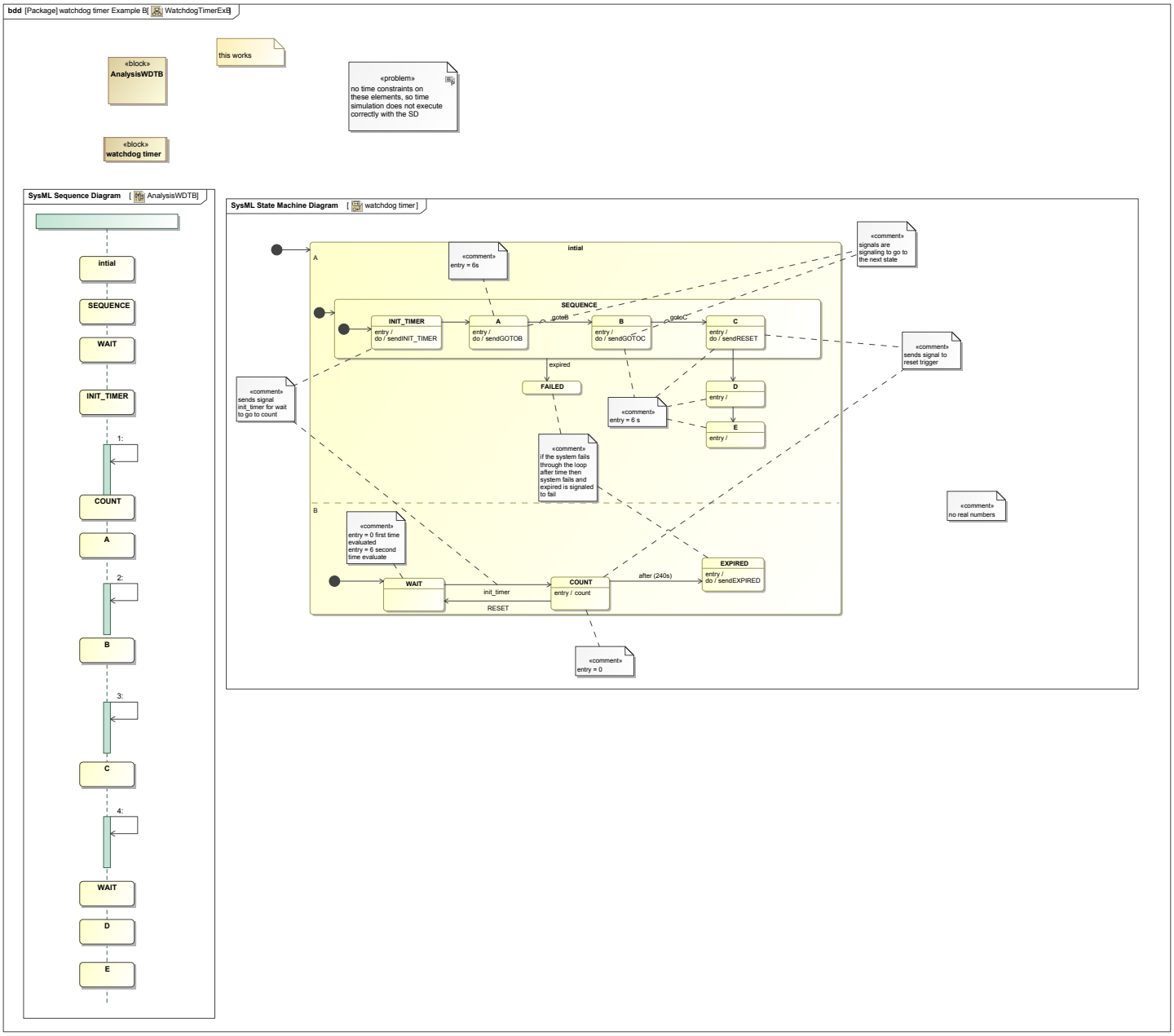


Figure 36. WatchdogTimerExB

In this example, there is a watchdog timer with a state machine that has two regions. There are two regions because there are two different sequences that run in relation to one another, they will also run in parallel to each other. The signals are easily sent through the two regions. The simulation should run as such:

Starting into the initial state, then entering into the sequence state in region A and the wait state in region B. In region B, the wait state is waiting for a signal to be sent to continue onto the next state, therefore it is idle. Then in region A, the simulation continues into init_timer state, that contains a do behavior, which owns an activity that will send a signal to region B to the wait state. Then the since the signal was sent the transition from wait to count will be executed. Now there is a timer running with count for 241 seconds. If after 241 seconds, the signal to reset is not given then the time will expire and the system will fail. But the simulation continues in region A, continuing onto state A with a do behavior signaling for the transition to state B, then state B's do behavior signaling for the transition into state C. At state C, there is a do behavior that then signals the reset signal for the region in B. For region B, the transition between count to wait is executed. Then region A finished out the simulation run by going into states D and states E. Note that in the do behaviors with the signals there was a duration constraint of 0 seconds, which is not necessarily needed if assuming it takes not time to go to the next state or send the next signal.

The sequence diagram represents the watchdog timer block. As the watch dog timer simulation runs, the simulation goes through the state machine. The sequence diagram represents that state machine as it runs through the simulation. The time duration and the time

constraints seen on the SD give a better description of how the state machine should should run. The SD is owned by the analysis block. The sequence diagram runs as such:

Starting in initial state invariant, moving to sequence, and then wait. Then going to init_timer and sending the signal init_timer which goes to count. Then entering into state invariant A, which sends the signal gotoB. Entering state invariant B, which sends the signal gotoC. Then entering into C, which sends the signal to reset, then going into wait state invariant. Finally, continuing into D and E.

This diagram shows how and where the simulation should go as time continues, which can also provide the best understanding for how the state machine should run.

3.6.14.3 STM Example A

This watchdog example is supposed to show how the watchdog actually sends the failed/expired signal when the duration constraint is not met. The original goal of the watchdog is to make it through the sequence in under 5 seconds, but the do activity in state A has a 10 seconds duration constraint inside of it, so the watchdog will go over the max of 5 seconds and send the expired signal to the sequence. If the do activity was anything less than five seconds, the sequence will make it all the way to the final state E. If the duration constraint is exactly five seconds, then the final state is failed, but the sequence reaches state C.

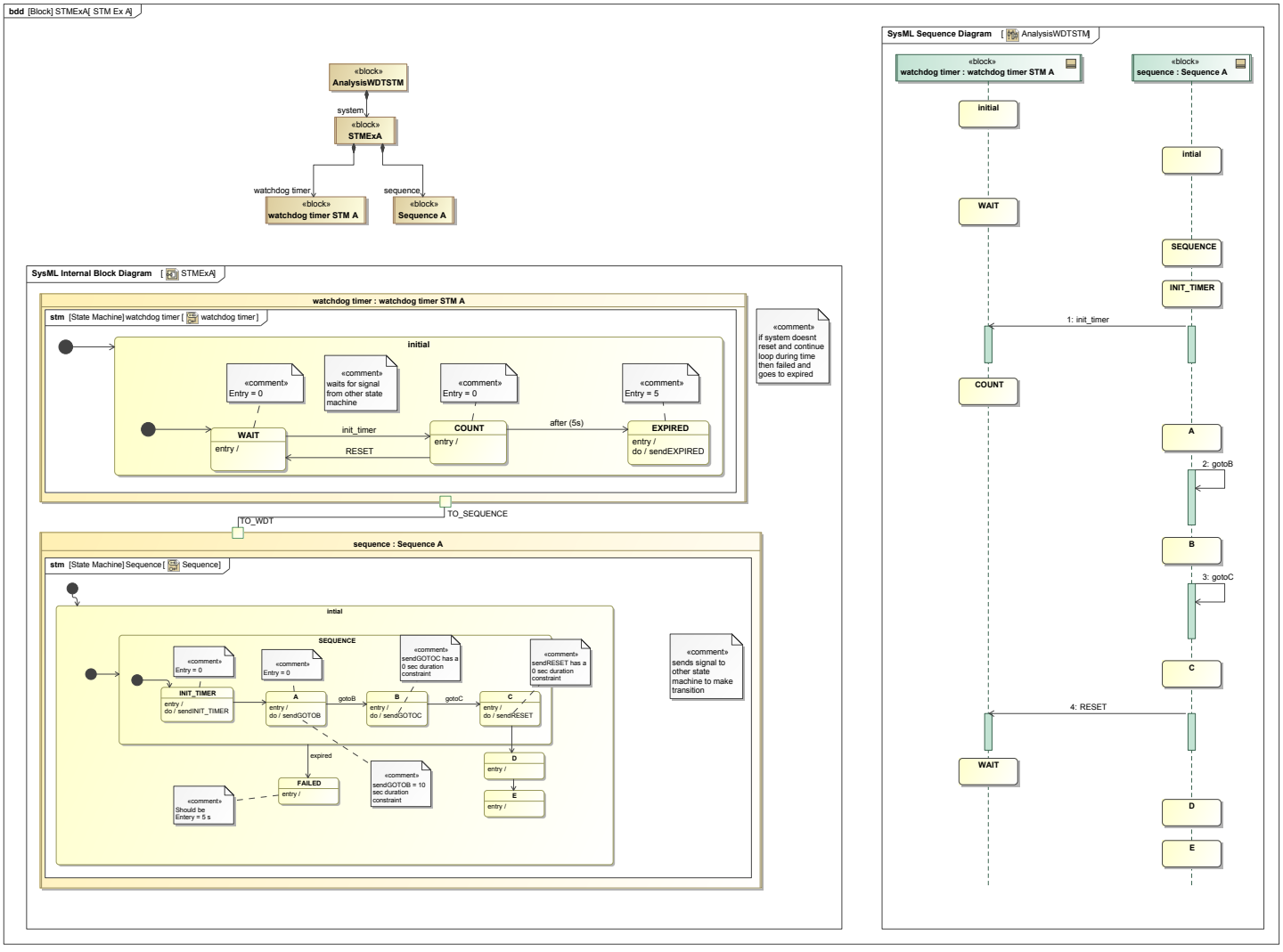


Figure 37. STM Ex A

Both of the sequence and watchdog timers are in the same analysis context so they both run in parallel. When t = 0, they both enter their initial state and then enter to their next states. In the sequence state machine, the state Init_Timer sends a signal to the watchdog which flows from ports on the Sequence and watchdog timer blocks over a connector. In the watchdog timer the wait state receives the signal sendInitTimer which sends the watchdog block into the count state. Meanwhile in the sequence block, the state machine is in state A, and has begun the do activity which waits for 10 seconds. Since the watchdog is only made to wait for 5 seconds, after 5 seconds have passed the count state transitions to the Expired state, which then sends the Expired signal through the ports and across the connector to sequence, which then transitions into the failed state.

The sequence diagram is a model for if the entire duration was under 5 seconds and the expired signal would not be sent. The sequence diagram follows the ideal flow for if the model was running as expected and would not time out.

3.6.14.4 STM Example B

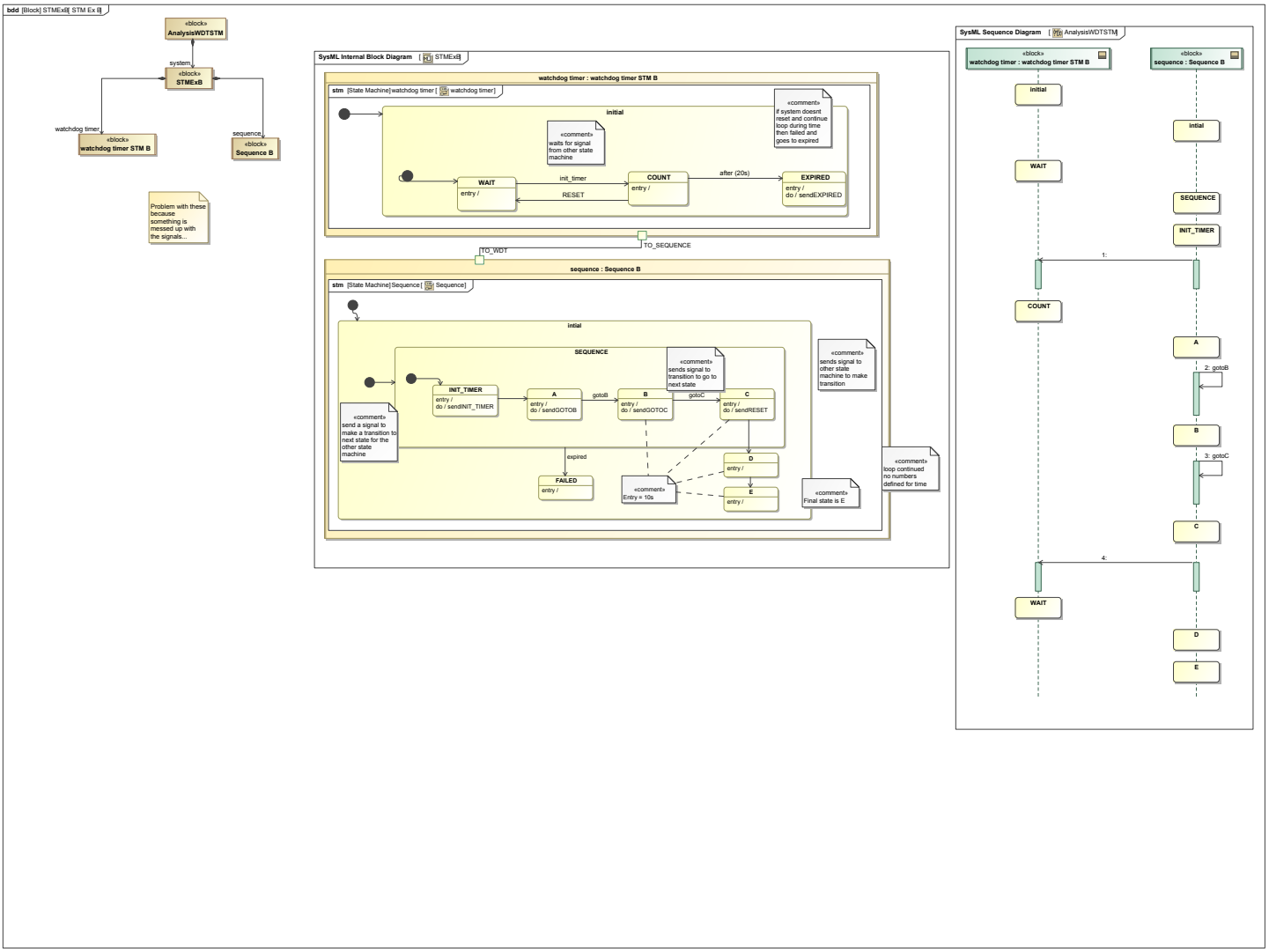


Figure 38. STM Ex B

In this simulation, there is another example of a watch dog timer, but in a different variation. On this example, there are two blocks owned by a single block, which can be considered the main system. Each block has its own state machine and communicate through a ports and connectors in an internal block diagram. The simulation runs as such:

In the beginning, the initial state is the first state entered in both of the state machines. The simulations will run in parallel because of the relationship to the whole system block, STMExB. Then the state machine owned by watchdog timer STM B will enter into the wait state, waiting for a signal to be sent to continue, therefore it is idle. Then in the sequence state machine, the simulation enters into the sequence state and then into the init_timer state. In the init_timer state, there is a do behavior that send the signal init_timer to the watchdog timer STM B state machine. The message passes through the ports and connectors to then signal the transition between state wait and state count. Once the simulation is in the state count the timer starts to count for 20 seconds. In the sequence state machine, the simulation continues into the state A that has a do behavior with an action that has a duration constraint of 10 seconds and a send signal gotoB. After 10 seconds, the signal gets sent and goes into the state B. Then entering into the do behavior of state B, there is a call signal that signals to transition to state D, then transition into state E.

3.6.15 Parallel Region Example

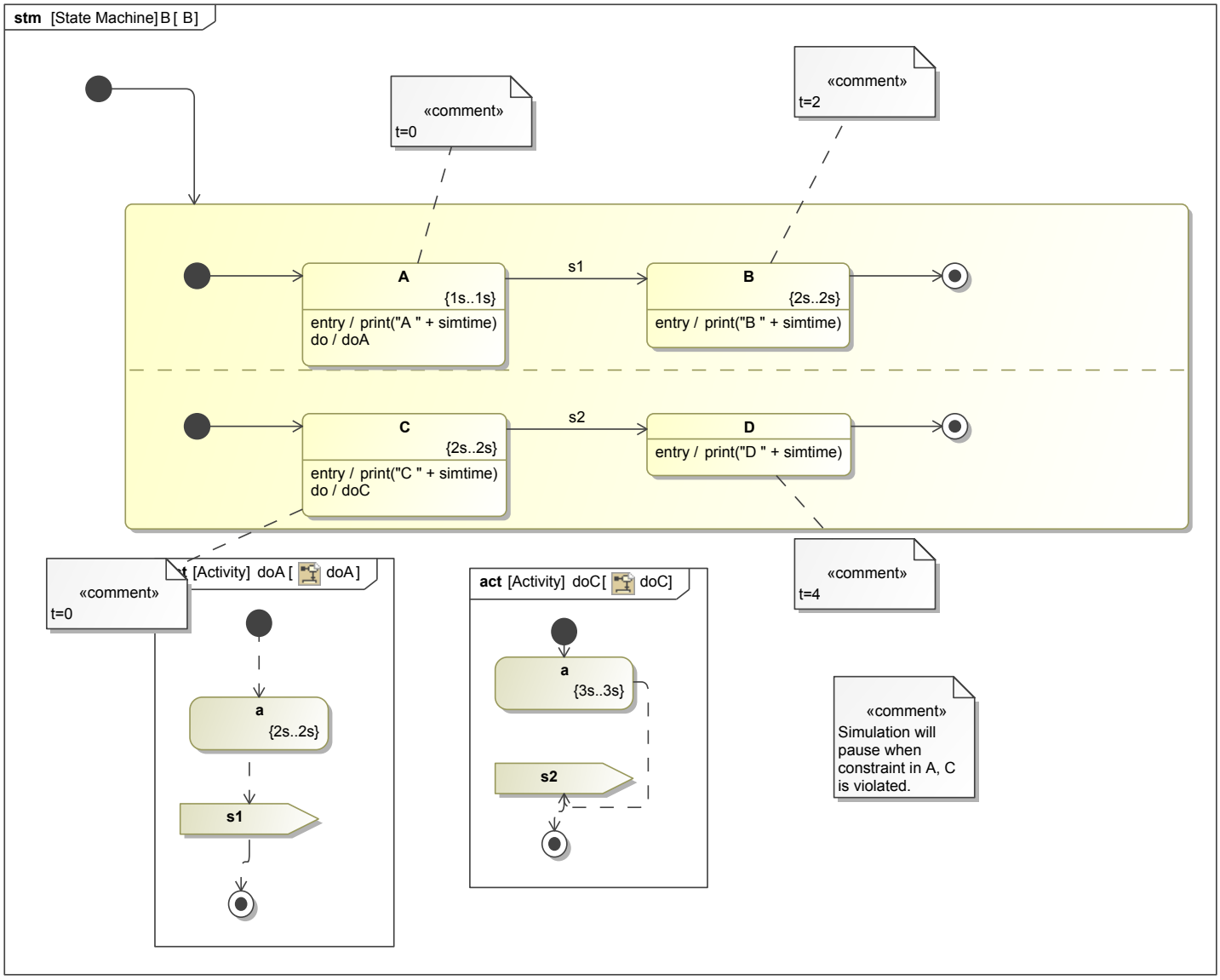


Figure 39. B

The duration constraints will effect the model as such:

Entering into the superstate, then into the two parallel regions. In the the parallel regions there are nested states, A and C. State A has a duration constraint and a do behavior activity. The the constraints are not violated, but the state has a constraint of one, while the behavior has a constraint of two seconds. Therefore state A will run for two seconds and then second the signal s1. Then there will be a transition into state B with signal, s1. Then state B has a duration constraint of two seconds, therefore will be in the state for at least two seconds transitioning into the final node. While this is happening, the second region is also active. State C also has a duration constraint and a do behavior. The constraint on the state is two seconds, while the nested constraint is three seconds. So the state will run for three seconds and then send the signal s2 to transition into state D, then to the final node.

3.6.16 Example N

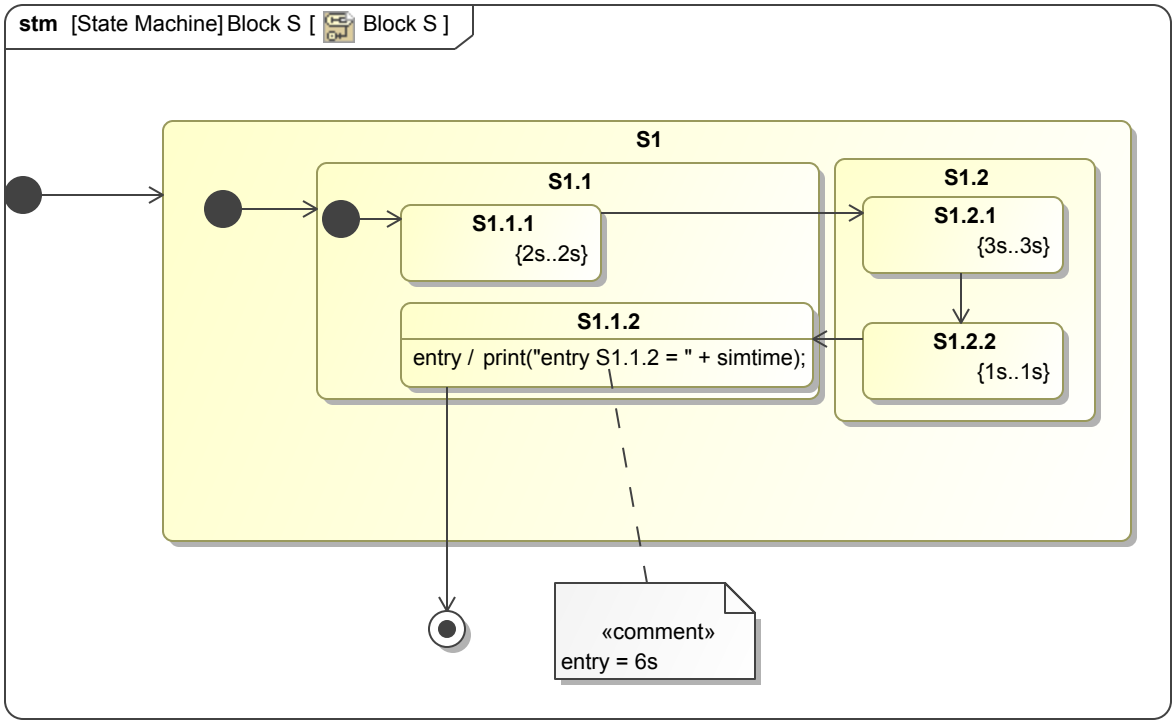


Figure 40. Block S

The state machine runs as follows: Enters into S1, S1.1, and S1.1.1 when t = 0, then there is the 2 second constraint on the state S1.1.1, which then transitions into state S1.2.1 where the simulation is then frozen. This freeze is due to a null transition out of the state. It is a good modeling practice to have a signal, guard, or effect on transitions when modeling state machines.

3.6.17 Time Envelope Example

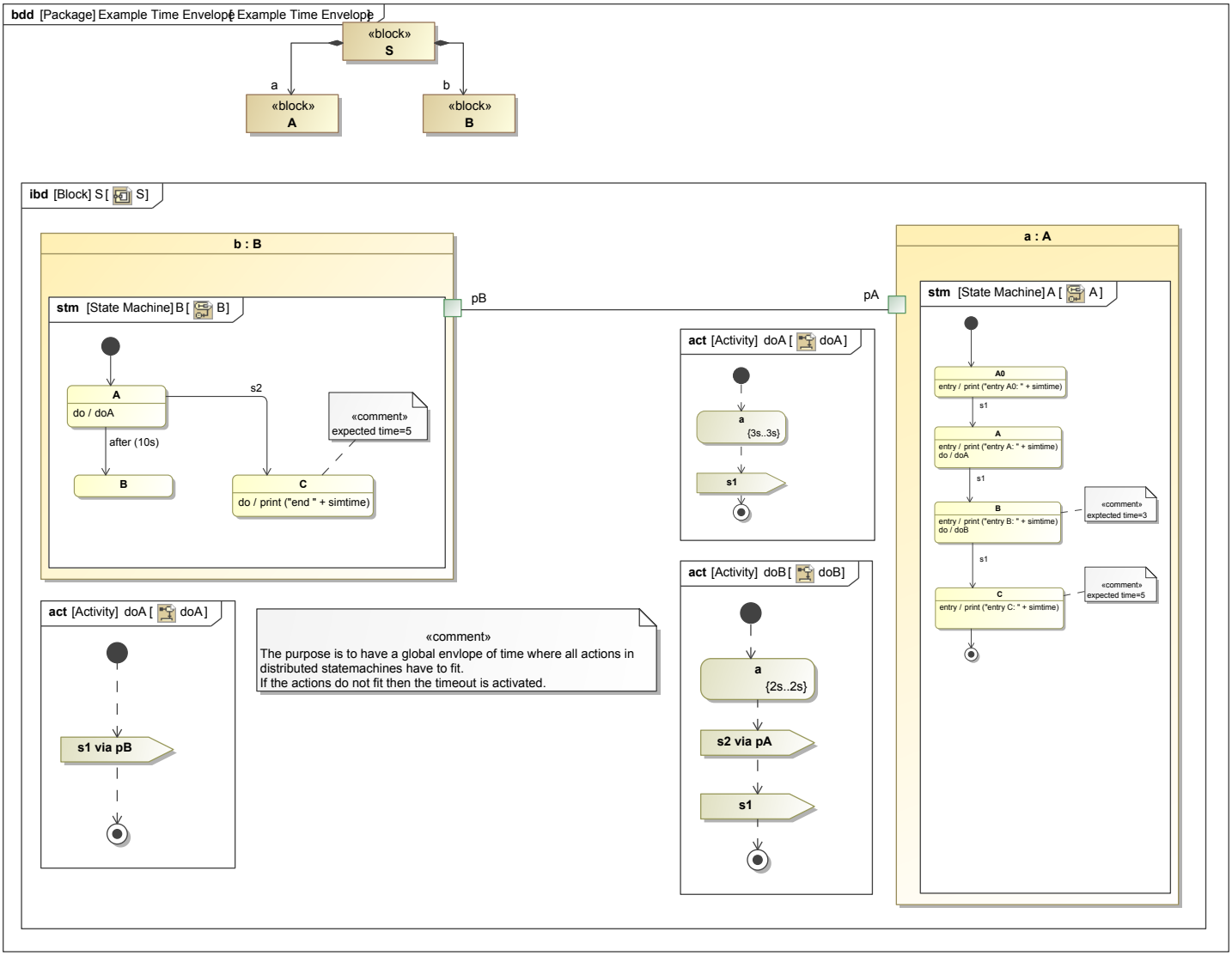


Figure 41. Example Time Envelope

3.6.18 Example O

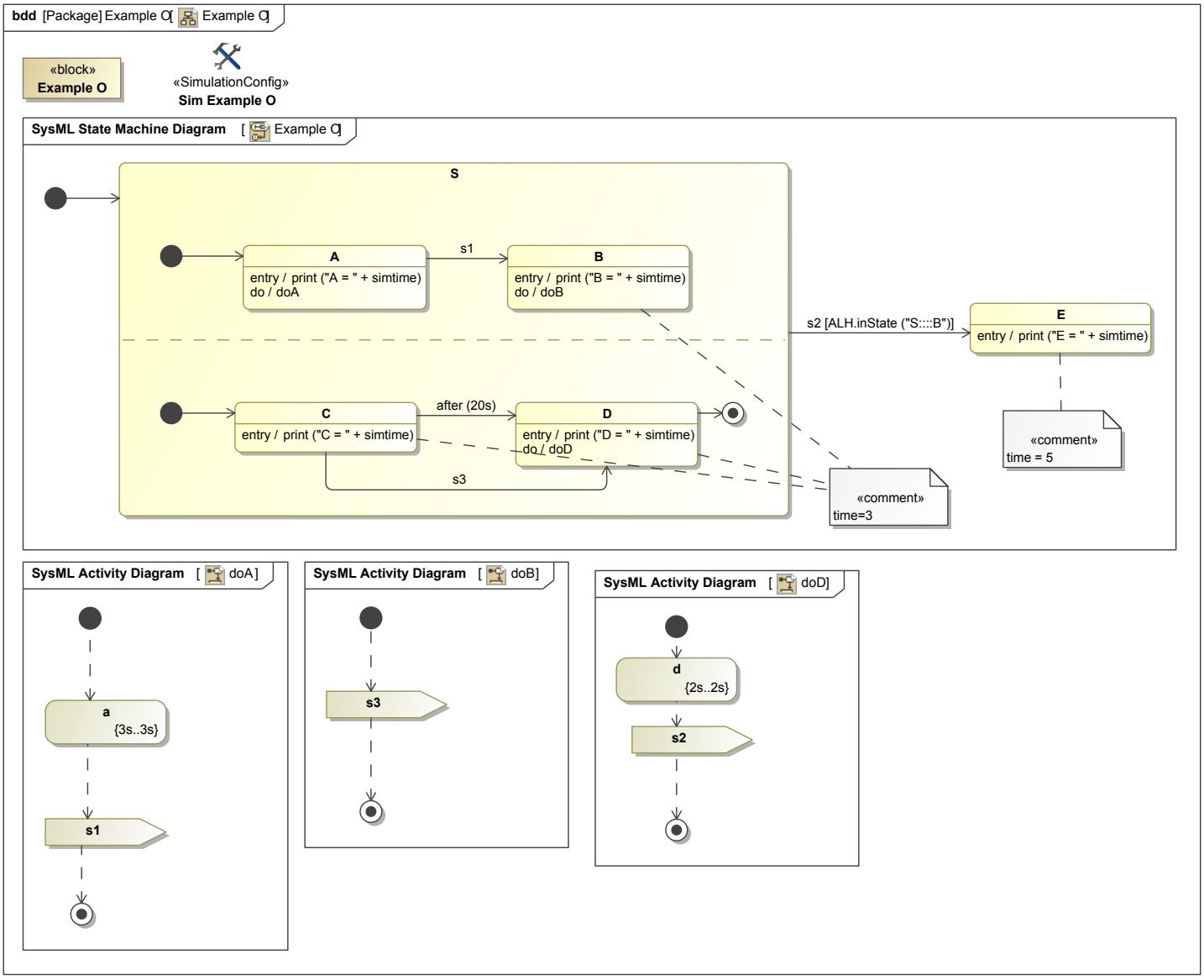


Figure 42. Example O

3.6.19 Threaded watchdog with regions

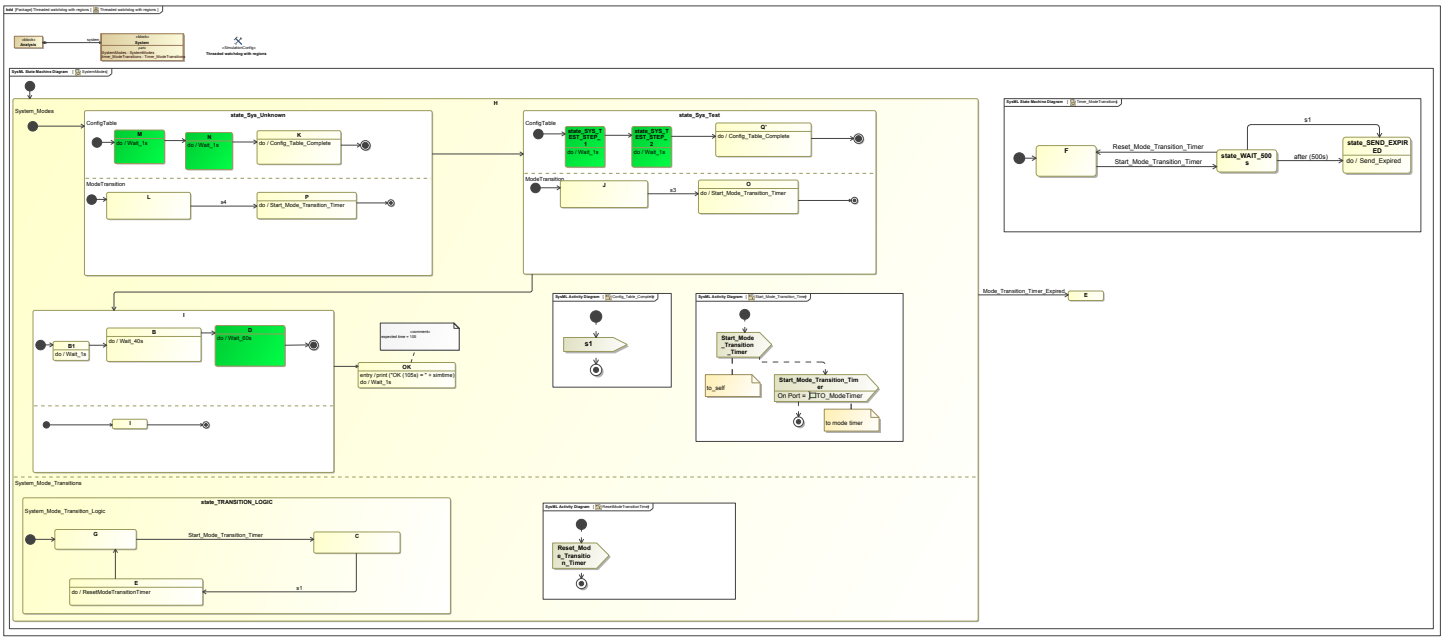


Figure 43. Threaded watchdog with regions

3.6.20 State Invariant Timing Example B

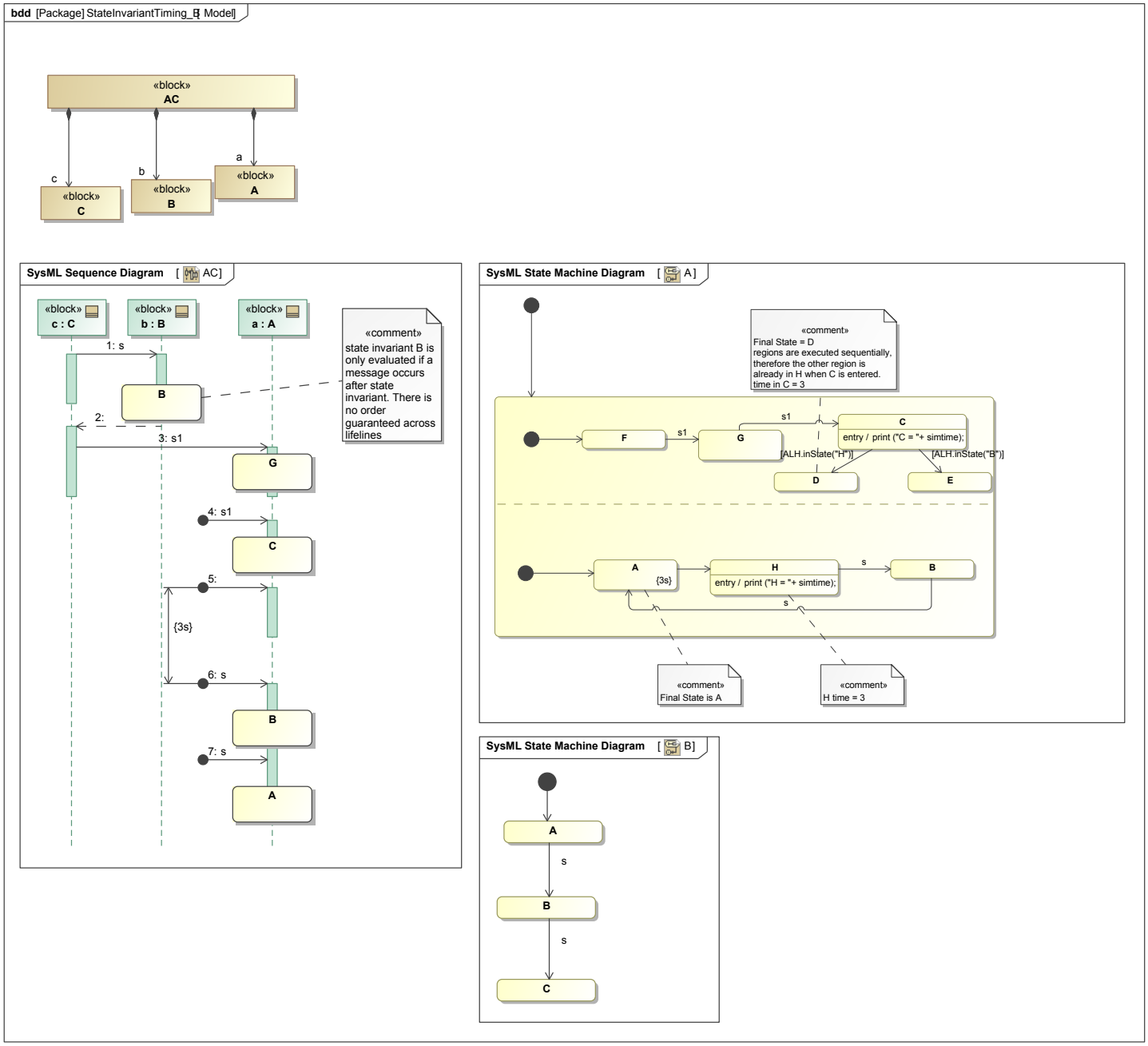


Figure 44. Model

3.6.21 State Invariant Timing Example A

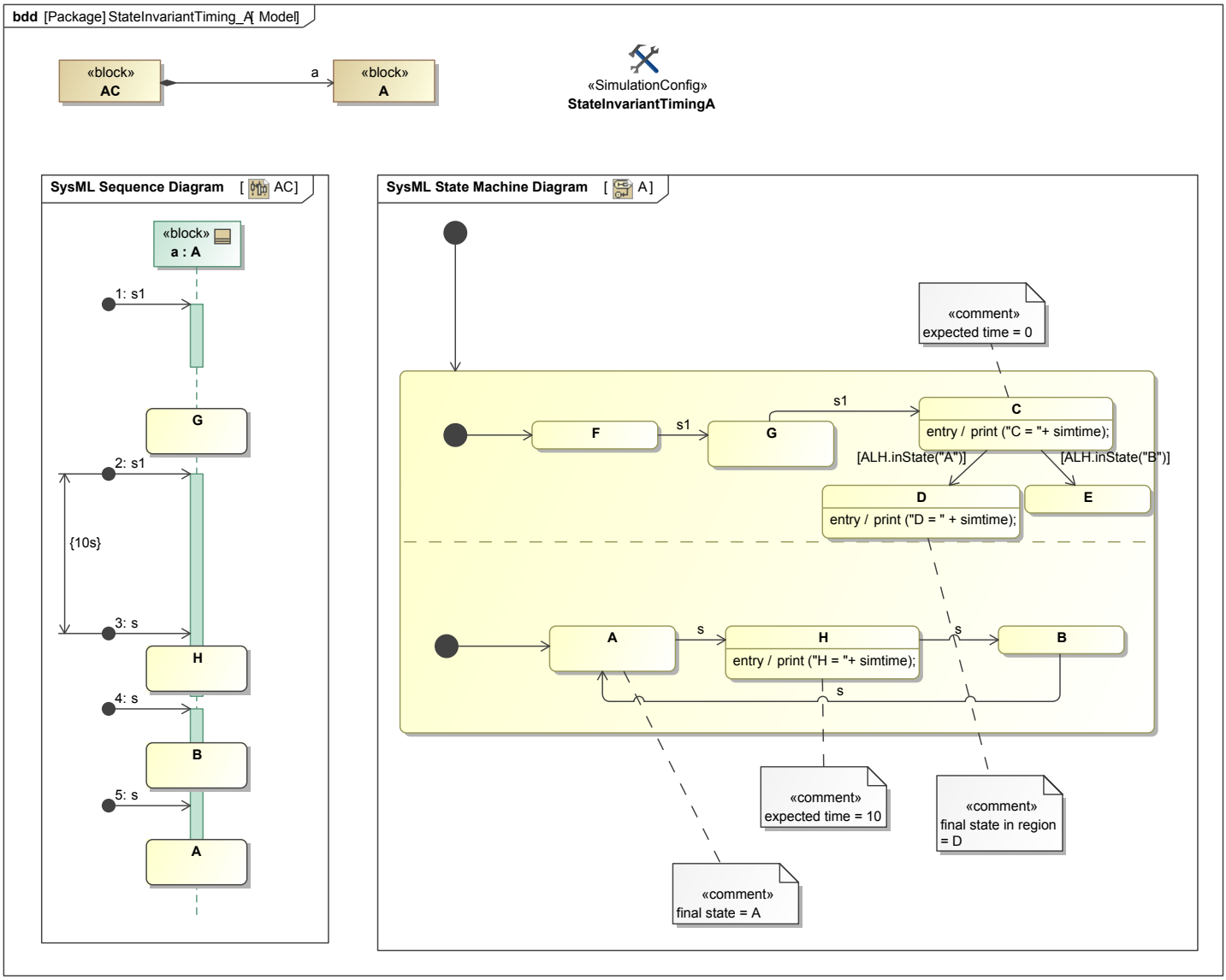


Figure 45. Model

3.7 Known Uses

Examples of real usages of the pattern.

3.8 Analysis

Analysis discusses how the time modeling pattern helps to analyze a system model.

The time modeling pattern can be broken down in to the behavioral diagrams, which can then be broken down into their owned elements. A top level analysis compares how the diagrams interact with one another, while a more in depth analysis would evaluate how the owned elements interact with each other and even other diagrams. In the time modeling pattern the analysis focuses on how the time is used amongst the elements and diagrams.

The main tool used analyze the time in the behavioral diagrams is the Cameo Simulation Toolkit, more information on the features provided by the tool, see the Tooling section.

The time modeling pattern is structured in a way that allows the sequence diagram to analyze the behavior of the state machine. The sequence diagram is created in order to verify the behavior of the state machine. Constraints on a sequence diagram, such as duration constraints, are implemented on sequence diagrams to better understand how the timing in a state machines is expected to work. The sequence diagrams and state machines are run in the same analysis context, so they are run in parallel which allows for the sequence diagrams to be directly compared to the timing of the state machines.

3.9 Related Patterns

3.10 Tooling

The execution of the models was performed using the Cameo Simulation Toolkit (CST), which is a plugin to MagicDraw enabling model execution for early system behavior simulation. CST analyzed the time modeling through executable models.

To create simulations, a simulation configuration is required. The simulation configuration is instantly created when a simulation configuration diagram is created. The simulation configuration defines how the simulation runs.

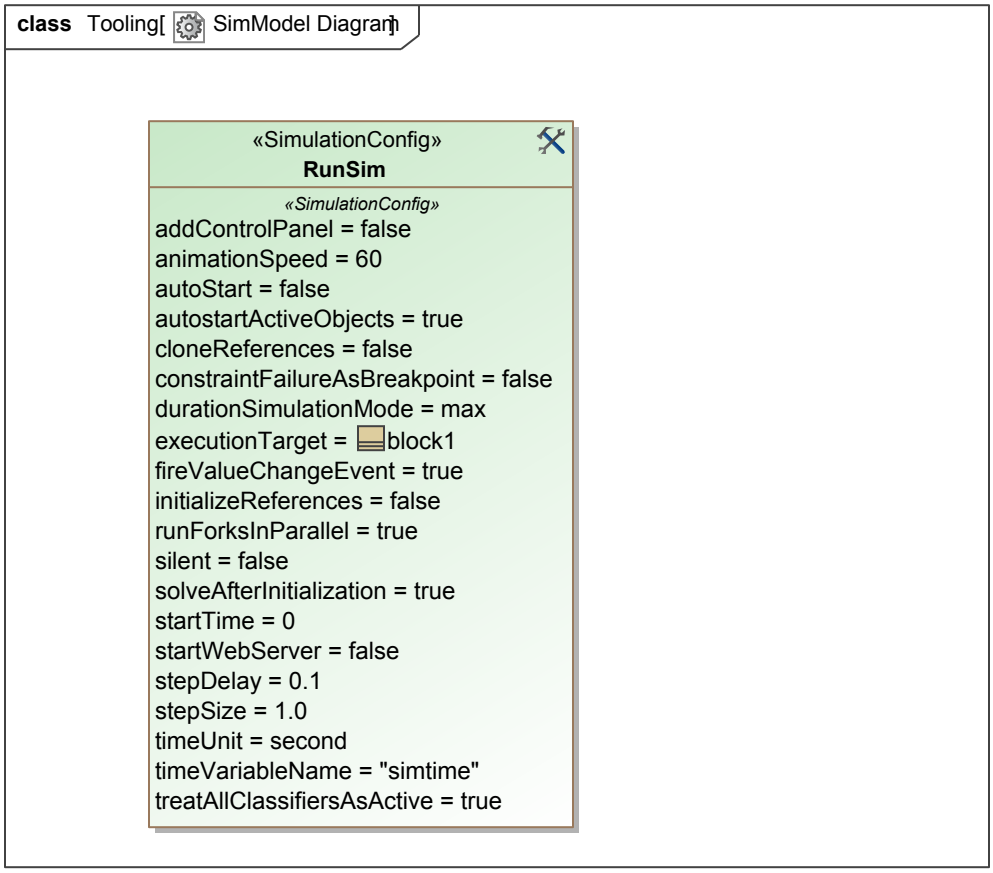


Figure 46. SimModel Diagram

Fig. 1 shows the simulation configuration element that was a general use in all the simulation examples. There are a few important tags that need to be created. The timing of the simulation must be created based on "real" time.

As you see in the image above, there is a "startTime", "stepDelay", "stepSize", and "timeUnit". The "startTime" should be set to 0, unless otherwise required. The "stepDelay" is set to 0.1, where each step of time will be delayed 0.1 moments for the simulation. The "stepSize" increases the simulation time. The "timeUnit" defines the unit of time, such as seconds, milliseconds, etc. These tags create a global clock that does not differ compared to a computer based clock that varies with every use. Another important tool that can be utilized in this simulation configuration is the "timeVariableName". The tag defines a name for the simulation time, which can be used in user scripts. The user script can do many things, such as print the time an element is entered, exited, etc. in the console window.

The "durationSimulationMode" defines the duration or time constraints the simulation will use. For any element, there are options to place constraints on time for when the element starts or how long the simulation is in the element. The "durationSimulationMode" tag defines if it should use maximum (max), minimum (min), average, or random. Depending on the duration constraints, the simulation will run differently or the same.

The execution target is the most important tag because it will run the simulation you specify. Most diagrams are owned by another element, in the examples there are state machines owned by blocks. The blocks are then selected in the execution target for the simulation configuration to know that the simulation should be run with this block.

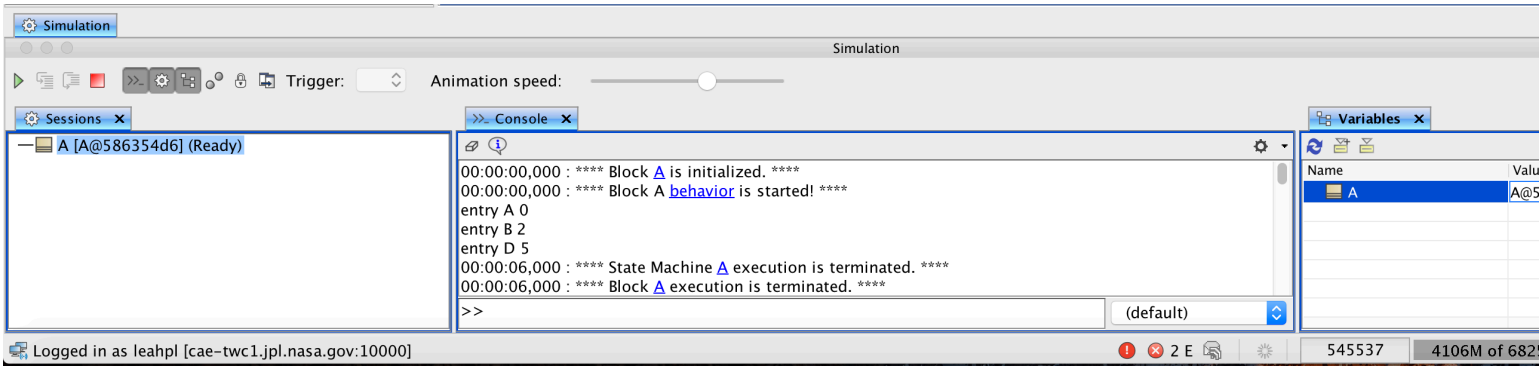


Figure 47. Fig. 2 Simulation Window in CST This image is an example of the pop-up window that appears when simulating wit

The simulation window, in Fig. 2, will appear when right clicking the simulation configuration element, go to simulation, then run. If the simulation configuration has auto-start on, then the simulation will instantly run. The window will pop-up to running the simulation on the bottom of the screen. A session window, a console window, and a variables window will all appear. Before the simulation is executed, the simulation clock can be shown by right clicking the block under sessions and clicking the "show simulation clock". CST can also create their own Sequence Diagram when running the simulation with right clicking the block under variables and clicking "Create Sequence Diagram". Another form of this can be done for every simulation ever run with Sequence Diagram(SD) Generator Configuration, an element can be found in Simulation Configuration Diagram to generate a SD every time. In the console window, the simulation will state errors that happen during the simulation and executions that happened including the time it took. In the examples, the console will also have printed summaries of the simulation time as it enters through every state or activity.

For more information about Cameo Simulation Tool Kit refer to the User Guide from No Magic.

https://docs.nomagic.com/display/CST185/User+Guide

4 Roll-up Patterns

Many common analyses can be modeled purely in SysML using its parametric and behavioral semantics and modeling patterns, and can be applied to systems that involve structural, behavioral and parametric diagrams. Universal and reusable behavior patterns allow system modelers a method to apply standard constructs to their domain specific system.

4.1 Static Roll-up Patterns

4.1.1 Simple Static Roll-up Pattern

Table 2. <>

Model Element	Role
Anti-pattern	

4.1.1.1 Intent

Roll-up calculations of system resources (e.g. total mass, cost, power, or another system dimension) are among the most common use cases in systems engineering. System engineers want to perform calculations based on the specific values of all the components in the system. The Simple Static Roll-up pattern provides system modelers a solution to solve common domain issues, and consists of standard constructs that are applicable to most domains. If the modeler desires to perform analysis of a system where the values of the subsystem are independent of the system's behavior the Simple Static Roll-up pattern should be reused and applied in their domain specific construct.

4.1.1.2 Motivation

The overall motivation for system engineers to apply the pattern is the desire to have requirements that are traced to to design artifacts, and to have a method to assert that a system design satisfies a set of requirements. Through SysML patterns, analysis specifications can be defined and the modeler can verify, for instance, whether the resource consumption of a system satisfies an initial requirement. If a system modeler wants to perform analysis of a complex system that consists of components whose value properties are state independent, and the modeler is familiarized with running simulations, the Simple Static Roll-up Pattern can be configured to their system.

The Simple Static Roll-up supports the analysis of a system design which is independent of the component's operational states. The Simple Static Roll-up pattern can be applied to a model in which the product breakdown structure of the system is specified.

The components aggregated in the roll-up must share a common value property that does not change (static) throughout the component's life-cycle. Through the application of the pattern, each component's static values can be recursively propagated up a hierarchy of components characterized by these values.

To apply the pattern to a system, the modeler needs to augment the components of the system design for a particular resource in order to propagate the values up the system hierarchy. The pattern must be configured to the system to specify the analysis context using the system's design, states, properties, and relationships. Only then can the pattern provide a context as to whether the design satisfies a set of formalized requirements.

4.1.1.3 Concept

The Simple Static Roll-up pattern consists of the following model elements; the Simple Static Resource Roll-up Aspect block, the Simple Static Resource Roll-up Aspect parametric diagram, and the static values of the components in a system that are to be summed.

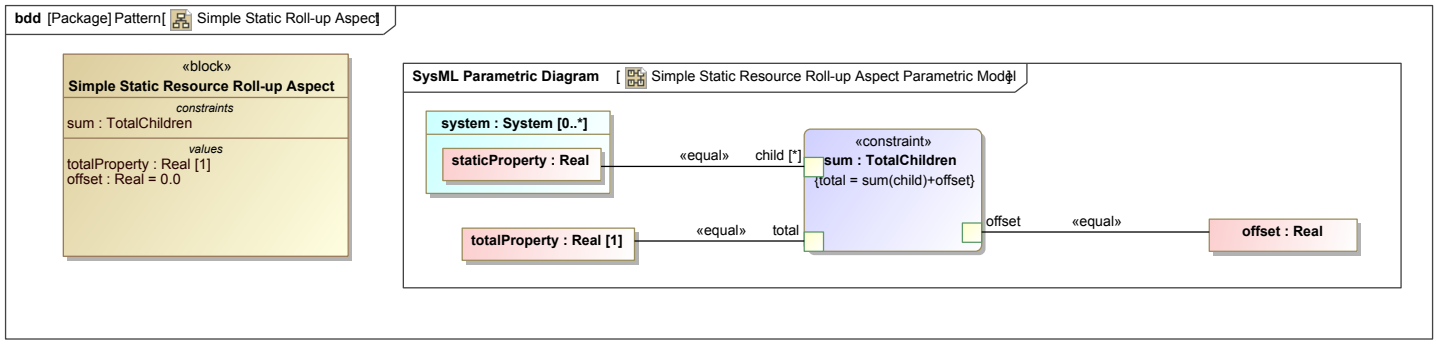


Figure 48. Simple Static Roll-up Aspect

One method to propagate the static values of a system is by applying an aspect block where a constraint is applied, and the total value (totalProperty) of the static property (staticProperty) is calculated. The aspect block owns a parametric model where the static property of the system is constrained. In the parametric model the owner's constraint, sum, is constrained to the expression total = sum(child)+offset. For each component that is a part of the Recursive Static Resource Roll-up Aspect block, the static and the total property will be constrained. The parameter, child, is equal to the staticProperty of a top level hierarchical component, and is denoted with the multiplicity * to represent that there can be an infinite amount of components. The parameter offset is equal to the aspect's value property offset property. The constraint's parameter, total, which calculates the sum of the children is equal to the aspect's value property totalProperty.

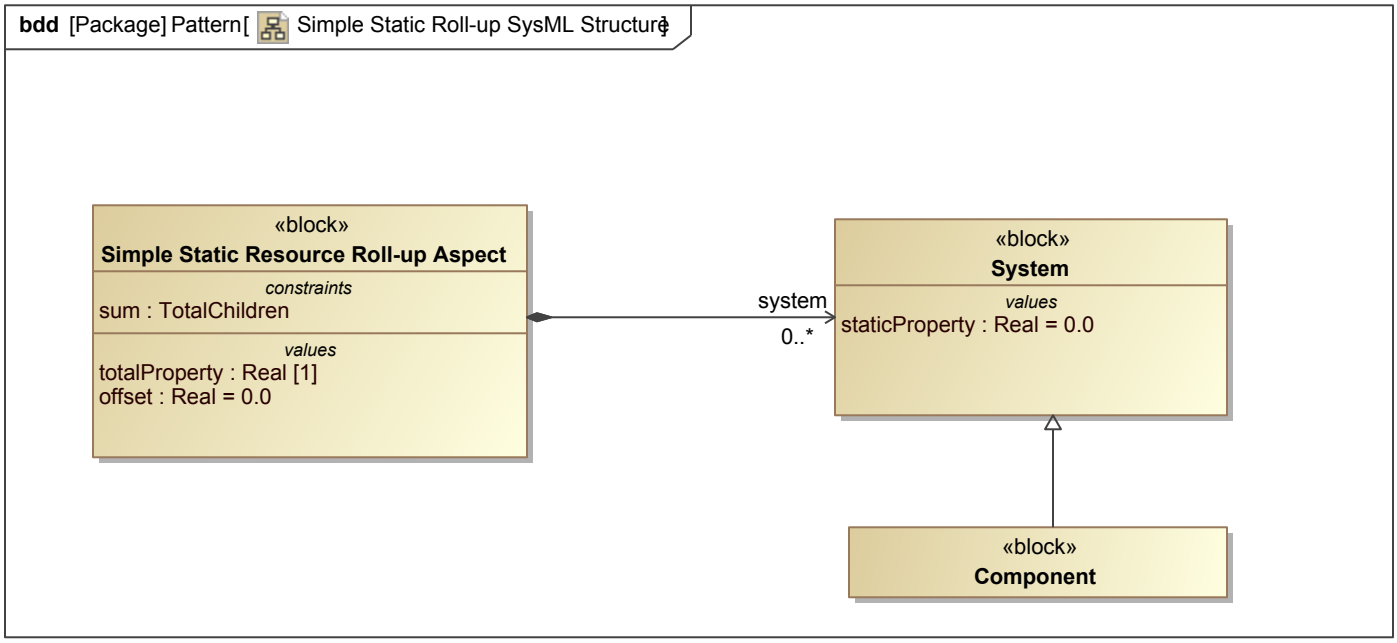


Figure 49. Simple Static Roll-up SysML Structure

The SysML structure of the Simple Static Roll-up Pattern is dependent on the properties of the Simple Static Resource Roll-up Aspect block and the relationship between the aspect block and components in the system. The static values of a system that are to be propagated will be inherited by the top level system component (System). To calculate the values of every component in the system each block participating in the roll-up must specialize the system. The relationship between the Simple Static Resource Roll-up Aspect block and the top level system component (System) is a whole-part relationship.

Table 3. <>

Model Element	Role
Simple Static Resource Roll-up Aspect	The Simple Static Resource Roll-up Aspect contains the value properties (totalProperty, offset) that are needed to calculate the total of all the staticProperty values, the TotalChildren constraint, and a parametric model where the properties are constrained.

Model Element	Role
totalProperty	The inherited value property, totalProperty, is the sum of all the children in the system hierarchy. The totalProperty property represents the final value for the roll-up calculation of the staticProperty of the system.
offset	The value property, offset, can be an optional value to compensate for a certain quality that impacts the system.
System	The System block represents a modeler's system which will be augmented to the system's appropriate name (ex. Power Subsystem, Telescope). The System is specialized by components that will inherit the staticProperty property. The system is a part of the Simple Static Resource Roll-up Aspect.
staticProperty	The value property, staticProperty, is a property that doesn't change over time, and all components that participate in the system will inherit this value. The default value of the staticProperty needs to be initialized to zero. The name of the property should be changed to the modeler's system specific property they want to calculate.
Component	The block, Component, represents a subsystem/subcomponent of a system that will inherit the staticProperty property. The purpose of the block in the structure is just to show the relationship (generalization) that is required to follow the Simple Static Roll-up behavioral pattern. In reality, the modeler will have many components in their system that they want to calculate the total value from.

4.1.1.4 Consequences

There are several trade-offs to consider when configuring the Simple Static Roll-up pattern to the modeler's system.

Action	Consequence
Redefining the static value properties of the system's components	By redefining the static value properties of the system's components, the modeler can modify the values. If the static value of a component isn't redefined, the value will default to the value of the base class. Redefinition provides a method for specifying the component's attributes such that the modeler can view and modify the aggregated value properties of the component's generalization hierarchy. This creates a full specification of the component being the leaf element in this hierarchy.
Creating instance specifications to define the values of the system's components	By creating instance specifications to define the initial values of the system, the modeler is able to track different configurations. New configurations for a scenario can be easily created and viewed in an instance table. The value properties are visible in the block specification.
Modifying the system model when applying the roll-up pattern to the system	A consequence of needing to modify the system design results in an authority/ownership problem. An example of this occurs when an authority owns and modifies components of the system design and system decomposition. If another authority requires to perform a roll-up analysis of this design, they will need to augment the system design in order to define the relation between the system properties, e.g. the mass of one component is the sum of the mass of its children. A possible SysML work around is to use multi-classification at the instance specification level i.e. the instance specification is specified by the component classifier and the roll-up aspect classifier. The drawback to this approach is that the value properties are not visible in the block specification and only in the instance specification level.
Subsetting all composition relationships of the system's hierarchy	The modeler must subset every element in the hierarchy which can become burdensome depending on the number of components and relationships in the system. A possible solution would be to use a reasoning tool to automate this process.

Usage of the Pattern	Explanation of Conflict

Configuring the Simple Static pattern to a system that consists of part-whole relationships.	If the relationships between components in a system, where the Simple Static Resource Roll-up Aspect is applied, have a part-whole relationship the wrong roll-up pattern is being used. Instead the modeler should use Recursive Static roll-up pattern.

4.1.1.5 Implementation

An instance of how the Dynamic Roll-up Pattern can be applied, is by calculating the total data consumption of a Hybrid Sport Utility Vehicle's subsystem components (Brake Control Module, Powertrain Control Module, Body Control Module).

The first step is to specify the design of the system and its decomposition. In our example, the physical system decomposition of a satellite is modeled. The subsystems (Braking System, Power Control Unit) are composed of physical components Powertrain Control Module, Brake Control Module, and Body Control Module. After specifying the design, the next step is to modify the system with the Dynamic Roll-up analysis pattern. The analysis needs to explain (by simulation) the total data consumption of the system according to the design. When using the Dynamic Roll-up pattern, every element in the composition hierarchy whose data properties are to be summed need to generalize the Dynamic Roll-up block. The composition relationships between the physical components need to be subset with the subRollUp part property of the Dynamic Roll-up block. The relationships in the system need to be subset because the the value of the subcomponent property , totalDataRate, is computed as the union of the values of all properties that subset it. Therefore, all components participating need to have their subcomponent properties subset by this property.

4.1.1.5.1 Simple Static Volume Example

The Simple Static pattern should be used if a modeler's system is linear. The breakdown structure of the system should only consist of one component or subsystem, and is specialized by an infinite amount of subcomponents. The unique quality of the pattern is that every subcomponent of the system shares only one static property that is independent of the component's behavior. By applying the pattern to a product breakdown structure, the modeler can calculate the sum of all the static property values. In this example, the sum of all the values for volume will be calculated.

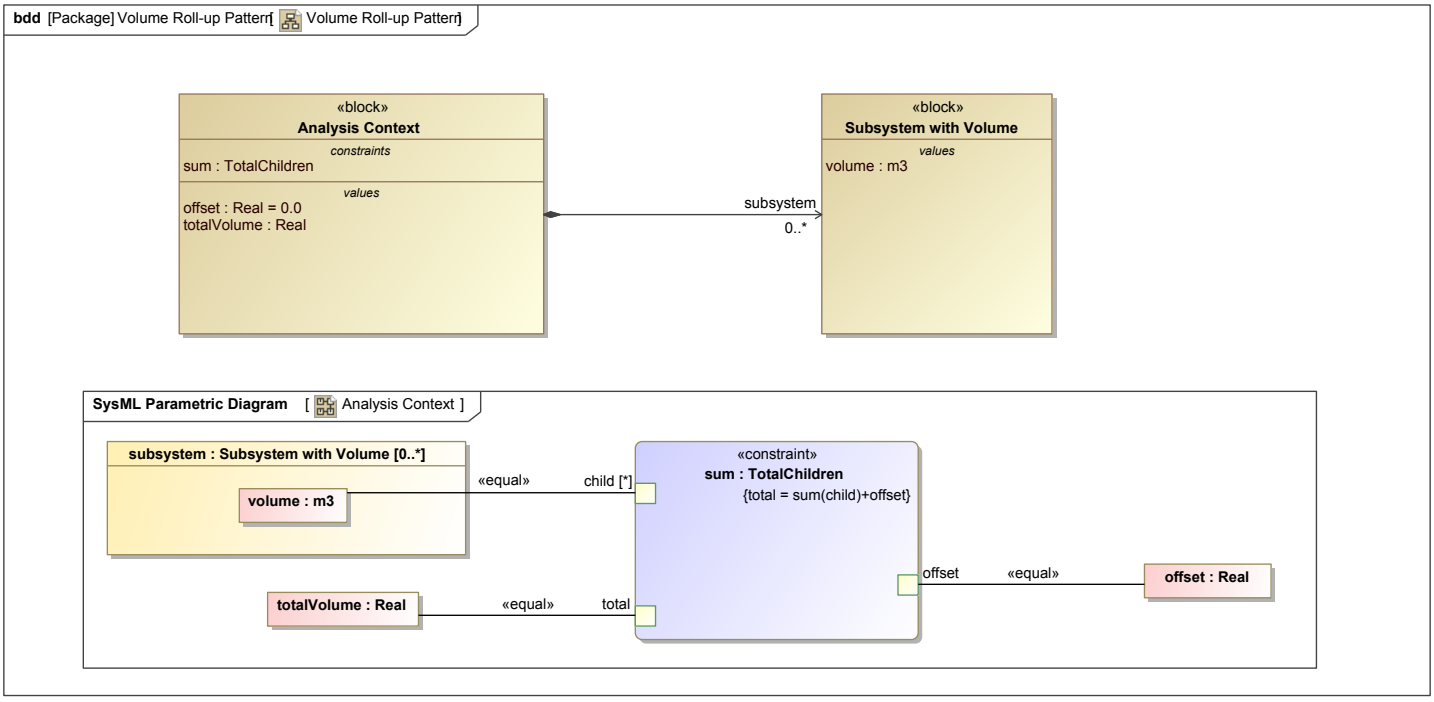


Figure 50. Volume Roll-up Pattern

To apply the structure of the pattern to the Subsystem with Volume system, the Simple Static Roll-up Aspect needs to be augmented to reflect the value (volume) that is to be propagated in the system. The derived value property totalProperty has been modified to totalVolume and ownStaticProperty has been changed to volume. In the parametric model of the Analysis Context block, the totalVolume is equal to the total parameter, the volume of the Subsystem with Volume is equal to the child parameter, and the value of offset is equal to the offset parameter of the constraint.

The Analysis block has been created to decouple the system design from its actual analysis, and for displaying the results of the volume propagation. The relationship between the Analysis Context block and the top level hierarchical component of the system is a whole-part relationship.

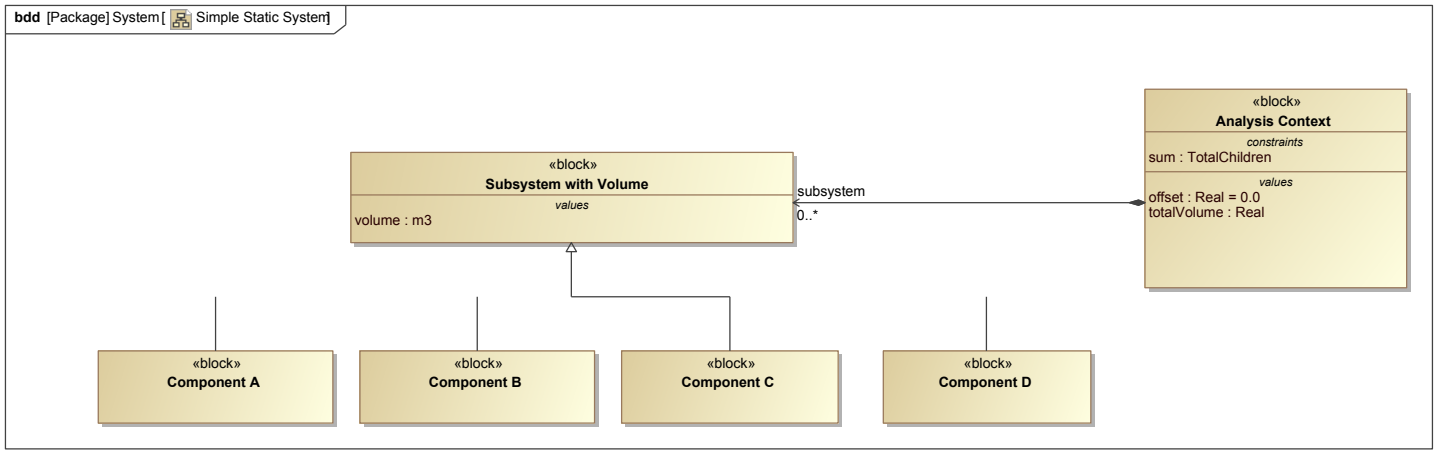


Figure 51. Simple Static System

All components of the system specialize the subsystem, and will inherit it's behavior and volume property. Either through redefinition or through instance specifications can the value of the volume property of the components be specified.

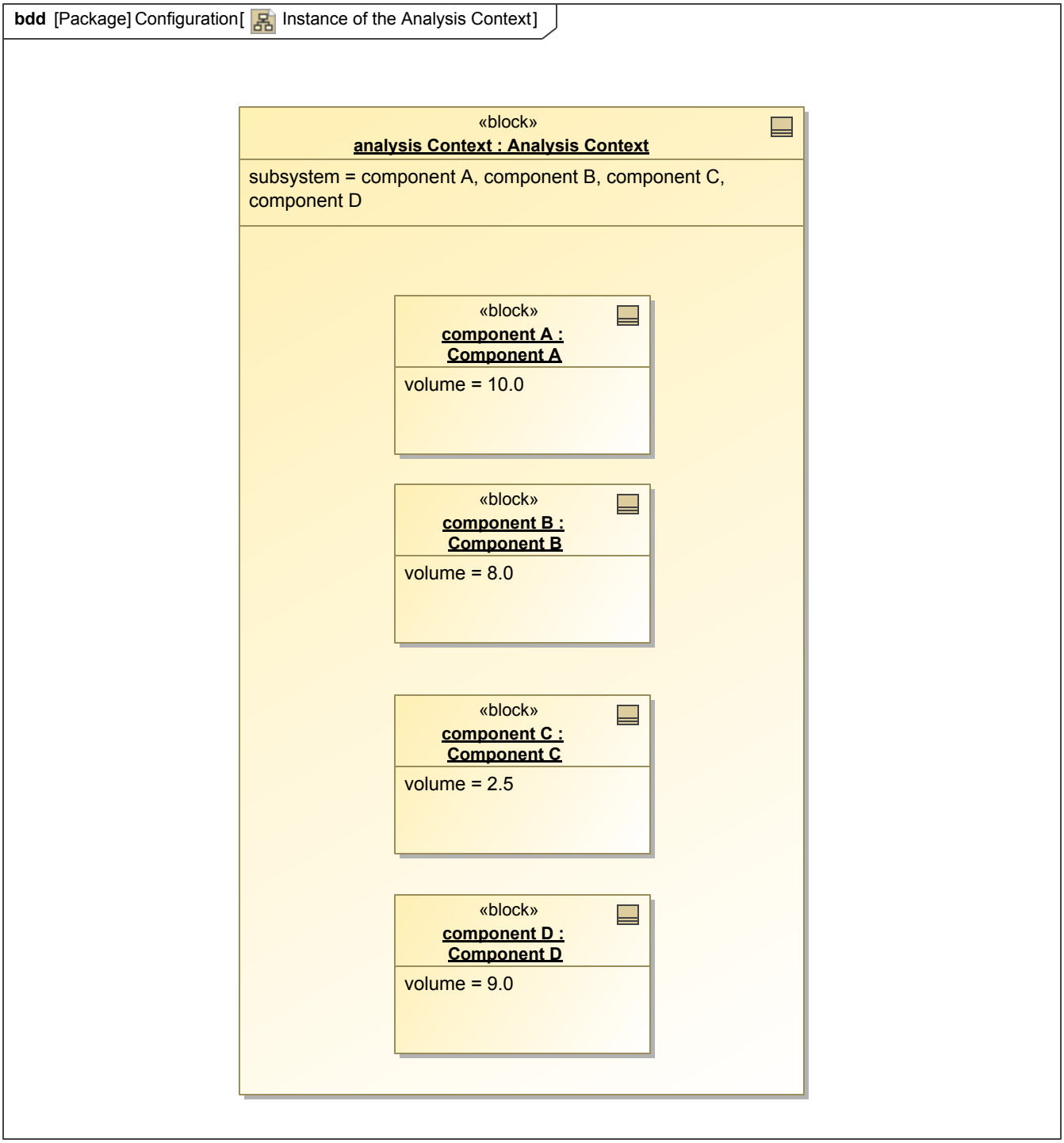


Figure 52. Instance of the Analysis Context

In this example, to define the value of volume for each component instance specifications have been created. In the instances of Components A, B, C, D the value for volume has been modified.

Table 4. volume Configuration

Name	volume : m3
component A	10.0
component B	8.0
component C	2.5
component D	9.0

Table 5. Analysis Results

Name	totalVolume : Real	subsystem : Subsystem with Volume
analysis Context at 2017.04.17 10.02	29.5	analysis Context.component A analysis Context.component B analysis Context.component C analysis Context.component D

4.1.1.6 Known Uses

4.1.1.7 Analysis

The Simple Static Roll-up pattern helps to analyze a system model by demonstrating that the resource usage according to the system design satisfies a corresponding requirement for every defined specified scenario.

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

After the operational scenarios and configurations the configured analysis can then be run using a simulation engine, such as Cameo Simulation Toolkit. This provides several outputs such as a timeline of the states of the individual components which shows state changes of individual components during the simulation, rolled up resource and margin , and value profiles, which show the total value over time of the rolled up value. These products represent different views that can be used by the engineer to analyze the results.

4.1.1.8 Tooling

The following tooling supports the modeler in the implementation of the Simple Static Roll-up pattern in MagicDraw.

4.1.1.8.1 Cameo Simulation Toolkit

NoMagic's Cameo Simulation Toolkit (CST) is implemented and supports the Simple Static Roll-up pattern by providing a method to perform analysis of a modeler's system.

After the roll-up pattern has been augmented to the system design, the values for the static property of the components participating in the roll-up need to be specified either through redefinition or instance specifications, and then analysis can be performed on the system by running a simulation with CST.

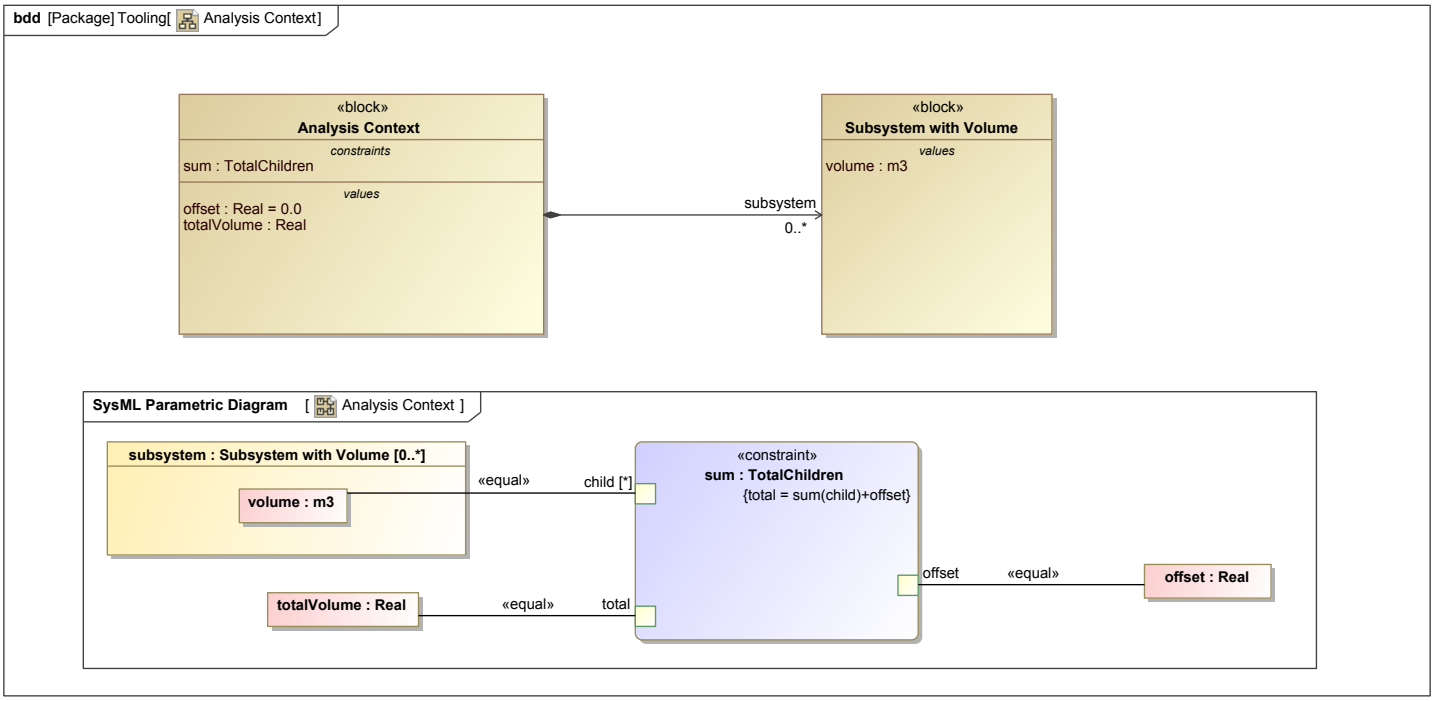


Figure 53. Analysis Context

The Analysis Context block is used to calculate the sum of all the children in the system. In the owned parametric model, the totalVolume is equal to the static property (totalVolume) of the roll-up of the system and an offset value.

To perform a simulation of the system, the modeler can either run a simulation from a component in the system design or to create a simulation configuration.

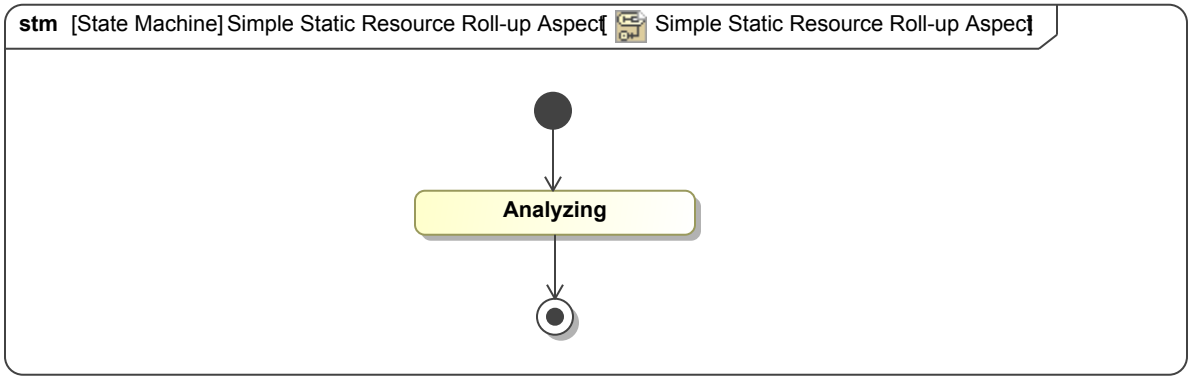


Figure 54. Simple Static Resource Roll-up Aspect

To automate the simulation process, the Analysis block needs to own a state machine diagram. In the state machine, there must be a initial and final node with some unspecific state in between. The roll-up is state independent, but the state machine is required if the modeler wants to automatically save the results of a simulation after running a simulation configuration. Without the the state machine the user would need to select the stop button to end the simulation.

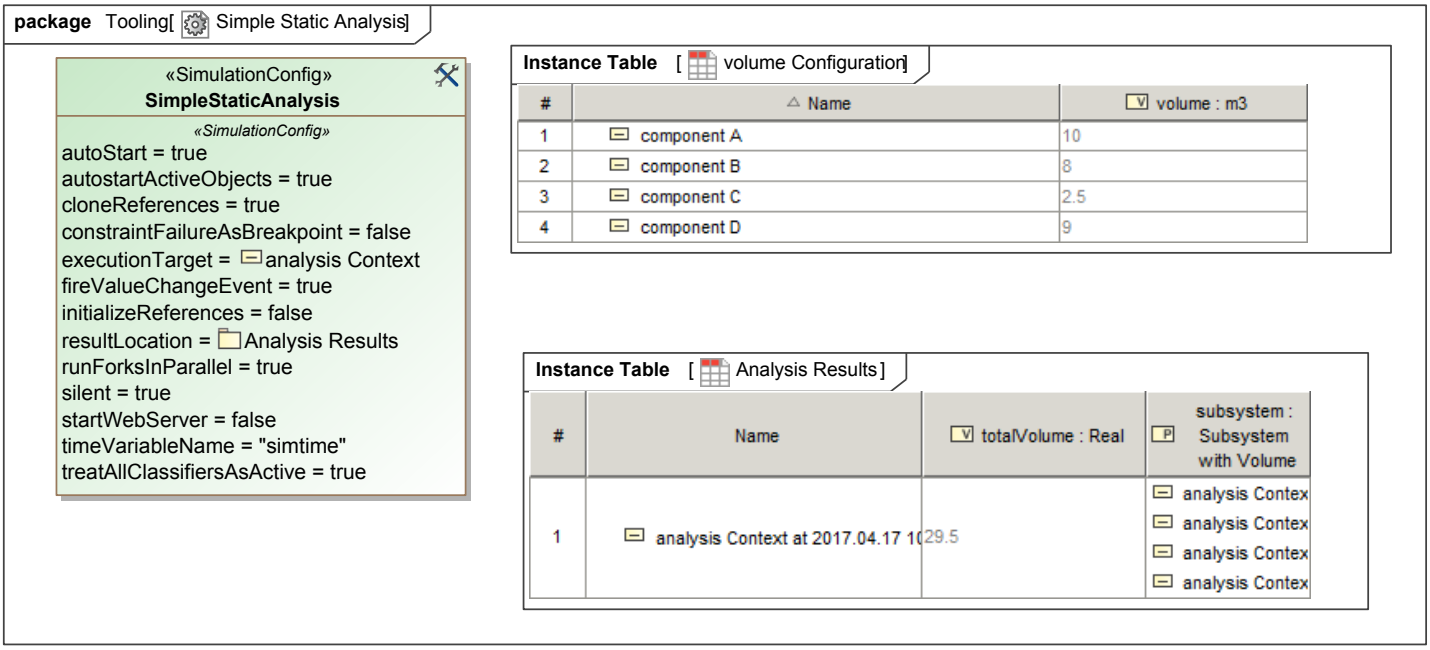


Figure 55. Simple Static Analysis

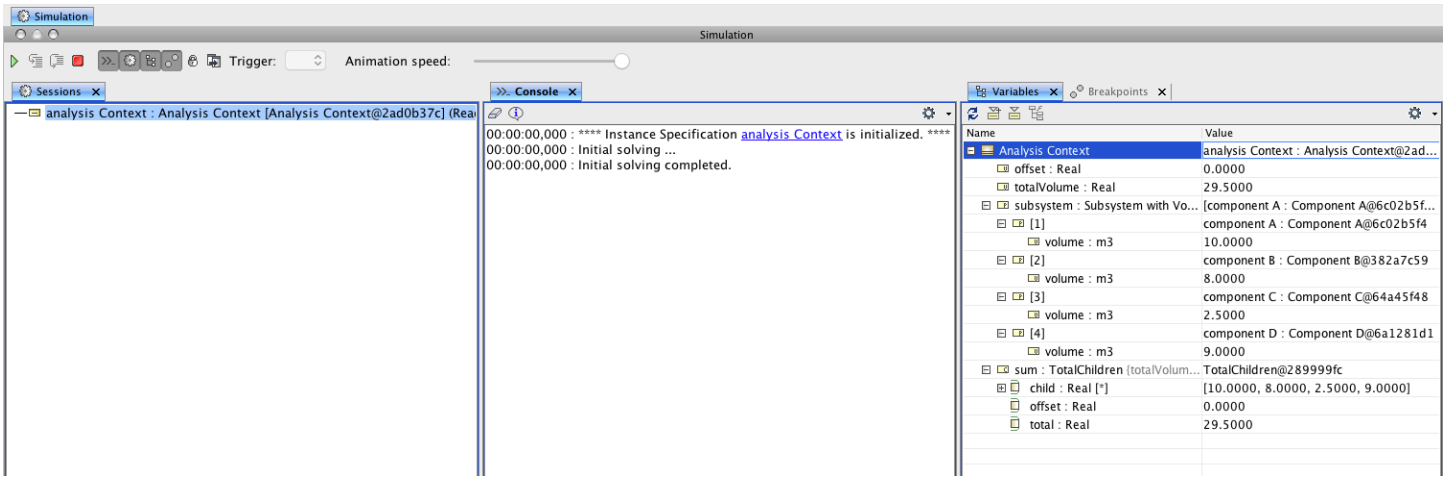
A simulation configuration can be used to automate the simulation process and exports the results as an instance specification. To create a Simulation Configuration, the executionTarget needs to equal the instance of the Analysis Context (or other top level hierarchical functional component).

After selecting to run the simulation, the resulting instance specification with the total value of the static property for the system can be saved in an instance table. This is done by configuring the property resultLocation to a package. In the Analysis Results instance table, a summary of the resulting final values of volume based on the previously configured blocks can be seen.

An overview of the values for each component during simulation can be seen in the variables section of the Simulation window.

When a Simulation of the Analysis context is initialized, initial solving of the variables occurs. During initial solving, the parametric model and state machine of the Analysis block, and the parametric model of the Simple Static Roll-up Pattern block are run.

As specified in the pattern, every component specializes the system, and will therefore inherit the static property (volume). In initial solving, the constraint's parameter child is equal to the value of all the children in the subsystem (Component A, B, C, D).
Initially there was no offset value specified, so the value for the totalVolume of in the Analysis Context is equal the parameter child.



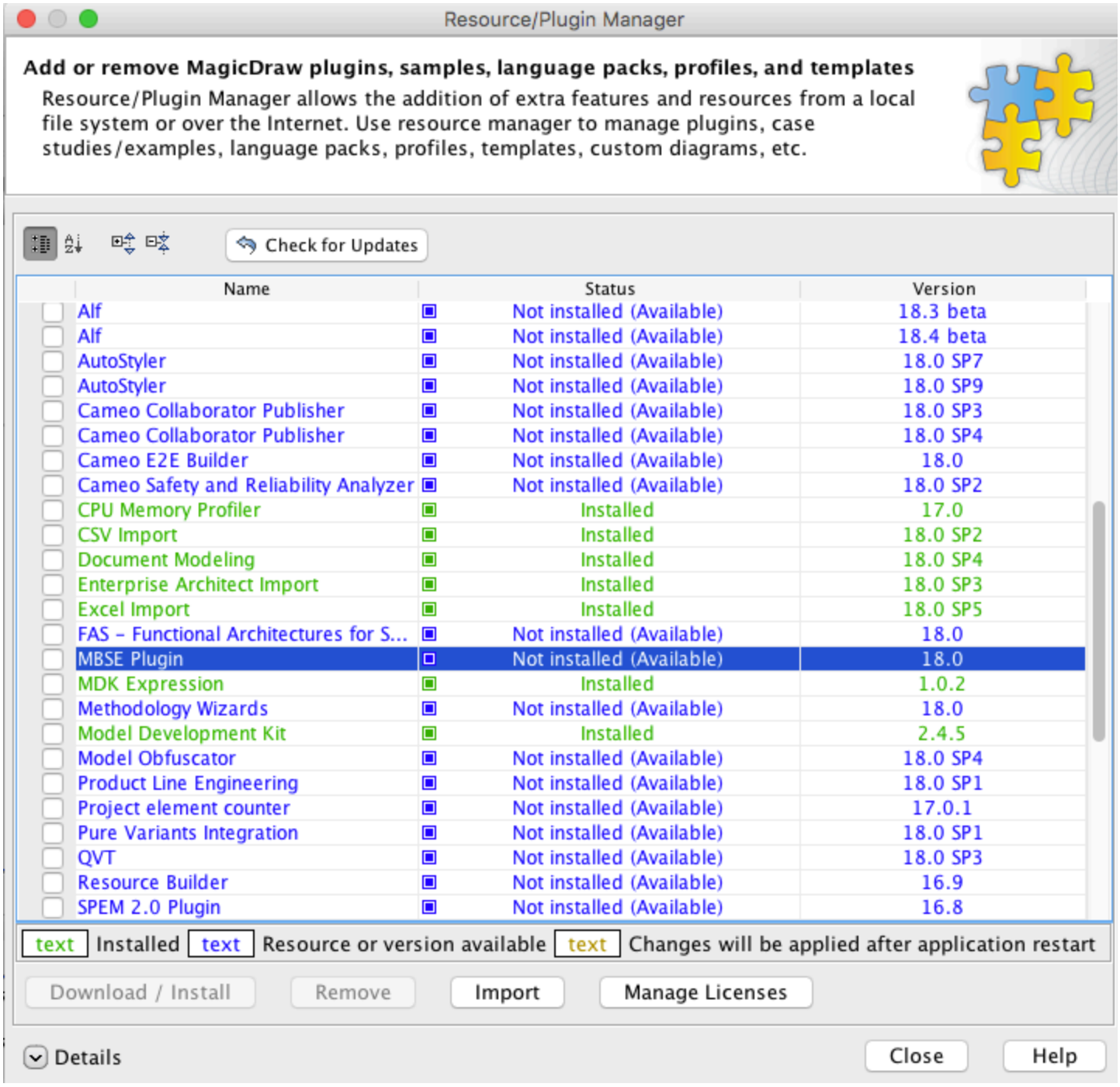
4.1.1.8.2 Model Construction

The general usage of the reasoner component is to aid systems engineers in pattern based reasoning.

Other useful resources that aid the implementation the pattern include NoMagic's Rollup Pattern Wizard for MagicDraw version 18.5, and the MBSE plug-in for MagicDraw version 17.0 sp4.

MBSE Plugin

The MBSE plug-in can also be found in the Resource/Plugin Manager of MagicDraw which is compatible for MD 18.0.



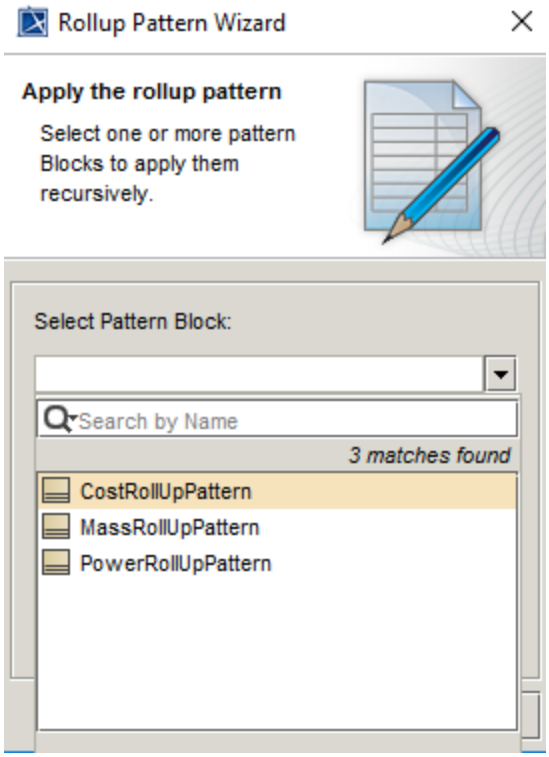
4.1.1.8.2.1 NoMagic Rollup Pattern Wizard

The NoMagic Rollup Pattern Wizard is used to apply the specified rollup pattern to a system. After defining the system and the rollup pattern, the user can apply the NoMagic Rollup Wizard to the system. The user can use the Rollup pattern by following these steps:

Applying a Rollup Pattern

Select the Block (or instance specification) in which you want to apply the pattern to and select **Tools** > **Apply Rollup Pattern**

The Rollup Pattern Wizard window will then appear as shown.



The Rollup Pattern Wizard provides three default rollup pattern blocks for cost, mass, and power (i.e. CostRollUpPattern, MassRollUpPattern, PowerRollUpPattern). The default pattern blocks can be found in the **MD Customization for SysML** auxiliary resource. The wizard also allows for customized Pattern Blocks as well.

Creating Customized Pattern Blocks

There are two methods for creating a customized pattern block that is specific to certain type of roll-up analysis.

1. Copy and pasting a default rollup pattern block from the MD Customization for SysML auxiliary resource
    1. Copy any existing rollup pattern block (i.e. CostRollUpPattern, MassRollUpPattern, PowerRollUpPattern)
    2. Paste into appropriate location of the model
    3. Modify the pattern block
        1. Change name, value properties (make sure to change the types of the value properties), part property, constraint block, and parametric diagram
2. Creating a Roll-up block from our pattern and applying the <<Pattern>> stereotype
    1. Follow the structure or copy elements of/from the OpenSE Cookbook/OpenSE Library
    2. Paste into appropriate location of the model
    3. If copying, modify the pattern block
        1. Change name, value properties (make sure to change the types of the value properties), part property, constraint block, and parametric diagram
    4. If creating the pattern block on own, modify the pattern block
        1. Set name, value properties (types of the value properties), part property, constraint block, and parametric diagram
        2. Apply <<Pattern>> Stereotype
            1. Set the stereotype of the pattern block to <<Pattern>>. The stereotype is located in the MD Customization for SysML auxiliary resource. This step will allow the Rollup Pattern Wizard to recognize the pattern block.

4.1.1.9 Related Patterns

List of other patterns that are related to the Simple Static Roll-up pattern, and the differences and similarities between them.

Pattern Name	Similarities	Differences

Recursive Static Roll-up Pattern	A similarity between the Simple Static roll-up and the Recursive Static roll-up pattern is that they both propagate the values of a static property in a system. This means the value of the property does not change over time due to its behavior.	The differences between the Recursive Static roll-up and the Simple Static roll-up pattern is in the application and relationships of the pattern to a system. When applying the Simple Static roll-up to a system it must have an un-complex structure, where there is a flat listing of components. However, when applying the Recursive Static Roll-up, the system can be structured in a nested compositional hierarchy. The aspect block of the Recursive Static roll-up needs to be specialized by every component in the structure and subsetted by its subRollup part property. While the aspect block of the Simple Static Roll-up is a whole-part relationship to only the top level component/system.
Dynamic Roll-up Pattern	There are no similarities between the behavior of the Simple Static and Dynamic roll-up patterns, and their relationship to the system.	The primary difference between the Simple Static roll-up and the Dynamic roll-up pattern is due to the system's behavior. The total value of the final dynamic property of the Dynamic Roll-up is equal to the component's values during a certain event in its lifecycle. Meanwhile, the value of the final static property of the Simple Static roll-up doesn't depend on the state of the components.

4.1.2 Recursive Static Roll-up Pattern

4.1.2.1 Intent

Roll-up calculations of system resources (e.g. total mass, cost, power, or another system dimension) are among the most common use cases in systems engineering. System engineers want to perform calculations based on the specific values of all the components in the system. The Recursive Static Roll-up pattern provides system modelers a solution to solve common domain issues, and consists of standard constructs that are applicable to most domains. If the modeler desires to perform analysis of a system where the values of the subsystem are independent of the system's behavior the Recursive Static Roll-up pattern should be reused and applied in their domain specific construct.

4.1.2.2 Motivation

The overall motivation for system engineers to apply the pattern is the desire to have requirements that are traced to to design artifacts, and to have a method to assert that a system design satisfies a set of requirements. Through SysML patterns, analysis specifications can be defined and the modeler can verify, for instance, whether the resource consumption of a system satisfies an initial requirement. If a system modeler wants to perform analysis of a complex system that consists of components whose value properties are state independent, and the modeler is familiarized with running simulations, the Recursive Static Roll-up Pattern can be configured to their system.

The Recursive Static Roll-up supports the analysis of a system design which is independent of the component's operational states. The Recursive Static Roll-up pattern can be applied to a model in which the product breakdown structure of the system is specified.

The components aggregated in the roll-up must share a common value property that does not change (static) throughout the component's life-cycle. Through the application of the pattern, each component's static values can be recursively propagated up a hierarchy of components characterized by these values.

To apply the pattern to a system, the modeler needs to augment the components of the system design for a particular resource in order to propagate the values up the system hierarchy. The pattern must be configured to the system to specify the analysis context using the system's design, states, properties, and relationships. Only then can the pattern provide a context as to whether the design satisfies a set of formalized requirements.

4.1.2.3 Concept

The structure of the Recursive Static Roll-up pattern consists of the following model elements; the Recursive Static Resource Roll-up Aspect block, and the Recursive Static Resource Roll-up Aspect parametric diagram.

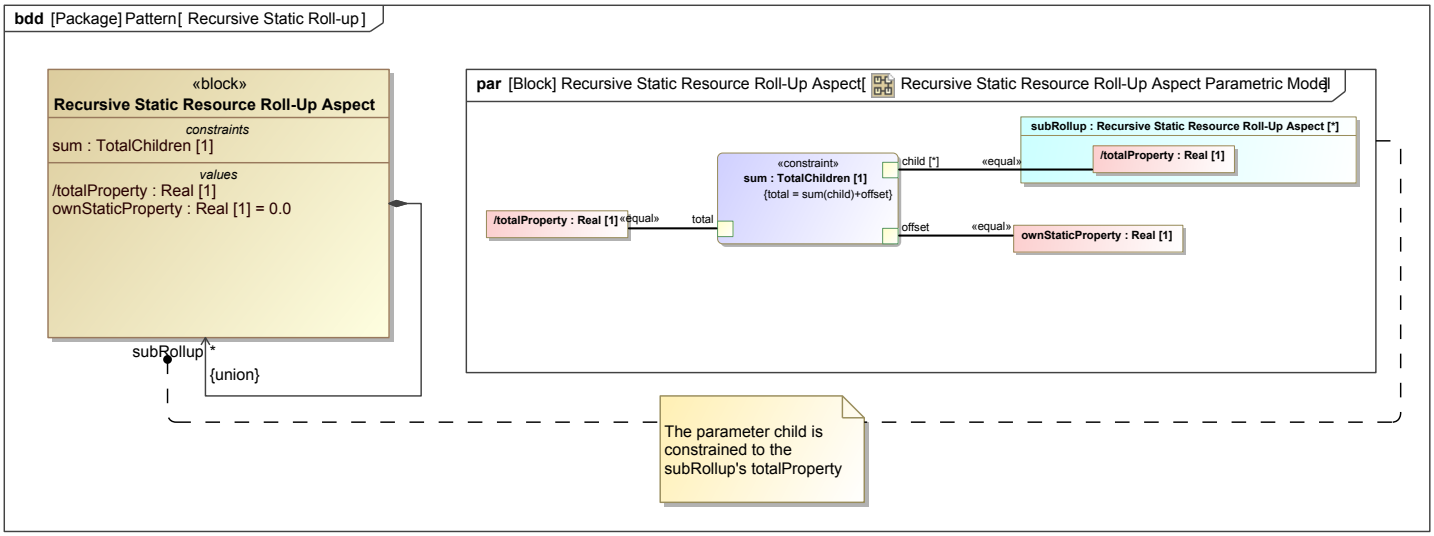


Figure 56. Recursive Static Roll-up

The Recursive Static Resource Roll-up Aspect has the value properties totalProperty and ownStaticProperty that are propagated in the roll-up, and a recursive part property subRollup. The aspect block owns a parametric model where the properties are constrained. In the parametric model the owner's constraint sum is constrained to the expression total = sum(child)+offset. For each component that specializes the Recursive Static Resource Roll-up Aspect block the static and total properties will be constrained. The parameter, child, is equal to the derived value property totalProperty of the subRollUp, and is denoted with the multiplicity * to represent that there can be an infinite amount of components. The parameter offset is equal to the property ownStaticProperty. The constraint's parameter, total, which calculates the sum of the children is equal to the derived value property totalProperty.

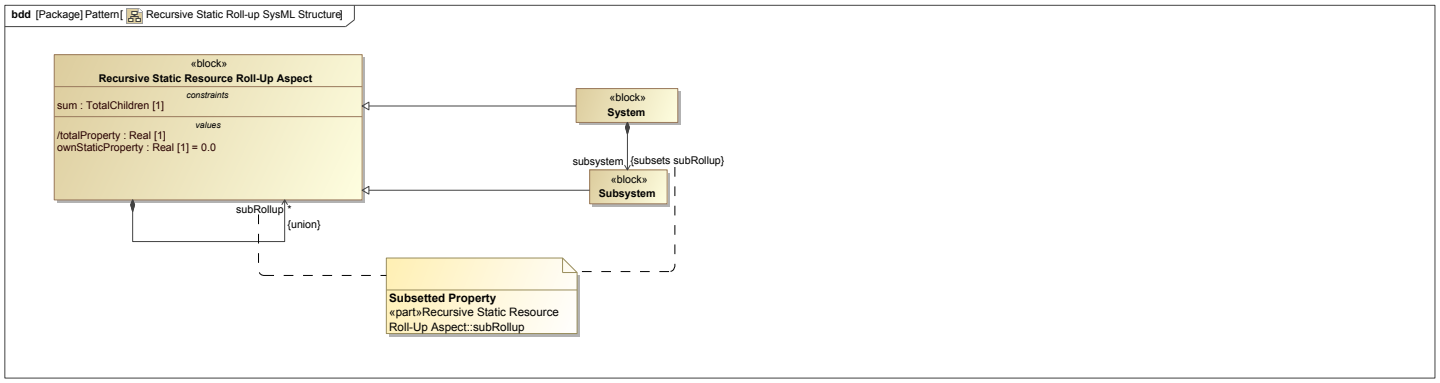


Figure 57. Recursive Static Roll-up SysML Structure

The SysML structure of the Recursive Static Roll-up Pattern is dependent on the properties of the Recursive Static Resource Roll-up Aspect block and the relationship between the aspect block and components in the system. To calculate the values of each component of the system each block participating in the roll-up must specialize the Recursive Static Resource Roll-up Aspect block. In addition, each subsystem must have a composition relationship to the system or to its parent. Every subsystem/subcomponent that composes the system must subset the subRollup part property of the Recursive Static Resource Roll-up Aspect.

Table 6. <>

Model Element	Role
System	The System block is decomposed into the components that will inherit the value properties totalProperty and ownStaticProperty and specializes the Recursive Static Resource Roll-up Aspect. The System block represents a modeler's system which will be augmented to the system's appropriate name (ex. Power Subsystem, Telescope).
Recursive Static Resource Roll-Up Aspect	The Recursive Static Resource Roll-up Aspect contains the value properties (totalProperty, ownStaticProperty) that are needed to calculate the total of all the staticProperty values, the TotalChildren constraint, and a parametric model where the properties are constrained.

Model Element	Role
totalProperty	The inherited value property, totalProperty, is the sum of all the children in the system hierarchy. The totalProperty is final value for the roll-up calculations of the system.
ownStaticProperty	The local value property, ownStaticProperty, of the Recursive Static Resource Roll-up Aspect block is used to offset the total value of a child.

4.1.2.4 Consequences

There are several trade-offs to consider when configuring the Recursive Static Roll-up pattern to the modeler's system.

Action	Consequence
Redefining the static values of the system's components	By redefining the static values of the system's components the modeler can easily change the inherited attributes by redefining and modifying the values. If the static value of a component isn't redefined, the value will default to the value of the base class. However one could argue that redefinition is the proper approach to specifying the components attributes because the modeler can view and modify the inherited aggregated value properties of it's classifier.
Creating instance specifications to define the static values of the system's components	By creating instance specifications to define the initial values of the system, the modeler is able to track different configurations. New configurations for a scenario can be easily created and viewed in an instance table. The value properties are visible in the block specification.
Modifying the system model when applying the roll-up pattern to the system	A consequence of needing to modify the system design results in an authority/ownership problem. An example of this occurs when an authority owns and modifies components of the system design and system decomposition. If another authority requires to perform a roll-up analysis of this design, they will need to augment the system design in order to define the relation between the system properties, e.g. the mass of one component is the sum of the mass of its children. A possible SysML work around is to use multi-classification at the instance specification level i.e. the instance specification is specified by the component classifier and the roll-up aspect classifier. The drawback to this approach is that the value properties are not visible in the block specification and only in the instance specification level.
Subsetting all composition relationships of the system's hierarchy	The modeler must subset every element in the hierarchy which can become burdensome depending on the number of components and relationships in the system. A possible solution would be to use a reasoning tool to automate this process.

Usage of the Pattern	Explanation of Conflict
Configuring the Recursive Static Roll-up pattern to a system without subsetting the relationship between components of the system.	The composition relationships between components of a hierarchical system needs to subset the Recursive Static Resource Roll-up Aspect block's part property subRollup. Without subsetting the relationship the static values will be unable to propagate. If after performing a simulation of the system and the expected resulting final value is 0 navigate to either the structure of the system or the simulation configuration window. A likely cause for the error and the resulting value is due to an un-subsetted relationship.

4.1.2.5 Implementation

There are two possible methods for applying the Recursive Static Roll-up pattern to a system for performing simulation analysis. These methods include either through redefinition of the static value properties or through instance specifications.

4.1.2.5.1 Recursive Static Instance Example

In this example, we will apply the Recursive Static Roll-up to a subsystem of a motorcycle to demonstrate how the pattern can be applied and reconfigured to a system to calculate the static property of volume. Instance specifications of the system will be created to modify the values of each subcomponent.

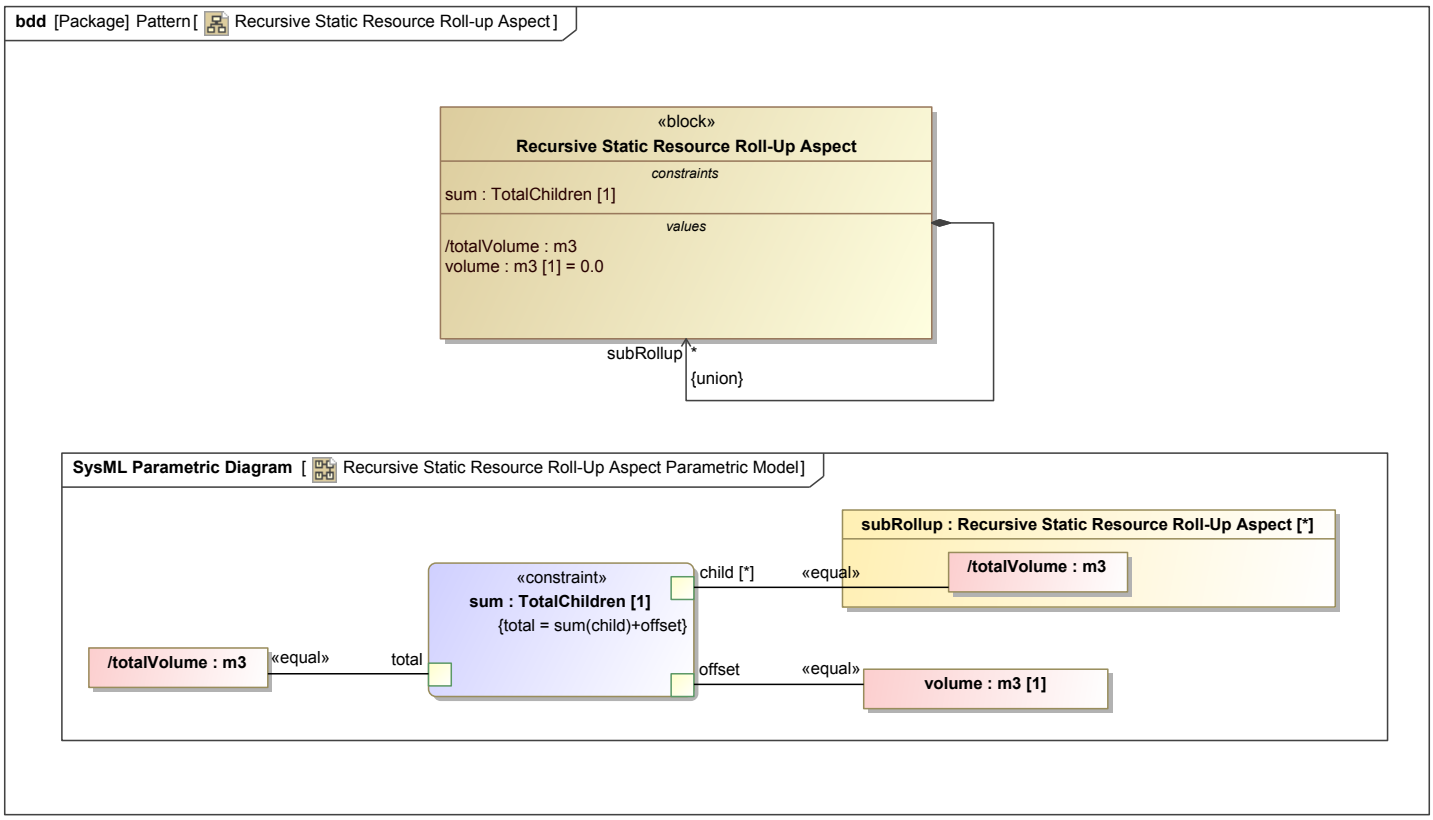


Figure 58. Recursive Static Resource Roll-up Aspect

To apply the structure of the pattern to the motorcycle system, the Recursive Static Roll-up Aspect needs to be augmented to reflect the value that is to be propagated in the system. The derived value property totalProperty has been modified to totalVolume and ownStaticProperty has been changed to volume.

In the parametric model of the Recursive Static Resource Roll-up Aspect the totalVolume is equal to the total parameter, the totalVolume of the subRollup part is equal to the child parameter, and the value of volume is equal to the offset parameter of the constraint.

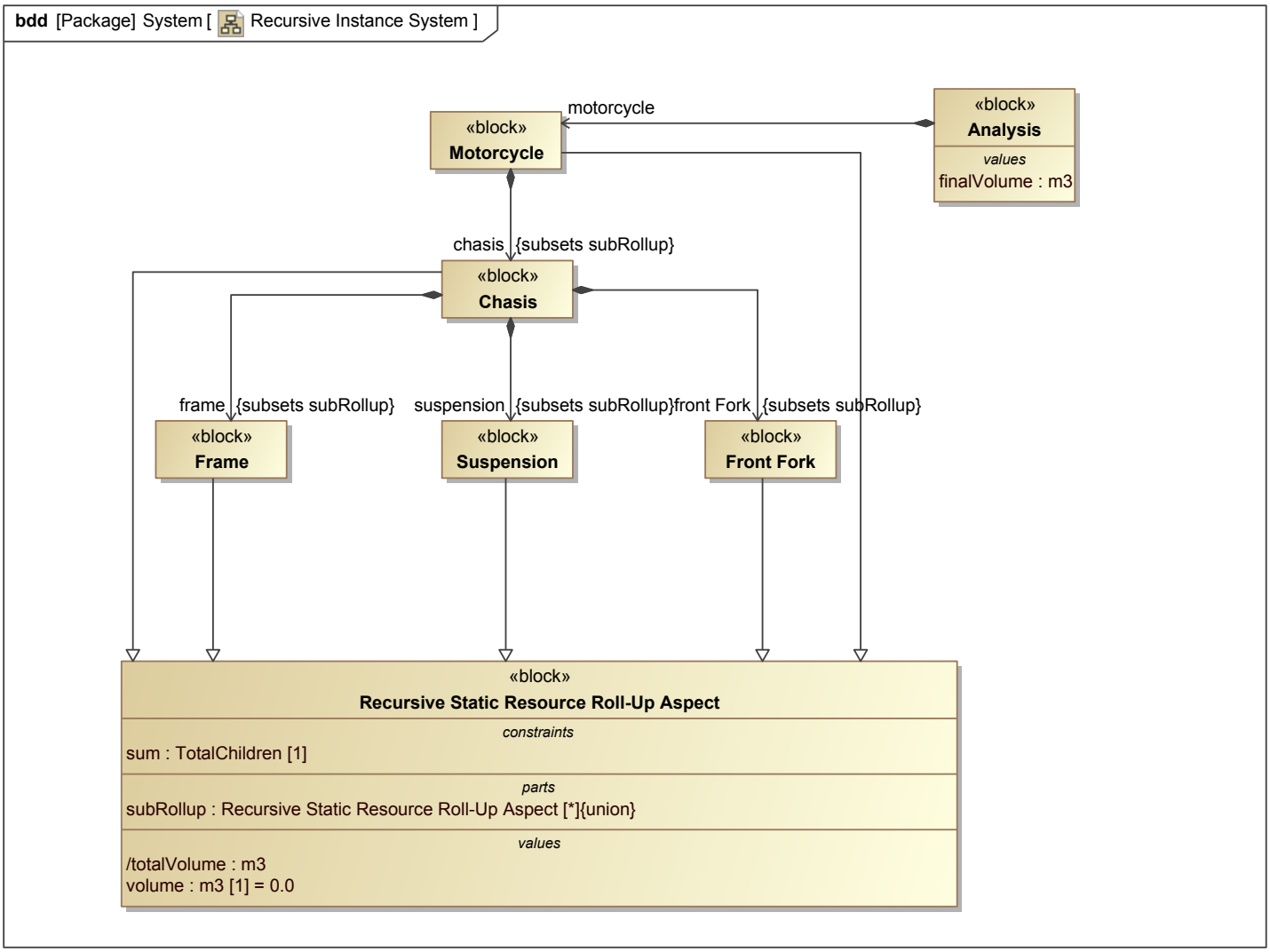


Figure 59. Recursive Instance System

All components of the hierarchy specialize the Recursive Static Resource Roll-up Aspect block, and will inherit its behavior and properties. As seen in the bdd above, every part of the motorcycle system subset the aspect block's part property subRollup.

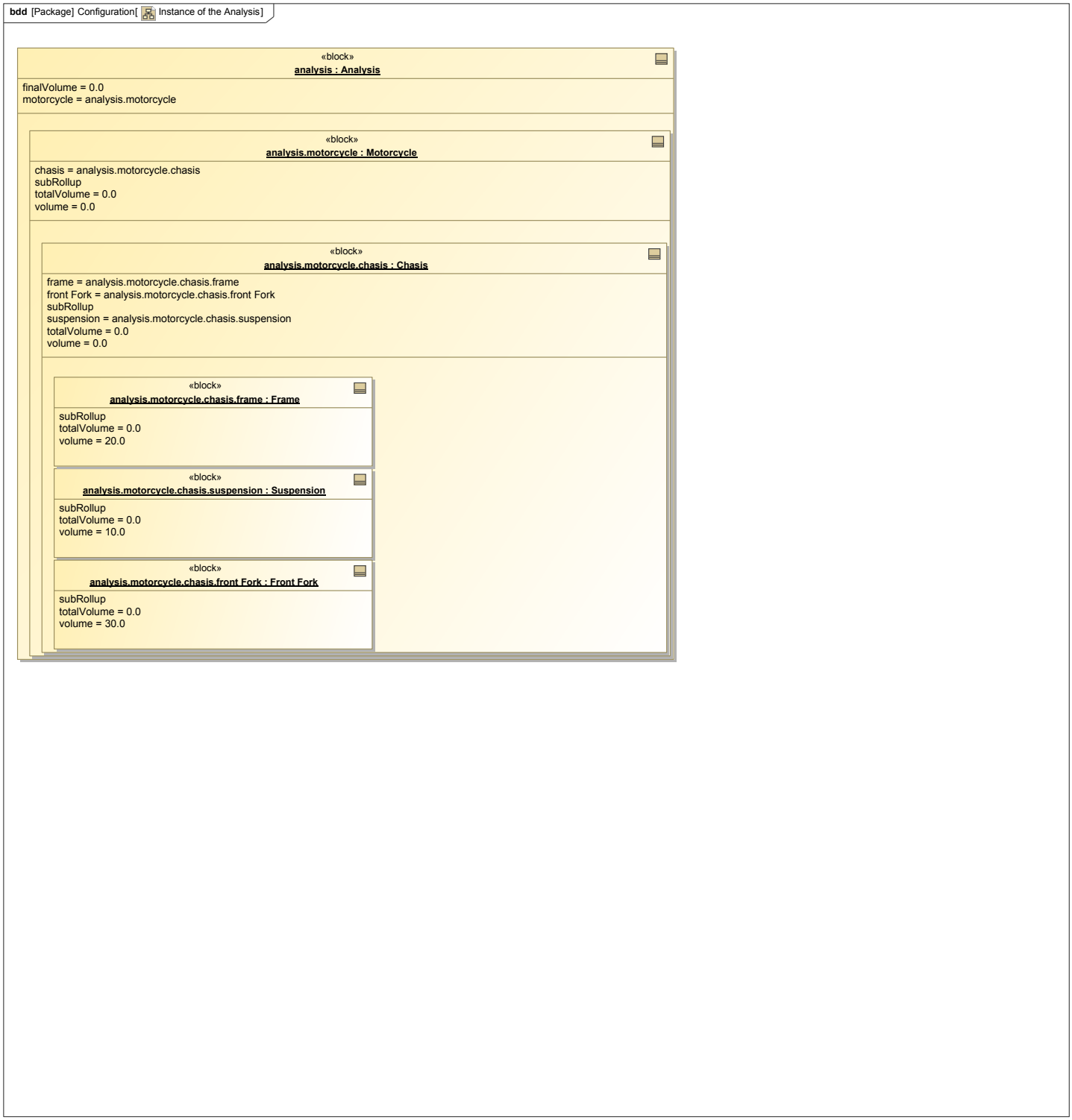


Figure 60. Instance of the Analysis

To define the value of volume for each component, instance specifications have been created. In the instances of the Front Fork, Suspension, and Frame the value for volume has been modified.

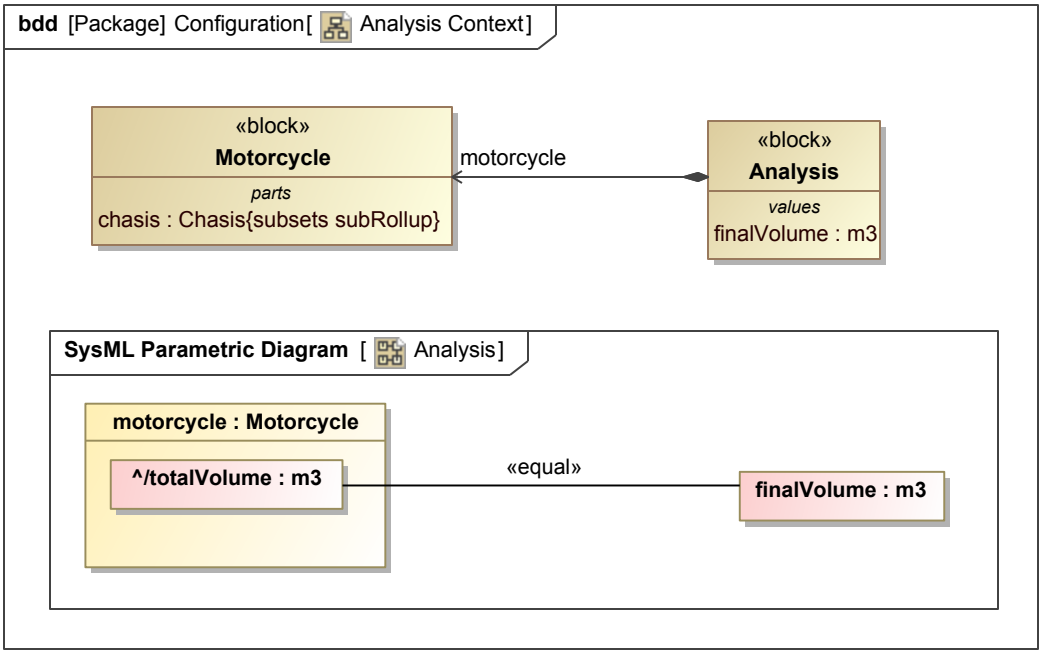


Figure 61. Analysis Context

The Analysis block has been created for an easier context for displaying the results of the volume propagation. A parametric model has been created to constrain the finalVolume to equal the totalVolume of the roll-up.

Table 7. Configuration

Name	volume : m3
analysis.motorcycle.chasis.frame	20.0
analysis.motorcycle.chasis.front Fork	30.0
analysis.motorcycle.chasis.suspension	10.0

Table 8. Analysis Results

Name	finalVolume : m3
analysis at 2017.04.19 00.14	60.0
analysis at 2017.04.24 10.25	60.0

4.1.2.5.2 Recursive Static Block Example

In this example, we will apply the Recursive Static Roll-up to a subsystem of a motorcycle to demonstrate how the pattern can be applied and reconfigured to a system to calculate the static property of volume. The values of the static properties for the blocks participating in the roll-up will be redefined.

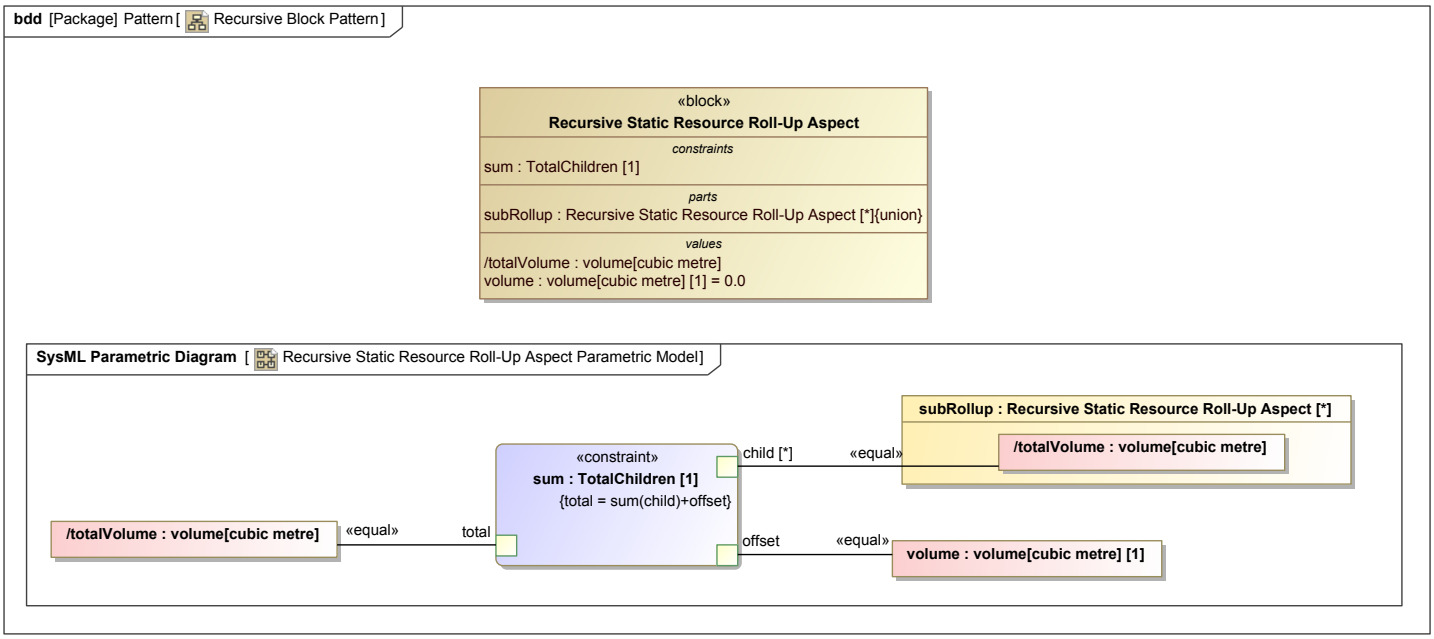


Figure 62. Recursive Block Pattern

To apply the structure of the pattern to the motorcycle system, the Recursive Static Roll-up Aspect needs to be augmented to reflect the value that is to be propagated in the system. The derived value property totalProperty has been modified to totalVolume, and ownStaticProperty has been changed to volume.

In the parametric model of the Recursive Static Resource Roll-up Aspect the totalVolume is equal to the total parameter, the totalVolume of the subRollup part is equal to the child parameter, and the value of volume is equal to the offset parameter of the constraint.

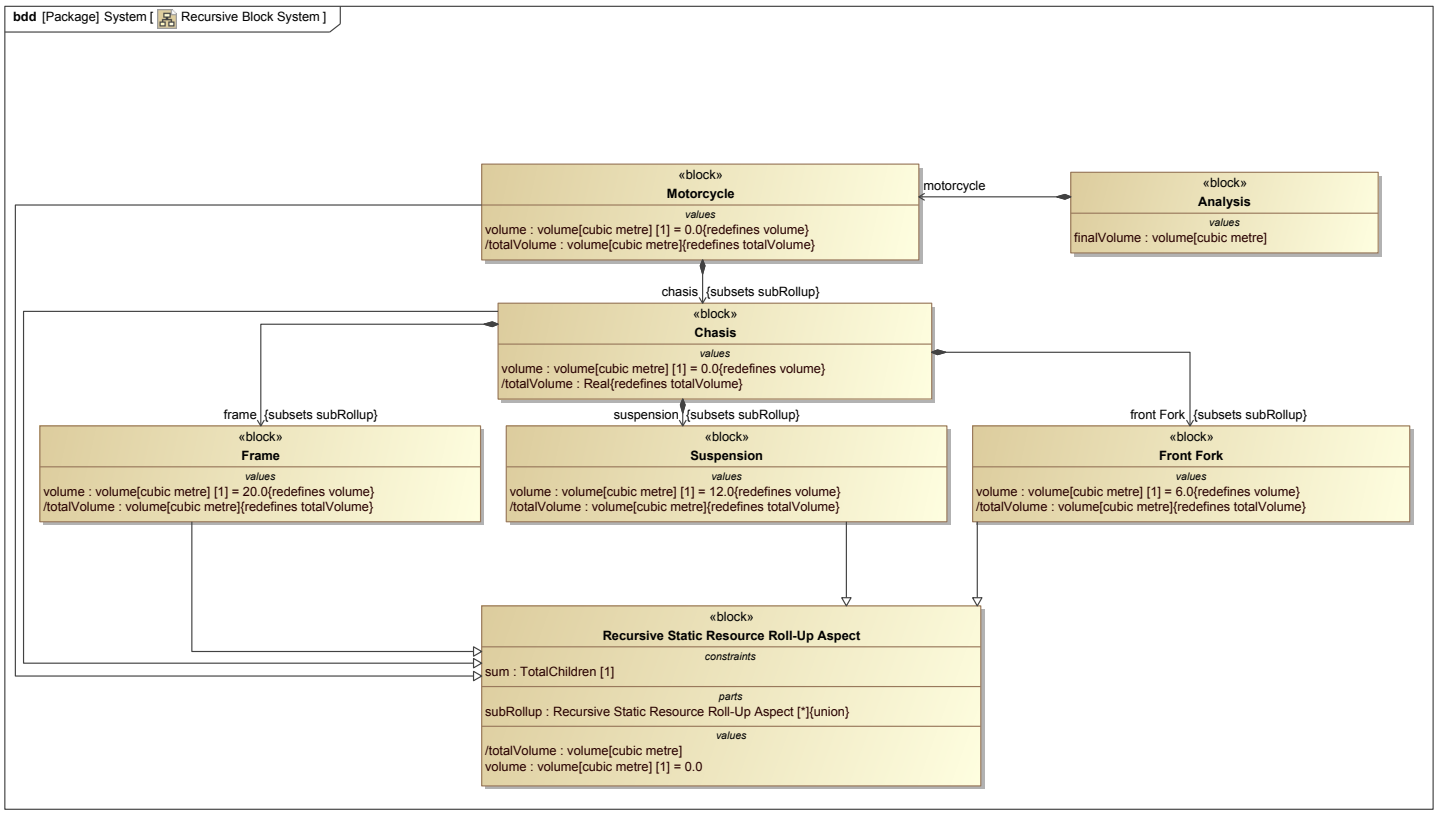


Figure 63. Recursive Block System

All components of the hierarchy specialize the Recursive Static Resource Roll-up Aspect block, and will inherit its behavior and properties. Additionally, every part of the motorcycle system subset the aspect block's part property subRollup.

In this method, the static properties (totalVolume and volume) of each component in the system (Motorcycle, Chassis, Front Fork, Suspension, and Frame) have been redefined and the default values have been specified. Through this method, the user will need to modify the default values of the blocks when there is a change to a value of a component.

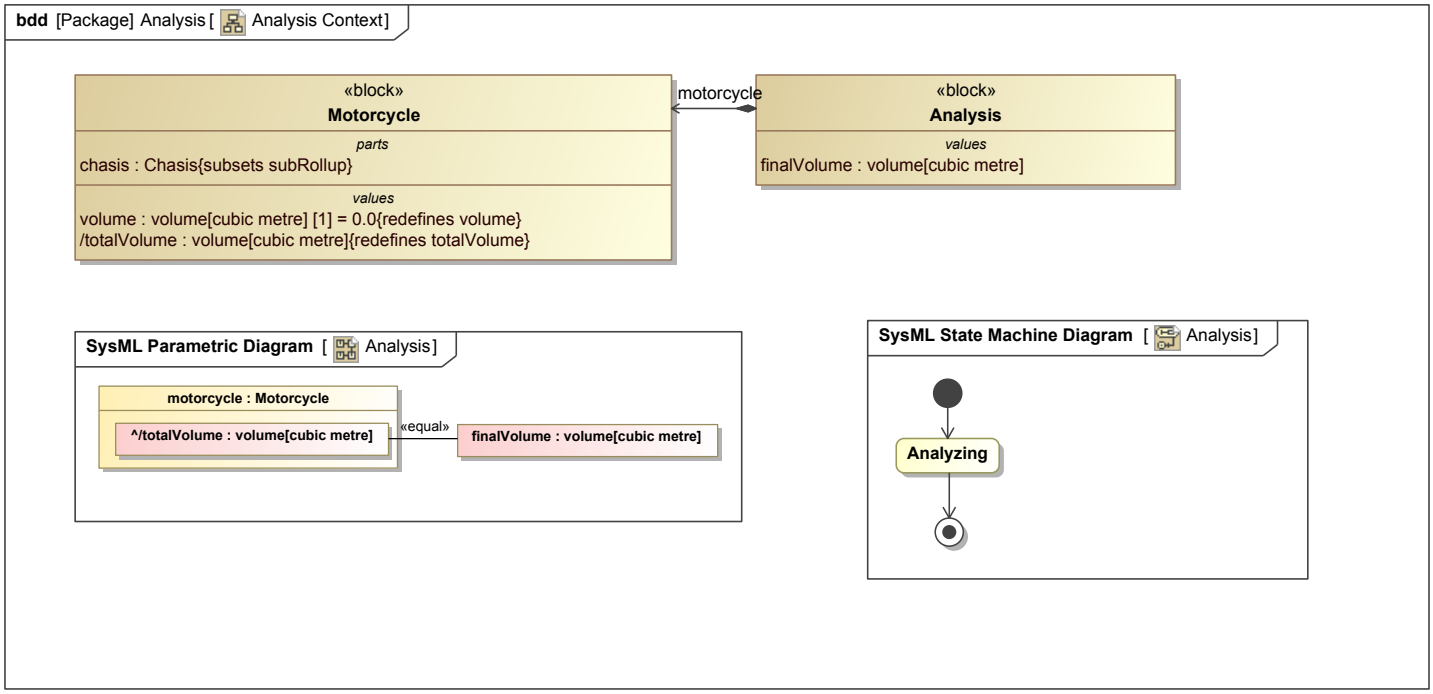


Figure 64. Analysis Context

The Analysis block has been created for an easier context for displaying the results of the volume propagation. A parametric model has been created to constrain the finalVolume to equal the totalVolume of the roll-up.

Table 9. Analysis Results

Name	finalVolume : volume[cubic metre]
analysis at 2017.04.19 07.32	38.0
analysis at 2017.04.24 10.01	38.0
analysis at 2017.04.24 10.02	48.0

4.1.2.6 Known Uses

The TMT SysML model applies a recursive static roll-up pattern to calculate the total mass of the components in the Alignment and Phasing System (APS). A main objective of the TMT model is to capture operational scenarios and demonstrate that requirements are satisfied by the design. The roll-up pattern facilitates the analysis of the system design against mass requirements.

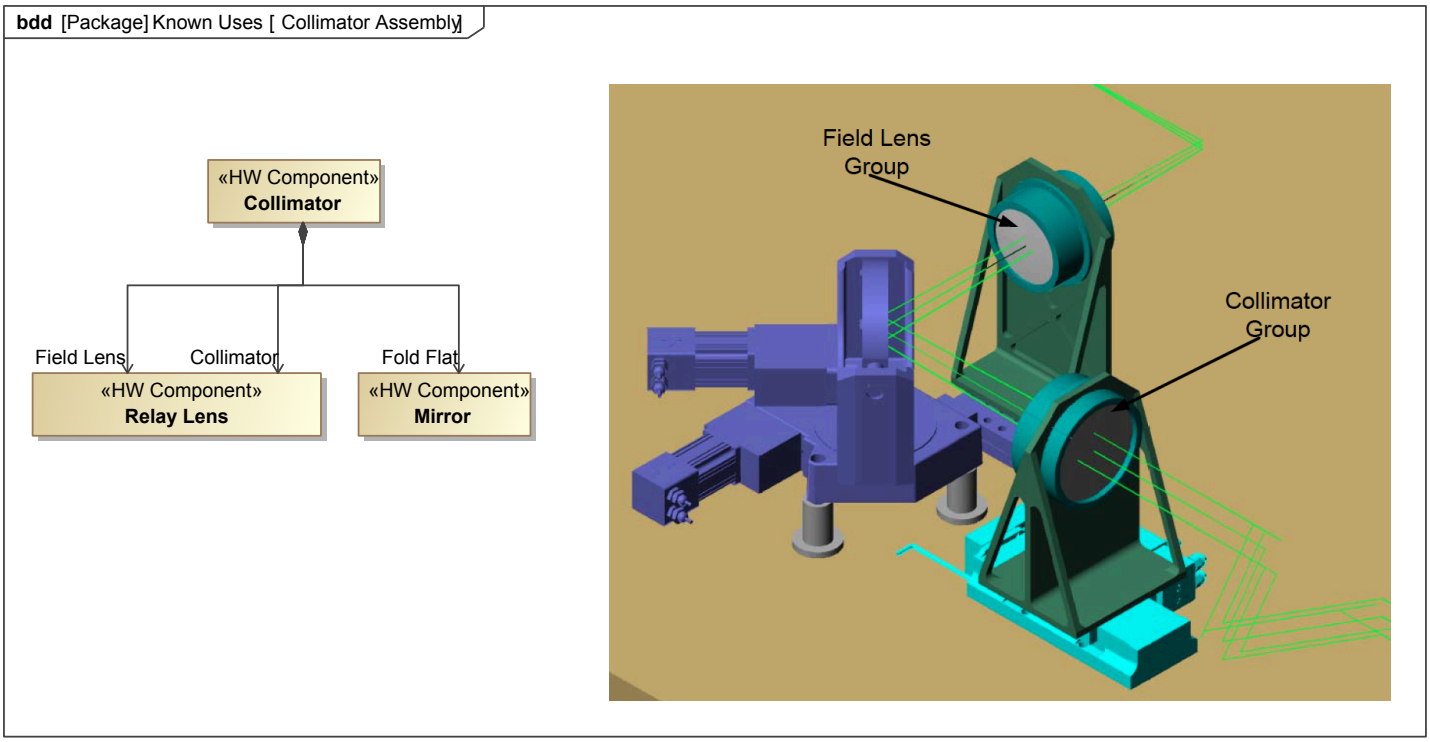


Figure 65. Collimator Assembly

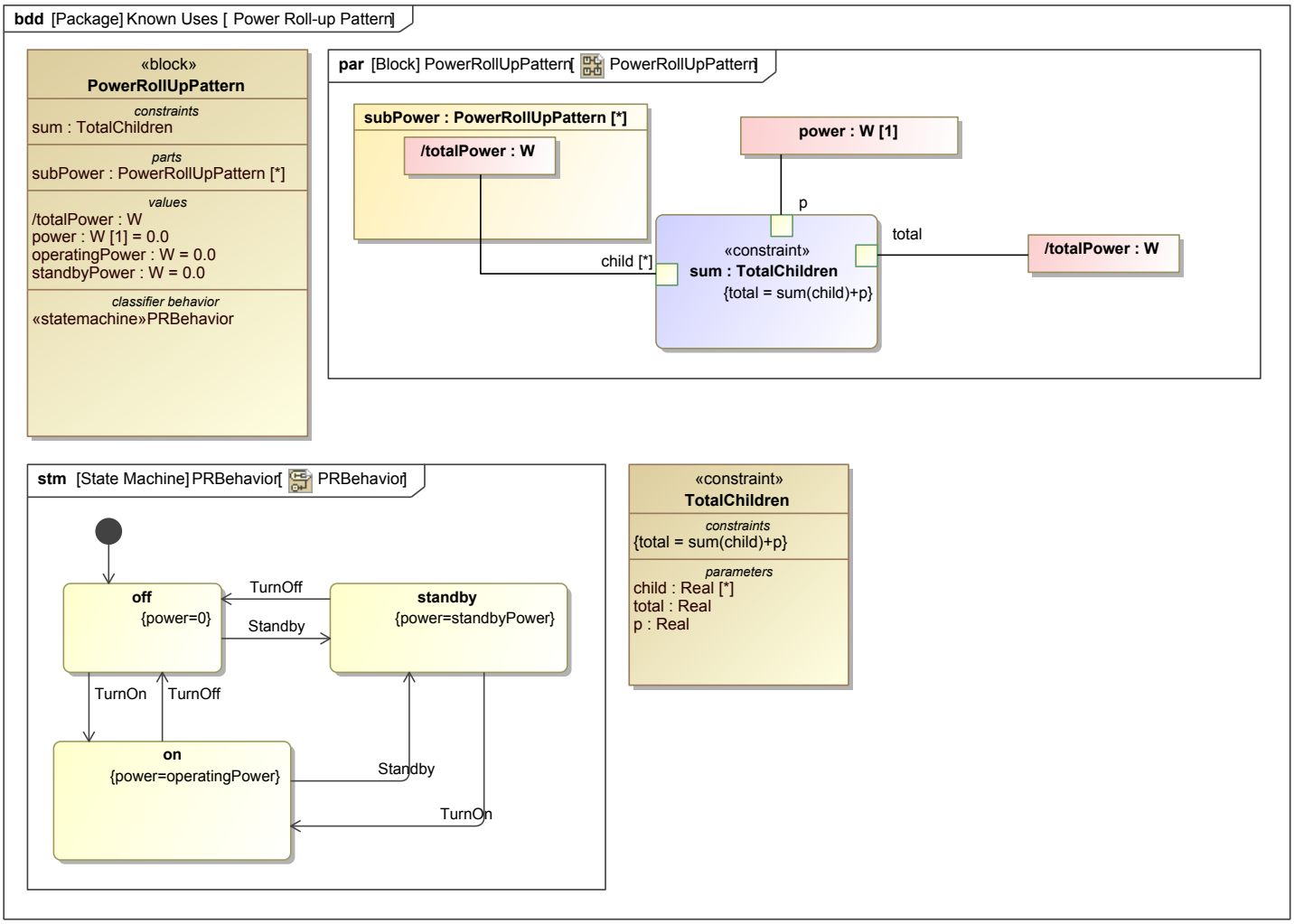


Figure 66. Power Roll-up Pattern

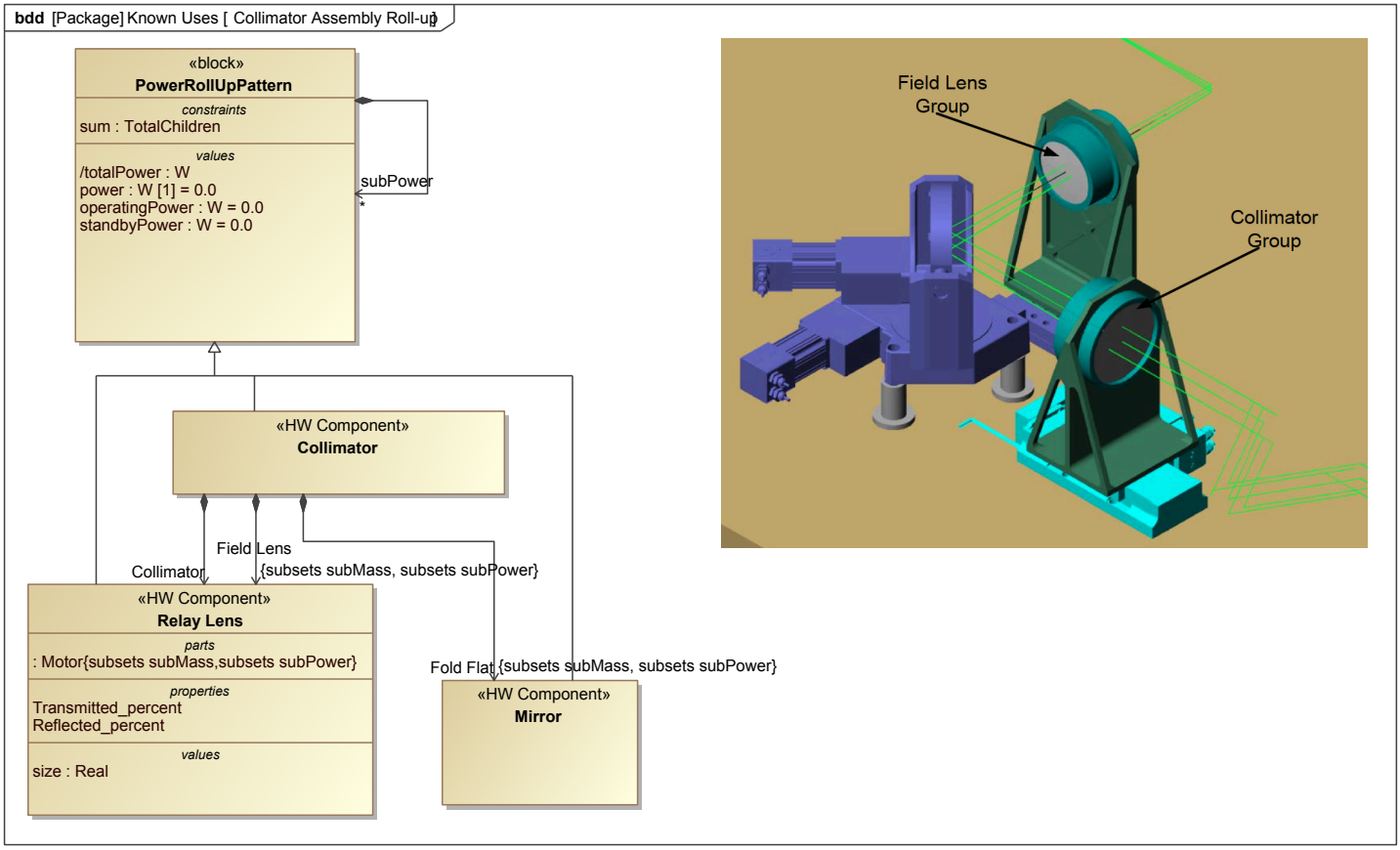


Figure 67. Collimator Assembly Roll-up

Figure 68. Simulation Roll-up

Table 10. Satisfy Peak Power Result

Name	enc : Check Peak Power Compliance	fac : Check Peak Power Compliance	powerPeakLimitEnclosure : W	powerPeakLimitSummitFacilityBuildings : W	peakPowerEnc : W	p
satisfy Peak Power Explanation at 2016.03.29 12.49	pass	pass				
peak Power Limit Scenario Online at 2016.04.30 00.49					420.0	5
peak Power Limit Scenario Online at 2016.04.30 00.52					420.0	5
peak Power Limit Scenario Online at 2016.04.30 00.54					420.0	5

Name	enc : Check Peak Power Compliance	fac : Check Peak Power Compliance	powerPeakLimitEnclosure : W	powerPeakLimitSummitFacilityBuildings : W	peakPowerEnc : W	p
peak Power Limit Scenario Online at 2016.07.25 14.28					460.0	5
peak Power Limit Scenario Online at 2016.09.29 15.38					460.0	5
peak Power Limit Scenario Online at 2017.02.26 18.29					460.0	5
peak Power Limit Scenario Online.aps operational blackbox specification jpl.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl1.peak Power Limit Requirement JPL			8100.0	4100.0		

Name	enc : Check Peak Power Compliance	fac : Check Peak Power Compliance	powerPeakLimitEnclosure : W	powerPeakLimitSummitFacilityBuildings : W	peakPowerEnc : W	p
peak Power Limit Scenario Online.aps operational blackbox specification jpl1.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl2.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl2.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl3.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl3.peak Power Limit Requirement TMT			8500.0	4200.0		

Name	enc : Check Peak Power Compliance	fac : Check Peak Power Compliance	powerPeakLimitEnclosure : W	powerPeakLimitSummitFacilityBuildings : W	peakPowerEnc : W	p
peak Power Limit Scenario Online.aps operational blackbox specification jpl4.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl4.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl5.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl5.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps realization.peak Power Limit Requirement JPL			8100.0	4100.0		

Name	enc : Check Peak Power Compliance	fac : Check Peak Power Compliance	powerPeakLimitEnclosure : W	powerPeakLimitSummitFacilityBuildings : W	peakPowerEnc : W	p
peak Power Limit Scenario Online.aps realization.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps realization1.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps realization1.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps realization2.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps realization2.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps realization3.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps realization3.peak Power Limit Requirement TMT			8500.0	4200.0		

Name	enc : Check Peak Power Compliance	fac : Check Peak Power Compliance	powerPeakLimitEnclosure : W	powerPeakLimitSummitFacilityBuildings : W	peakPowerEnc : W	p
peak Power Limit Scenario Online.aps realization4.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps realization4.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps realization5.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps realization5.peak Power Limit Requirement TMT			8500.0	4200.0		

4.1.2.7 Analysis

The Recursive Static Roll-up pattern helps to analyze a system model by demonstrating that the resource usage according to the system design satisfies a corresponding requirement for every defined specified scenario.

After applying the Roll-up pattern to the system model, an analysis context needs to be specified. In this analysis context, an operational scenario (in either a sequence or activity diagram) and the configuration values need to be specified either through redefinition or instance specifications. The configuration of the quantitative properties of a system model for a particular analysis.

The operational scenarios and configurations of the configured analysis can be run using a simulation engine, such as Cameo Simulation Toolkit. This provides several outputs such as a rolled up resource and margin or value profiles, which show the total value over time of the rolled-up value. These products represent different views that can be used by the engineer to analyze the results.

4.1.2.8 Tooling

The following tooling supports the modeler in the implementation of the Recursive Static Roll-up pattern in MagicDraw.

4.1.2.8.1 Cameo Simulation Toolkit

NoMagic's Cameo Simulation Toolkit (CST) is implemented and supports the Recursive Static Roll-up Pattern by providing a method to perform analysis of a modeler's system.

After the roll-up pattern has been augmented to the system design, the values for the static property of the components participating in the roll-up need to be specified either through redefinition or instance specifications, and then analysis can be performed on the system by running a simulation with CST.

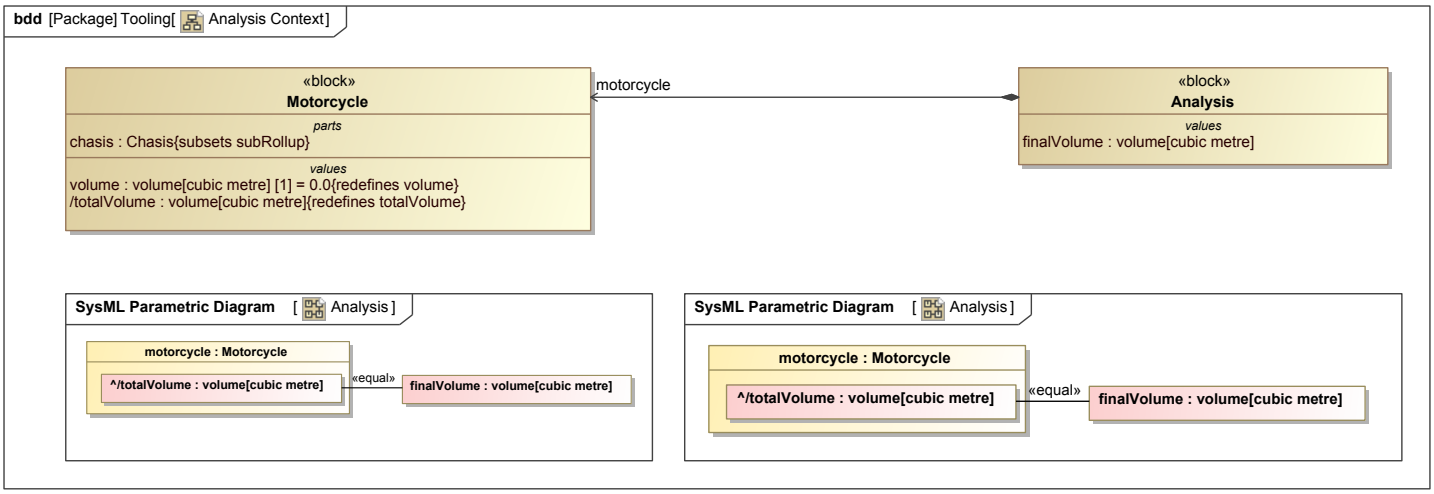


Figure 69. Analysis Context

An Analysis block can be be created to separate the concerns of systems design, its analysis, and to display the results of the static property propagation. A parametric model must be created to constrain the finalVolume to equal the total static property (totalVolume) of the roll-up.

To perform a simulation of the system, the modeler can either run a simulation from a component in the system design or to create a simulation configuration.

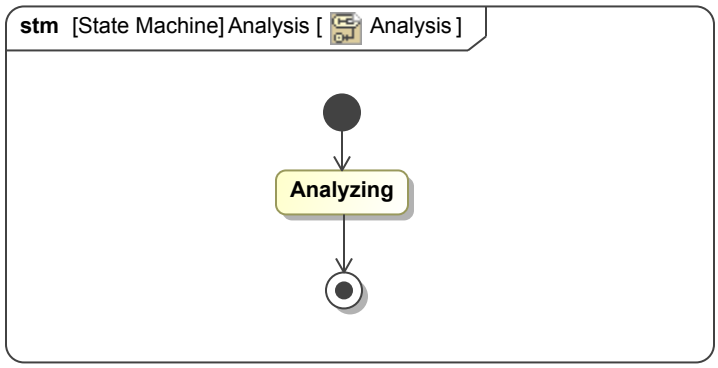


Figure 70. Analysis

To automate the simulation process, the Analysis block needs to own a state machine diagram. In the state machine, there must be a initial and final node with some unspecific state in between. The roll-up is state independent, but the state machine is required if the modeler wants to automatically save the results of a simulation after running a simulation configuration. Without the the state machine the user would need to select the stop button to end the simulation.

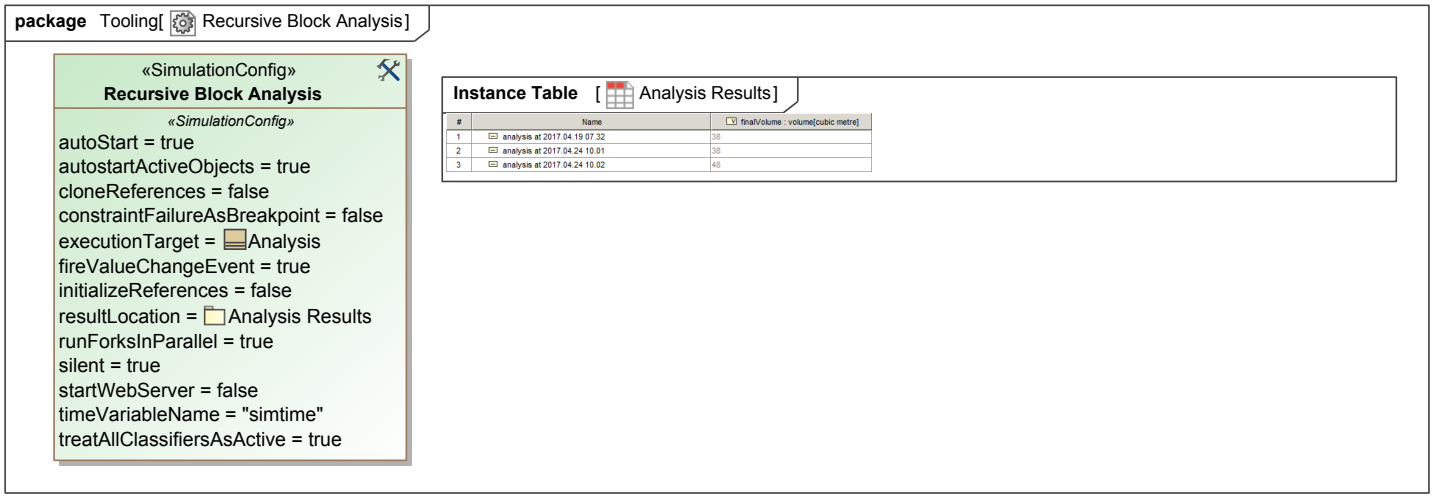


Figure 71. Recursive Block Analysis

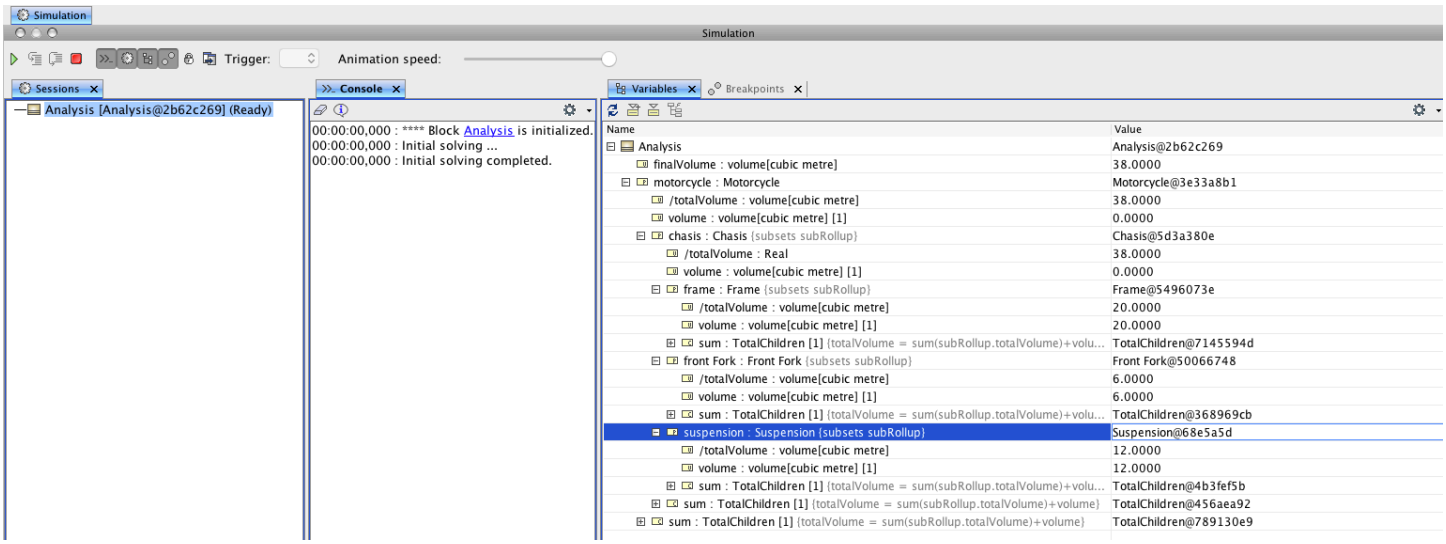
A simulation configuration can be used to automate the simulation process and exports the results as an instance specification. To create a Simulation Configuration, the executionTarget needs to equal the Analysis block (or other top level hierarchical functional component).

After selecting to run the simulation, the resulting instance specification with the total value of the static property for the system can be saved in an instance table. This is done by configuring the property resultLocation to a package. In the Analysis Results instance table, a summary of the resulting final values of volume based on the previously configured blocks can be seen.

An overview of the values for each component during simulation can be seen in the variables section of the Simulation window.

When a Simulation of the Analysis context is initialized, initial solving of the variables occurs. During initial solving, the parametric model and state machine of the Analysis block, and the parametric model of the Recursive Static Roll-up Pattern block are run.

Since very component in the system specializes the Recursive Static Roll-up Pattern block, the constraint expression for each component will be initialized and the values for totalVolume will change. Initially there was no offset value specified, so the default value for volume of each component (Front Fork, Frame, Suspension) is equal to the totalVolume value after initial solving. The value of the totalVolume for the Chassis is equal to the value of its parts (Front Fork, Frame, Suspension).



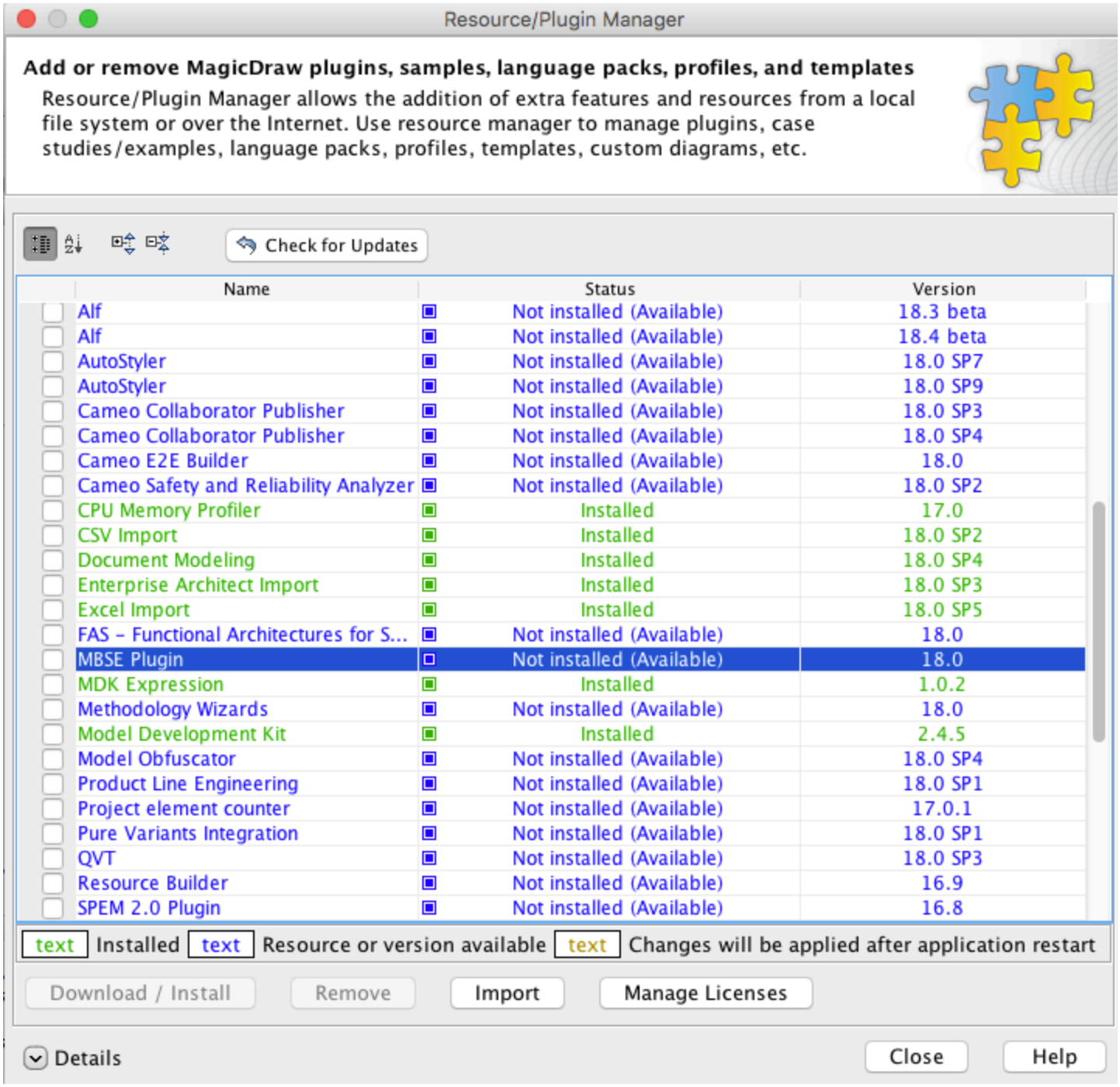
4.1.2.8.2 Model Construction

The general usage of the reasoner component is to aid systems engineers in pattern based reasoning.

Other useful resources that aid the implementation the pattern include NoMagic's Rollup Pattern Wizard for MagicDraw version 18.5, and the MBSE plug-in for MagicDraw version 17.0 sp4.

MBSE Plugin

The MBSE plug-in can also be found in the Resource/Plugin Manager of MagicDraw which is compatible for MD 18.0.



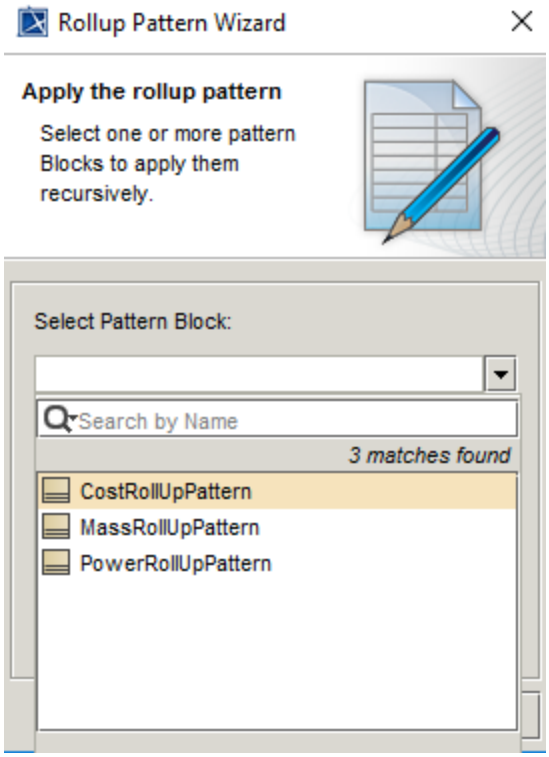
4.1.2.8.2.1 NoMagic Rollup Pattern Wizard

The NoMagic Rollup Pattern Wizard is used to apply the specified rollup pattern to a system. After defining the system and the rollup pattern, the user can apply the NoMagic Rollup Wizard to the system. The user can use the Rollup pattern by following these steps:

Applying a Rollup Pattern

Select the Block (or instance specification) in which you want to apply the pattern to and select **Tools** > **Apply Rollup Pattern**

The Rollup Pattern Wizard window will then appear as shown.



The Rollup Pattern Wizard provides three default rollup pattern blocks for cost, mass, and power (i.e. CostRollUpPattern, MassRollUpPattern, PowerRollUpPattern). The default pattern blocks can be found in the **MD Customization for SysML** auxiliary resource. The wizard also allows for customized Pattern Blocks as well.

Creating Customized Pattern Blocks

There are two methods for creating a customized pattern block that is specific to certain type of roll-up analysis.

1. Copy and pasting a default rollup pattern block from the MD Customization for SysML auxiliary resource
    1. Copy any existing rollup pattern block (i.e. CostRollUpPattern, MassRollUpPattern, PowerRollUpPattern)
    2. Paste into appropriate location of the model
    3. Modify the pattern block
        1. Change name, value properties (make sure to change the types of the value properties), part property, constraint block, and parametric diagram
2. Creating a Roll-up block from our pattern and applying the <<Pattern>> stereotype
    1. Follow the structure or copy elements of/from the OpenSE Cookbook/OpenSE Library
    2. Paste into appropriate location of the model
    3. If copying, modify the pattern block
        1. Change name, value properties (make sure to change the types of the value properties), part property, constraint block, and parametric diagram
    4. If creating the pattern block on own, modify the pattern block
        1. Set name, value properties (types of the value properties), part property, constraint block, and parametric diagram
        2. Apply <<Pattern>> Stereotype
            1. Set the stereotype of the pattern block to <<Pattern>>. The stereotype is located in the MD Customization for SysML auxiliary resource. This step will allow the Rollup Pattern Wizard to recognize the pattern block.

4.1.2.9 Related Patterns

List of other patterns that are related to the Recursive Static Roll-up pattern, and the differences and similarities between them.

Table 11. Similarities and Differences

Pattern Name	Similarities	Differences
Simple Static Roll-up Pattern	A similarity between the Recursive Static roll-up and the Simple Static roll-up pattern is that they both propagate the values of a static property in a system. This means the value of the property does not change over time due to its behavior.	The differences between the Recursive Static roll-up and the Simple Static roll-up pattern is in the application and relationships of the pattern to a system. When applying the Recursive Static Roll-up, the system can be structured in a nested compositional hierarchy. However, the system in which the Simple Static Roll-up can be applied only to an un-complex structure where there is a flat listing of components. The aspect block of the Recursive Static roll-up needs to be specialized by every component in the structure and subsetted by its subRollup part property. While the aspect block of the Simple Static Roll-up is a whole-part relationship to only the top level component/ system.
Dynamic Roll-up Pattern	A similarity between the Recursive Static roll-up and the Dynamic Roll-up is that they both have sub component that is typed by itself (subRollup). The composition relationships between the components of each system need to subset the part property.	The primary difference between the Recursive Static roll-up and the Dynamic roll-up pattern is due to the system's behavior. The total value of the final dynamic property of the Dynamic Roll-up is equal to the component's values during a certain event in its lifecycle. Meanwhile, the value of the final static property of the Recursive Static roll-up doesn't depend on the state of the components.

4.2 Dynamic Roll-up Pattern

4.2.1 Intent

Roll-up calculations of system resources (e.g. total mass, cost, power, or another system dimension) are among the most common use cases in systems engineering. System engineers want to perform calculations based on the specific values of all the components in the system. The Dynamic Roll-up pattern provides system modelers a solution to solve common domain issues, and consists of standard constructs that are applicable to most domains. If the modeler desires to perform analysis of a system where the values of the subsystem are dynamically changing based on the system's behavior the Dynamic Roll-up pattern should be reused and applied in their domain specific construct.

4.2.2 Motivation

The overall motivation for system engineers to apply the pattern is the desire to have requirements that are traced to to design artifacts, and to have a method to assert the system design satisfies a set of requirements. Through SysML patterns, analysis specifications can be defined and the modeler can verify, for instance, whether the resource consumption of a system satisfies an initial requirement. If a system modeler wants to perform analysis of a complex system that consists of components whose value properties are behavior dependent, the component's values depend on the operating state, and the modeler is familiarized with running simulations the Recursive Static Roll-up Pattern can be configured to their system.

The Dynamic Roll-up supports the analysis of a system design which is dependent on the component's operational states. The Dynamic Roll-up pattern can be applied to a model in which the product breakdown structure of the system is specified.

The components aggregated in the roll-up must share a common set of operational specific value properties that change dynamically throughout the component's life-cycle. Through the application of the pattern, each component's dynamic values can be recursively propagated up a hierarchy of components characterized by these values.

To apply the pattern to a system, the modeler needs to augment the components of the system design for a particular resource in order to propagate the values up the system hierarchy. The pattern must be configured to the system to specify the analysis context using the system's design, states, properties, relationships, and operational scenarios. Only then can the pattern provide a context as to whether the design satisfies a set of formalized requirements.

4.2.3 Concept

The SysML structure of the Dynamic Roll-up pattern consists of the Dynamic Resource Roll-up Aspect block where the properties and behavior of the system are defined. The Dynamic Roll-up pattern consists of the Dynamic Resource Roll-up Aspect block which is parent to the constraint sum, a state machine in which the flow of operations is specified, and parametric model where the value properties are constrained to perform calculations.

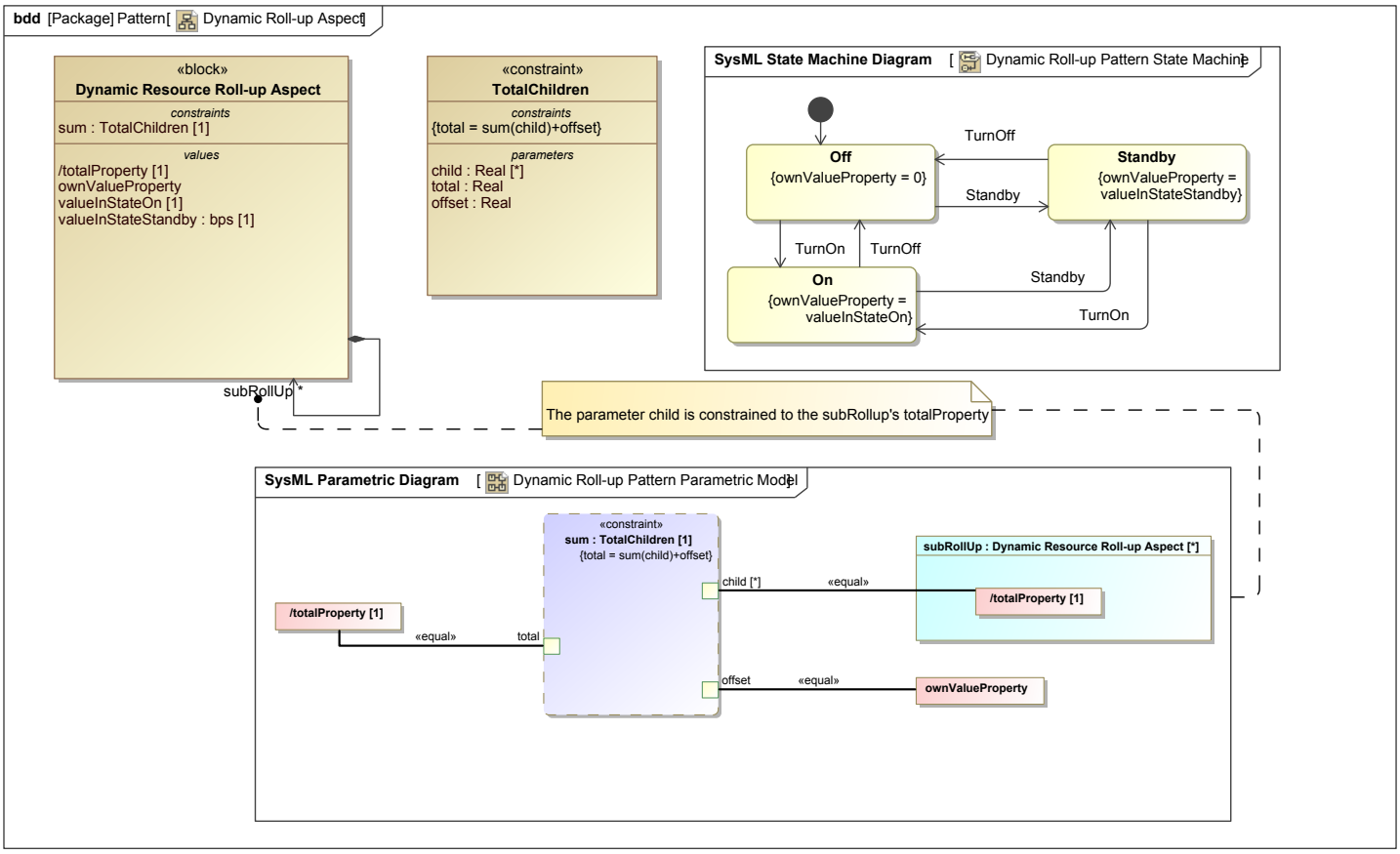


Figure 72. Dynamic Roll-up Aspect

The Dynamic Resource Roll-up Aspect has the value properties totalProperty, ownValueProperty, valueInStateOn, and valueInStateStandby that are propagated in the roll-up, and a recursive part property named subRollup. The aspect block is parent to a parametric model where the properties are constrained. In the parametric model the owner's constraint, sum, is constrained to the expression total = sum(child)+offset. For each component that specializes the Recursive Static Resource Roll-up Aspect block the value properties totalProperty and ownValueProperty will be constrained. The parameter, child, is equal to the derived value property totalProperty of the subRollUp, and is denoted with the multiplicity * to represent that there can be an infinite amount of components. The parameter offset is equal to the property ownValueProperty. The constraint's parameter, total, which calculates the sum of the children is equal to the derived value property totalProperty. During simulation, the value of ownValueProperty for a component depends on its operating state. The operating states valueInStateOn and valueInStateStandby become equal to ownValueProperty.

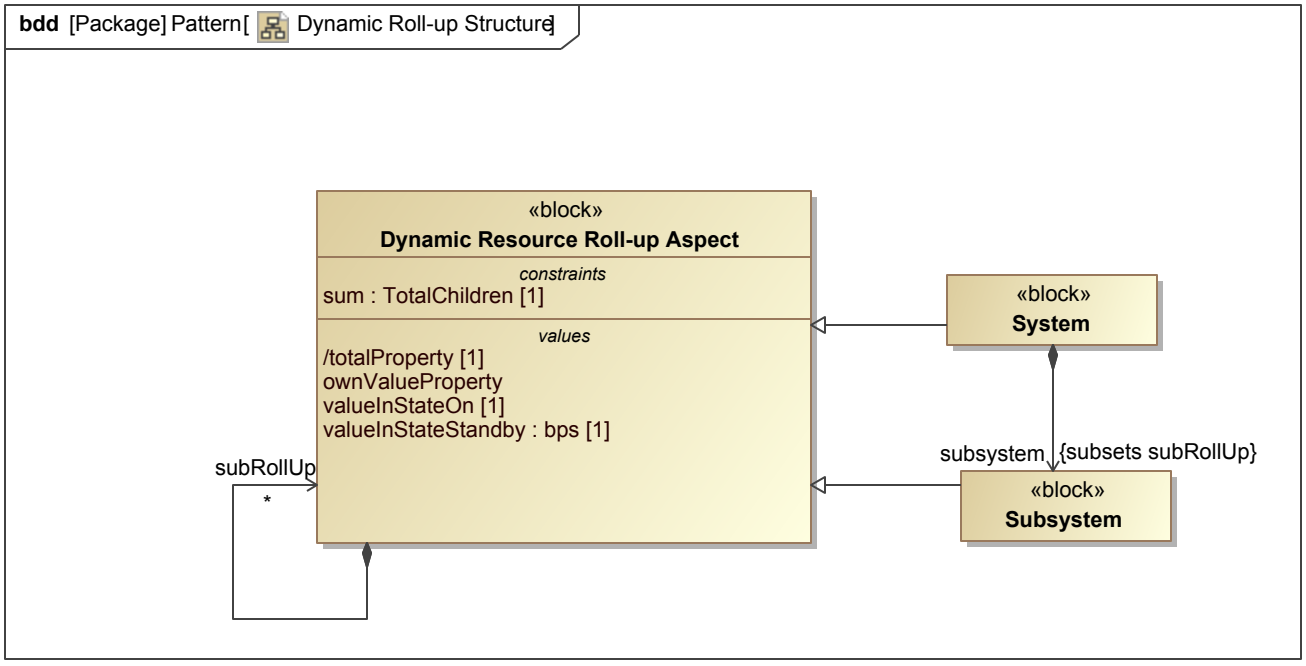


Figure 73. Dynamic Roll-up Structure

The SysML structure of the Dynamic Roll-up Pattern is dependent on the properties of the Dynamic Resource Roll-up Aspect block and its relationship to components in the modeler's system. To calculate the values of each component of the system each block participating in the roll-up must specialize the Dynamic Resource Roll-up Aspect block. The relationship between the subcomponents/ subsystems and a system is a part-whole relationship, and must subset the subRollup part property of the Dynamic Resource Roll-up Aspect.

Table 12. <>

Model Element	Role
Dynamic Resource Roll-up Aspect	The Roll-up Pattern block owns the elements the part property subRollUp, the constraint sum, and the value properties totalProperty (derived), valueProperty, valueInStateOn, and valueInStateStandby. The block owns all values that will be redefined throughout the system, and after simulation the values will be propagated. The pattern is considered recursive because the Roll-up Pattern block consists of a part subRollUp which is of the same type. The subRollUp can be applied to an infinitesimal amount of children.
totalProperty	The inherited value property, totalProperty, is the product of the sum of all the children in the system hierarchy. The totalProperty is final value for the roll-up calculations of the system.
ownValueProperty	The local value property, ownValueProperty, of the Dynamic Resource Roll-up Aspect block is used to offset the total value of a child. During a system's lifecycle, the operational value property becomes equal to ownValueProperty.
valueInStateOn	The behavior dependent value property, valueInStateOn, of the Dynamic Resource Roll-up Aspect block is used to specify a components value in the operational state TurnOn. If during a system's lifecycle the operational state is in TurnOn, the property ownValueProperty is equal to valueInStateOn.
valueInStateStandby	The behavior dependent value property, valueInStateStandby, of the Dynamic Resource Roll-up Aspect block is used to specify a components value in the operational state Standby. If during a system's lifecycle the operational state is in Standby, the property ownValueProperty is equal to valueInStateStandby.
Value Property	The value properties of the Dynamic Resource Aspect consist of operational, local, and global properties that are propagated among a hierarchy of components to calculate a system resource.
Dynamic Resource Aspect	The Dynamic Resource Aspect contains the value properties that are to be propagated in the roll-up, a state machine where the state-dependent behavior is defined, and a parametric model where the properties are constrained.

4.2.4 Consequences

There are several trade-offs to consider when configuring the Dynamic Roll-up pattern to the modeler's system.

Action	Consequence
Redefining the values of the operational specific components	By redefining the static values of the system's components the modeler can easily change the inherited attributes by redefining and modifying the values. If the static value of a component isn't redefined, the value will default to the value of the base class. However one could argue that redefinition is the proper approach to specifying the components attributes because the modeler can view and modify the inherited aggregated value properties of it's classifier.
Creating instance specifications to define the values of the operational specific components	After all components in the system have been configured to generalize the Roll-up pattern block every component in the system will inherit the pattern's value properties.
Modifying the system model when applying the roll-up pattern to the system	A consequence of needing to modify the system design result in an authority/ownership problem. A SysML work around is to create an instance tree of the product breakdown structure, and to only apply the pattern at the instance level.
Subsetting all composition relationships of the system's hierarchy	The modeler must subset every element in the hierarchy which can become burdensome depending on the number of components and relationships in the system. A possible solution would be to use a reasoning tool to automate this process.
The state machine is owned by the Resource aspect and is inherited by the components of the system	Every component of the system specializes the roll-up aspect which owns a state machine where the behavior of the system is specified. These components will then inherit this state machine. This can be a concern if a component doesn't share the same states that are specified in the state machine. A modeler needs to create a new state machine for this component which will modify the classifier behavior. See State Machine Redefinition as example how to modify the classifier behavior in this case.

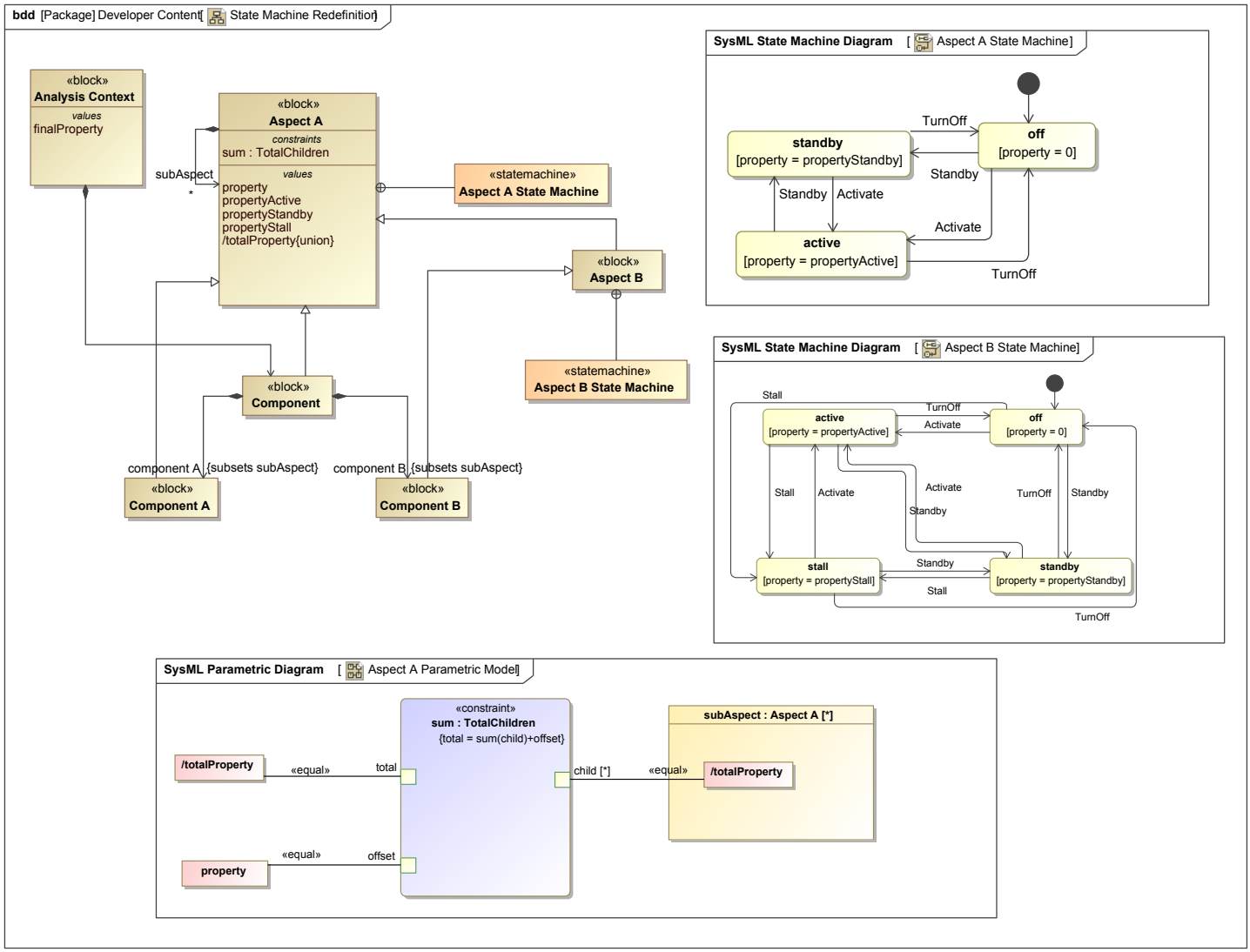


Figure 74. State Machine Redefinition

Conflicts that occur while applying or using the Dynamic Roll-up Pattern.

Usage of the Pattern	Explanation of Conflict
Configuring the Dynamic Roll-up pattern to a system without subsetting the relationship between components of the system.	The composition relationships between components of a hierarchical system needs to subset the Dynamic Roll-up Aspect block's part property subRollup. Without subsetting the relationship the dynamic values will be unable to propagate. If after performing a simulation of the system and the expected resulting final value is 0 navigate to either the structure of the system or the simulation configuration window. A likely cause for the error and the resulting value is due to an un-subsetted relationship.

4.2.5 Implementation

An example of how the Dynamic Roll-up Pattern can be applied is to calculate the total data consumption of a Hybrid Sport Utility Vehicle's subsystem components (Brake Control Module, Powertrain Control Module, Body Control Module). The analysis needs to explain (by simulation) the total data consumption of the system according to the design.

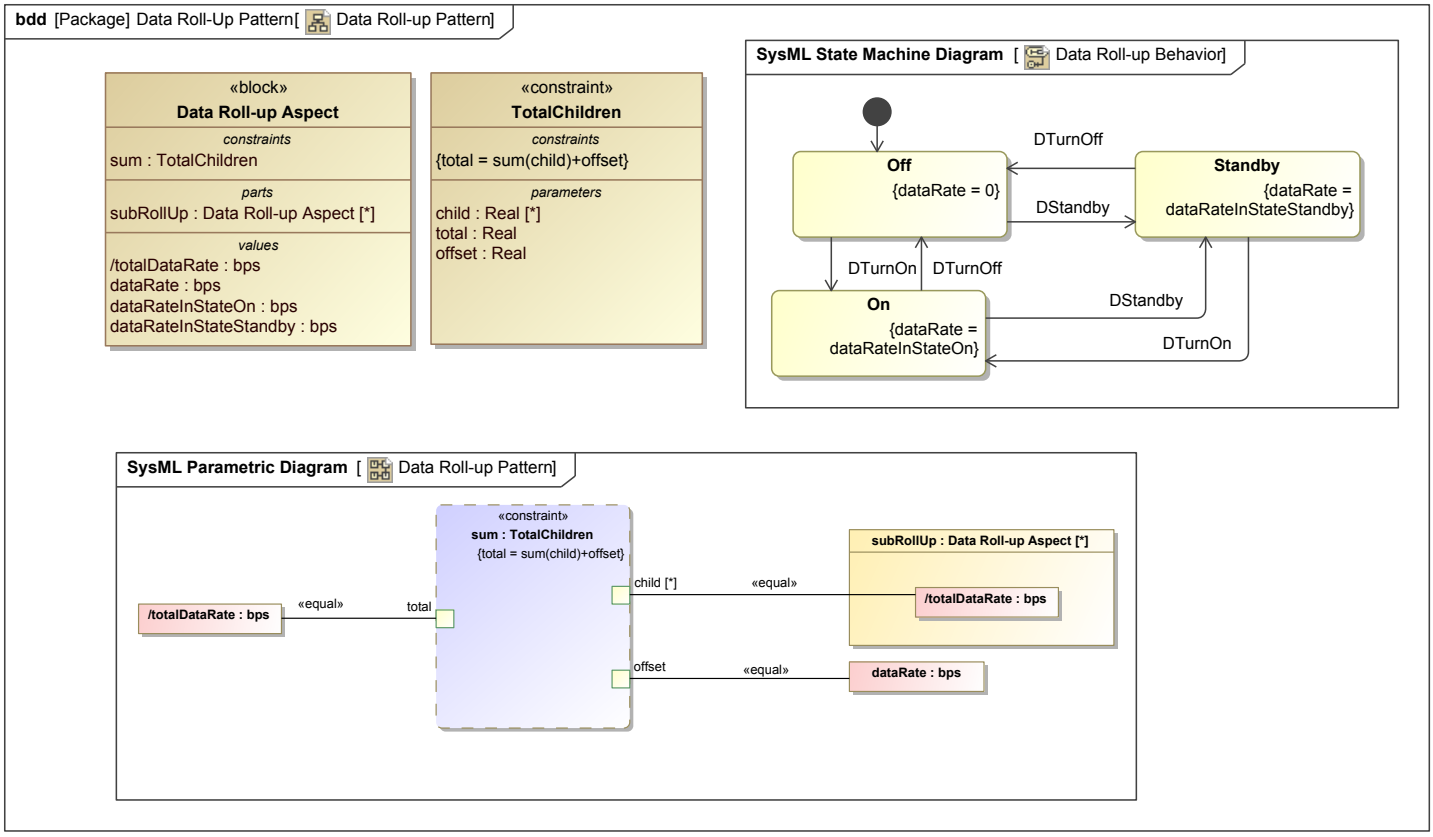


Figure 75. Data Roll-up Pattern

The naming of the value properties of the Dynamic Roll-up pattern block need to be modified to the main property (data) that is being summed in the system. The properties of the Data Roll-up Pattern block then become totalDataRate, dataRate, dataRateInStateOn, and dataRateInStateStandby.

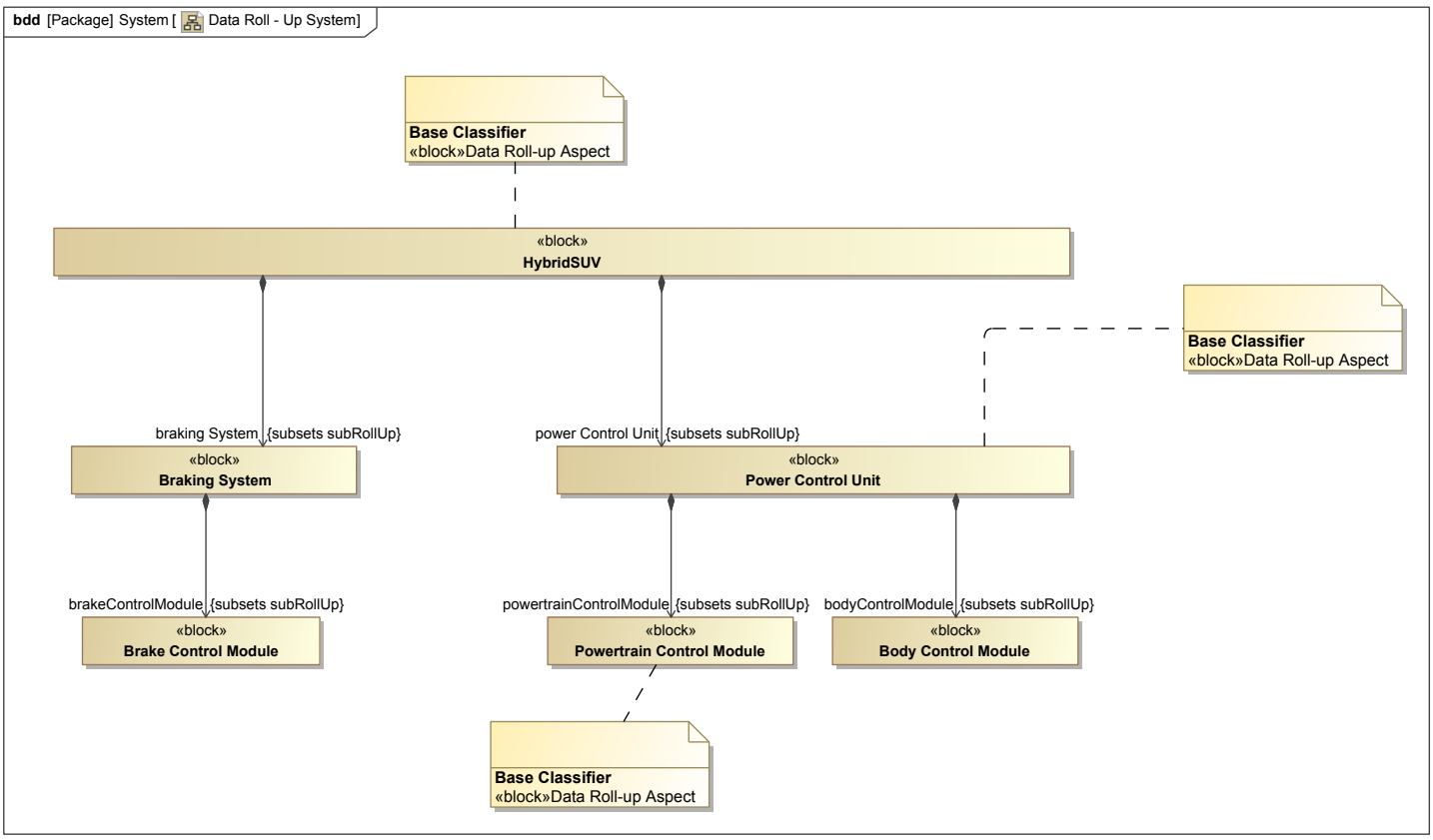


Figure 76. Data Roll - Up System

The first step is to specify the design of the system and its decomposition. The subsystems (Braking System, Power Control Unit) are composed of physical components Powertrain Control Module, Brake Control Module, and Body Control Module. After specifying the design, the next step is to modify the system with the Dynamic Roll-up analysis pattern. When using the Dynamic Roll-up pattern, every element in the composition hierarchy whose data properties are to be summed need to generalize the Dynamic Roll-up block. The composition relationships between the physical components need to be subset with the subRollUp part property of the Dynamic Roll-up block. The relationships in the system need to be subset because the the value of the subcomponent property , totalDataRate, is computed as the union of the values of all properties that subset it. Therefore, all components participating need to have their subcomponent properties subset by this property.

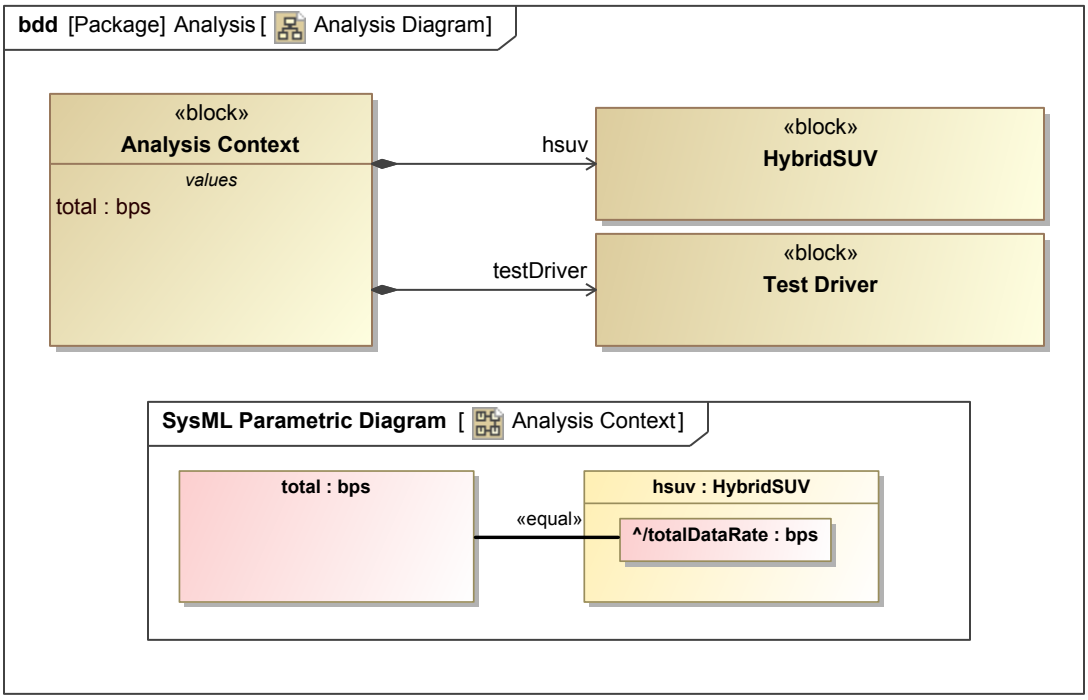


Figure 77. Analysis Diagram

After configuring the Roll-up pattern to the system an analysis context needs to be specified. In particular, the block Analysis Context is used to represent this and is defined in it's owning parametric model. The analysis context is composed of a black box (Analysis Driver) where the analysis is performed and a component (HybridSUV). In addition, concrete analysis blocks corresponding to operational scenarios are needed to be defined. The analysis block (Analysis Driver) specializes the abstract analysis blocks. The Analysis Driver inherits the Analysis Context block's parametric model and defines a sequence diagram that serves as a scenario driver. The scenario driver can become a use case that can be analyzed using Cameo Simulation Toolkit. After defining the behavior of the Analysis Driver block, the Analysis Context will apply the signals and automate the simulation process.

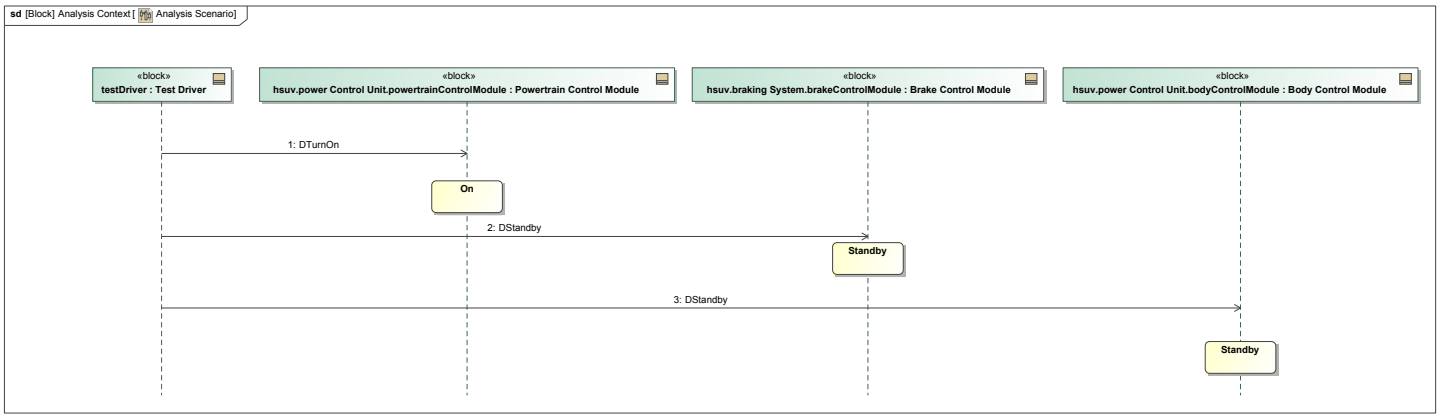


Figure 78. Analysis Scenario

To preconfigure the operational states of the system and to automate the simulation process the Analysis Context block needs to define a sequence diagram that serves as a scenario driver. The scenario driver becomes a use case that can be analyzed using Cameo Simulation Toolkit.

The Test Driver lifeline is required to send messages (signals) to the other components whose dynamic properties are specified. Due to this sequence diagram, the value of the total data rate will be equal to the value of the dataRateInStateOn of the Powertrain Control Module, the value of the dataRateInStateStandby of the Brake Control Module, and the value of the dataRateInStateStandby of the Body Control Module.

Table 13. Analysis Configuration

Name	dataRateInStateOn : bps	dataRateInStateStandby : bps
analysis Context.hsuv.braking System.brakeControlModule	6.0	5.0
analysis Context.hsuv.power Control Unit.bodyControlModule	8.0	3.0
analysis Context.hsuv.power Control Unit.powertrainControlModule	10.0	5.0

Table 14. Dynamic Analysis Results

Name	total : bps
analysis Context at 2017.04.26 11.15	18.0

4.2.5.1 Recursive Activity Example

In this example, we will apply the Recursive Dynamic Roll-up to a notional coffee machine in order to demonstrate how the pattern can be applied to properties of activities.

This example model represents a notional Coffee Machine consisting of a Coffee Grinder, Coffee Pot, Brewer, and Water Heater as shown below. The classifier behavior of Coffee Machine is the activity Make Coffee which consists of actions with the behaviors Grind Beans, Heat Water, Brew Coffee, and Keep Warm. In the activity diagram pictured below, swim lanes are used to allocate the actions in the coffee making process to their associated structural element. For example, the action with the behavior Grind Beans is allocated to the block Coffee Grinder. It is important to note that the swim lane allocation mode should be set to 'usage'. This setting may be toggled in the in the pop-up menu after right clicking the swim lane.

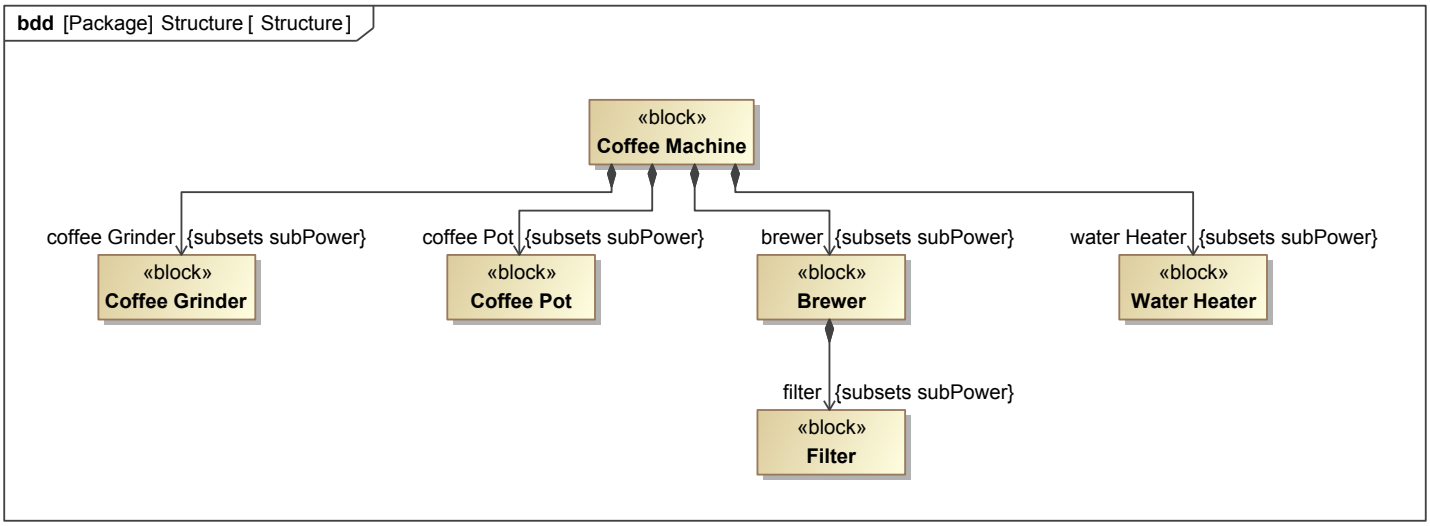


Figure 79. Structure

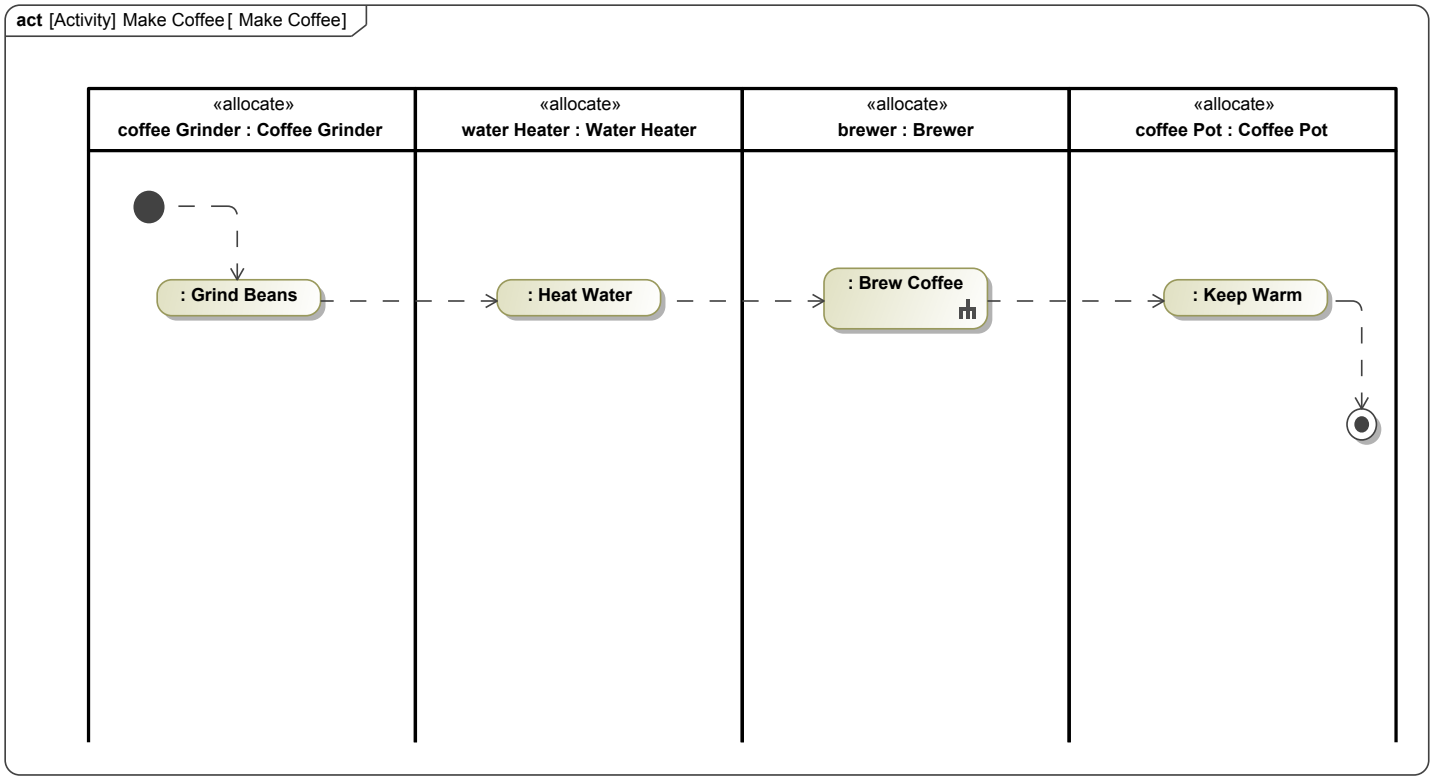


Figure 80. Make Coffee

In the case where there are nested activities, such as the for the Brew Coffee activity, the swim lane allocation must match the structure as demonstrated in the diagram below. Since Filter Coffee is an action within Brew Coffee, the swim lane allocates the action to the structural block Filter as well as to the parent structural block Brewer. Again, swim lane allocation mode should be set to 'usage'.

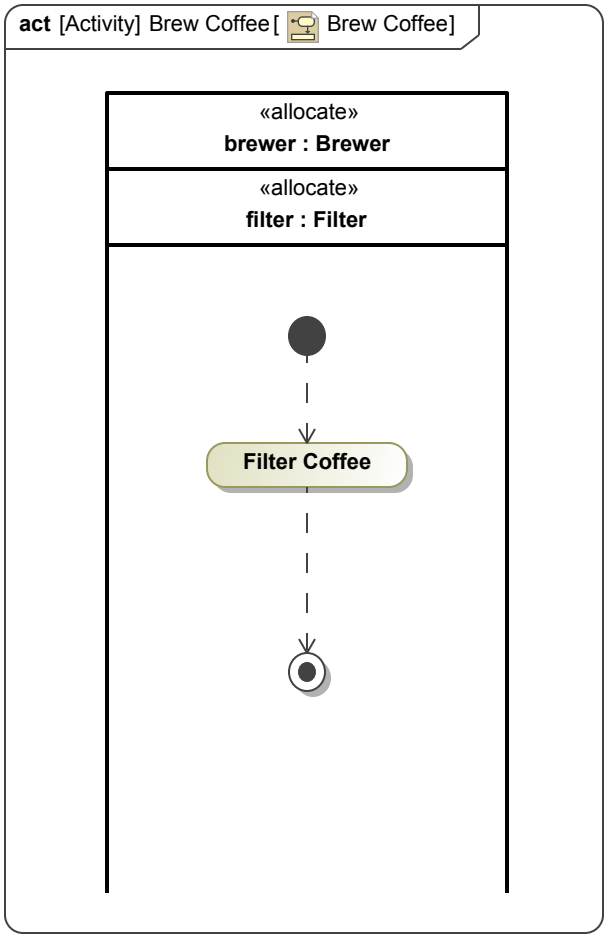


Figure 81. Brew Coffee

We wish to apply the Recursive Dynamic Roll-Up pattern such that the power used for each action in the coffee making process may be tracked at the system level over time. This use case is unique when compared to previous examples that applied the pattern to blocks and instances of blocks as opposed to activities. To apply the pattern to activities, a mechanism must exist for relating the the 'power' value property specified in the roll up pattern to a power draw property associated with each activity in the coffee making process. This mechanism is achieved using properties on activities, constraints, and inheritance.

Below is an activity decomposition of the activity Make Coffee which shows the conversion of the Make Coffee activity into a Class Diagram. This diagram view provides the ability to represent, analyze, and document activity hierarchies in structure diagrams. From this view, it is evident that the Make Coffee activity is composed of Keep Warm, Grind Beans, Brew Coffee, and Heat Water activities. It is also evident that duration constraints have been applied to those activities. For example, the activity Grind Beans has a duration of 30 seconds.

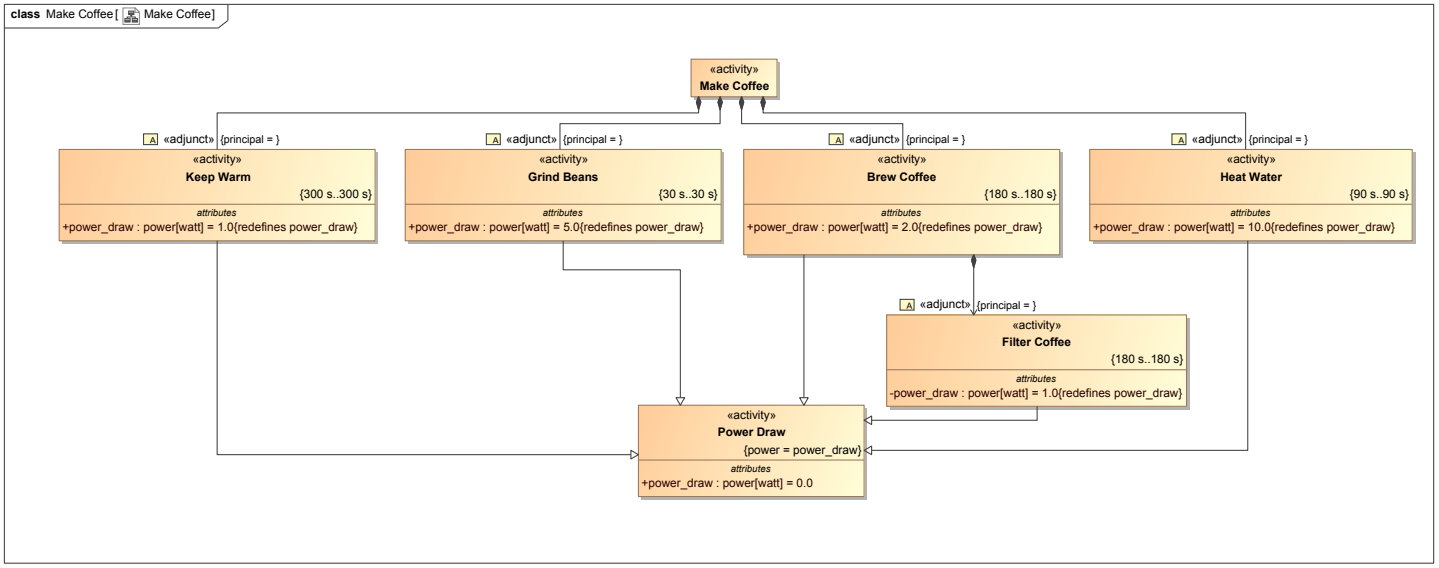


Figure 82. Make Coffee

To construct the mechanism which relates the 'power' value property of the roll-up pattern to a power draw property associated with each activity, a generic Power Draw activity is created from which all leaf activities inherit. The generic activity Power Draw has a property named power_draw which has units of watts and a default value of zero. Each activity that specializes Power Draw also has the power_draw property which redefines the power_draw property of Power Draw, thus enabling each activity to have a unique value for power_draw.

The generic Power Draw activity also has a constraint which enforces the condition that 'power = power_draw'. This constraint is the linkage between the 'power' value property of the Recursive Dynamic Roll-Up pattern and the power_draw properties of each activity. Now, when the Power Roll-Up pattern is applied to the Coffee Machine block, the resulting rolled-up value will be the total power draw of all activities. Furthermore, this constraint explains the categorization of this pattern as a dynamic roll-up rather than a static roll-up. Because the *constraint* sets the 'power' value property in the roll-up pattern equal to the power_draw property of each activity, the power_draw property may dynamically adjust while still maintaining equivalence to the 'power' value property. When values within a subsystem are dynamically changing based on the system's behavior, the pattern is classified as dynamic.

This pattern facilitates the simulation of the Coffee Machine block in order to produce a timeline chart as well as power profile and power-per-time calculation. The simulation may be constructed using the Analysis Context pattern and Simulation Configuration diagrams as shown below.

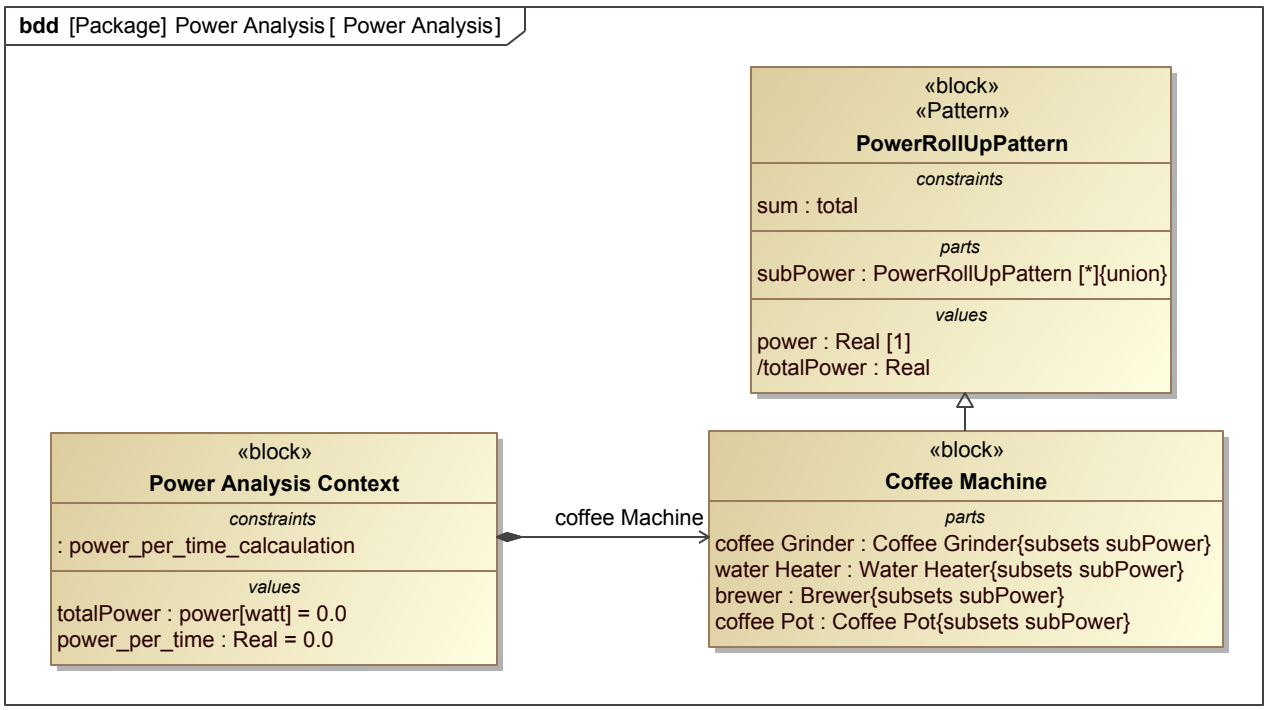


Figure 83. Power Analysis

The Power Analysis Context serves as the parent of the Coffee Machine in order to separate the model analysis from the model structure. The Power Roll-Up pattern is applied to the Coffee Machine such that Coffee Machine inherits from PowerRollUpPattern as shown above. Additional documentation on the power roll-up pattern may be found in the Power Roll-up Pattern section. The parametric diagram (shown below) equates the resulting totalPower of the Coffee Machine to the totalPower of the Power Analysis Context. Using the same concept, additional calculations may be performed by using constrains owned by Power Analysis Context. In this example, a power-per-time calculation is performed which utilizes the 'totalPower' obtained from the Power Roll-Up Pattern.

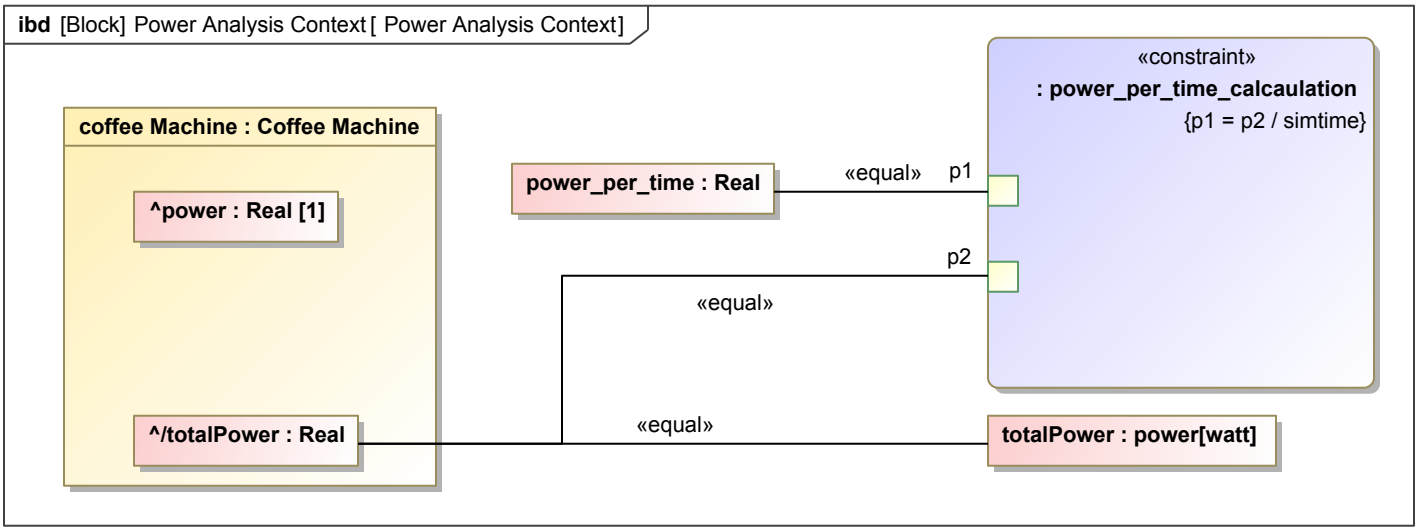


Figure 84. Power Analysis Context

This pattern may be applied in cases where a model structure already exists and the user wishes to maintain separation between the existing model and the simulation model. By leveraging a specialization relationship between the simulation model and the existing model, a copy of the existing model may be created. Elements within the specialized simulation model should redefine elements from the general existing model. The specialization and redefinition of the model may be completed using the Systems Reasoner tool. An example of utilizing specialization and redefinition in order to achieve a separate simulation model is shown in MEV-CBE Power Roll Up Example .

Simulation for a power analysis may be configured using a Simulation Configuration Diagram as shown below. Results may be visualized using timeline charts and time series charts.

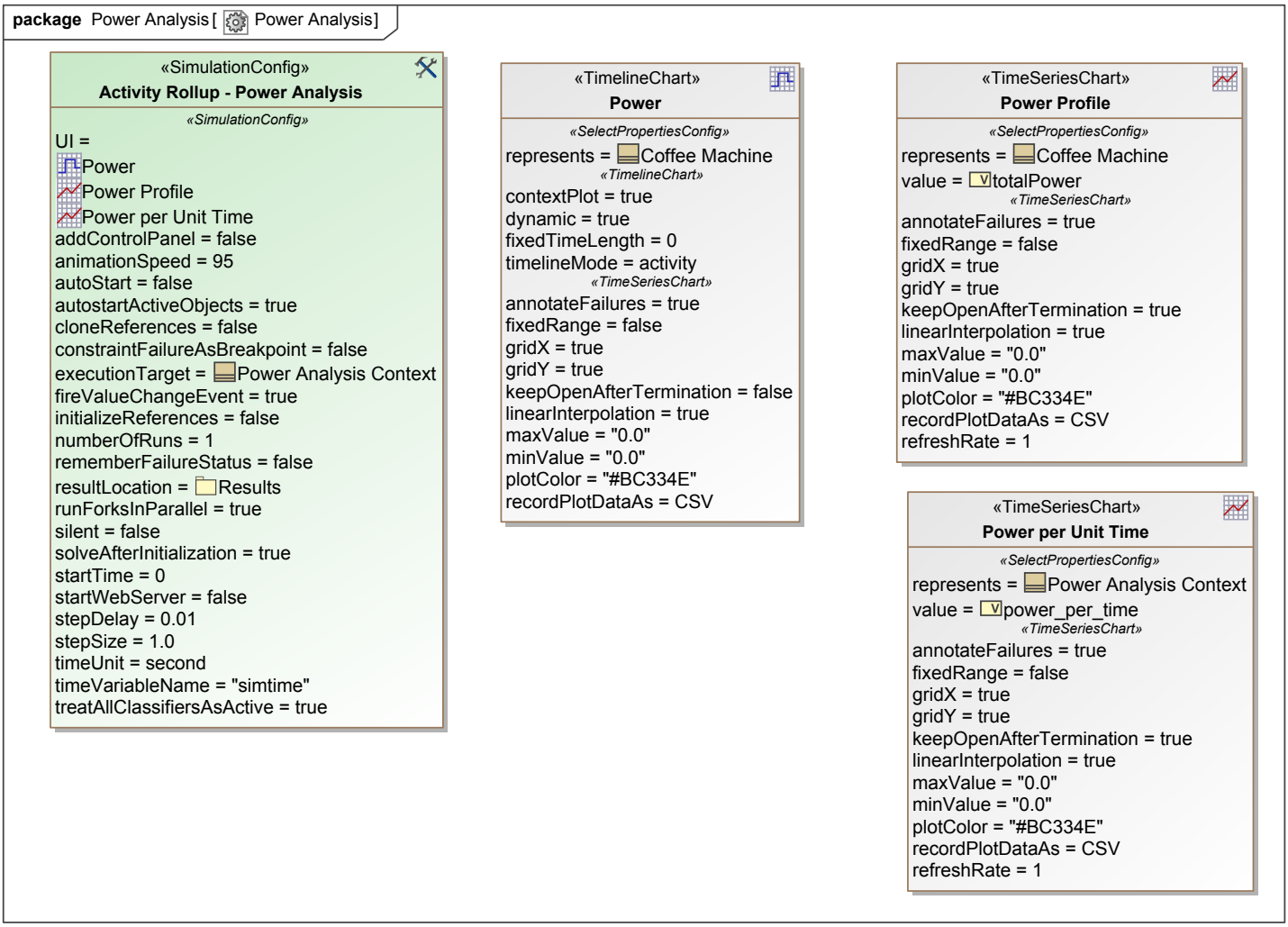


Figure 85. Power Analysis

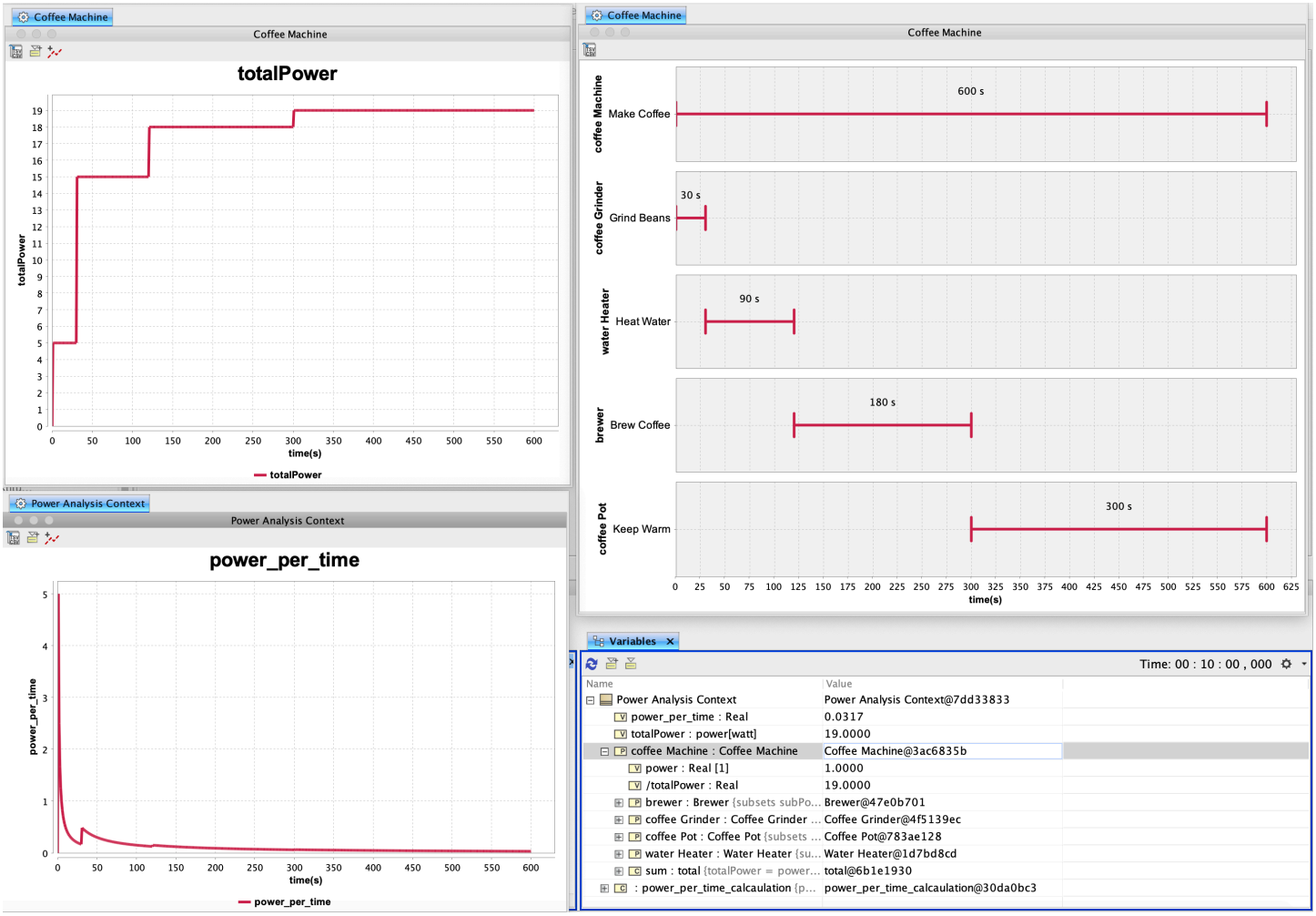


Figure 86. Visualized Simulation Results

Results of the simulation may be consolidated in tabular format as shown below.

Table 15. Power Analysis Results

Name	coffee Machine : Coffee Machine	totalPower : power[watt]	power_per_time : Real
power Analysis Context at 2020.02.24 15.20	power Analysis Context.coffee Machine	18.0	0.03
power Analysis Context at 2020.02.24 15.23	power Analysis Context.coffee Machine1	18.0	0.03
power Analysis Context at 2020.02.24 15.33	power Analysis Context.coffee Machine2	18.0	0.03
power Analysis Context at 2020.02.24 15.53	power Analysis Context.coffee Machine3	18.0	0.03
power Analysis Context at 2020.02.24 15.55	power Analysis Context.coffee Machine5	18.0	0.03
power Analysis Context at 2020.02.24 15.55	power Analysis Context.coffee Machine4	18.0	0.03
power Analysis Context at 2020.02.24 15.56	power Analysis Context.coffee Machine6	18.0	0.03
power Analysis Context at 2020.02.25 08.00	power Analysis Context.coffee Machine7	18.0	0.03

Name	coffee Machine : Coffee Machine	totalPower : power[watt]	power_per_time : Real
power Analysis Context at 2020.02.25 14.52	power Analysis Context.coffee Machine8	18.0	0.03
power Analysis Context at 2020.03.04 16.06	power Analysis Context.coffee Machine9	18.0	0.03
power Analysis Context at 2020.03.04 16.18	power Analysis Context.coffee Machine10	19.0	0.03166666666666667
power Analysis Context at 2020.03.04 16.23	power Analysis Context.coffee Machine11	19.0	0.03166666666666667
power Analysis Context at 2020.03.04 16.26	power Analysis Context.coffee Machine12	19.0	0.03166666666666667
power Analysis Context at 2020.03.04 16.32	power Analysis Context.coffee Machine13	18.0	0.0375
power Analysis Context at 2020.03.04 16.37	power Analysis Context.coffee Machine14	19.0	0.03166666666666667
power Analysis Context at 2020.03.04 16.45	power Analysis Context.coffee Machine15	19.0	0.03166666666666667
power Analysis Context at 2020.03.04 16.50	power Analysis Context.coffee Machine16	19.0	0.03166666666666667
power Analysis Context at 2020.03.20 10.05	power Analysis Context.coffee Machine17	19.0	0.03166666666666667
power Analysis Context at 2020.03.20 10.06	power Analysis Context.coffee Machine18	19.0	0.03166666666666667
power Analysis Context at 2020.03.20 10.06	power Analysis Context.coffee Machine19	19.0	0.03166666666666667
power Analysis Context at 2020.03.20 10.08	power Analysis Context.coffee Machine20	19.0	0.03166666666666667
power Analysis Context at 2020.03.20 10.11	power Analysis Context.coffee Machine21	19.0	0.03166666666666667
power Analysis Context at 2020.03.20 10.13	power Analysis Context.coffee Machine22	19.0	0.03166666666666667

4.2.6 Known Uses

A main objective for TMT related analysis is to provide state-dependent power roll-ups for different operational scenarios and to demonstrate that requirements are satisfied by the design and duration analysis of the operational use cases. In this example, a walk-through of how the TMT applies the Dynamic Roll-up pattern for power analysis will be shown.

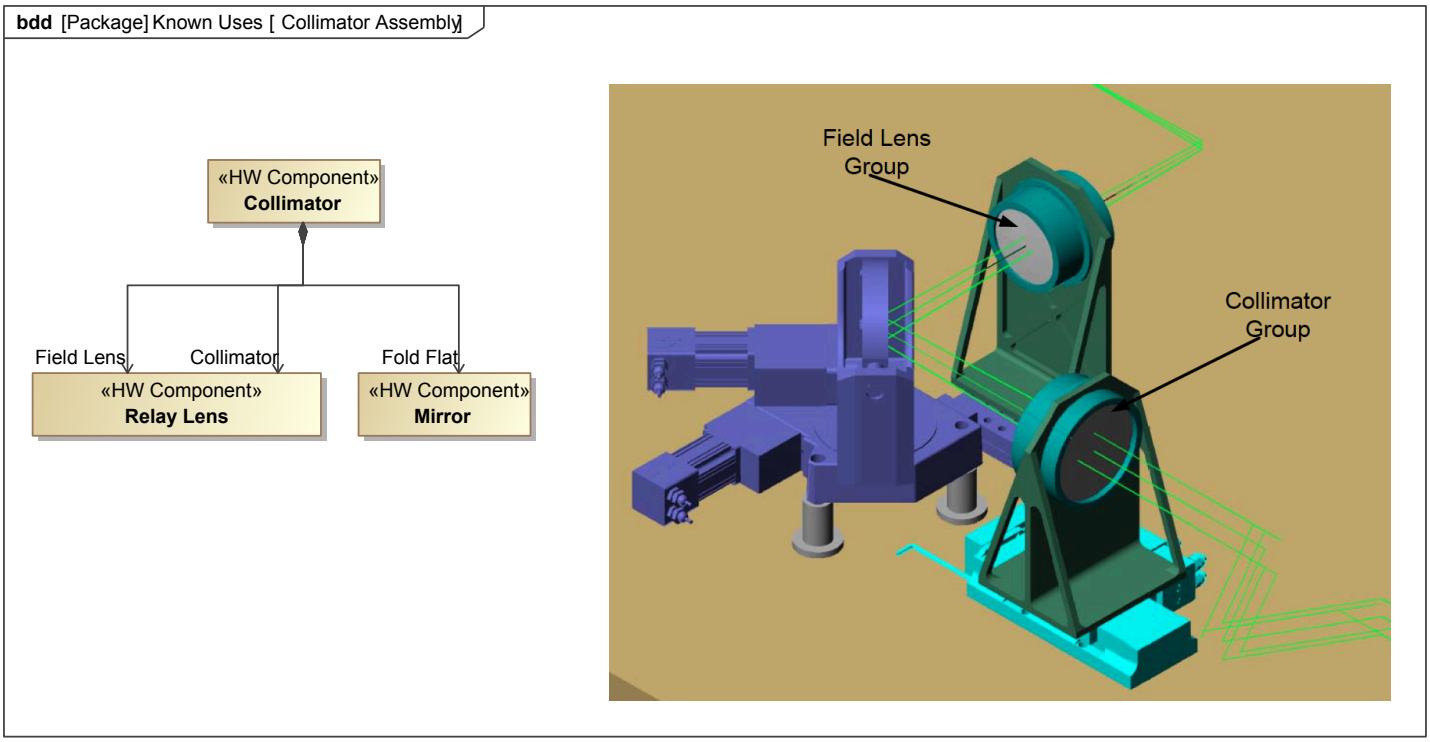


Figure 87. Collimator Assembly

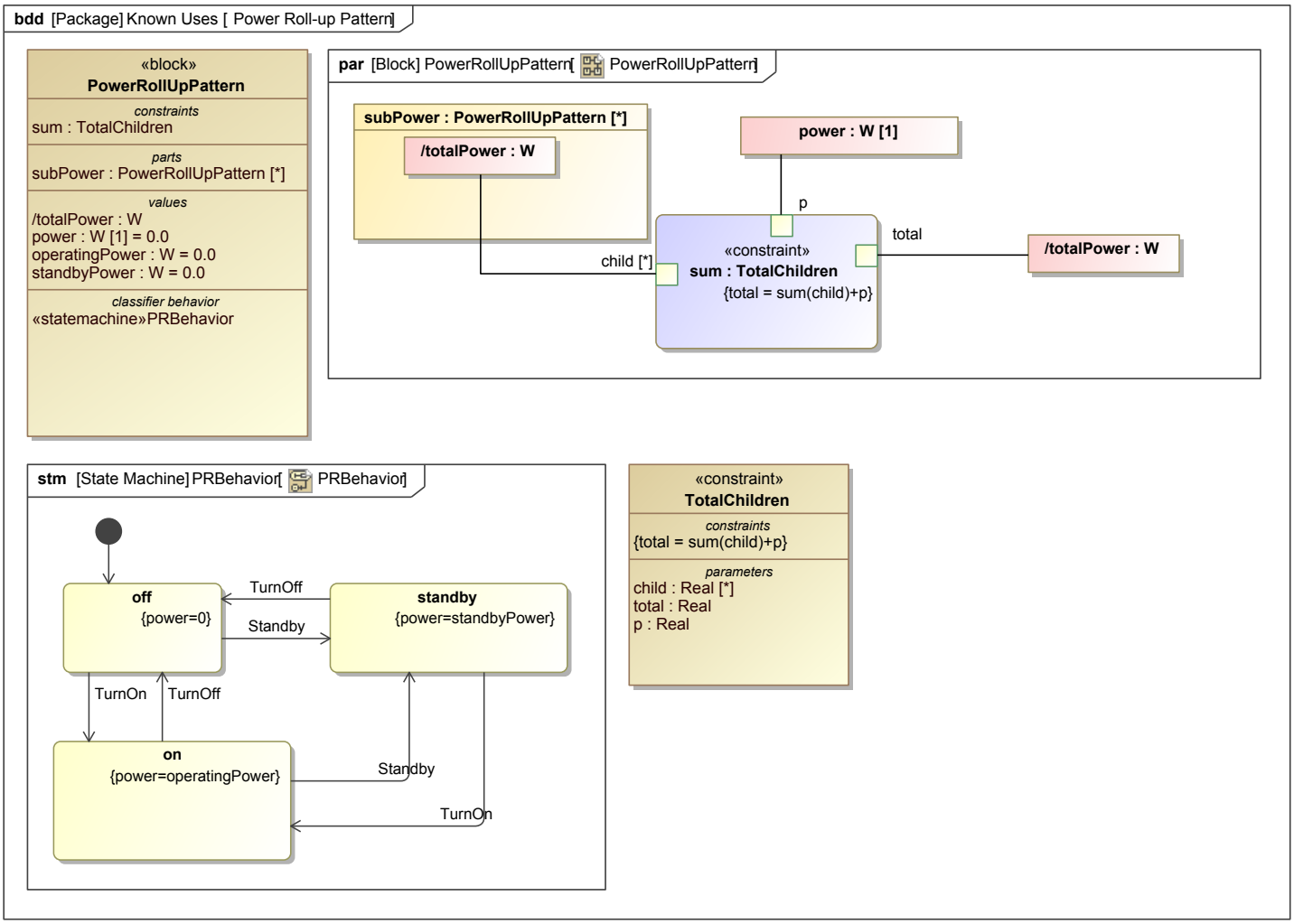


Figure 88. Power Roll-up Pattern

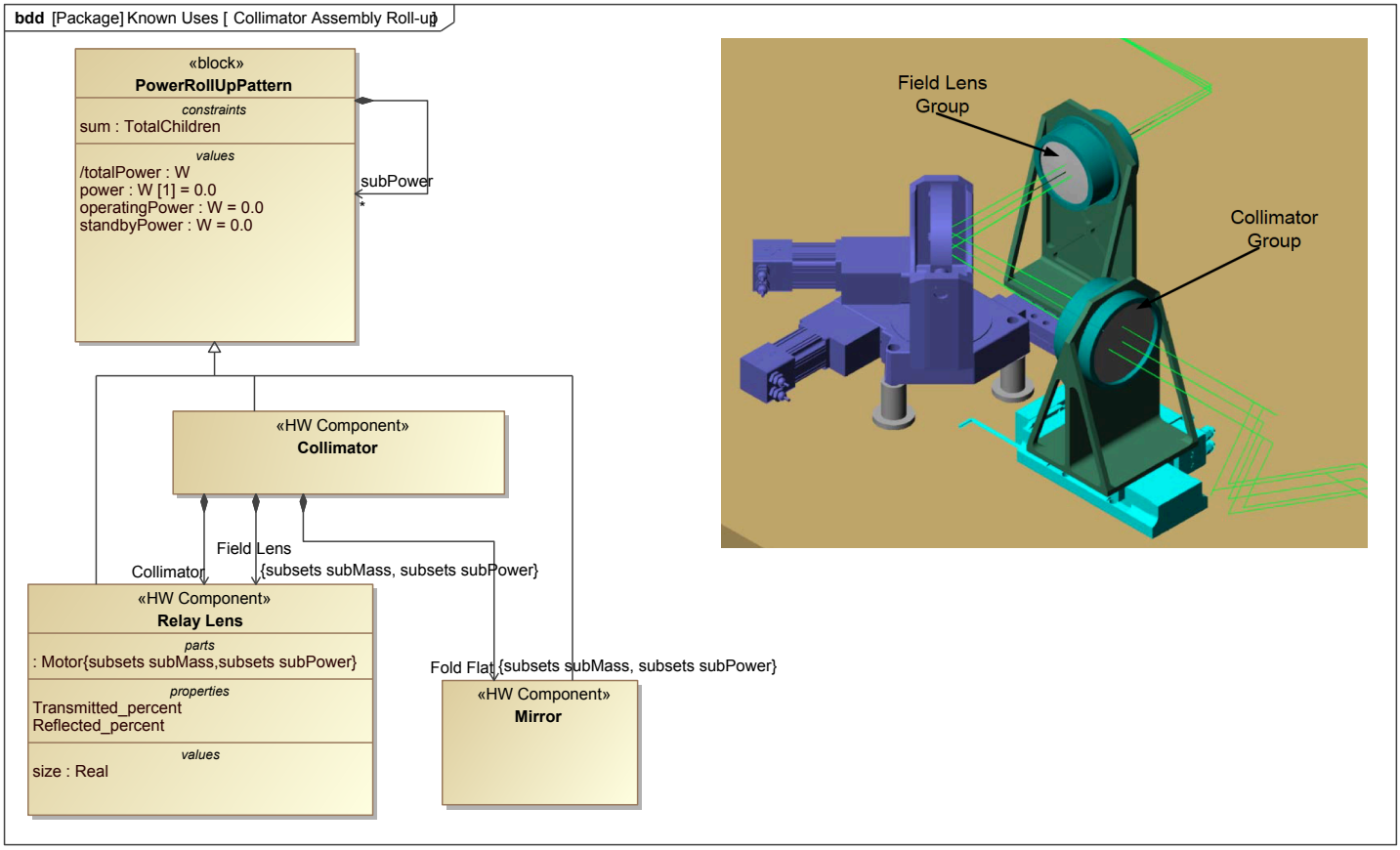


Figure 89. Collimator Assembly Roll-up

Figure 90. Simulation Roll-up

Table 16. Satisfy Peak Power Result

Name	enc : Check Peak Power Compliance	fac : Check Peak Power Compliance	powerPeakLimitEnclosure : W	powerPeakLimitSummitFacilityBuildings : W	peakPowerEnc : W	p
satisfy Peak Power Explanation at 2016.03.29 12.49	pass	pass				
peak Power Limit Scenario Online at 2016.04.30 00.49					420.0	5
peak Power Limit Scenario Online at 2016.04.30 00.52					420.0	5
peak Power Limit Scenario Online at 2016.04.30 00.54					420.0	5

Name	enc : Check Peak Power Compliance	fac : Check Peak Power Compliance	powerPeakLimitEnclosure : W	powerPeakLimitSummitFacilityBuildings : W	peakPowerEnc : W	p
peak Power Limit Scenario Online at 2016.07.25 14.28					460.0	5
peak Power Limit Scenario Online at 2016.09.29 15.38					460.0	5
peak Power Limit Scenario Online at 2017.02.26 18.29					460.0	5
peak Power Limit Scenario Online.aps operational blackbox specification jpl.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl1.peak Power Limit Requirement JPL			8100.0	4100.0		

Name	enc : Check Peak Power Compliance	fac : Check Peak Power Compliance	powerPeakLimitEnclosure : W	powerPeakLimitSummitFacilityBuildings : W	peakPowerEnc : W	p
peak Power Limit Scenario Online.aps operational blackbox specification jpl1.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl2.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl2.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl3.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl3.peak Power Limit Requirement TMT			8500.0	4200.0		

Name	enc : Check Peak Power Compliance	fac : Check Peak Power Compliance	powerPeakLimitEnclosure : W	powerPeakLimitSummitFacilityBuildings : W	peakPowerEnc : W	p
peak Power Limit Scenario Online.aps operational blackbox specification jpl4.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl4.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl5.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps operational blackbox specification jpl5.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps realization.peak Power Limit Requirement JPL			8100.0	4100.0		

Name	enc : Check Peak Power Compliance	fac : Check Peak Power Compliance	powerPeakLimitEnclosure : W	powerPeakLimitSummitFacilityBuildings : W	peakPowerEnc : W	p
peak Power Limit Scenario Online.aps realization.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps realization1.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps realization1.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps realization2.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps realization2.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps realization3.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps realization3.peak Power Limit Requirement TMT			8500.0	4200.0		

Name	enc : Check Peak Power Compliance	fac : Check Peak Power Compliance	powerPeakLimitEnclosure : W	powerPeakLimitSummitFacilityBuildings : W	peakPowerEnc : W	p
peak Power Limit Scenario Online.aps realization4.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps realization4.peak Power Limit Requirement TMT			8500.0	4200.0		
peak Power Limit Scenario Online.aps realization5.peak Power Limit Requirement JPL			8100.0	4100.0		
peak Power Limit Scenario Online.aps realization5.peak Power Limit Requirement TMT			8500.0	4200.0		

4.2.7 Analysis

The Dynamic Roll-up pattern helps to analyze a system model by demonstrating that the resource usage according to the system design satisfies a corresponding requirement for every defined specified scenario.

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

4.2.8 Tooling

The following tooling supports the modeler in the implementation of the Dynamic Roll-up pattern in MagicDraw.

4.2.8.1 Cameo Simulation Toolkit

NoMagic's Cameo Simulation Toolkit (CST) is implemented and supports the Dynamic Recursive Roll-up Pattern by providing a method to perform analysis of a modeler's system.

After the roll-up pattern has been augmented to the system design, the values of the behavior specific properties of the components participating in the roll-up need to be specified either through redefinition or instance specifications, and then analysis can be performed on the system by running a simulation with CST.

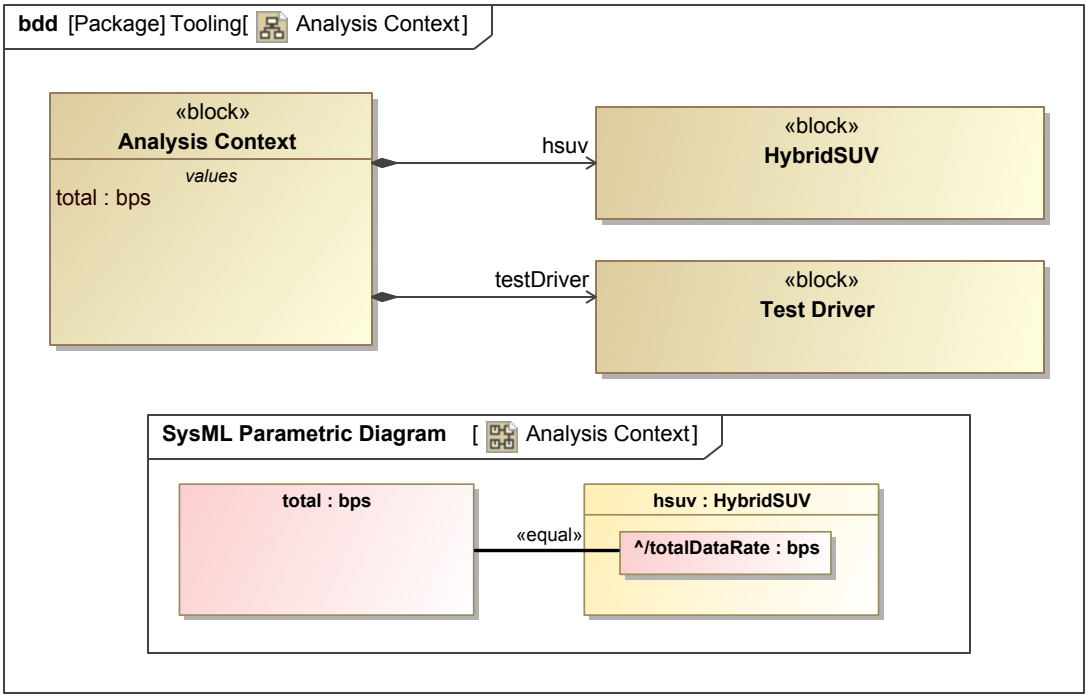


Figure 91. Analysis Context

After configuring the Roll-up pattern to the system an analysis context needs to be specified. In particular, the block Analysis Context is used to represent this and is defined in it's owning parametric model. An Analysis Context block and Test Driver block can be created as an easier context to display the results of the dynamic property propagation. The analysis context is composed of a black box (Analysis Driver) where the analysis is performed and a design block (HybridSUV). In addition, concrete analysis blocks corresponding to operational scenarios are needed to be defined. A parametric model must be created to constrain the total (data rate) to equal the total dynamic property (totalDataRate) of the roll-up. The Test Driver block may be required depending if and how the operational scenario for the system is specified. The analysis block (Analysis Driver) specializes the abstract analysis blocks.

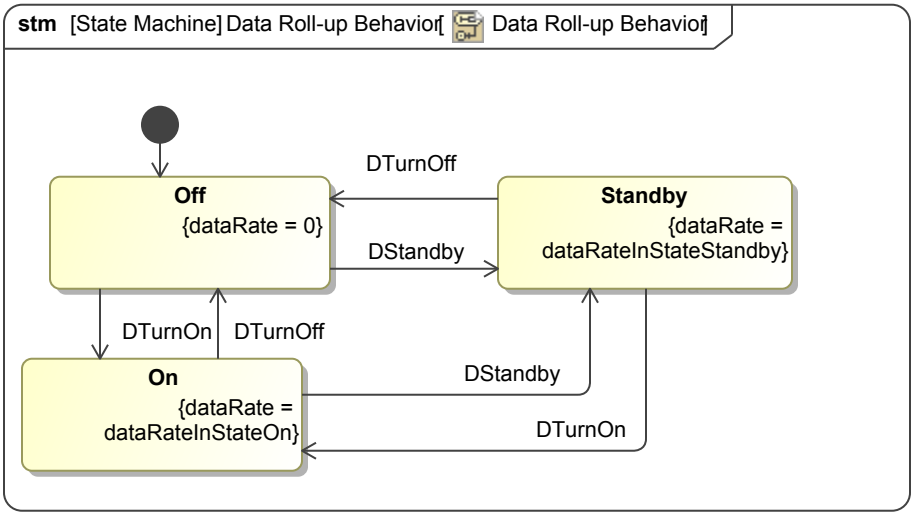


Figure 92. Data Roll-up Behavior

The behavior of the system is defined in a state machine, and is inherited by all components participating in the roll-up. When running a simulation with CST, the user has the option to specify the behavior of the component's lifecycle either at run time or through simulation configurations.

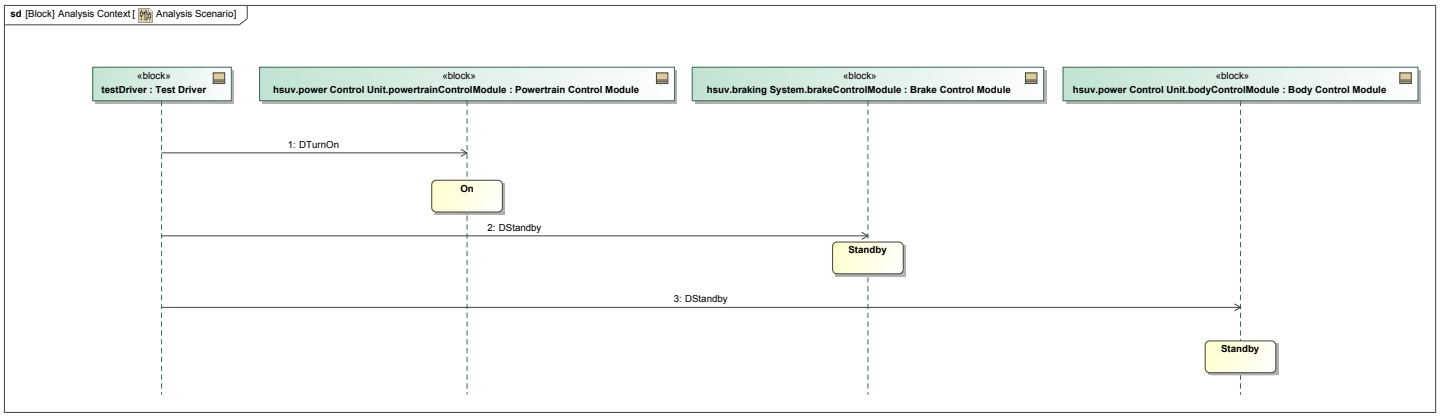


Figure 93. Analysis Scenario

To preconfigure the operational states of the system and to automate the simulation process the Analysis Context block needs to define a sequence diagram that serves as a scenario driver. The scenario driver becomes a use case that can be analyzed using Cameo Simulation Toolkit.

The Test Driver lifeline is required to send messages (signals) to the other components whose dynamic properties are specified. Due to this sequence diagram, the value of the total data rate will be equal to the value of the dataRateInStateOn of the Powertrain Control Module, the value of the dataRateInStateStandby of the Brake Control Module, and the value of the dataRateInStateStandby of the Body Control Module.

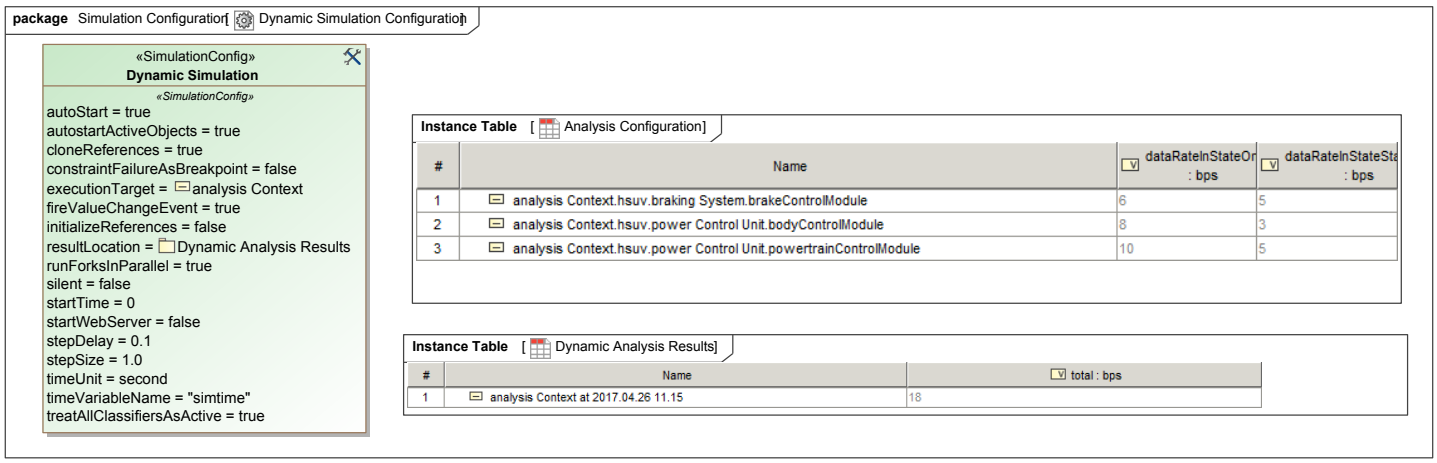


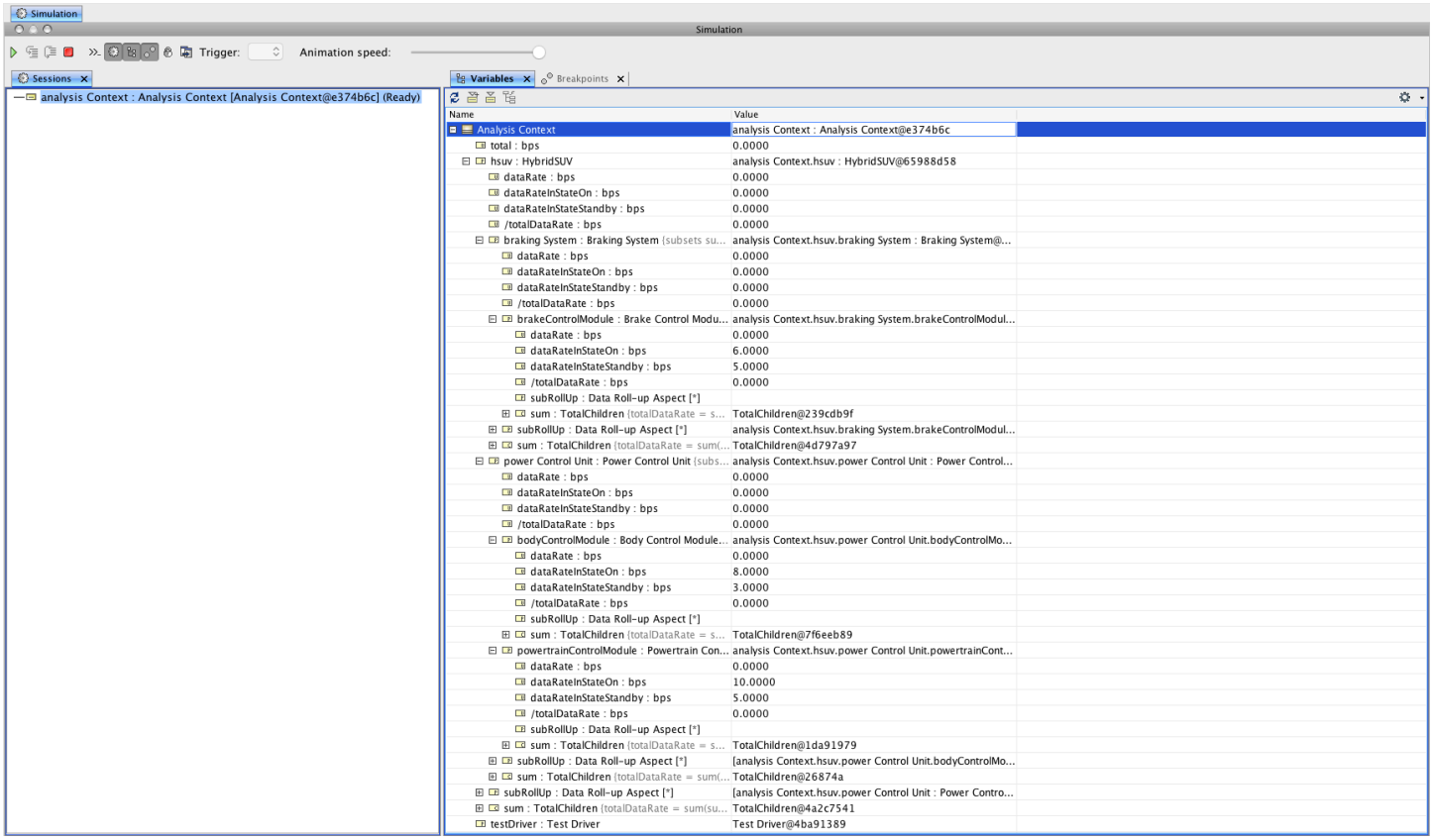
Figure 94. Dynamic Simulation Configuration

A simulation configuration can be used to automate the simulation process and exports the results as an instance specification. To create a Simulation Configuration, the executionTarget needs to equal the Analysis instance specification (or other top level hierarchical functional component).

After selecting to run the simulation, the resulting instance specification with the total value of the dynamic property for the system can be saved in an instance table. This is done by configuring the property resultLocation to a package. In the Analysis Results instance table, a summary of the resulting final values of the data rate based on the previously configured instance specifications can be seen.

An overview of the values for each component during simulation can be seen in the variables section of the Simulation window.

When a Simulation of the Analysis Context instance is initialized, initial solving of the variables occurs. During initial solving, the total data rate of the system and any other components where the value properties were not defined in an instance specification remains to be zero. The values are zero because the sequence model of the Analysis Context does not launch until the simulation has been run. Once the simulation has began, the signals defined the scenario will proceed and the values for dataRate will be propagated throughout the system. This occurs because each component specializes the Dynamic Roll-up Pattern block where the behavior is specified (in the state machine). The total value of the Analysis Context is equal to the value of its parts (Brake Control Module, Body Control Module, Powertrain Control Module) during each of their respective states.



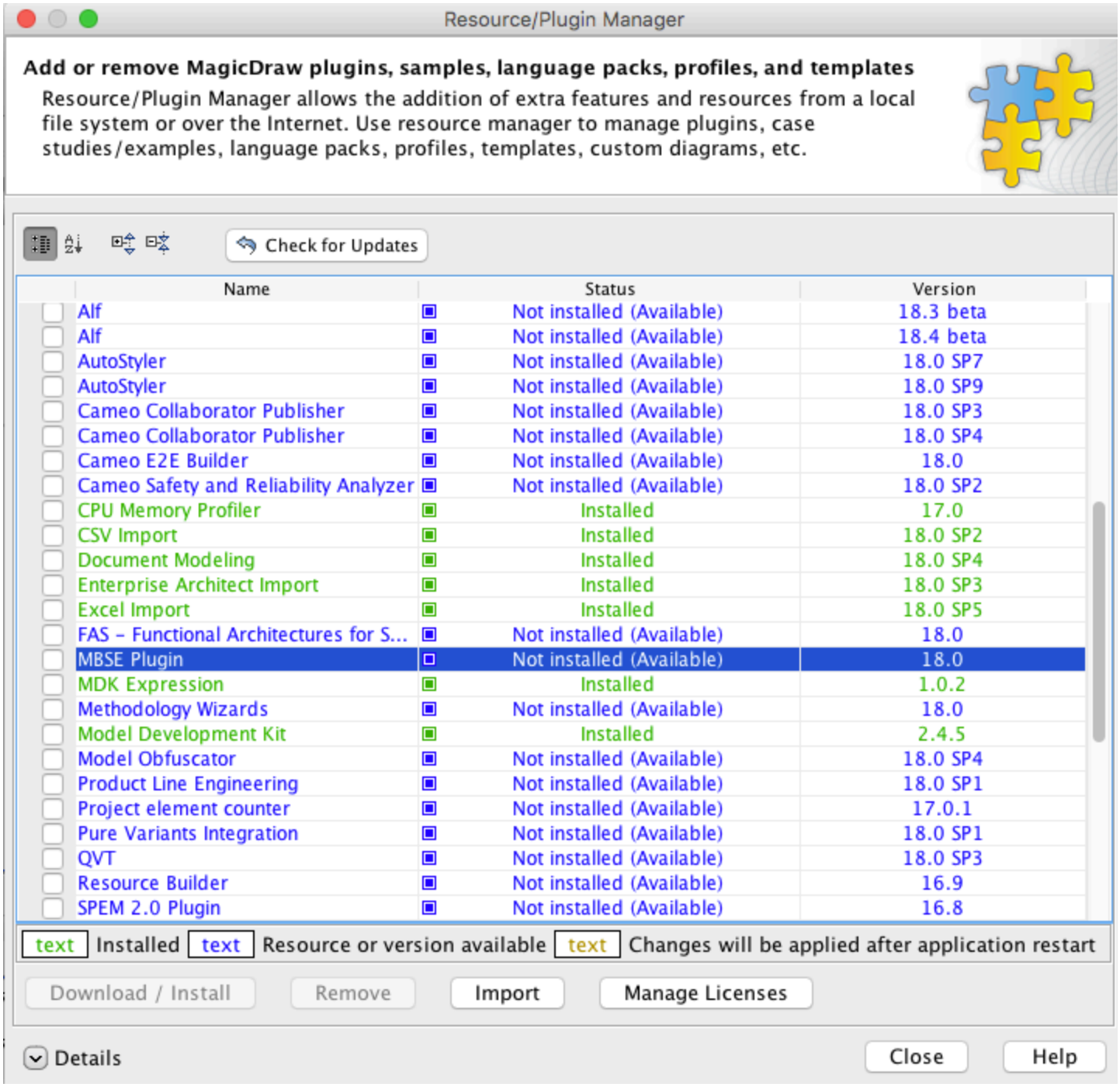
4.2.8.2 Model Construction

The general usage of the reasoner component is to aid systems engineers in pattern based reasoning.

Other useful resources that aid the implementation the pattern include NoMagic's Rollup Pattern Wizard for MagicDraw version 18.5, and the MBSE plug-in for MagicDraw version 17.0 sp4.

MBSE Plugin

The MBSE plug-in can also be found in the Resource/Plugin Manager of MagicDraw which is compatible for MD 18.0.



4.2.9 Related Patterns

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

Pattern Name	Similarities	Differences
Simple Static Roll-up Pattern	There are no similarities between the behavior of the Dynamic roll-up and Simple Static roll-up patterns and their relationship to their respective systems.	The primary difference between the Dynamic roll-up and the Simple Static roll-up pattern is due to the behavior of their systems. The total value of the final dynamic property of the Dynamic Roll-up is equal to the component's values during a certain event in its lifecycle. Meanwhile, the value of the final static property of the Simple Static roll-up doesn't depend on the state of the components.

Recursive Static Roll-up Pattern	A similarity between the Dynamic roll-up and the Recursive Static roll-up is that they both share a sub component that is typed by itself (subRollup). The composition relationships between the components of their systems subset the part property.	The primary difference between the Dynamic and the Recursive Static roll-up pattern is due to the behavior of their systems. The total value of the final dynamic property of the Dynamic Roll-up is equal to the component's values during a certain event in its lifecycle. Meanwhile, the value of the final static property of the Recursive Static roll-up doesn't depend on the state of the components.

5 Error Budgeting

5.1 Intent

Technical resources of systems such as mass, cost, or power are frequently managed by budgets. Budgets represent the maximum possible usage of a resource. Complex systems may have a requirement that defines the overall budget of a particular technical resource with each subsystem contributing to the usage of the resource. The intent of this pattern is to model the budgeting of error in SysML models.

5.2 Motivation

Throughout the development cycle of a system, a maximum possible and current best estimate of a technical resource usage exist. These values may change as the design process evolves, and changes in low level systems propagate to affect the aggregate error. Monitoring the allowable and current measures of error throughout the system is particularly important in ensuring that requirements are satisfied. The motivation of this modeling pattern is to be able to trace the maximum allowable, current best estimate (CBE), and growth of error within a SysML model. Changes in the overall error may be tracked as the system changes, and incorporating error allocation values into the model allows margin calculations to be completed. Combining the system analysis with system requirements allows an automated verification of requirements to be simulated.

This pattern is applicable to systems that must perform within a defined level of error. The roll-up nature of the pattern favors systems containing multiple layers of subsystems with error values that contribute to the overall error of the system. Changes in error of one system component are automatically propagated to higher level systems. Additionally, this pattern can be applied to managing the allocation of error to specific subsystems.

5.3 Concept

The SysML structure of error budgeting resembles other roll-up patterns for system metrics and utilizes a recursive structure to allow application to systems of varying size. The error budgeting pattern consists of the *Error Roll-Up Pattern Element* block that serves as a common super-type to other roll-up schemes (*Root Sum Squared Roll-Up, Sum Roll-Up, Product Roll-Up*). All sub-types of *Error Roll-Up Pattern Element* inherit error requirement (Req), current best estimate (CBE), and margin value properties. The composite association from the *Error Roll-Up Pattern Element* to itself creates a recursive relationship that allows the model to be flexible to systems with varying degrees of sub-levels. The multiplicity ranging from zero to infinity ensures that the lowest level sub-systems do not have components and that the pattern may be applied to a system with infinite sub-systems. Each system component must subset *subError* in order to successfully propagate values through the system.

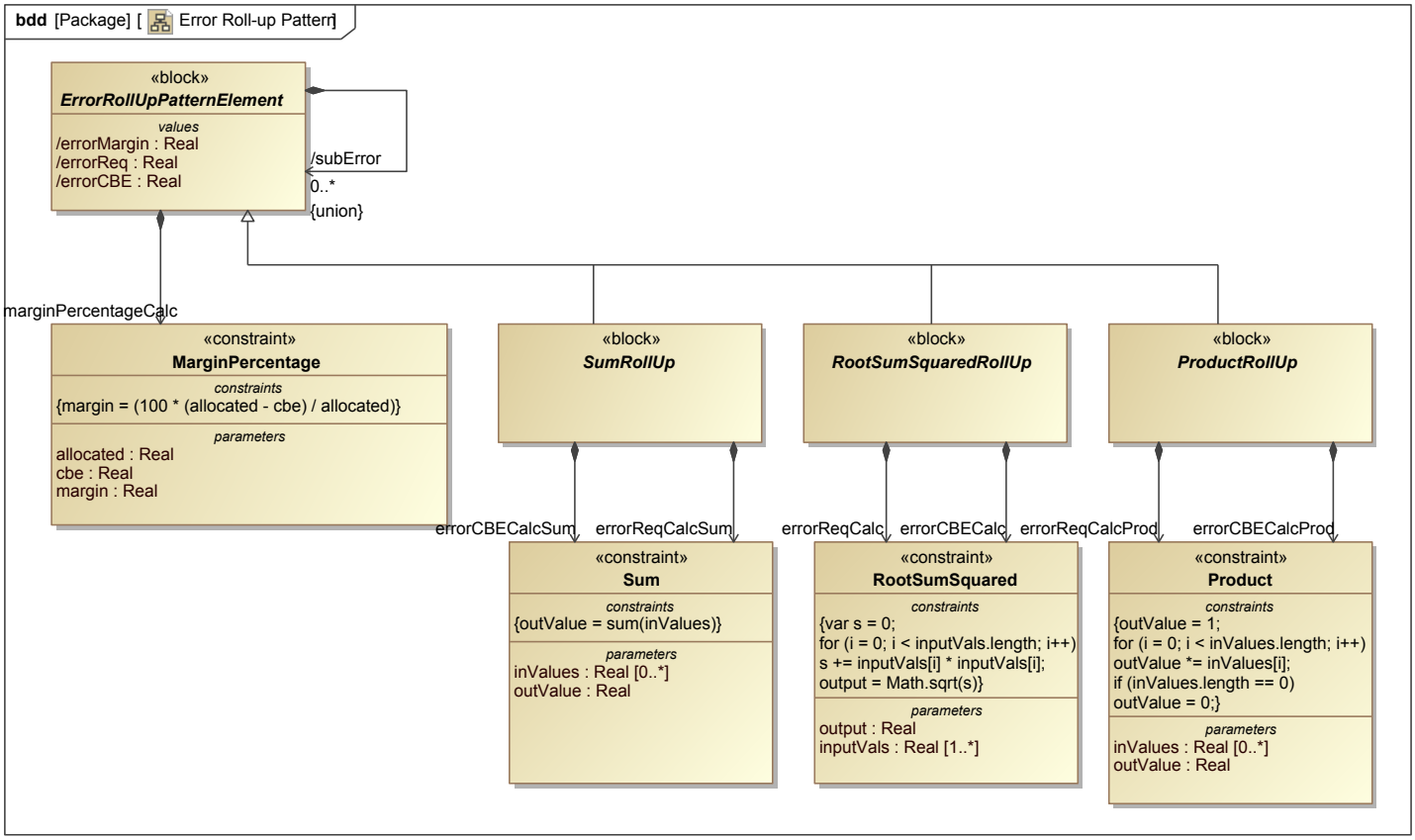


Figure 95. Error Roll-up Pattern

Since the same margin calculation is desired for all components, the calculation (represented as a constraint) is modeled as a child of the *Error Roll-Up Pattern Element.* As a result, the parametric diagram of the margin constraint may be defined at the *Error Roll-Up Pattern Element* super-type level and is inherited by all sub-types. The parametric diagram for the margin calculation is seen at the bottom of the figure above.

Various roll-up schemes, such as a root sum squared, sum, or product, may be selected to model how values propagate from lower level systems to high level systems. Calculations for each scheme are modeled as a constraint that is a child of the roll-up scheme. Constraint specifications are designed such that they are applicable to any number of inputs from any number of sub-components. The parametric diagrams for the root sum squared roll-up are seen in the figure below, and other roll-up schemes follow a similar setup.

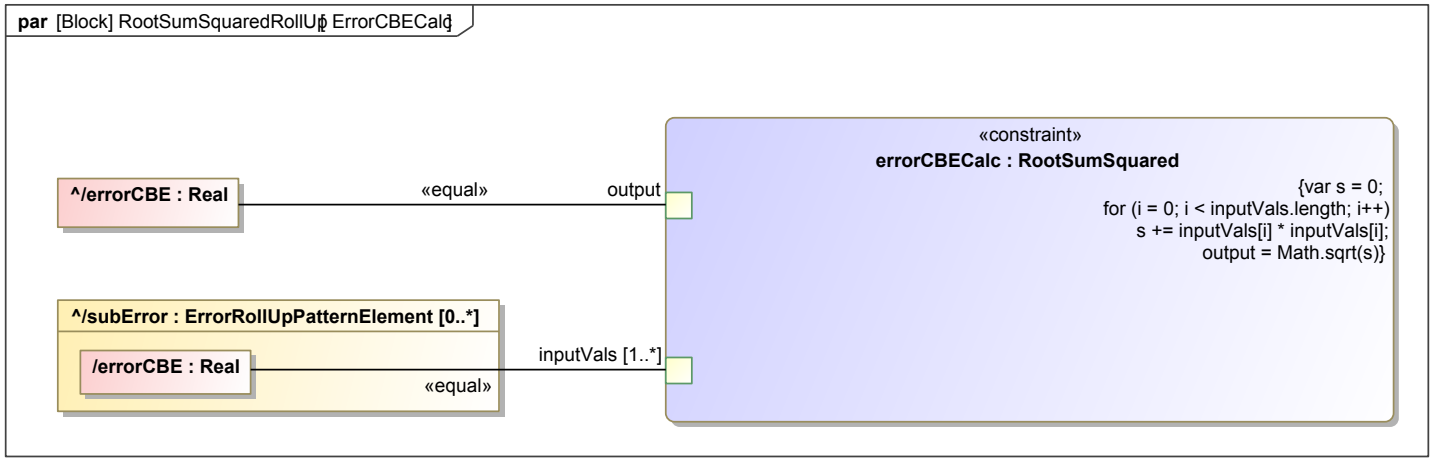


Figure 96. ErrorCBECalc

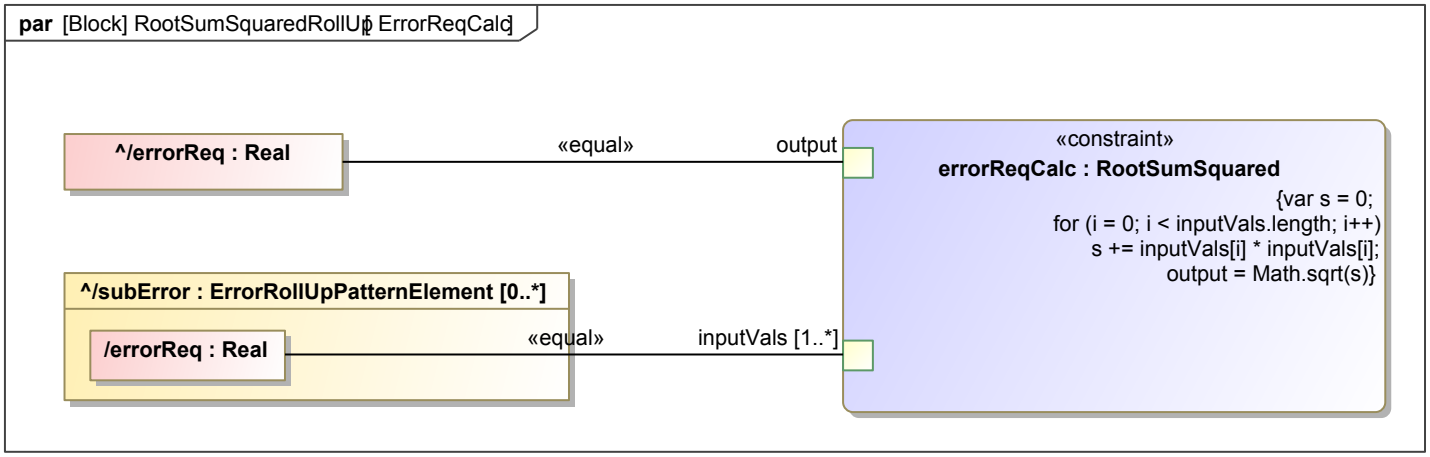


Figure 97. ErrorReqCalc

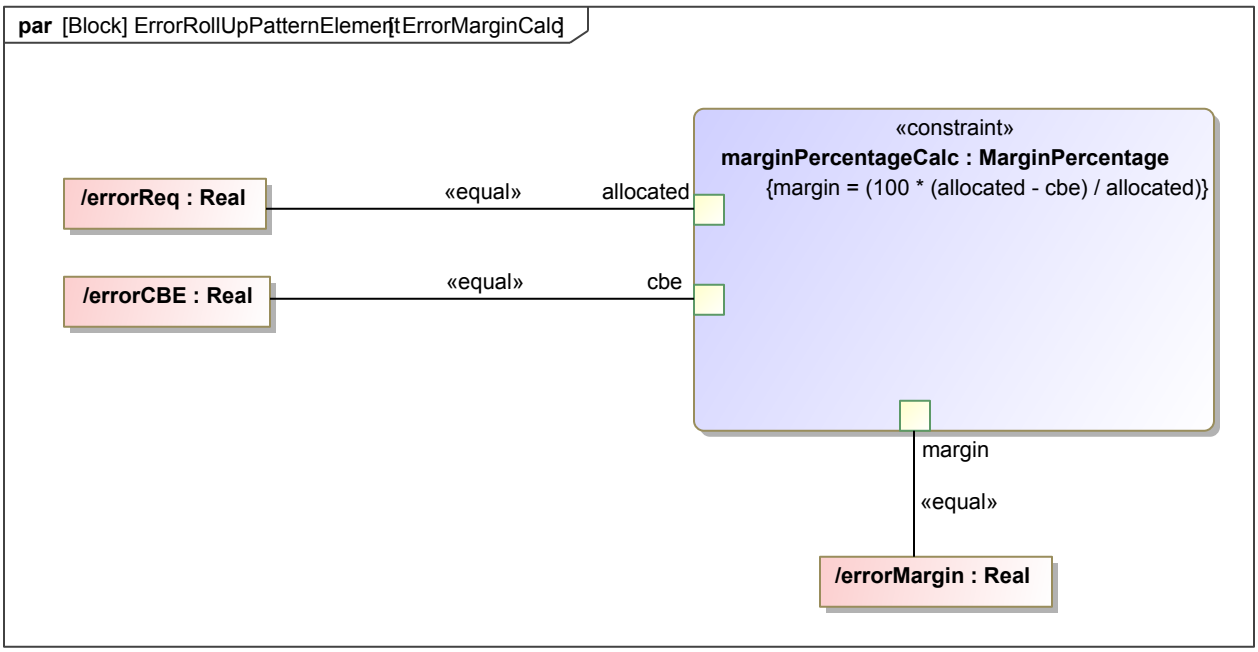


Figure 98. ErrorMarginCalc

5.4 Consequences

Actions implementing inheritance, multiplicity, redefinition, and recursion allow the error budgeting pattern to be implemented.

Table 17. T1

Action	Consequence
Creating generalization relation	Creating a generalization relation between the ErrorRollUpPatternElement and all system components allows each component to inherit the value properties necessary to implement the pattern.
Flexible multiplicity of recursive relationship	Modeling the multiplicity of the recursive relationship allows the ErrorRollUpPatternElement to reference all sub components, regardless of the number. The multiplicity allows the pattern to be applied to systems with varying layers.
Creating composite relation to self	Creating a composite relation from the ErrorRollUpPatternElement to itself allows the pattern to act recursively. A recursive pattern may be applied to complex and non-flat level systems of any size.
Subsetting subError property	Subsetting the subError property is required in order for values from lower level components to propagate to the highest level of the system.
Redefining static values of system components	Redefining static values of system components allows inherited attributes to be easily modified. Redefinition allows the values of CBE to accumulate through each layer of the system.

Creating instance specifications of component static values	Creating instance specifications allows different model configurations to be simulated and results may be stored in an instance table.

5.5 Implementation

An example of the application of the Error Budgeting pattern is in the context of a microscope. In oder to achieve optimal performance, proper alignment of microscope components must be achieved to ensure that illumination is ideal while viewing the specimen. Köhler illumination is the condition at which the light emitted from the lamp filament is focused at the plane of the condenser aperture and is achieved by precisely locating the filament and condenser of the microscope.

The model consists of the *Microscope Alignment for Kohler Illumination* block which is parent to the *Filament Alignment* and *Condenser Alignment* blocks, both of which represent the position of the filament and condenser components, respectively. By assuming that alignment error will propagate through the system based on a root sum squared pattern, each system block specializes the *RootSumSquaredRollUp* block. The figure below shows the setup of the model.

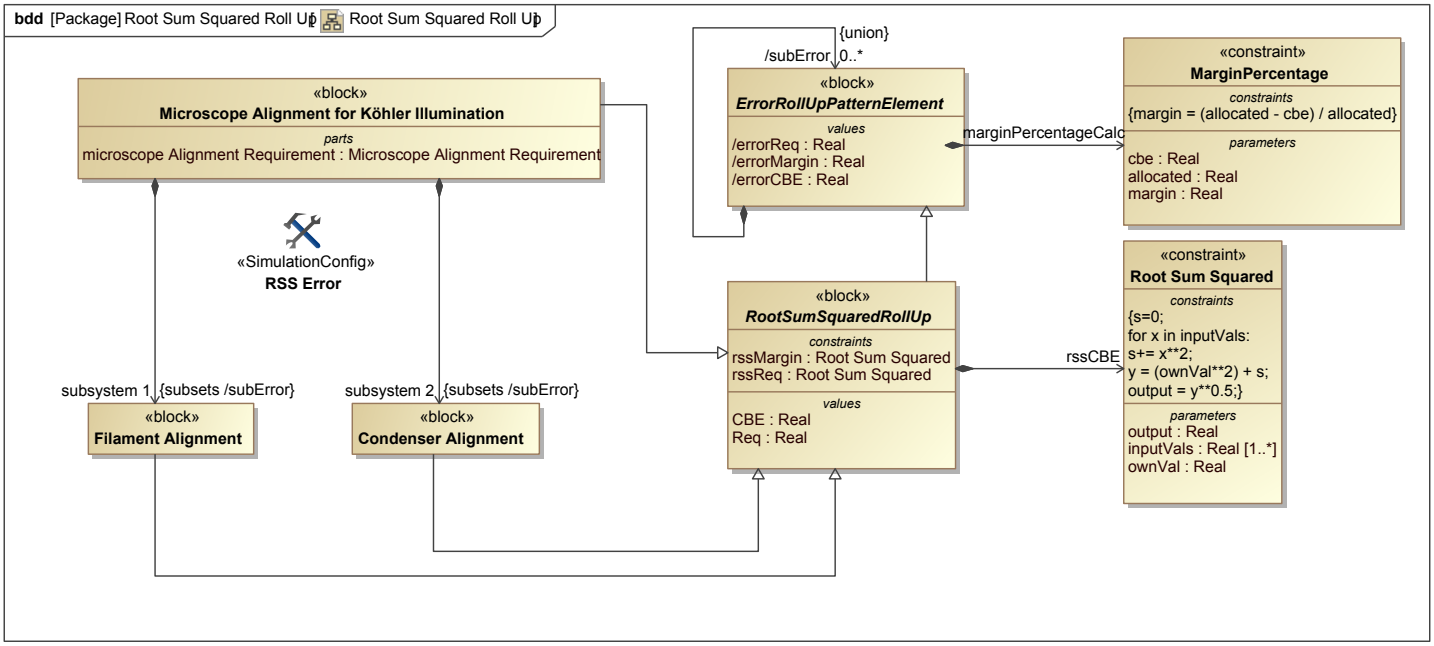


Figure 99. Root Sum Squared Roll Up

In order to simulate the model, an instance of the *Microscope Alignment for Kohler Illumination* block is created. In the instance, initial values for the filament alignment and condenser alignment error are specified.

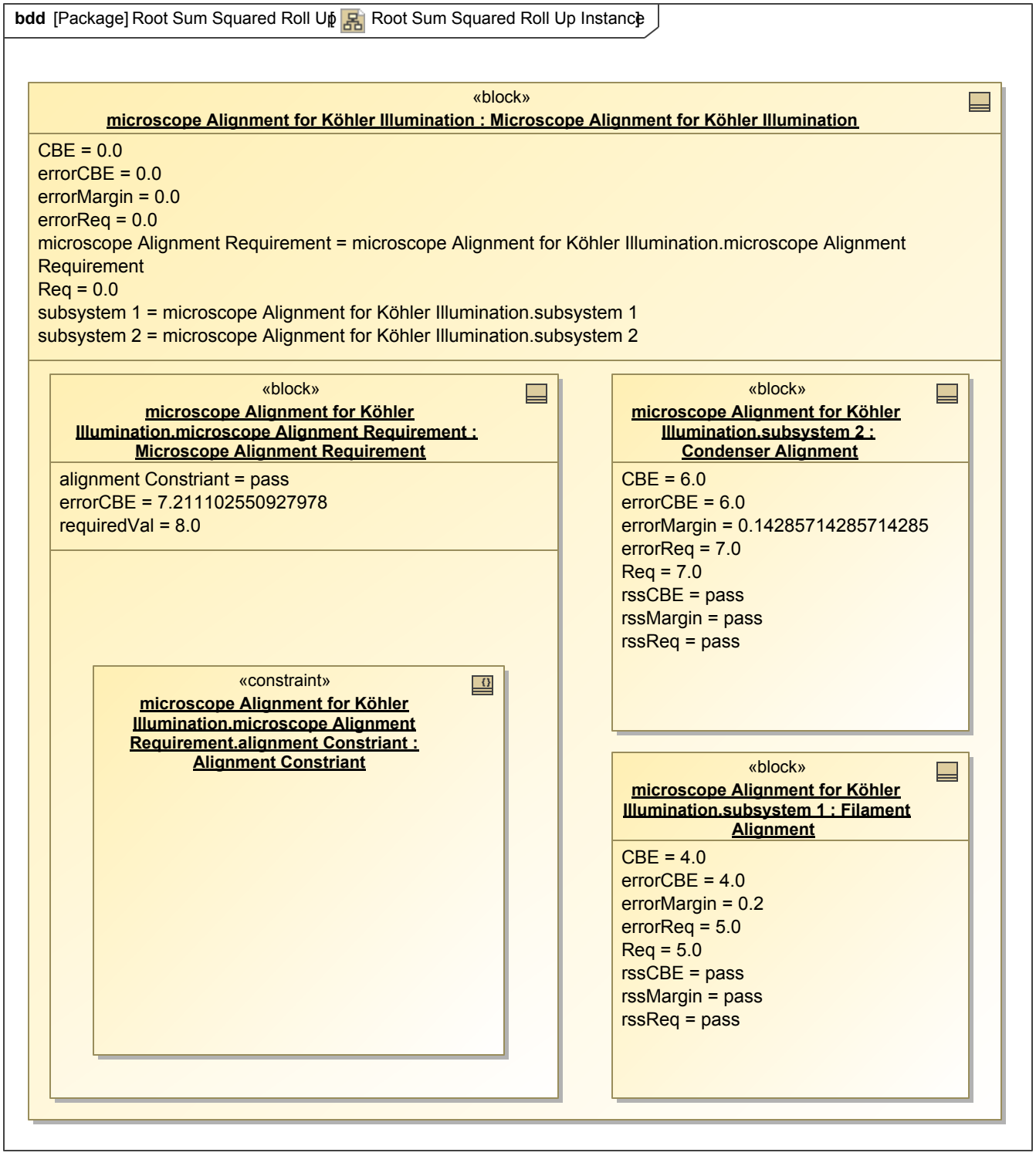


Figure 100. Root Sum Squared Roll Up Instance

In order to tie the simulation results to a requirement, a *Microscope Alignment Requirement* block is a child of the *Microscope Alignment for Kohler Illumination* block and owns a constraint called *Alignment Constraint.* The constraint block if representative of the alignment requirement and enforces the relation that the CBE value must be no greater than the allotted value. The figure below shows the relationships between blocks, constraints, and requirements.

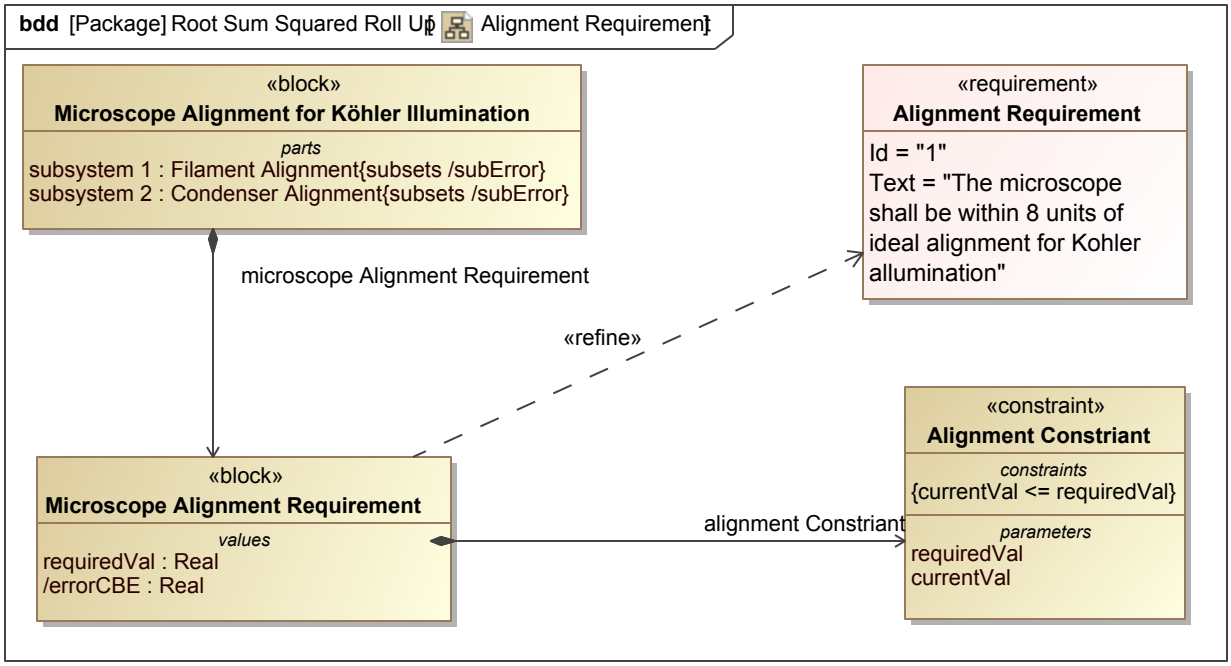


Figure 101. Alignment Requirement

5.6 Known Uses

The Thirty Meter Telescope (TMT) model utilizes this error budgeting pattern to capture decomposition of error, perform a roll-up of current best estimates from subsystems, calculate error margins, and verify that system requirements are met. System requirements, system design, and system error parameters are tied together to create an integrated approach of system analysis.

Tracking accuracy of expected system performance compared to system requirements may be demonstrated in the context of the Alignment and Phasing System (APS) of the Thirty Meter Telescope (TMT). An example requirement for which error may be tracked throughout the systems refers to the alignment error of the M3 to APS interface:

"APS shall measure the position of the telescope pupil to an accuracy of 0.03% of the diameter of the pupil."

The *Telescope Pupil Alignment Requirement* is further specified by the accuracy requirement and has a constraint which enforces that the value of the current best estimate is no greater than the required value.

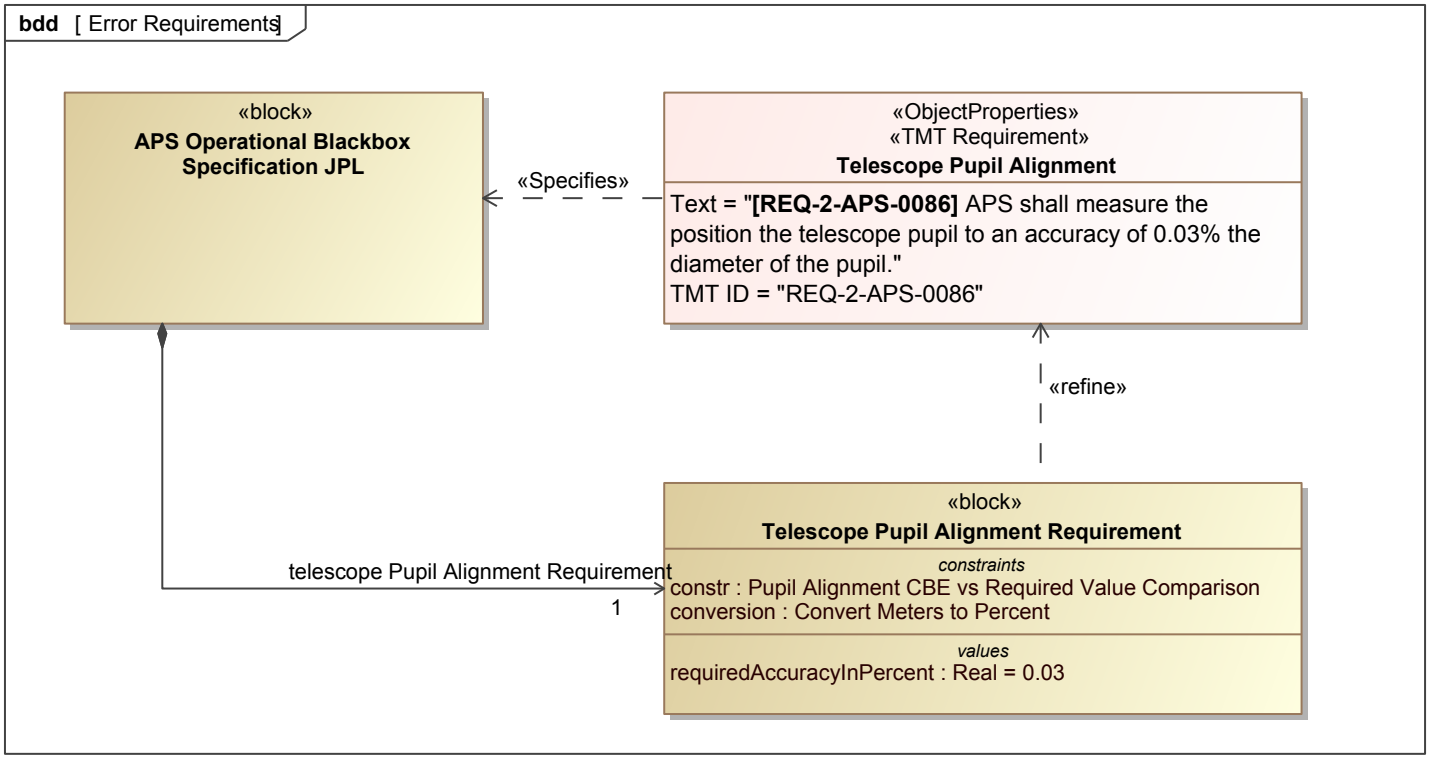


Figure 102. Error Requirements

The structure of the error budget is defined by an error budget tree that shows the hierarchy of system blocks that contain error values that contribute to the overall budget. The figures below show excerpts of the error breakdown tree.

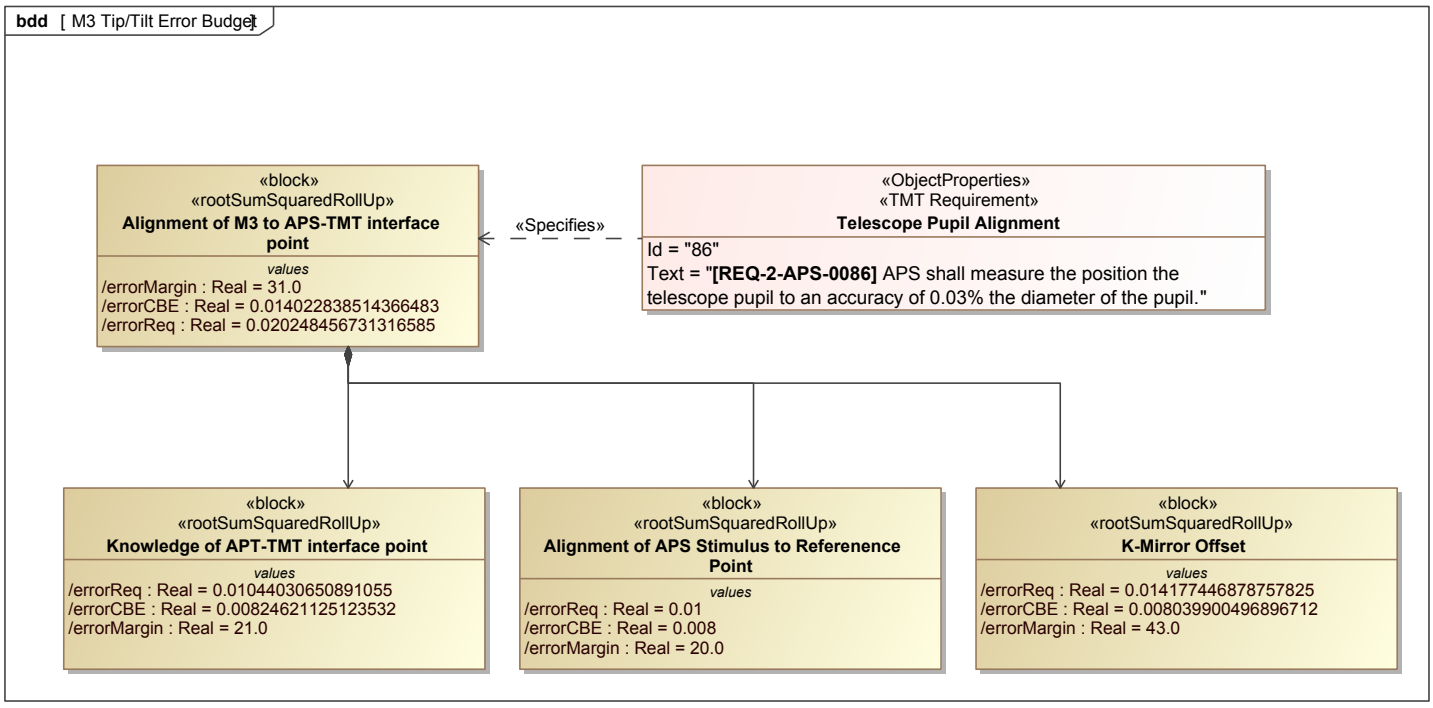


Figure 103. M3 Tip/Tilt Error Budget

This is the top of the M3 alignment error budget that ties the requirement to the top of the error budget tree.

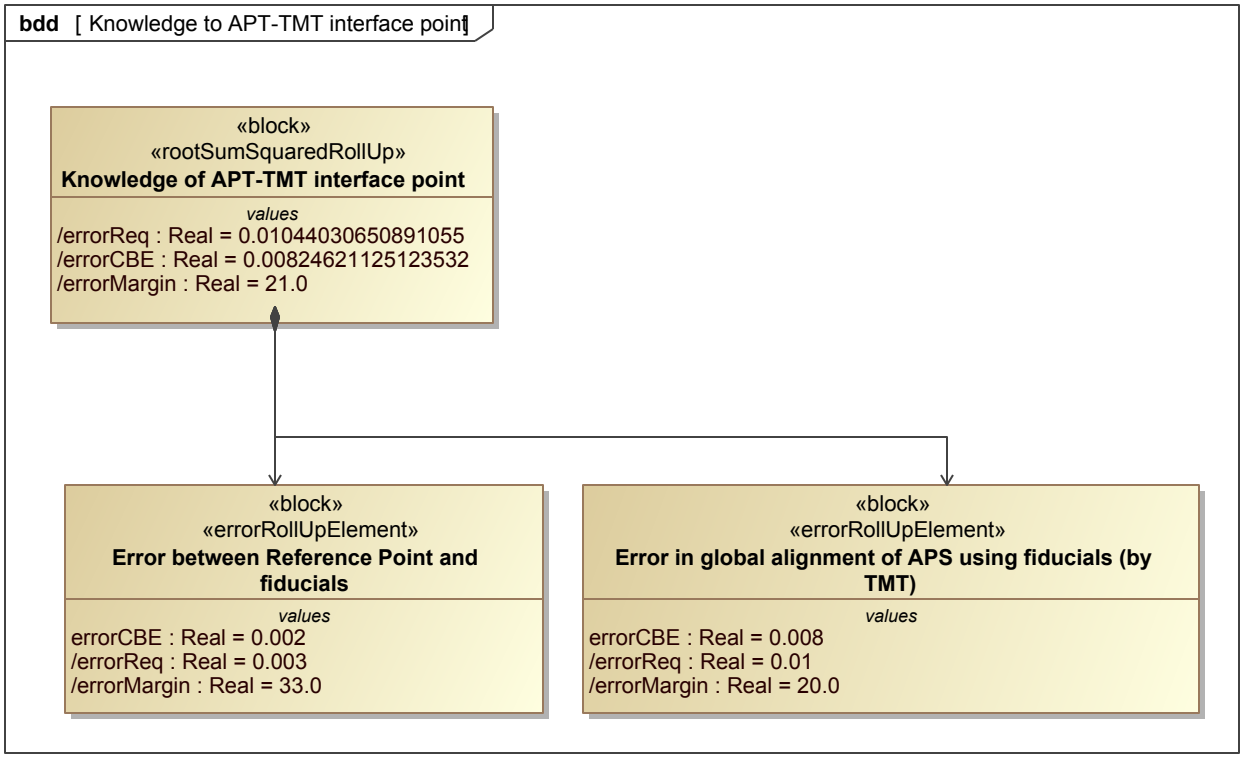


Figure 104. Knowledge to APT-TMT interface point

The pattern is applied to a system as shown in the figure below. Each block in the system inherits from a specific error roll-up type and subsets *subError* of *Error Roll Up Pattern Element.* Initial values for requirement and current best estimate are defined for leaf level blocks.

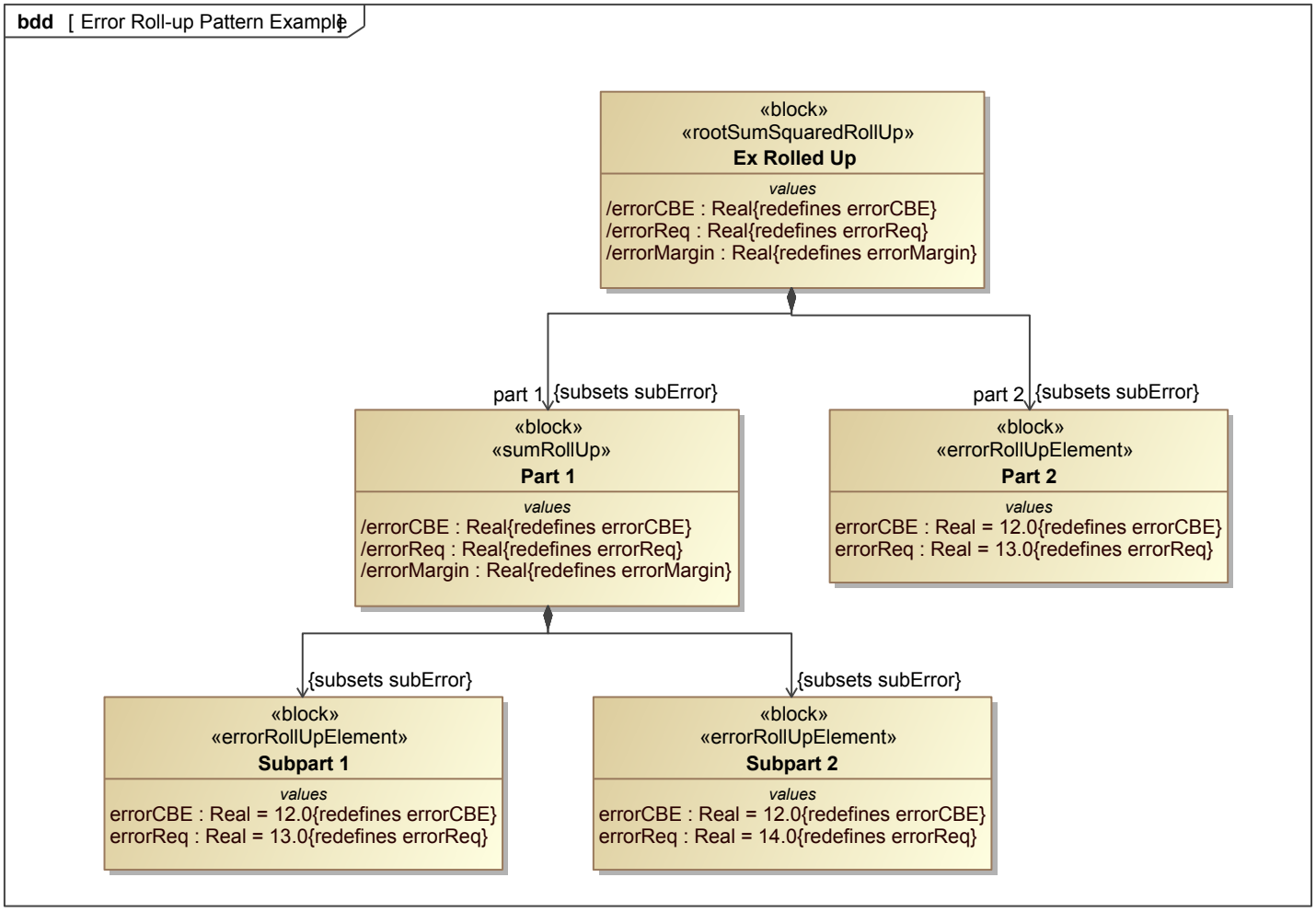


Figure 105. Error Roll-up Pattern Example

Requirements are tied to the product breakdown structure via a constraint which evaluates the comparison between the required value and the current best estimate value and returns a pass or fail verdict.

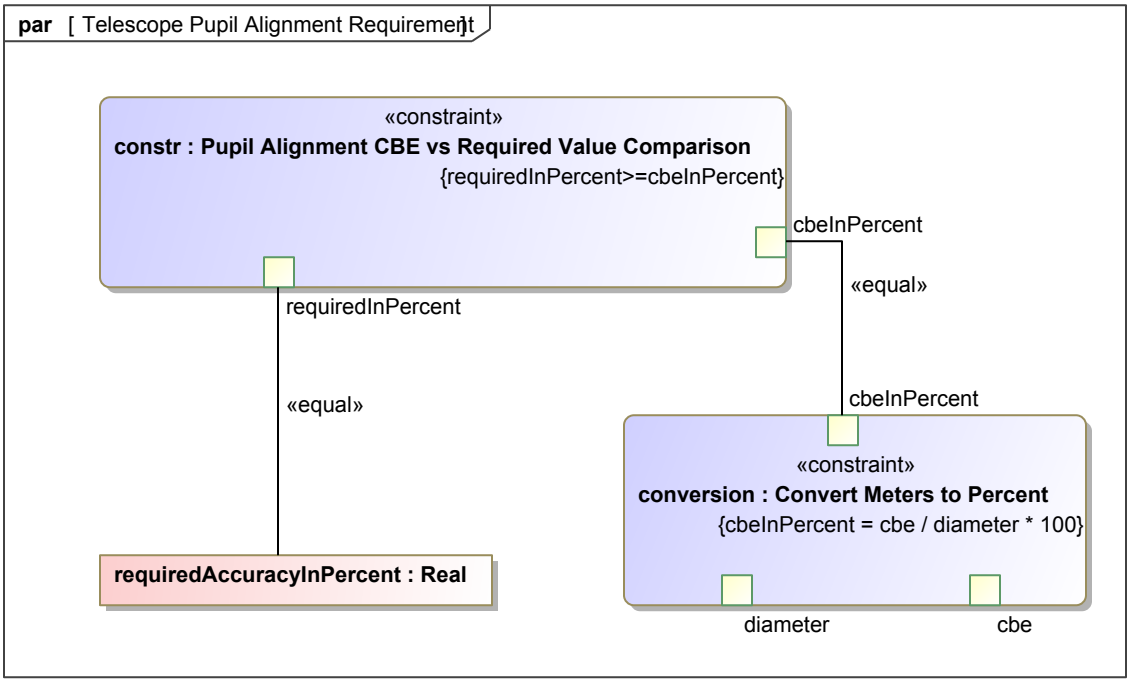


Figure 106. Telescope Pupil Alignment Requirement

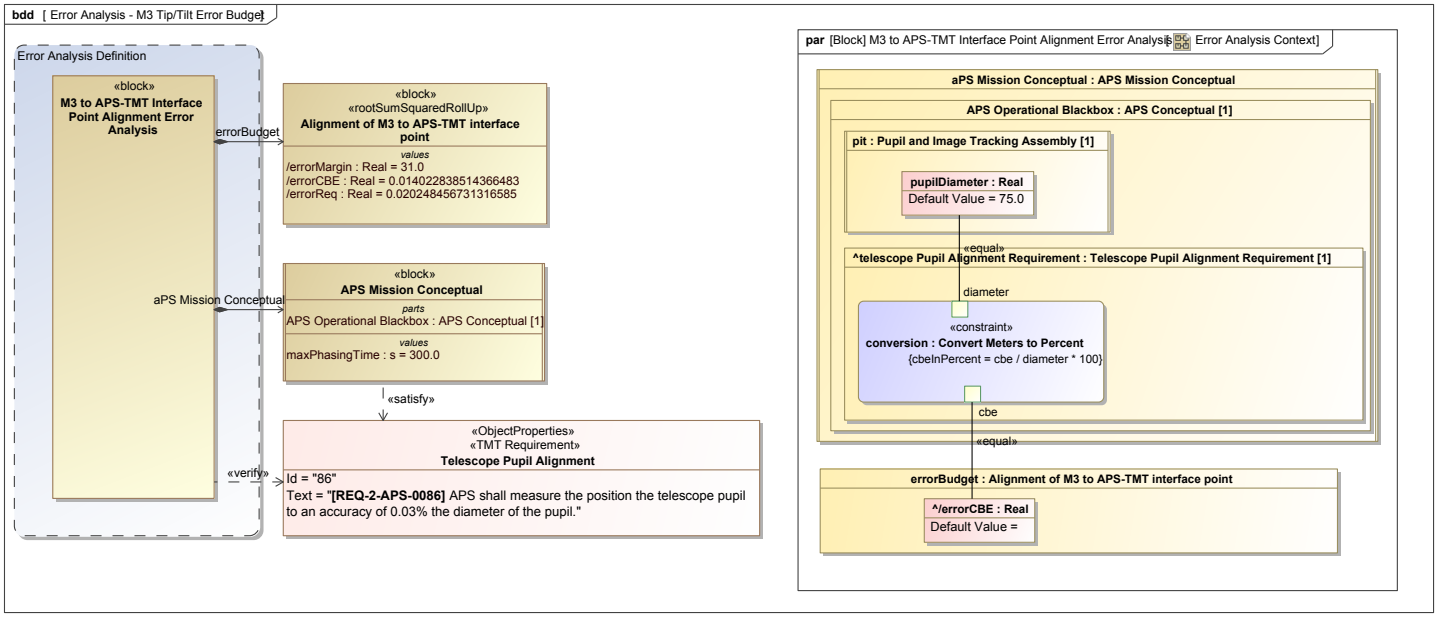


Figure 107. Error Analysis - M3 Tip/Tilt Error Budget

The M3 Tip/Tilt Error Budget diagram shows how the error budget ties together in the model. The error analysis block ties together the top of the error budget, the APS conceptual model and the requirement. The <> block is used to start the analysis using the Cameo Simulation Toolkit. The analysis results are stored in a results table which show the history of the top result.

5.7 Tooling

Cameo Simulation Toolkit (CST)

NoMagic's Cameo Simulation Toolkit (CST) is implemented in the model and supports the Error Budgeting Pattern by providing a method to perform analysis of a system. Once the Error Budgeting pattern has been integrated into the system design, the values for the static property of the leaf level components participating in the roll-up need to be specified. Analysis of the system performed by CST propagates the values through system levels via redefinition.

An Analysis block can be be created to tie together the top of the error budget, the conceptual model, and the requirement, as seen in the TMT example model. A parametric model must be created to constrain the system parameter that is the focus of the roll-up.

A simulation from a component in the system design or a simulation configuration may be used to manually or automatically run the analysis using CST. To automate the simulation process, the top level analysis block needs to own a state machine diagram with an initial, final node, and state in between as seen in the figure below.

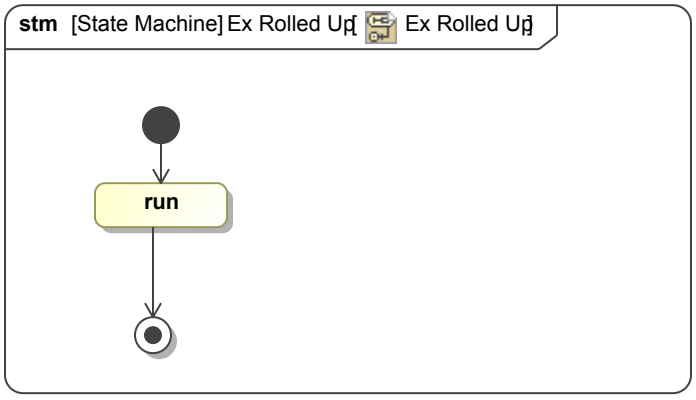


Figure 108. Ex Rolled Up

While the roll-up of required and current best estimate values is independent of the state of the model, the state machine is required to automatically save the results of a simulation after running a simulation configuration. The simulation configuration is created by selecting the top level hierarchical component as the execution target and is used to automate the simulation process. Results are exported as an instance specification.

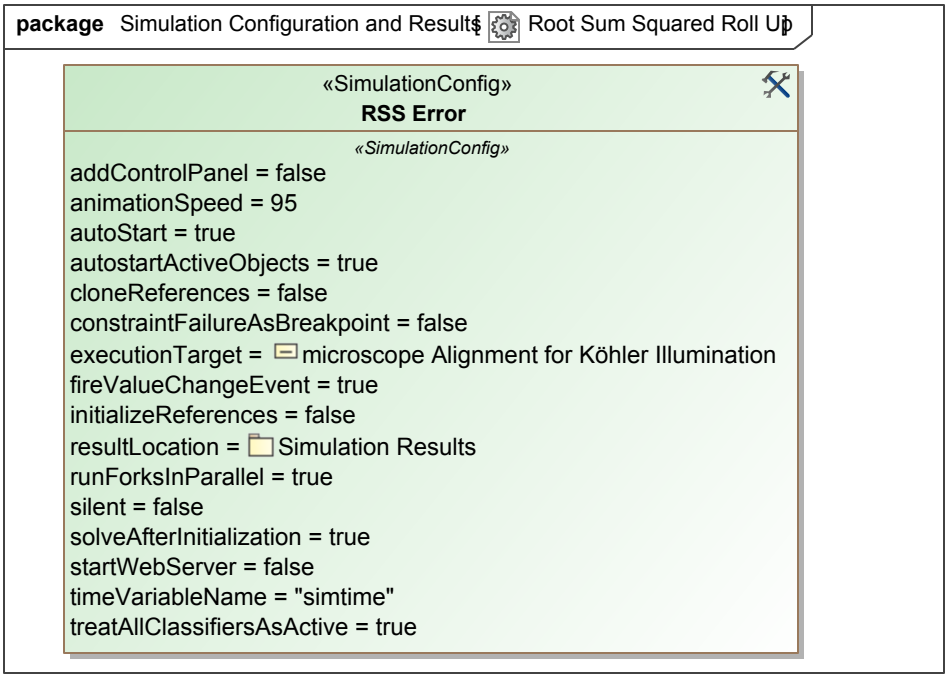


Figure 109. Root Sum Squared Roll Up

The simulation may be manually run by right clicking the top level hierarchical component and navigating to simulation and selecting run.

When CST is initialized either automatically or manually, the system components and their values can be observed in the CST simulation window.

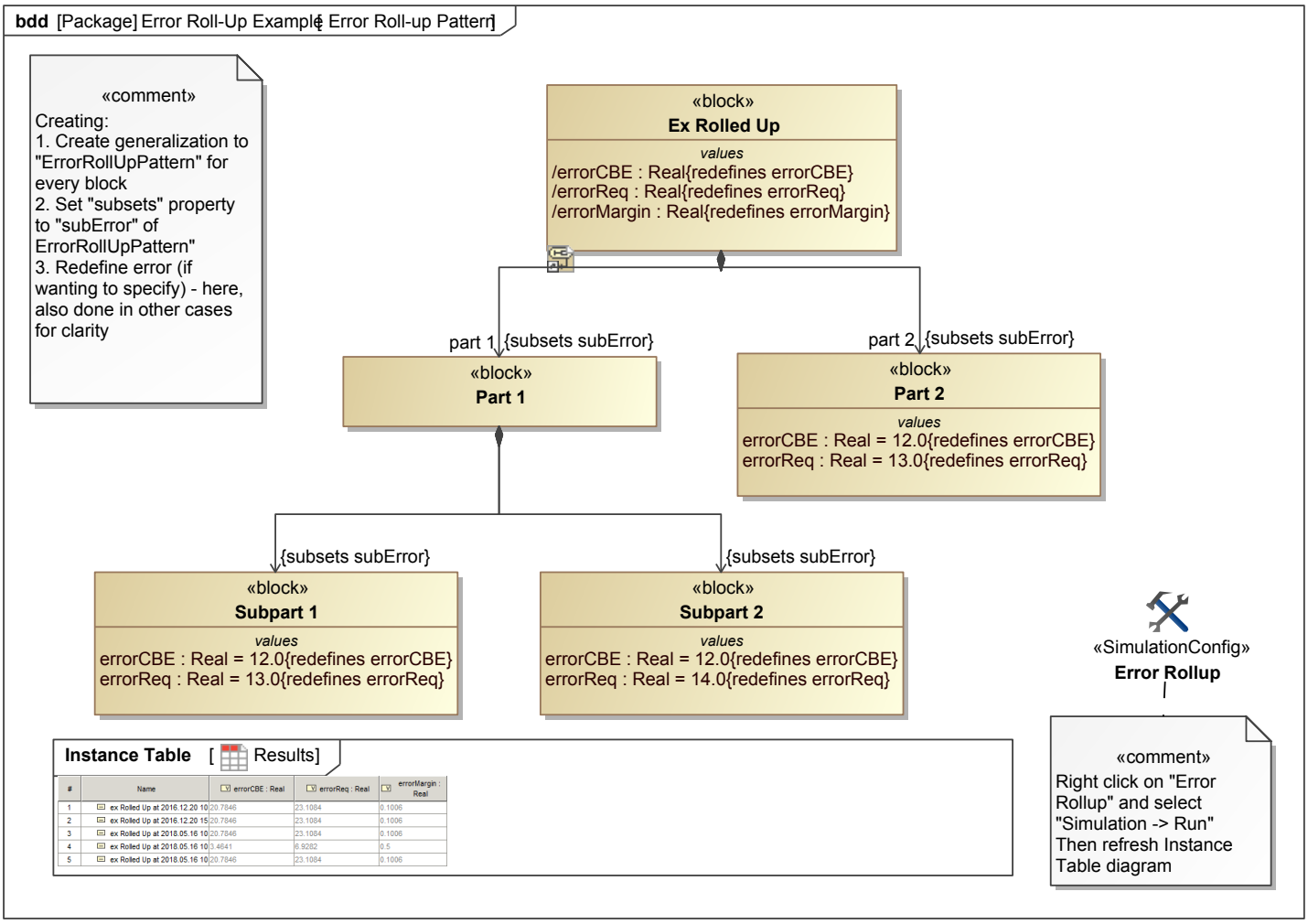


Figure 110. Error Roll-up Pattern

The model may be configured to simply show the resulting calculations in an instance table, or it can be tied to requirements to verify requirement satisfaction. The addition of a constraint that is representative of the requirement may be used to evaluate the calculated CBE value with respect to the required value. The resulting instance table then includes a pass or fail evaluation of the current system design as seen in the table below.

Table 18. M3 to APS-TMT Interface Point Alignment Error Analysis Results

Name	constr : Pupil Alignment CBE vs Required Value Comparison
m3 to APS-TMT Interface Point Alignment Error Analysis.alignment of M3 to APS-TMT interface point	
m3 to APS-TMT Interface Point Alignment Error Analysis.alignment of M3 to APS-TMT interface point1	
m3 to APS-TMT Interface Point Alignment Error Analysis.alignment of M3 to APS-TMT interface point2	
m3 to APS-TMT Interface Point Alignment Error Analysis.alignment of M3 to APS-TMT interface point3	
m3 to APS-TMT Interface Point Alignment Error Analysis.alignment of M3 to APS-TMT interface point4	
m3 to APS-TMT Interface Point Alignment Error Analysis.alignment of M3 to APS-TMT interface point5	

Name	constr : Pupil Alignment CBE vs Required Value Comparison
m3 to APS-TMT Interface Point Alignment Error Analysis.alignment of M3 to APS-TMT interface point6	
m3 to APS-TMT Interface Point Alignment Error Analysis.alignment of M3 to APS-TMT interface point7	
m3 to APS-TMT Interface Point Alignment Error Analysis.alignment of M3 to APS-TMT interface point8	
m3 to APS-TMT Interface Point Alignment Error Analysis.aps mission conceptual.aps conceptual.telescope Pupil Alignment Requirement	fail
m3 to APS-TMT Interface Point Alignment Error Analysis.aps mission conceptual1.aps conceptual.telescope Pupil Alignment Requirement	pass
m3 to APS-TMT Interface Point Alignment Error Analysis.aps mission conceptual2.aps conceptual.telescope Pupil Alignment Requirement	pass
m3 to APS-TMT Interface Point Alignment Error Analysis.aps mission conceptual3.aps conceptual.telescope Pupil Alignment Requirement	pass
m3 to APS-TMT Interface Point Alignment Error Analysis.aps mission conceptual4.aps conceptual.telescope Pupil Alignment Requirement	fail
m3 to APS-TMT Interface Point Alignment Error Analysis.aps mission conceptual5.aps conceptual.telescope Pupil Alignment Requirement	pass
m3 to APS-TMT Interface Point Alignment Error Analysis.aps mission conceptual6.aps conceptual.telescope Pupil Alignment Requirement	pass
m3 to APS-TMT Interface Point Alignment Error Analysis.aps mission conceptual7.aps conceptual.telescope Pupil Alignment Requirement	pass
m3 to APS-TMT Interface Point Alignment Error Analysis.aps mission conceptual8.aps conceptual.telescope Pupil Alignment Requirement	pass

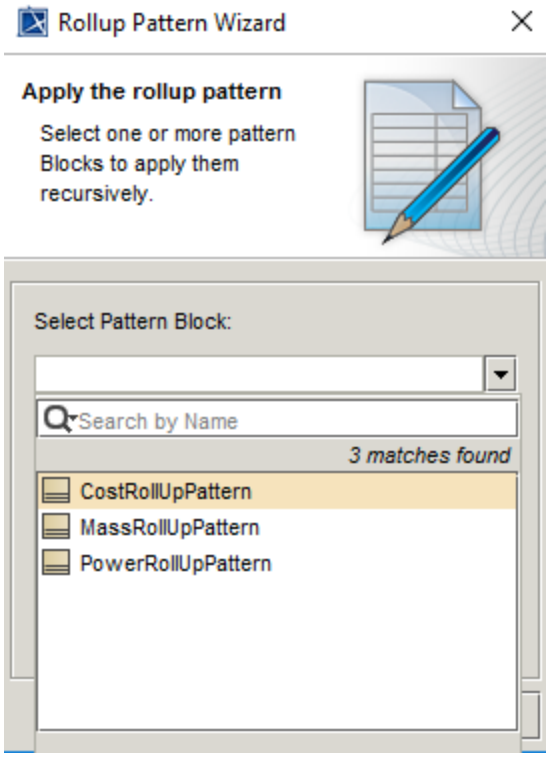
5.7.1 NoMagic Rollup Pattern Wizard

The NoMagic Rollup Pattern Wizard is used to apply the specified rollup pattern to a system. After defining the system and the rollup pattern, the user can apply the NoMagic Rollup Wizard to the system. The user can use the Rollup pattern by following these steps:

Applying a Rollup Pattern

Select the Block (or instance specification) in which you want to apply the pattern to and select **Tools** > **Apply Rollup Pattern**

The Rollup Pattern Wizard window will then appear as shown.



The Rollup Pattern Wizard provides three default rollup pattern blocks for cost, mass, and power (i.e. CostRollUpPattern, MassRollUpPattern, PowerRollUpPattern). The default pattern blocks can be found in the **MD Customization for SysML** auxiliary resource. The wizard also allows for customized Pattern Blocks as well.

Creating Customized Pattern Blocks

There are two methods for creating a customized pattern block that is specific to certain type of roll-up analysis.

1. Copy and pasting a default rollup pattern block from the MD Customization for SysML auxiliary resource
    1. Copy any existing rollup pattern block (i.e. CostRollUpPattern, MassRollUpPattern, PowerRollUpPattern)
    2. Paste into appropriate location of the model
    3. Modify the pattern block
        1. Change name, value properties (make sure to change the types of the value properties), part property, constraint block, and parametric diagram
2. Creating a Roll-up block from our pattern and applying the <<Pattern>> stereotype
    1. Follow the structure or copy elements of/from the OpenSE Cookbook/OpenSE Library
    2. Paste into appropriate location of the model
    3. If copying, modify the pattern block
        1. Change name, value properties (make sure to change the types of the value properties), part property, constraint block, and parametric diagram
    4. If creating the pattern block on own, modify the pattern block
        1. Set name, value properties (types of the value properties), part property, constraint block, and parametric diagram
        2. Apply <<Pattern>> Stereotype
            1. Set the stereotype of the pattern block to <<Pattern>>. The stereotype is located in the MD Customization for SysML auxiliary resource. This step will allow the Rollup Pattern Wizard to recognize the pattern block.

5.8 Related Patterns

List of other patterns that are related to the Recursive Static Roll-up pattern, and the differences and similarities between them.

Table 19. T1

Pattern Name	Similarities	Differences
Simple Static Roll-Up Pattern	The Error Budgeting Patter is similar to the Simple Static roll-up pattern in that they both propagate values of a static property in a system. The value of the property does not change over time due to system behavior or current state.	The Error Budgeting Pattern and the Simple Static roll-up pattern are different in that the application of the patterns apply to different system structures. When applying the Error Budgeting Pattern, the system can be structured in a nested compositional hierarchy. The Simple Static Roll-up can be applied only to an un-complex structure where there is a flat listing of components. *The Error Roll Up Pattern Element* block of the Error Budgeting pattern needs to be specialized by every component in the structure and subsetted by its *subError* part property, while the aspect block of the Simple Static Roll-up is a whole-part relationship to only the top level component or system.
Recursive Static Roll-Up Pattern	The Error Budgeting Pattern and the Recursive Static Roll-Up are similar in that they both utilize a recursive approach and have a sub-component typed by itself. Each component block participating in the roll-up must specialize the pattern block and each subsystem must have a composition relationship to the system or to its parent. Every subsystem and subcomponent that composes the system must subset the subRollup part property.	There are not any notable differences between the Error Budgeting Pattern and the Recursive Static Roll-Up Pattern. The Error Budgeting Pattern implements at recursive static roll-up in the context of error tracking. The only difference between the two patterns is the presence of a constraint which calculates the margin for each component.
Dynamic Roll-Up Pattern	The Error Budgeting Pattern and the Dynamic roll-up are similar in that that they both utilize a recursive approach and share a sub component that is typed by itself. The composition relationships between the components of their systems subset the part property.	The Error Budgeting Pattern and the Dynamic Roll-Up Pattern are different due to the behavior of their systems. The value of the final dynamic property of the Dynamic Roll-up is equal to the component's values during a certain event in its lifecycle. The value of the final static property of the Error Budgeting Pattern is independent of the state of the components.