



75 Fifth Street NW, Suite 213
Atlanta, GA 30332, USA
Voice: +1-404-592-6897
Web: www.InterCAX.com
E-mail: info@intercax.com

ParaMagic™ v16.6 sp1 Tutorials

Table of Contents

1	Introduction and review.....	3
1.1	<i>Introduction.....</i>	3
1.2	<i>Short Review of SysML.....</i>	4
1.3	<i>Short Review of Solving Equations.....</i>	6
2	SysML Parametrics Tutorial - Addition.....	7
2.1	<i>Objective.....</i>	7
	What the User Will Learn.....	7
2.2	<i>Step-by-Step Tutorial.....</i>	7
	Step I Create Project.....	7
	Step II Create Infrastructure.....	8
	Step III Create Structural Model.....	9
	Step IV Create Constraints.....	11
	Step V Create Parametrics Model.....	12
	Step VI Validate Parametrics Model.....	14
	Step VII Create an Instance.....	15
	Step VIII Solve the Instance.....	19
3	SysML Parametrics Tutorial - Satellite.....	21
3.1	<i>Objective.....</i>	21
	What the User Will Learn.....	21
3.2	<i>Step-by-Step Tutorial.....</i>	21
	Step I Create Project.....	21
	Step II Create Infrastructure.....	22
	Step III Create Structural Model.....	22
	Step IV Create Constraints.....	23
	Step V Create Parametrics Model.....	26
	Step VI Validate Parametrics Model.....	28
	Step VII Create an Instance.....	28
	Step VIII Solve the Instance.....	32
4	SysML Parametrics Tutorial - LittleEye.....	35
4.1	<i>Objective.....</i>	35
	What the User Will Learn.....	35
4.2	<i>Step-by-Step Tutorial.....</i>	36
	Step I Create Project.....	36

Step II Create Infrastructure	36
Step III Create Structural Model.....	36
Step IV Create Constraints	37
Step V Create Parametrics Model.....	37
Step VI Validate Parametrics Model	40
Step VII Create an Instance	40
Step VIII Solve the Instance	41
5 SysML Parametrics Tutorial - CommNetwork	45
5.1 Objective.....	45
What the User Will Learn.....	45
5.2 Step-by-Step Tutorial.....	45
Step I Create Project	45
Step II Create Infrastructure	45
Step III Create Structural Model.....	46
Step IV Create Constraints	49
Step V Create Parametrics Model(s)	49
Step VI Validate Parametrics Model	51
Step VII Create an Instance	51
Step VIII Solve the Instance	55
6. SysML Parametrics Tutorial - Orbital	56
6.1 Objective.....	56
What the User Will Learn.....	56
System Requirements	57
6.2 Step-by Step Tutorial.....	57
Step I Create Project	57
Step II Create Infrastructure	57
Step III Create Structural Model.....	57
Step IV Create Constraints	59
Step V Create Parametrics Model.....	61
Step VI Validate Parametrics Model	62
Step VII Create an Instance	62
Step VIII Solve the Instance	67
7. SysML Parametrics Tutorial - HomeHeating	70
7.1 Objective.....	70
What the User Will Learn.....	70
7.2 Step-by Step Tutorial.....	71
Step I Create Project	71
Step II Create Infrastructure	71
Step III Create Structural Model.....	71
Step IV Create Constraints	72
Step V Create Parametrics Model.....	73
Step VI Validate Parametrics Model	74
Step VII Create an Instance	74
Step VIII Solve the Instance	76

1 INTRODUCTION AND REVIEW

1.1 Introduction

The primary purposes of SysML up to this point have been Documentation, precise specification of system design, and Communication, sharing the design among multiple parties. Adding parametric execution to SysML enables additional purposes,

- Consistency, enforcing internal relationships to insure a coherent, self-consistent data set;
- Simulation, evaluating the performance, cost and other parameters of the system design;
- Verification, integrating checks of system properties against requirements.

Our general approach for tutorials on creating SysML model with parametrics follows

- I. Create Project
- II. Create Infrastructure
- III. Create Structural Model
- IV. Create Constraints
- V. Create Parametric Model
- VI. Validate Parametric Model
- VII. Create an Instance
- VIII. Solve the Instance

Steps I and III are equivalent to those already performed by SysML users and it is generally straightforward to add parametrics to existing models. Step II, Infrastructure, requires the user to import a standard module into each project to enable parametrics execution. Step IV, Constraints, has the user define the generic mathematical relationships to be used. Step V “wires up” the connections between numerical attributes in the structural model and the constraint equations, using one or more SysML Parametric diagrams. Step VI, Validation, checks that the parametrics model is properly constructed for the InterCAX plug-in. In Step VII, the user must create a specific example of the model, populating some of the attributes in the model with real numbers and identifying others as unknowns to be calculated in Step VIII. This outline is not offered as a general methodology for building parametric models, so much as a helpful outline for organizing the detailed instructions.

Before the user can reproduce these tutorials, the user must install and configure

- MagicDraw 16.6
- MagicDraw SysML 16.6 plug-in
- ParaMagic 16.6, the InterCAX SysML Parametrics plug-in for MagicDraw
- Mathematica 6 or 7 (Wolfram Research)

Several features of the tutorial models are specific to the MagicDraw and ParaMagic 16.6 and may not work correctly with earlier versions. Contact NoMagic Inc or InterCAX for further information.

The fifth and sixth tutorials use two additional tools:

- Microsoft Excel
- MATLAB (The MathWorks, Inc.) with the Simulink toolkit

In each case, refer to the installation instructions in the appropriate user guide. It is also necessary to modify ParaMagic so that it points to the copy of Mathematica. We assume that MagicDraw, ParaMagic, MATLAB and Excel (if required) are all installed on the user’s local machine. Mathematica may be local or accessed through a web services interface.

The first tutorial, Addition, starts with three objects in the simplest possible relationship, $a + b = c$, and describes the steps in minute detail for those unfamiliar not only with parametrics, but with the MagicDraw SysML plug-in (MagicDraw Version 16.6) as well. In the later tutorials, we will hide more of the procedural detail as we model more complex and realistic systems.

The second tutorial, Satellite, models the weight and power budgets of a satellite system, introducing concepts of hierarchy, requirements and multiple constraints. The third tutorial, LittleEye, models the operational capability of an unmanned aerial vehicle, introducing object-oriented programming in model design and non-arithmetic functions. The fourth tutorial, CommNetwork, introduces the use of simple elements to build up and simulate more complex networks.

The fifth and sixth tutorials provide an introduction to special features for interfacing to Excel and MATLAB. The Orbital tutorial shows how Excel may be used to load initial values into a model for space mission planning and to record parametric simulation results. The HomeHeating tutorial demonstrates how external functions programmed in MATLAB can be integrated into the parametric simulation.

As in many subjects, the best way to learn SysML Parametrics is by doing. The author recommends building the models described in the first four tutorials, comparing your results with the figures in the text and exploring variations. There are generally multiple ways to implement any model and, in a few cases, alternate procedures are described. The author would appreciate user feedback on errors and unclear descriptions in this document (info@intercax.com).

1.2 Short Review of SysML

SysML is a powerful and wide-reaching language for modeling systems. In this section, we will review a few aspects of SysML of special importance to parametrics, to help make sense of the detailed instructions in the tutorials for new users with limited SysML experience. This is not intended as a broad introduction or primer on SysML.

SysML supports three major classes of diagrams, which are ways at the looking at the system model:

- Structure diagrams, which describe what the system is composed of. Parametrics is part of structure and these diagrams are our principal focus in the tutorials.
- Behavior diagrams, which describe what the system does. We will not deal with any behavior diagrams, which do not support parametrics at this time.
- Requirements diagrams, which describe the design and performance objectives the system must meet. We introduce a requirements diagram in the tutorial Satellite, to show how parametrics can help build requirements checking into a system model.

With respect to structure diagrams, there are three important types. These are illustrated in Figure 1.1.

- Block definition diagrams (BDD), which describe the organization of the structure, the hierarchy of system, subsystems, and all the elements that make up the system. In Figure 1.1, the Body, Engine, and Wheels are elements that belong to the object Automobile, the ownership relationships shown in black. Our tutorials usually begin by creating a BDD.
- Internal block diagrams (IBD), which describe qualitative flows between elements. In Figure 1.1, gasoline flows from a tank in the Body to the Engine, as shown in red. In the tutorial CommNetwork, we use an IBD to keep track of message traffic channels between stations, but IBDs do not affect parametrics directly.
- Parametric diagrams (PAR), which describe quantitative relationships between properties of the elements. In Figure 1.1, mileage, which is a property of Automobile, is a function of the drag of

the Body, the efficiency of the Engine, and so forth in green. Creating and executing parametric diagrams is the primary focus of these tutorials.

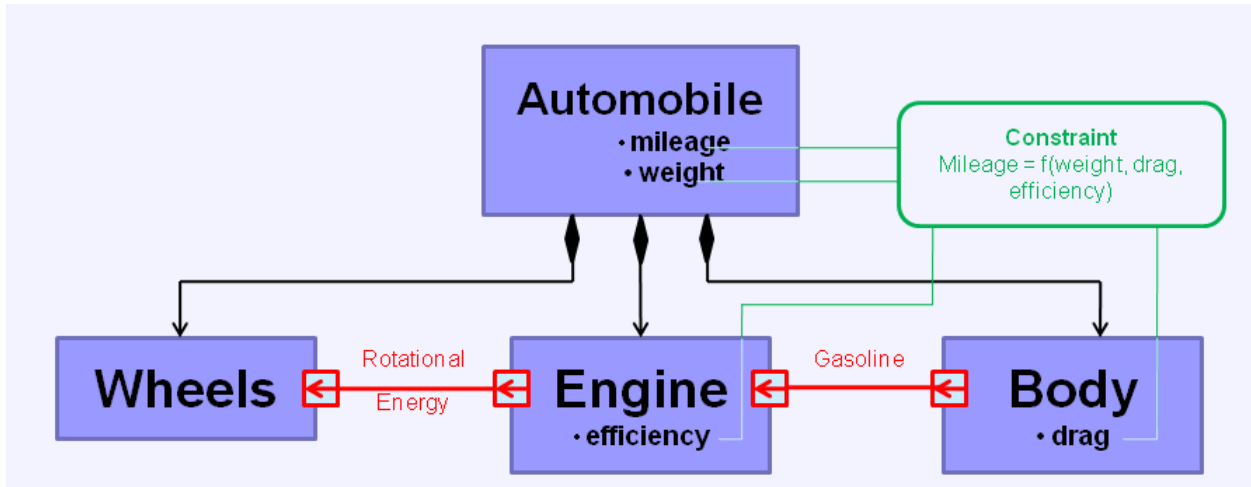


Figure 1.1 Structure diagram relationships

Finally, we need to clarify three types of objects within the system: blocks, part properties, and instances. Examples are illustrated in Figure 1.2.

- Blocks represent a generic object, like the Wheel in Figure 1.2a. A block may have value properties which describe it, like model number or radius, but these properties typically do not contain specific values.
- Part properties represent usages of a block; i.e. a block as part of some larger system. In Figure 1.2b, Front Wheel and Back Wheel are two separate roles that Wheel plays as part of Motorcycle.
- Instances represent a specific example of a generic object, like WhiteWallRadial in Figure 1.2c. The value properties have specific values, which may be fixed or calculated from other system values. ParaMagic executes parametric calculations for specific instances of system models.



Figure 1.2a Block

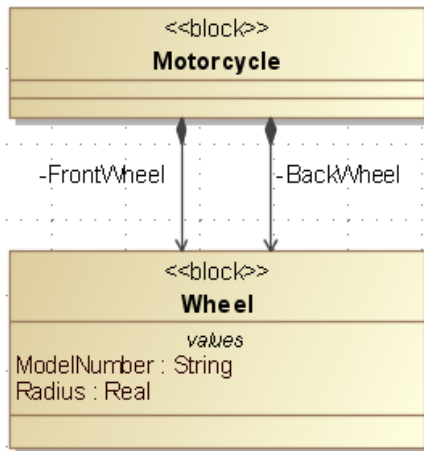


Figure 1.2b Part Properties

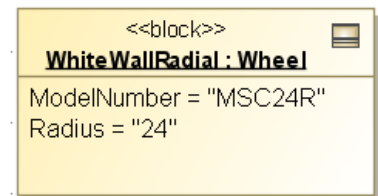


Figure 1.2c Instance of a Block

1.3 Short Review of Solving Equations

ParaMagic's primary function is to solve the often-complex network of parametric equations within the system model, so it is valuable to review a few concepts that will come up in the tutorials.

Causality is the organization of known and unknown variables in the equations. ParaMagic requires the assignment of a causality state to each variable, which can be done manually by the user or semi-automatically by the ParaMagic program. The allowable causality states are

- Given – a parameter with a known value provided by the user before the ParaMagic calculation.
- Target – a parameter with an initially unknown value that the user specifically wishes to calculate. Each ParaMagic calculation requires at least one target variable.
- Undefined – a parameter with an initially unknown value, that may be calculated in the process of solving for the target.
- Ancillary – an undefined parameter after its value has been calculated by ParaMagic. It can't be assigned before solution.

As an example, consider the two equation network

$$\mathbf{a} + \mathbf{b} = \mathbf{c} \qquad \mathbf{c} + \mathbf{d} = \mathbf{e}$$

Our objective is to calculate \mathbf{e} , which is assigned *target* causality. If we know beforehand that $\mathbf{a} = 3$, $\mathbf{b} = 2$, and $\mathbf{d} = 5$, these parameters would be assigned *given* causality. The remaining parameter, \mathbf{c} , could be assigned *undefined* causality. When \mathbf{c} is solved for in calculating \mathbf{e} , its causality changes to *ancillary*. It is also possible to assign both \mathbf{c} and \mathbf{e} to *target* causality, but multiplying targets unnecessarily may slow down solving for larger equation sets.

Assigning causality requires consideration of overconstraint/underconstraint. Underconstraint occurs when insufficient variables are assigned values (and *given* causality) to calculate the targets. For example, in the equations above, if we set $\mathbf{a} = 3$ and $\mathbf{d} = 5$, there are an infinite number of solutions for \mathbf{e} and the equation set is underconstrained. Alternately, if we assigned $\mathbf{a} = 3$, $\mathbf{b} = 2$, $\mathbf{d} = 5$, and $\mathbf{e} = 6$, the system is overconstrained and there are zero possible solutions for \mathbf{c} . In general, Mathematica, the solver engine for ParaMagic, will alert the user when overconstraint/underconstraint occurs, but some analysis by the user might be required to determine the correct number of knowns for complex equation sets.

A third issue to consider in assigning causality is reversibility. Some equations, like $\mathbf{c} = \mathbf{a} + \mathbf{b}$, are reversible or acausal. We can solve for \mathbf{c} knowing \mathbf{a} and \mathbf{b} , or we can solve for \mathbf{a} , knowing \mathbf{b} and \mathbf{c} . Other equations are not. $\mathbf{a} = \sin(\mathbf{b})$ can be solved uniquely for \mathbf{a} knowing \mathbf{b} , but may have multiple possible solutions for \mathbf{b} knowing \mathbf{a} (e.g. for $\mathbf{a} = 0$, \mathbf{b} can be equal to $0, \pi, 2\pi, \dots$). Similarly, $\mathbf{a} = \mathbf{minimum}(\mathbf{b}, \mathbf{c}, \mathbf{d})$ is not always reversible. ParaMagic treats these types of equations as "one-way" and causality assignments must take this into account.

2 SYSML PARAMETRICS TUTORIAL - ADDITION

2.1 Objective

Create a SysML project with three elements. Each element has one attribute. The attribute of the third element is the sum of the first two attributes. Create an instance of this model and solve for the third attribute parametrically.

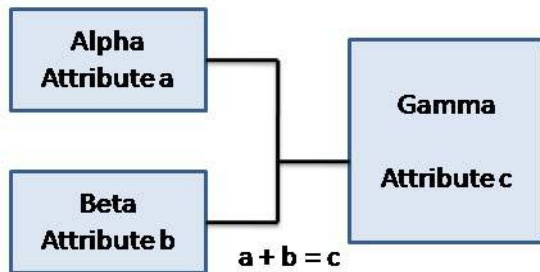


Figure 2.1 Outline of Objective

What the User Will Learn

- Creating the basic elements of SysML models in MagicDraw: blocks, properties, constraints, etc.
- Building a SysML parametrics model and diagram
- Creating an instance of the model with input and output parameters
- Opening and using the ParaMagic browser window
- Exporting a parametrics model to Mathematica

2.2 Step-by-Step Tutorial

Step I Create Project

1. Open MagicDraw
2. On MagicDraw menu bar, select Options→Perspectives→Perspectives and set to System Engineer
3. Create new project
 - a. On MagicDraw menu bar, select File→New Project,
 - b. In New Project window (Figure 2.2), choose SysML Project,
 - c. Set Name = Addition
 - d. Set or browse for location to save project.
4. Create a package within the project
 - a. RC (Right-click) on Data folder in Containment tree (Figure 2.3)
 - b. Select New Element→Package
 - c. Enter Name = Addition

Note for Mac Users: Right-Click is substituted by the Mac keystroke: Control-Click. While ParaMagic is compatible for both Windows and Macintosh operating systems, certain keystrokes and paths are bound to vary. The author will make note of any significant differences in ParaMagic for Mac users hereafter.

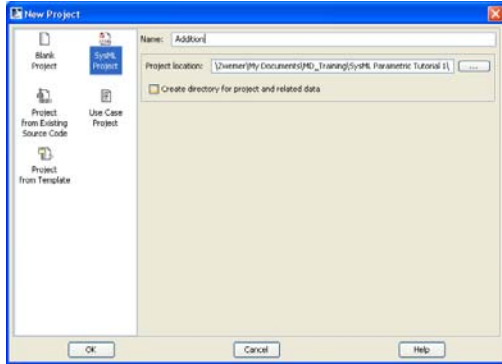


Figure 2.2 New Project window

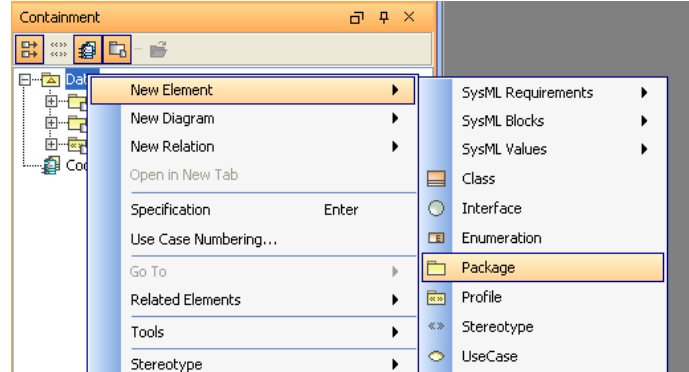


Figure 2.3 Creating Package in Data folder

Step II Create Infrastructure

5. Install ParaMagic Profile module

- a. On MagicDraw menu bar, select File→Use Module... and select as below (Figure 2.4). This loads a module containing all the ParaMagic features as part of the project.

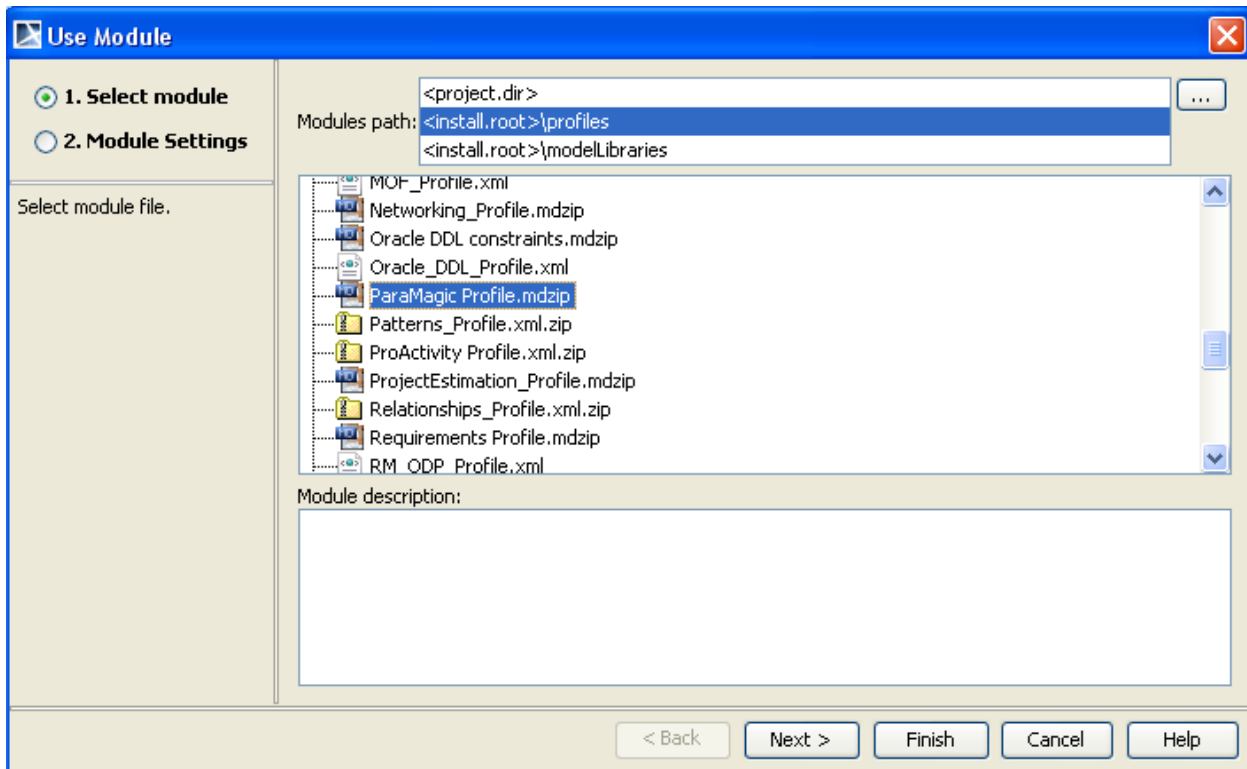


Figure 2.4 Use Module window

- i. Click Next (Figure 2.5)

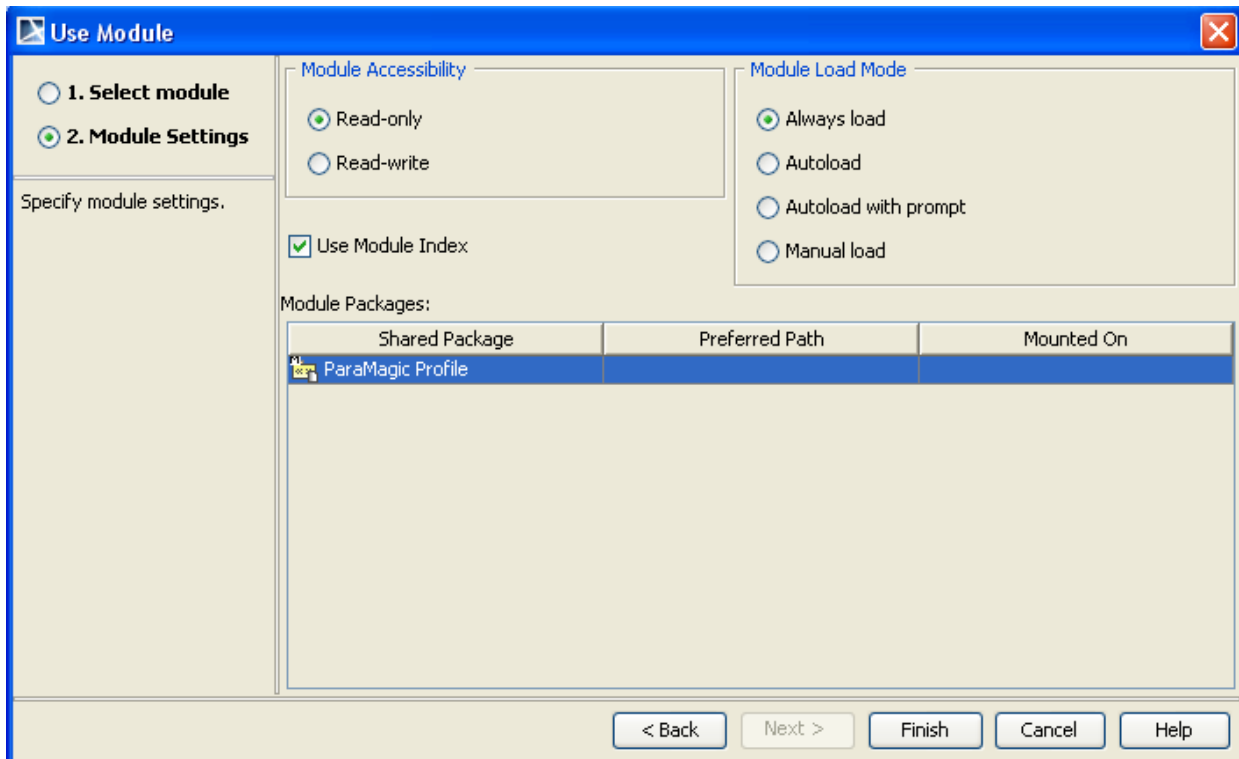


Figure 2.5 Module settings for ParaMagic Profile

- ii. Select Finish. If program asked whether to show auxiliary resources, click Yes to display other profiles present in Containment window tree.

Step III Create Structural Model

6. Create elements in model
 - a. RC (Right-click) Addition package in Containment window
 - i. Select New Element→SysML Blocks→Block
 1. Enter Name = Alpha in blank text box
 - b. RC Addition
 - i. Select New Element→SysML Blocks→Block
 1. Name = Beta
 - c. RC Addition
 - i. Select New Element→SysML Blocks→Block
 1. Name = Gamma
 - d. RC Alpha
 - i. Select New Element→SysML Properties→Value Property
 1. Name = A
 2. To set the Type of the property, DC (double click) on A and enter Real in the Value Property window (Fig. 1.6) or select Real from dropdown list
 - e. RC Beta
 - i. Select New Element→SysML Properties→Value Property
 1. Name = B
 2. Type = Real
 - f. RC Gamma
 - i. Select New Element→SysML Properties→Value Property

1. Name = C
2. Type = Real

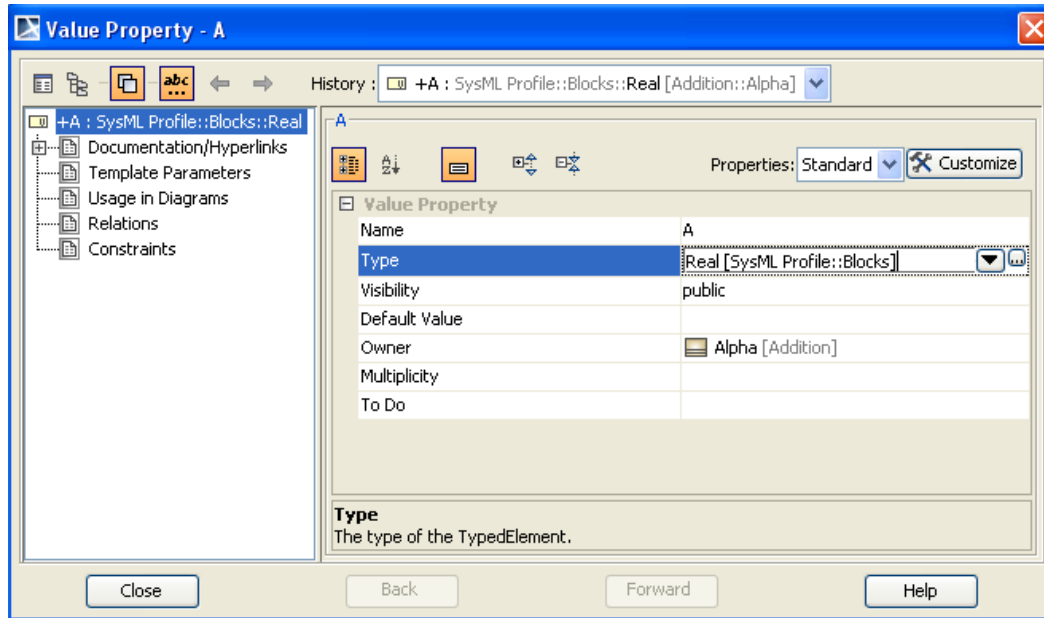


Figure 2.6 Value Property window, enter Type

7. Create a block definition diagram, a graphical view of the entire system
 - a. RC Addition
 - b. Select New Diagram→SysML Diagrams→SysML Block Definition Diagram
 - i. Name = AdditionBDD
 - c. Drag Alpha, Beta, and Gamma from Containment window to BDD (Figure 2.7)
 - i. To rearrange position, drag using top half of block.
 - ii. To resize block, click on top half of block and use cursors at corners
 - iii. If attributes (e.g. A) are not listed inside the blocks, display the attributes by,
 1. RC on block,
 2. choose Presentation Options,
 3. uncheck Suppress Attributes.

Alternatives: To show or hide the interior features of a block, right-click on the block and use the Symbol Properties, Edit Compartment and Presentation Options commands. Alternately, click on the block and look for small plus and minus icons on the left edge to display or suppress interior features.

8. Create relationships between elements of the model. Alpha and Beta can be considered as parts of Gamma.
 - a. Click on Gamma in Block Definition Diagram
 - b. Select Directed Composition arrow from floating toolbar (arrow with solid diamond at base).
 - c. Drag end of arrow to Alpha
 - d. Repeat steps a-c for Beta (Figure 2.8)

- e. This procedure causes blocks Alpha and Beta to be mirrored as Part Properties inside Gamma. Under the Gamma block in the Containment window, assign these Part Properties the names alp and bet, respectively.

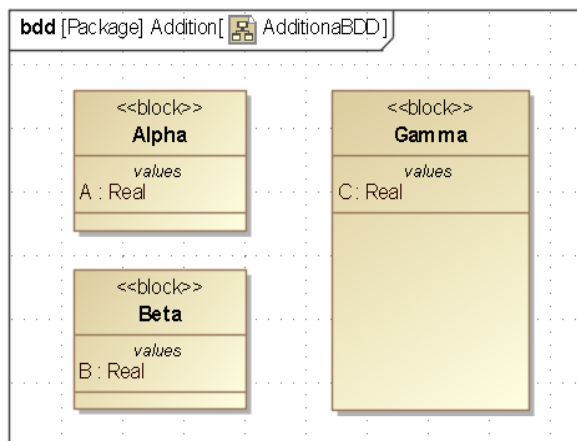


Figure 2.7 Block Definition Diagram

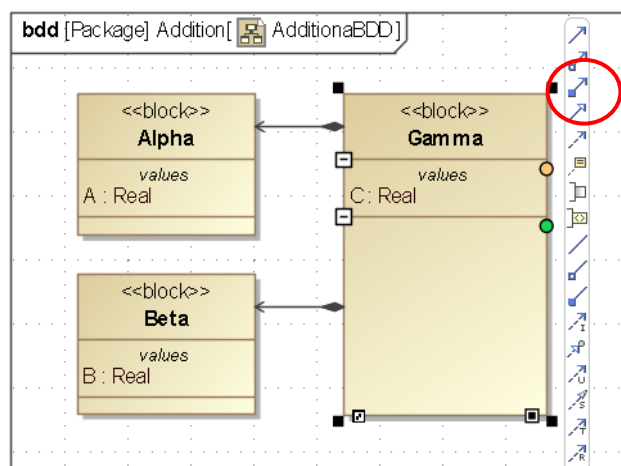


Figure 2.8 Block Definition Diagram with Floating Toolbar and Directed Composition Arrows

Step IV Create Constraints

9. Create a constraint block, which contains a mathematical relationship that the model will use.
 - a. RC Addition
 - i. Select New Element→SysML Blocks→Constraint Block
 1. Name = AdditionEqn
 - b. RC AdditionEqn
 - i. Select New Element→Constraint Parameter
 1. Name = a
 2. Type (of a) = Real
 - c. RC AdditionEqn
 - i. Select New Element→Constraint Parameter
 1. Name = b
 2. Type (of b) = Real
 - d. RC AdditionEqn
 - i. Select New Element→Constraint Parameter
 1. Name = c
 2. Type (of c) = Real
 - e. DC (double-click) AdditionEqn
 - i. In the window labeled Constraint Block – AdditionEqn (Figure 2.9),
 1. select Constraints
 2. click Create
 3. under Name, enter sum

4. under Specification, enter $c = a + b$ (putting a space between each parameter and operation is good procedure to enable Mathematica to parse the constraint expression correctly)
 5. click Apply
 6. click OK
 7. click Close
- f. Drag the constraint block, AdditionEqn, from the Containment window to the BDD diagram.

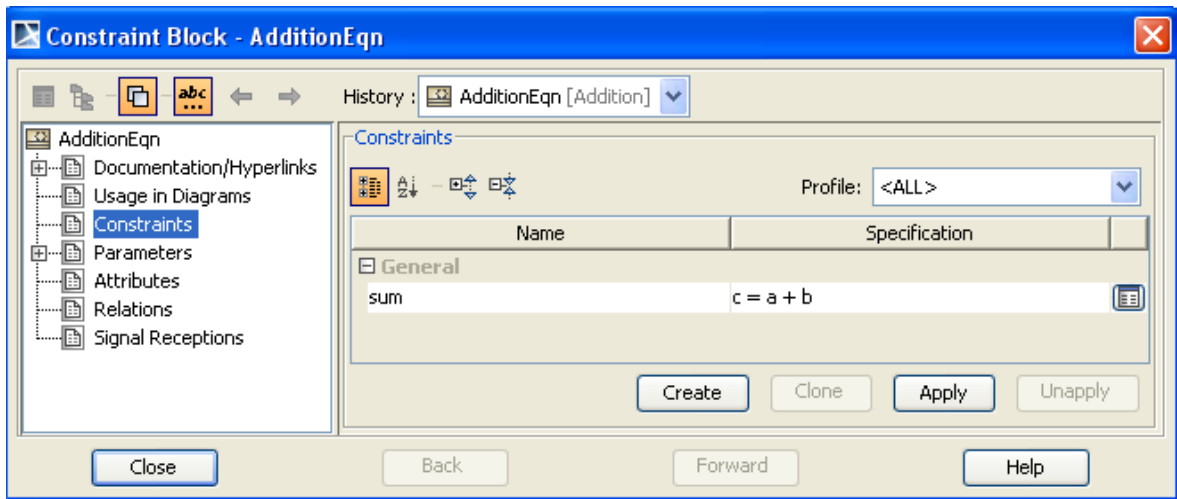


Figure 2.9 Constraint Block window showing Constraint

Step V Create Parametrics Model

10. Create a SysML Parametric diagram to define and display the relationships inside Gamma
 - a. RC Gamma
 - b. Select New Diagram → SysML Parametric Diagram
 - i. The Select Parts window will appear. Select Enter, and all the value properties and part properties of Gamma will appear automatically on the diagram.
 - ii. The diagram name is Gamma
 - c. Drag the Constraint Block AdditionEqn into the Gamma parametric diagram.
 - i. AdditionEqn will appear inside Gamma (in the Containment tree and in the parametric diagram) as a Constraint Property of the type AdditionEqn. Assign it a unique name, add.
 - ii. Use the Display Parameters icon (Figure 2.10) and SelectParameters window (Figure 2.11) to show parameters a, b and c.
 - iii. Arrange them at top of block as shown in Figure 2.13.
 - d. Display the internal structure of alp:Alpha by
 - i. RC on alp:Alpha

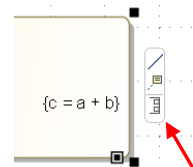


Figure 2.10

- ii. Select Edit Compartment→Structure from list
- iii. Select A from the Edit Compartment window (Figure 2.12)
- iv. Move it to the right side of window using central arrow or double arrow buttons
- v. Click OK
- vi. Repeat step i.-vi. for B in bet:Beta

Note: In general, Edit Compartment and Presentation Options provide control in displaying or hiding properties inside blocks, but displaying ports or parameters inside constraint blocks and constraint properties uses the floating toolbar icons like in Figure 2.10.

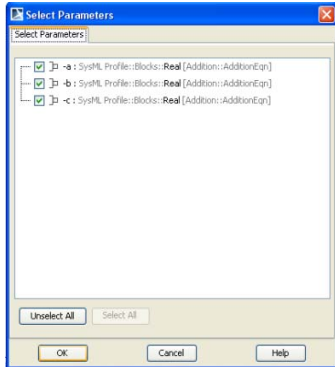


Figure 2.11 Select Parameters window

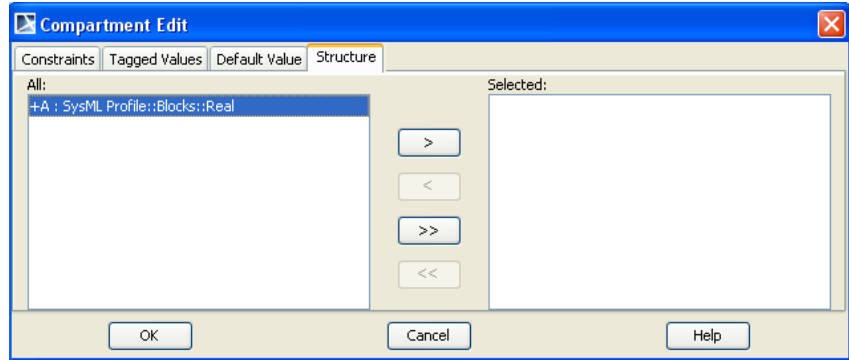


Figure 2.12 Edit Compartment window

- e. The parametric diagram should appear something like Figure 2.13.

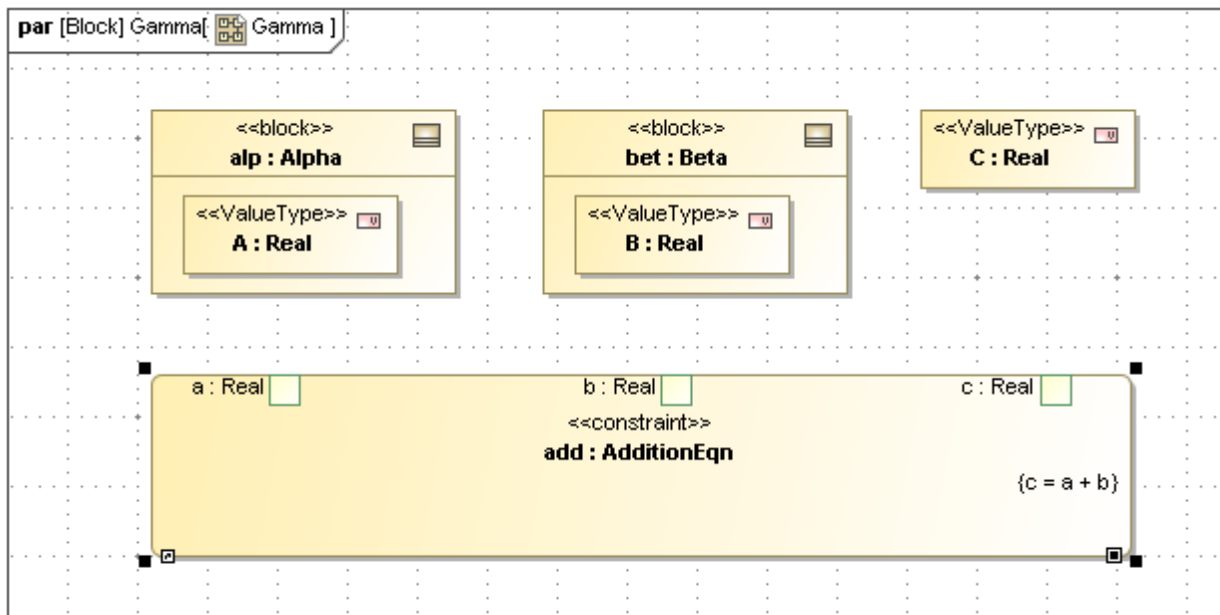


Figure 2.13 Parametric Diagram before Connectors added

- f. Create the connections between the model attributes (A, B, and C) and the constraint parameters (a, b, and c).
 - i. Click on parameter a in the add:AdditionEqn constraint block.

- ii. From the floating toolbar that appears, choose the Connector icon (simple straight line, see highlight in Figure 2.14) and drag the end to A in block alp:Alpha. If the connection is yellow, then the parameter is incorrectly linked to the block, alp:Alpha, instead of the value property, A:Real.
- iii. Repeat steps i. and ii. for b to B and c to C connectors.
- iv. At this stage, the parametric diagram should look like Figure 2.14 and the Containment tree should look like Figure 2.15.

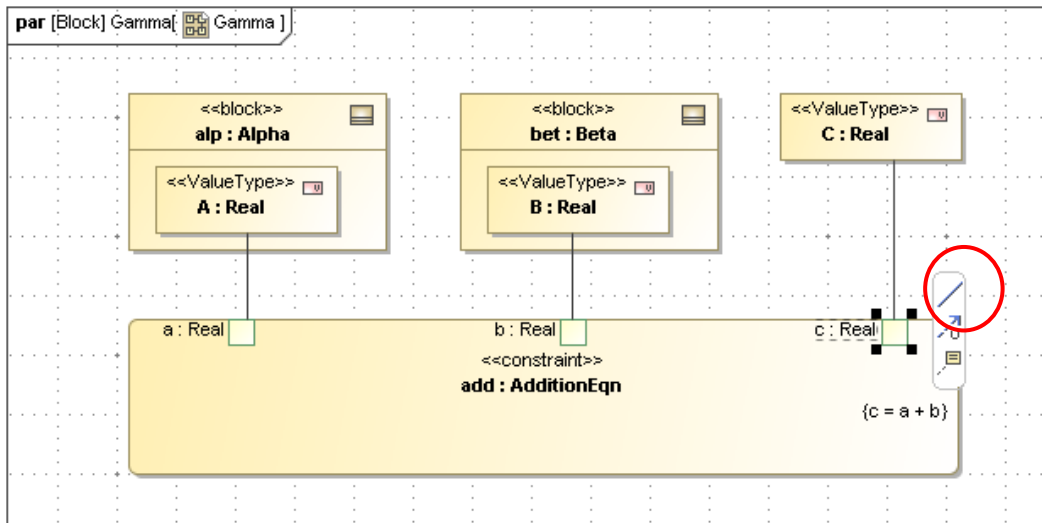


Figure 2.14 SysML Parametrics Diagram with connectors, connector icon highlighted

Step VI Validate Parametrics Model

11. To create the CXS_heading and validate the model schema,
 - a. Right-click on block Gamma in the Containment tree
 - i. select ParaMagic→Util→Create CXS_heading
 - b. If a CXS_heading object already exists, a prompt asks whether to overwrite. Press OK.
 - c. Enter 1.0 or some other string at the prompt for a version number. This feature allows the user to identify the specific schema version being validated
 - d. A new CXS_heading block will appear in the Containment tree. The CXS_heading block property root_cob

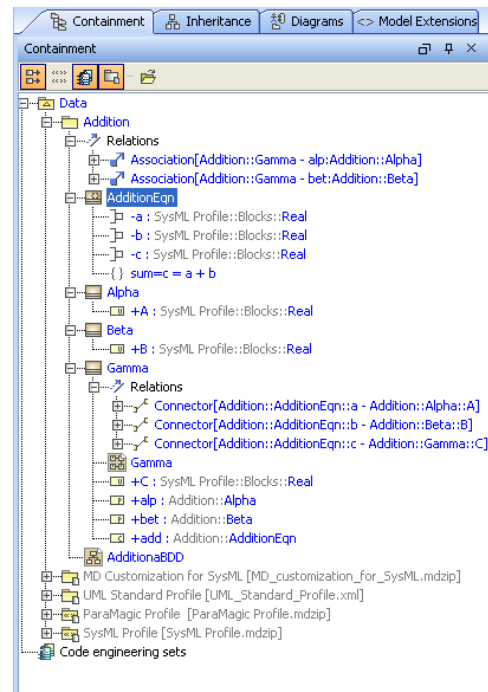


Figure 2.15 Containment Tree

points the parametrics solver at the root element containing the highest level parametric relationships, in this case, Gamma.

- e. RC on Addition in the Containment tree
 - i. select ParaMagic→Validate
- f. If the message reads “SysML block structure is valid. Warning: schema structure validation does not check for equation over-constraints, etc. –See Users Guide Limitations”, go on to Step VII. Otherwise, correct model errors, which will be specified in the message window at the bottom of the screen, and repeat validation.

Step VII Create an Instance

- 12. Create an instance. An instance is an example of the model with specific values assigned to the given parameters and which can be solved for the unknown(s).
 - a. RC Addition
 - i. Create New Element→Package
 - 1. Name = AdditionInstance01
 - b. RC AdditionInstance01
 - i. Create New Diagram→SysML Diagrams→SysML Block Definition Diagram
 - 1. Name = Instance01

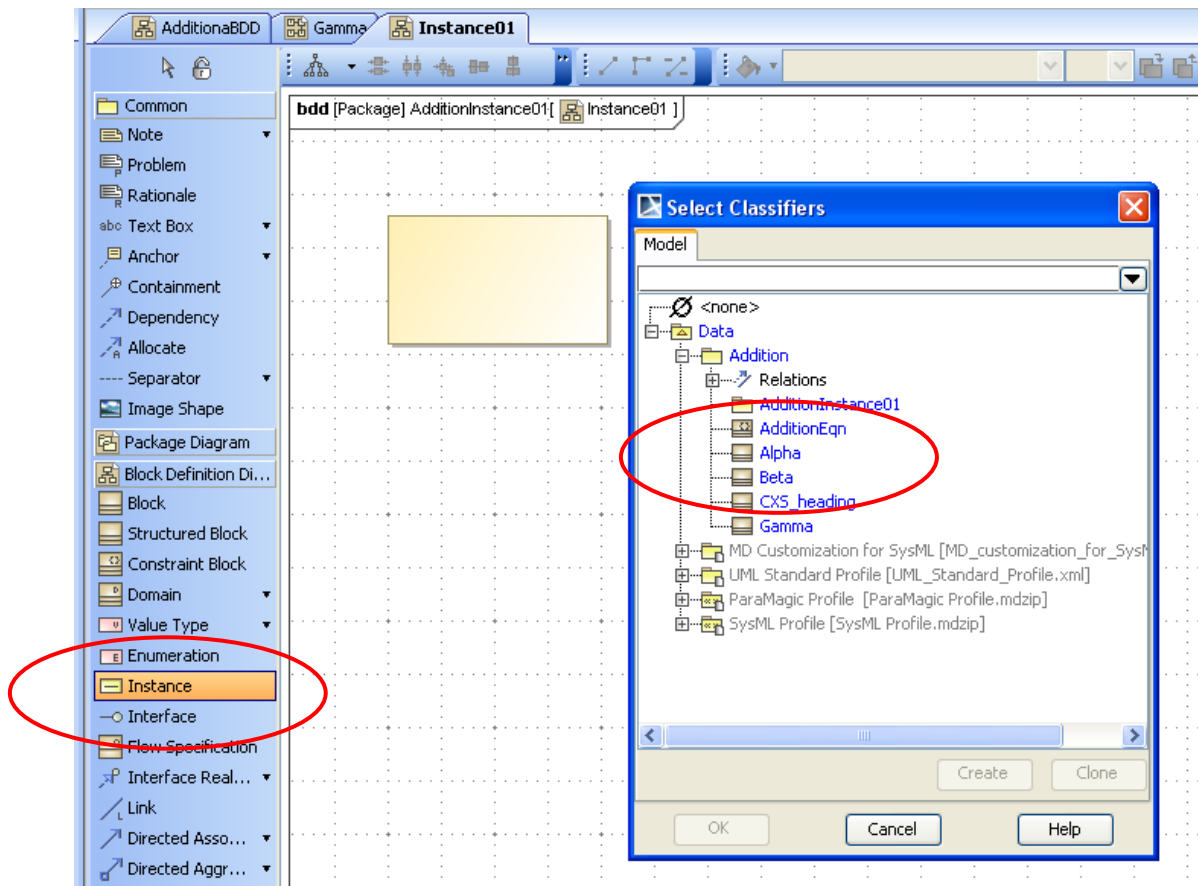


Figure 2.16 Creating an instance of Alpha inside AdditionInstance01

- c. Click on Instance from the central toolbar (highlighted in Figure 2.16) and click inside the diagram to draw an Instance block.
 - i. Select Alpha on the Select Classifiers window and press OK.
 - ii. DC :Alpha block on Instance01 and name alpha01.
 - iii. To display parameters inside alpha01,
 1. DC on alpha01 (see Figure 2.17)
 2. Select Slots,
 3. DC on A, and Click Close.

Alternative: In the next tutorial, we introduce the “Make Instance” utility to expedite the process of creating instance from each block in the schema.

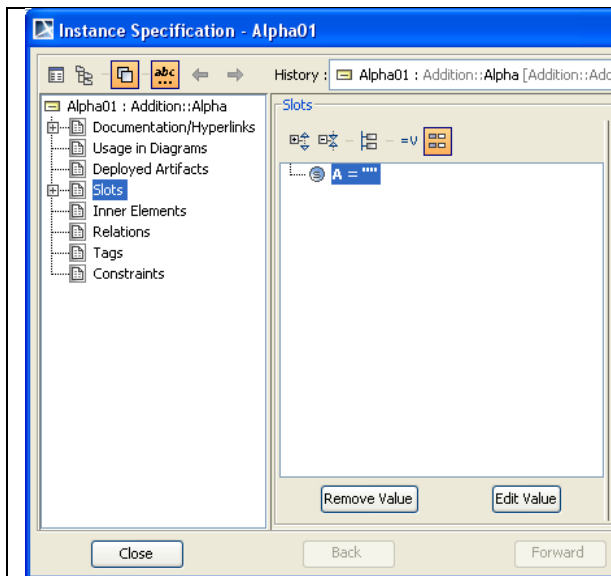


Figure 2.17 Instance Specification

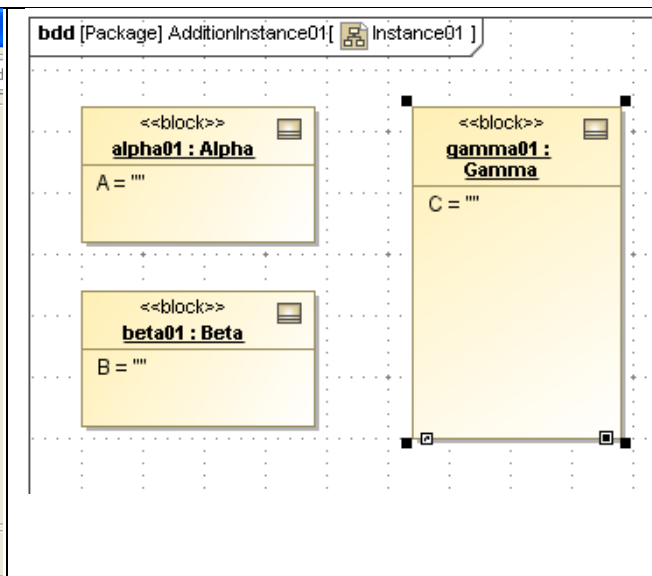


Figure 2.18 Instance diagram, preliminary

- d. Repeat process in step c for B in beta01:Beta and C in gamma01:Gamma. Instance diagram should appear similar to Figure 2.18.
- e. Click gamma01 and select connector (top solid line on floating toolbar).
- f. Draw the connector from gamma01 to alpha01
 - i. On the Select Association window, check the box by the association and click OK. See Figure 2.19.
 - ii. On the Create Slots window, check alp:Alpha and click OK (Figure 2.20).
- g. Repeat steps e and f for connector from gamma01 to beta01. See diagram in Figure 2.21.

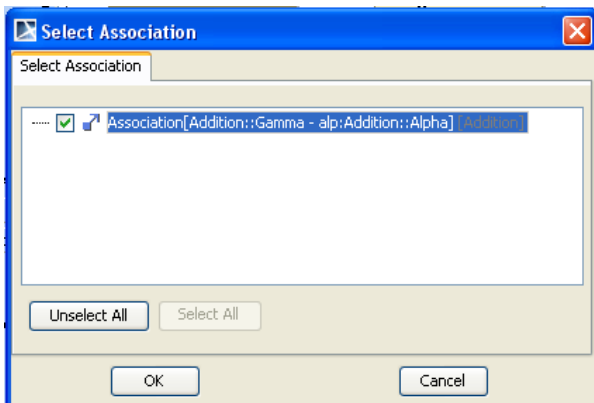


Figure 2.19 Select Association window

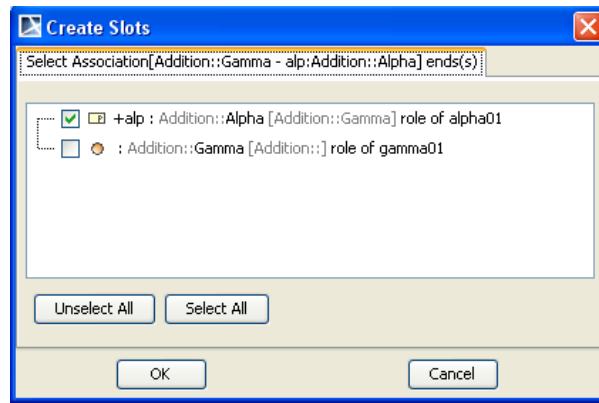


Figure 2.20 Create Slots window

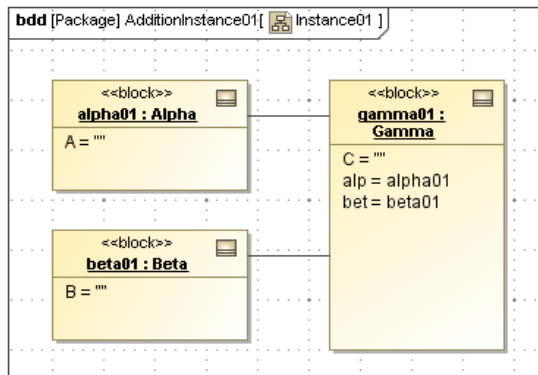


Figure 2.21 Instance diagram, preliminary

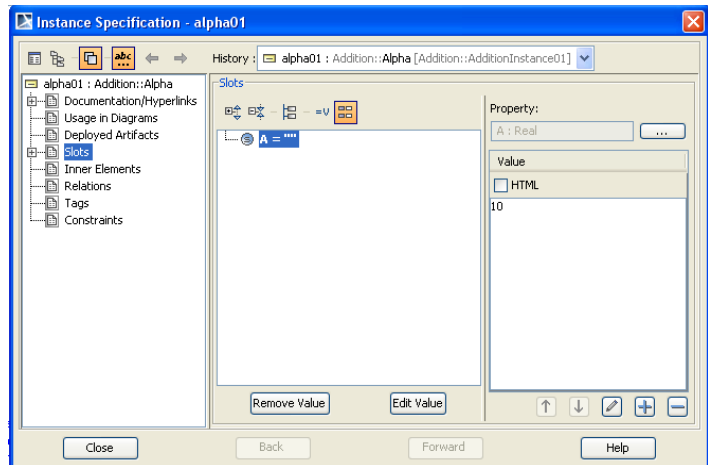


Figure 2.22 Setting value for A in Instance01

13. Add values to the instance and define inputs and outputs

- a. Add a value to A for solving in the instance.
 - i. Double-click on alpha01,
 - ii. click on Slots,
 - iii. click on A = "" in central Instance Specification window (Figure 2.22),
 - iv. enter "10" as value on right side of window.
 - v. Expand Slots (click on + box next to Slots),
 - vi. expand A = "",
 - vii. select Tags (see Figure 2.23).
 - viii. DC on <<causality>> type.
 - ix. Select "given" from dropdown list and Close.
- b. Repeat step a for beta01, specifying "20" as value.
- c. Repeat step a for gamma, but do not specify a value and for <<causality>>, select "target". Final instance diagram should appear similar to Figure 2.24.

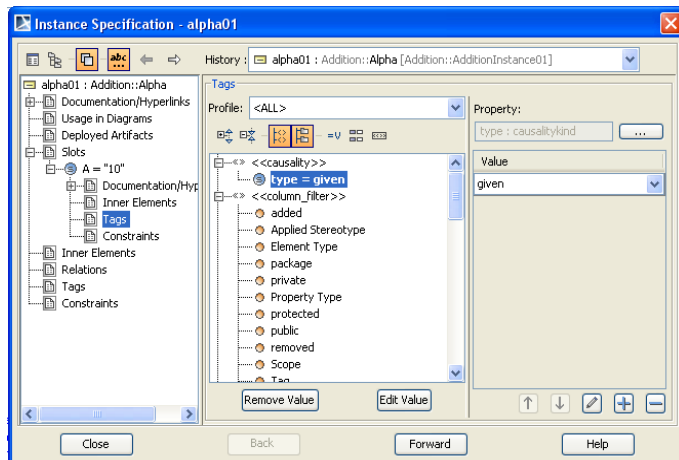


Figure 2.23 Instance diagram, preliminary

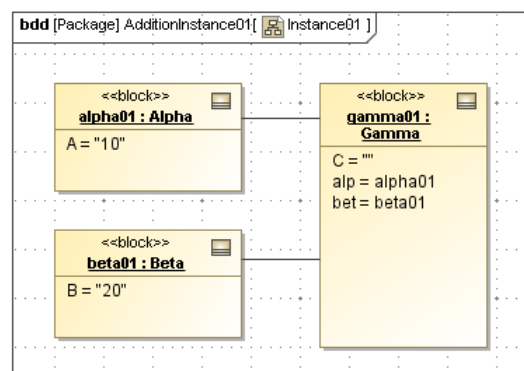


Figure 2.24 Instance diagram, final

Alternative: In the next tutorial, we introduce the “Assign default causalities” utility to minimize the process of assigning causality to each variable.

14. Create a CXI_heading, which contains a string that uniquely identifies the file generated in the following step
 - a. RC AdditionInstance01
 - i. Select ParaMagic→Util→Create CXI_heading,
 - ii. Assign an identifying string such as “1.0”, to the version,
 - iii. Click OK.

Step VIII Solve the Instance

15. Run the parametric solver
 - a. RC AdditionInstance01 in the Containment tree.
 - b. Select ParaMagic→Browse. Browser should show up like Figure 2.25.

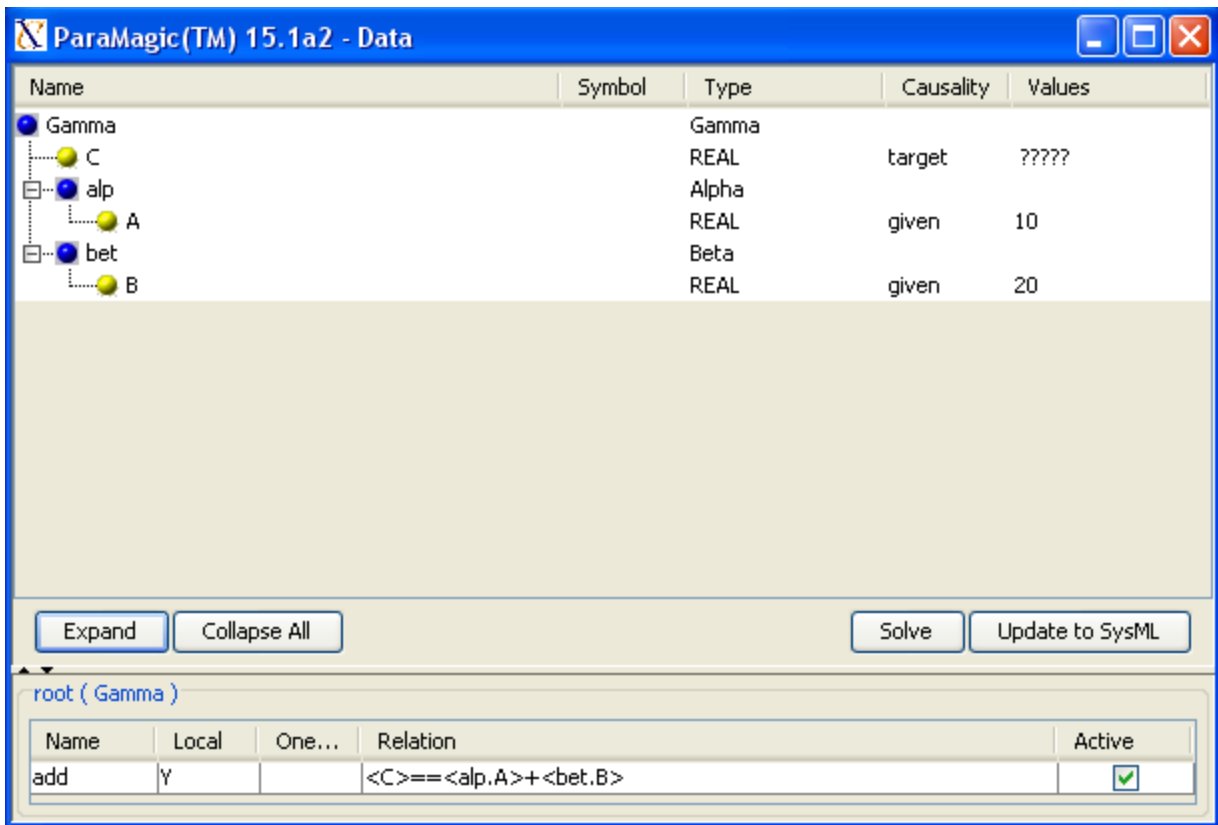


Figure 2.25 ParaMagic Browser window

- c. Press “Solve”
 - i. The ????? symbols in the target variables should change to their calculated values, in this case C = 30 (Figure 2.26).
- d. Press “Update SysML”. C=30 in the Containment tree or Instance diagram

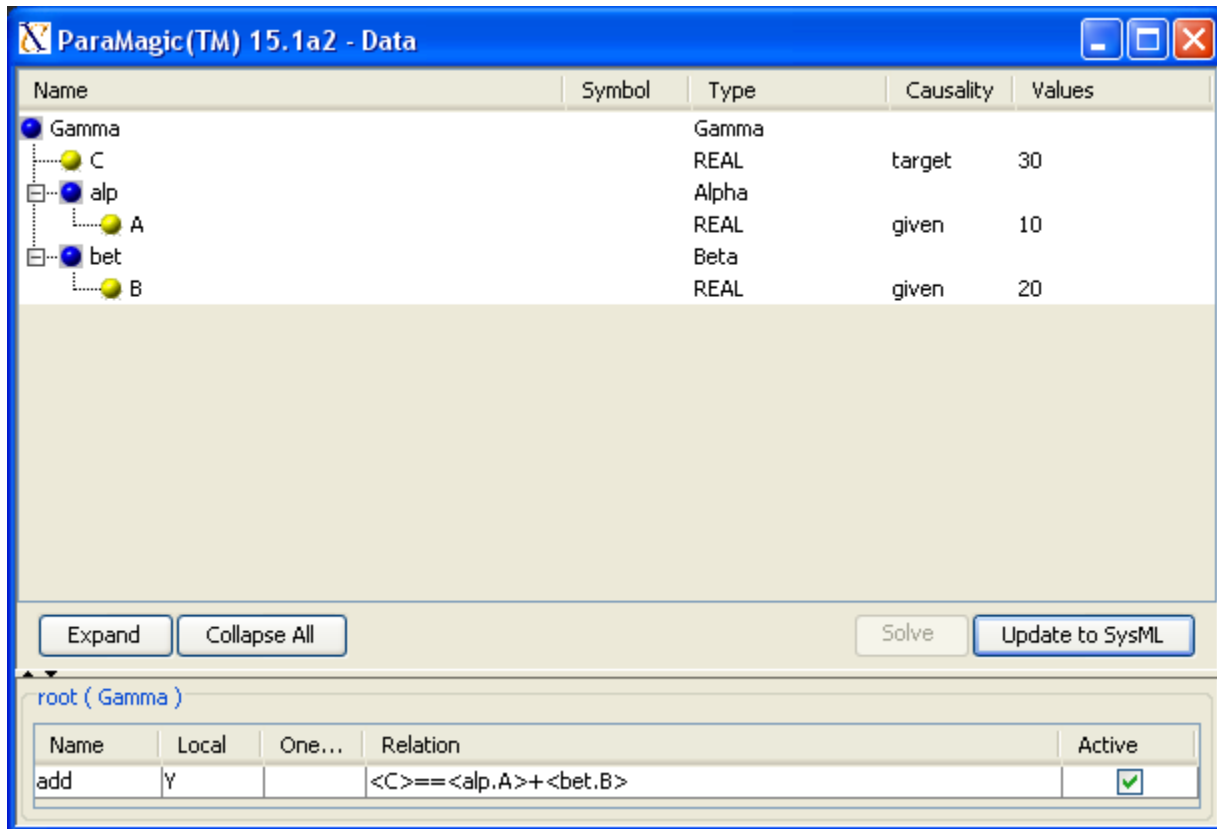
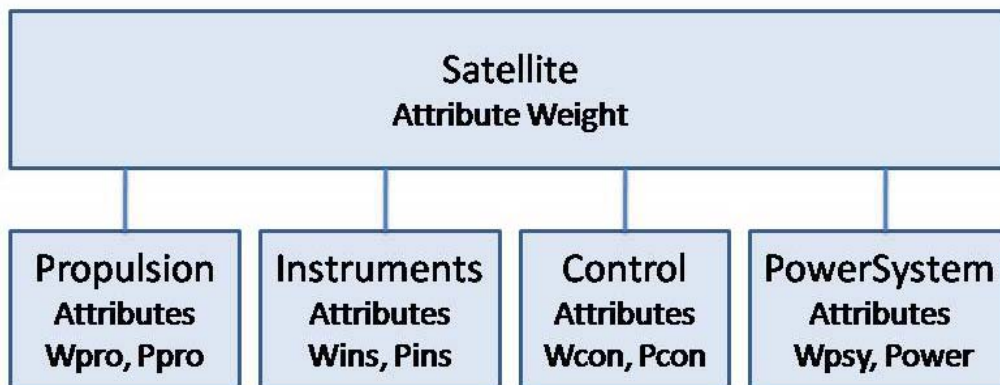


Figure 2.26 Browser window with solution

3 SYSML PARAMETRICS TUTORIAL - SATELLITE

3.1 Objective

Create a SysML project comprising a satellite system composed of four subsystems: Propulsion, Instrumentation, Control and PowerSystem. Each subsystem has a weight and there is a total weight budget for the satellite of 10,000 kilograms. The Propulsion, Instrumentation, and Control subsystems each draw electrical power from the PowerSystem, which can supply a maximum of 10 megawatts. Given the weights of all four subsystems and the power requirements of three, calculate the total weight and power demand for the system and compare against requirements.



$$\text{Weight} = W_{\text{pro}} + W_{\text{ins}} + W_{\text{con}} + W_{\text{psy}} \leq 10,000 \text{ kg}$$

$$\text{Power} = P_{\text{pro}} + P_{\text{ins}} + P_{\text{con}} \leq 10,000 \text{ kW}$$

Figure 3.1 Outline of Objective

What the User Will Learn

- Apply SysML parametrics to a realistic system with subsystems
- Working with multiple constraints
- Working with requirements
- Working with ValueTypes
- Changing causality – reversing the direction of calculation

3.2 Step-by-Step Tutorial

Step I Create Project

1. Create new SysML project
 - a. Name = Satellite
2. Create a package within the project
 - a. RC (Right-click) on Data folder in Containment tree (left column)
 - b. Choose New Element→Package
 - c. Enter Name = Satellite

Step II Create Infrastructure

3. Install ParaMagic Profile module, same as in the first tutorial.

Step III Create Structural Model

4. Create elements in model
 - a. RC on Satellite
 - i. Create New Element→Package
 1. Name = ValueTypes
 - ii. RC on ValueTypes, create New Element→SysML Values→ ValueType
 1. Name = Kilowatt
 2. Base Classifier = Real
 - a. DC on Kilowatt in Containment tree
 - b. In Value Type – Kilowatt window click to the right of Base Classifier row in table
 - c. Click on button with three dots
 - d. In Select Elements window, enter **Real** in Search by name text box
 - e. Select Real (SysML Profile::Blocks)
 - f. Click OK and Close.
 - iii. RC on ValueTypes, create New Element→SysML Values→ ValueType
 1. Name = Kilogram
 2. Base Classifier = Real
 - b. RC Satellite
 - i. Create New Element→SysML Blocks→Block
 1. Name = SatelliteSystem
 - ii. In SatelliteSystem, create three new value properties
 1. Name = Weight, Type = Kilogram
 2. Name = Weight_MOS, Type = Real
 3. Name = Power_MOS, Type = Real

Discussion – ValueTypes, Units, and Dimensions

We frequently want to apply units to a value property, for example, electrical power in our model will be expressed in kilowatts. We do this by assigning a ValueType to the value property. We can use the Type property to make this assignment. We create two new ValueTypes in step 4.a above, Kilograms and Kilowatts. ParaMagic expects valuetypes used for parameters to be subtypes of the Real valuetype, so we use the Base Classifier entry to make this assignment.

Assigning valuetypes to properties makes the model more exact and helps identify mismatches when block properties and constraint parameters are linked in parametric diagrams. Note that units are not assigned to value properties directly, so the Kilogram unit in the MD SysML profile is not used in step 4.b.ii.1 above. The full description of the relationship between ValueTypes, Units and Dimensions is described in the MagicDraw User Guide.

- c. RC on Satellite
 - i. Create New Element→SysML Blocks→Block

1. Name = Propulsion
- ii. In Propulsion, create two new block properties
 1. Name = Wpro, Type = Kilogram
 2. Name = Ppro, Type = Kilowatt
- d. Repeat step b. for Instrumentation, Control, and PowerSystem subsystems with attribute names as shown in Figure 3.2
- e. RC on Satellite
 - i. Create New Diagram→SysML Diagrams→SysML Block Definition Diagram
 - ii. Name the diagram SatelliteBDD.
 - iii. Drag SatelliteSystem, Propulsion, Instrumentation, Control and PowerSystem from the Containment tree into the diagram and arrange
- f. Click on SatelliteSystem, choose a Directed Composition arrow from the Links toolbar, and drag the link to Propulsion.
- g. Name part property Pro1
- h. Repeat steps f. and g. for Instrumentation, Control, and PowerSystem subsystems as shown in Figure 3.2.

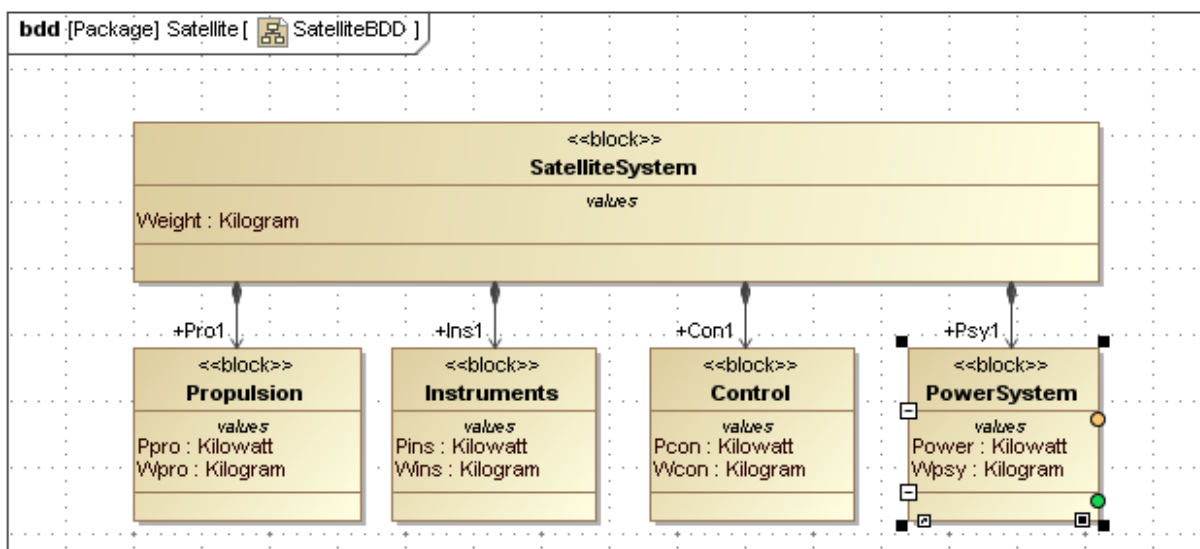


Figure 3.2 SysML Block Definition Diagram

Step IV Create Constraints

5. Inside Containment tree window, create two constraint blocks, which contains a mathematical relationship that the model will use for weight and power calculation
 - a. RC Satellite, select New Element→SysML Blokcks→Constraint Block, Name = WeightBalance
 - b. RC WeightBalance, create New Element→Constraint Parameter
 - i. Name = w, Type = Kilogram
 - c. Copy w, paste four times inside WeightBalance and rename to create parameters w1, w2, w3 and w4.

- d. DC (double-click) WeightBalance
 - i. In the window labeled Constraint Block
 1. select Constraints
 2. click Create
 3. under Name, enter weightbalance
 4. under Specification, enter $w = w1 + w2 + w3 + w4$
 5. click Apply
 6. click OK in Select Element dialog box
 7. click Close
- e. Create a second constraint block, name = PowerBalance, with four constraint parameters (p, p1, p2, p3), Type = Kilowatt, and one constraint, name = powerdemand, specification = $p = p1 + p2 + p3$, using the same process as steps as a.-d.
- f. Drag both new constraint blocks into the SatelliteBDD block definition diagram, which will appear similar to Figure 3.3.

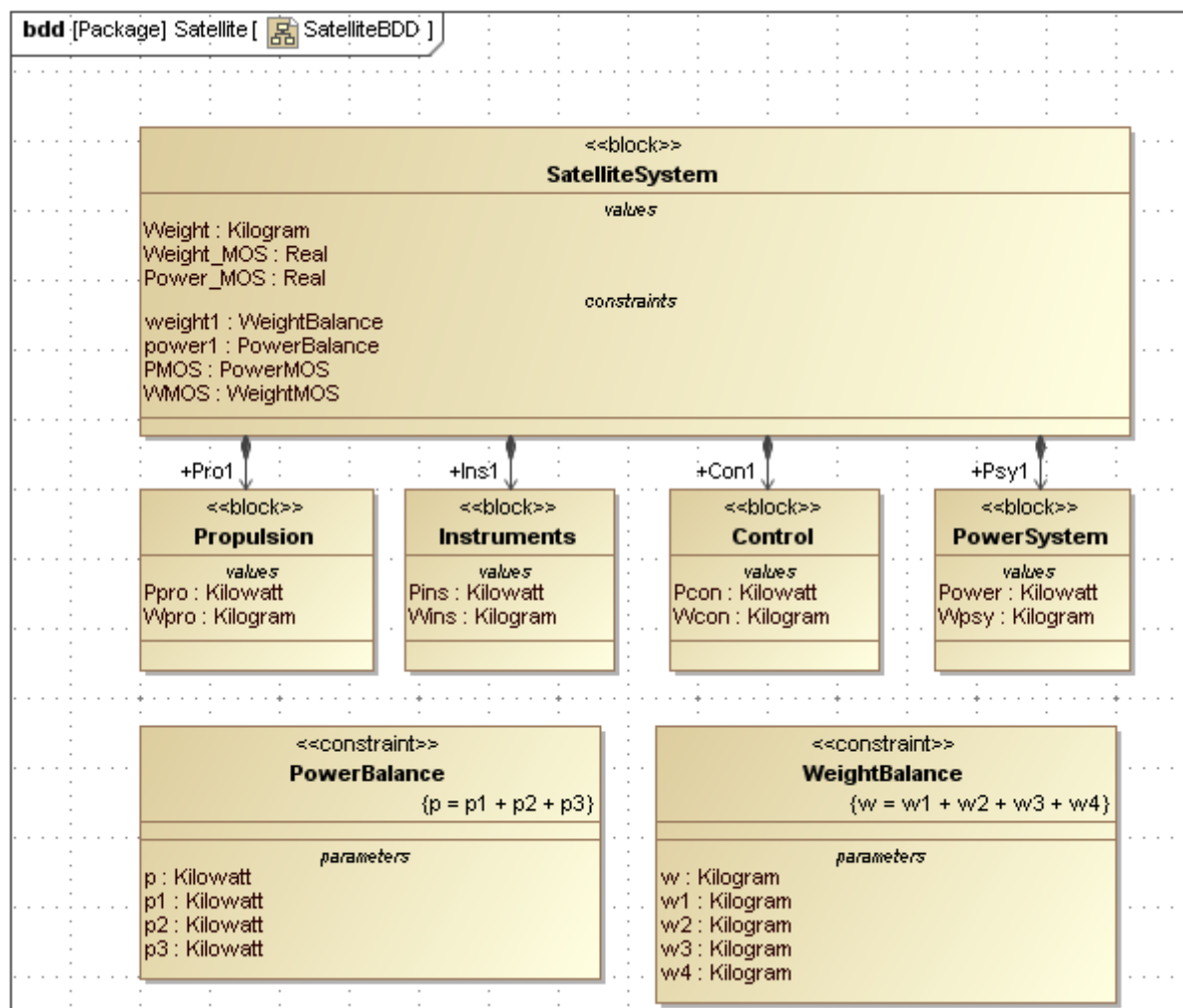


Figure 3.3 SysML Block Definition Diagram with Constraints

6. Create a Requirements diagram to show the specifications for the Satellite system. The purpose of a Requirements diagram is to make clear the requirements the system must meet and how these requirements tie to specific values of the model. For our purpose, we can pair each Requirement with a Constraint Block that mirrors it, making it easy to automatically verify the constraint when ParaMagic is executed.
 - a. RC on Satellite, create New Element→Package, Name = SatelliteReqs
 - b. RC on SatelliteReqs, create New Diagram→SysML Diagrams→SysML Requirements Diagram, Name = SatelliteReqs
 - c. Drag two requirements blocks from the center toolbar into the diagram
 - i. Name = WeightReqt, id = 1.0.1, text = Total system weight must be less than 10,000 kilograms
 - ii. Name = PowerReqt, id = 2.0.1, text = Total system power use must be less than 10,000 kilowatts
 - d. RC on SatelliteReqs, create New Element→SysML Blocks→Constraint Block
 - i. Name = WeightMOS (Margin of Safety)
 - e. RC on WeightMOS (in containment tree),
 - i. Create a New Element→Constraint Parameter, Name = mos, Type = Real
 - ii. Create a New Element→Constraint Parameter, Name = actual, Type = Kilogram
 - iii. Create a constraint, Name = wtmos, Specification = $mos = (10000 - actual)/10000$. This constraint calculates the margin of safety as the difference between the target value, 10,000 kg, and the actual weight as calculated, divided by the target value. It will be positive if system weight is below the target value, negative if it exceeds the target.
 - f. Repeat step e. for PowerMOS (see Figure 3.4 for specifics)
 - g. Drag PowerMOS and WeightMOS into the Requirements diagram
 - h. Use the Verify arrow (central toolbar) to show the relationship between each Constraint Block and the Requirement it verifies.

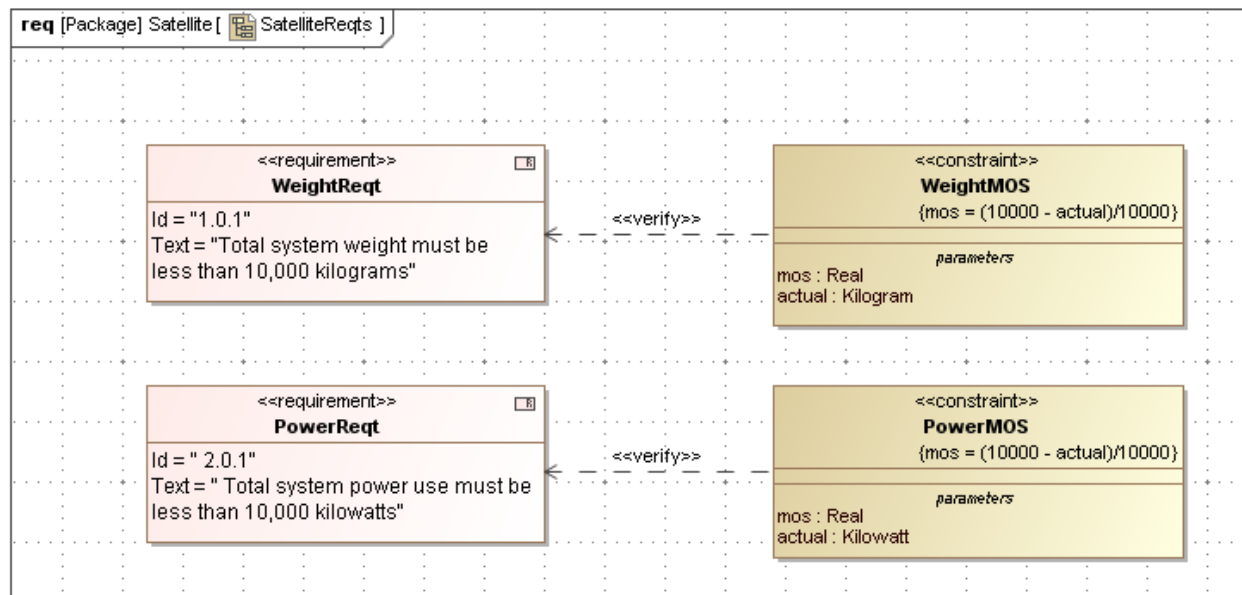


Figure 3.4 SysML Requirements Diagram

Step V Create Parametrics Model

7. Create a SysML Parametric diagram to define and display the relationships inside SatelliteSystem
 - a. RC SatelliteSystem
 - b. Create New Diagram→SysML Parametric Diagram, Name = SatelliteSystem
 - c. Use the Select Parts window to select the Weight Value Property and four Part Properties inside SatelliteSystem. Uncheck Value Properties: Weight_MOS and Power_MOS as they will be used in the second Parametric Diagram.
 - i. Use the Edit Compartment and Presentation Options commands to display the internal structure as shown in Figure 3.5
 - d. Drag a constraint property icon from the central toolbar into the parametric diagram
 - i. In Select/Create Part for Type window, choose WeightBalance
 - ii. Name the constraint property weight1.
 - iii. Click on weight1 and select the Display Parameters icon from the floating toolbar (see Figure 2.10). Click OK on Select Parameters window.
 - iv. Rearrange port icons inside weight1 as shown in Figure 3.5.
 - e. Repeat step d for a Constraint Property of the type of PowerBalance.
 - f. Add connectors (central toolbar or floating toolbar) from the Value or Part Property attributes to the Constraint Property parameters as shown in Figure 3.5.

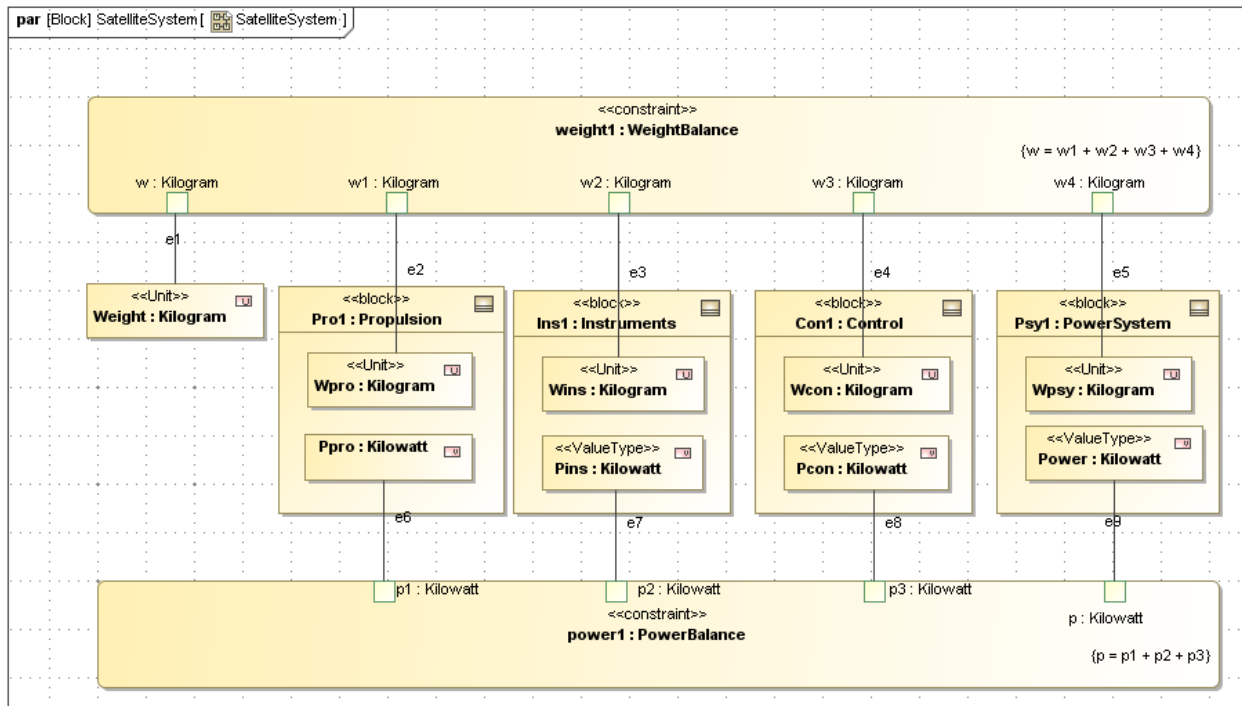


Figure 3.5 SysML Parametrics Diagram

8. Create a second SysML Parametric diagram to define and display the calculation of the requirements verification relationships inside SatelliteSpecification
 - a. RC SatelliteSystem
 - b. Create New Diagram→ SysML Parametric Diagram, Name = SatelliteVerify
 - c. Use the Select Parts Window to allow only the Part Property, Psy1, and the Value Properties: Weight, Weight_MOS, and Power_MOS, to appear in the parametric diagram.
 - d. Use Edit Compartment to show the Value property, Power, inside Psy1.
 - e. Drag the WeightMOS constraint block inside the diagram and give it the name WMOS as a constraint property of SatelliteSystem. Display the constraint parameters and arrange as shown in Figure 3.6.
 - f. Repeat for the PowerMOS constraint block.
 - g. Add connectors from the Value or Part Property attributes to the Constraint Property parameters as shown in Figure 3.6.

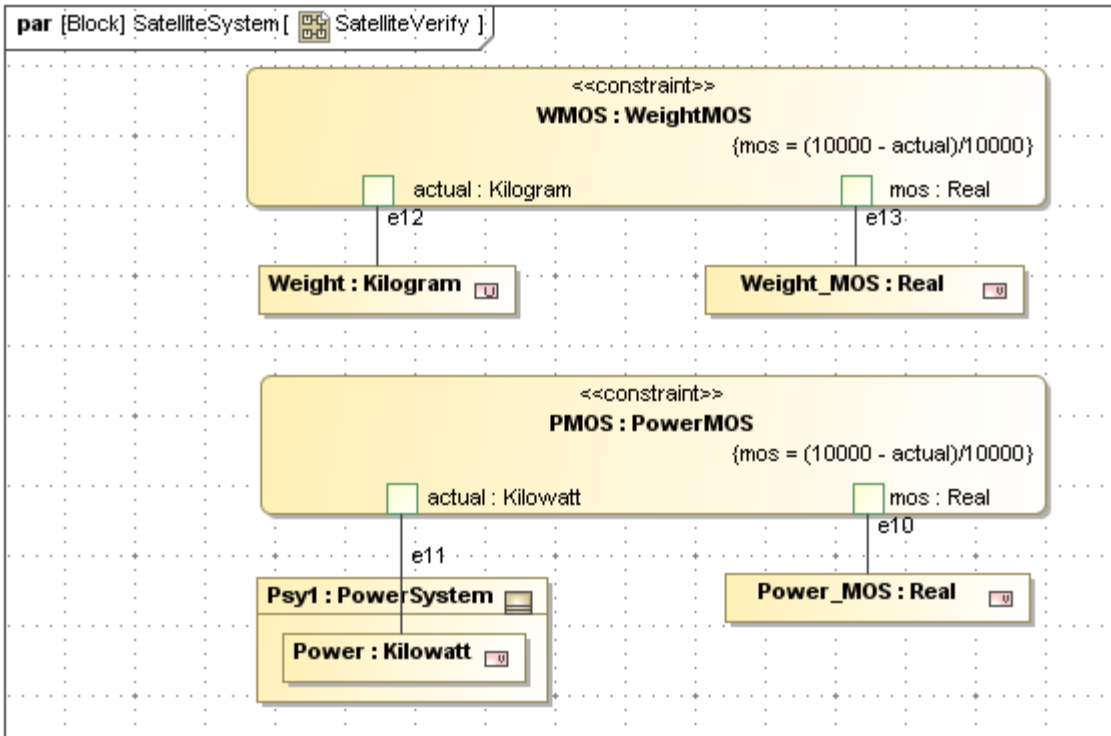


Figure 3.6 SysML Parametrics Diagram

Step VI Validate Parametrics Model

9. To create the CXS_heading and validate the model schema,
 - a. RC on SatelliteSystem in the Containment tree and select ParaMagic→Util→Create CXS_heading
 - b. RC on Satellite in the Containment tree and select ParaMagic→Validate.

Step VII Create an Instance

10. Create an instance by creating a new block definition diagram containing the model, the elements whose attributes are givens or unknowns in the calculation. In the first tutorial, we created the diagrams and instances individually. In this example, we use the Instantiation Wizard, which can create complex instances more efficiently, particularly where there are many elements.
 - a. RC on SatelliteSystem (the root block) and select Create Instance...
 - i. The Automatic Instantiation Wizard (new since MagicDraw 16.5) is launched as in Figure 3.7. Click Next.

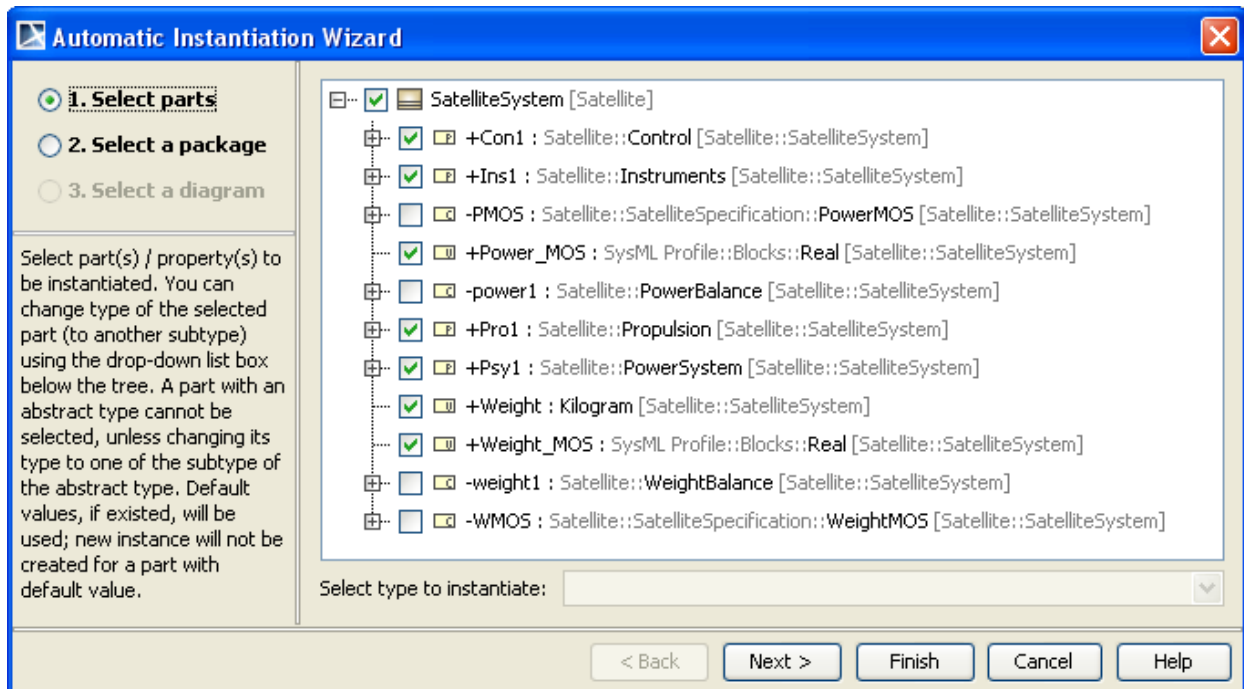


Figure 3.7 Automatic Instance Generation, Part 1

- ii. Click Next.
- iii. Click Create and create the Instance_01 package in the Satellite package (see Figure 3.7b)
- iv. Click Next.

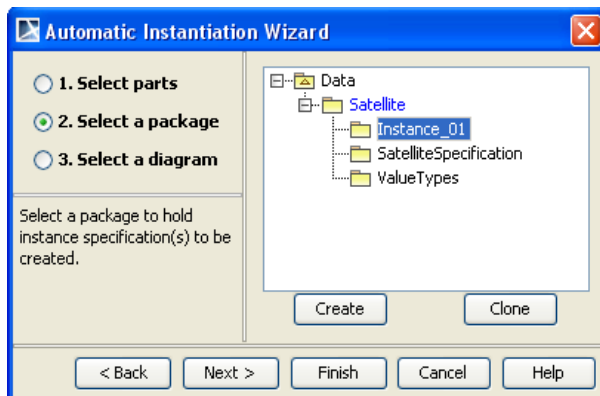


Figure 3.7b Automatic Instance Generation, Part 2

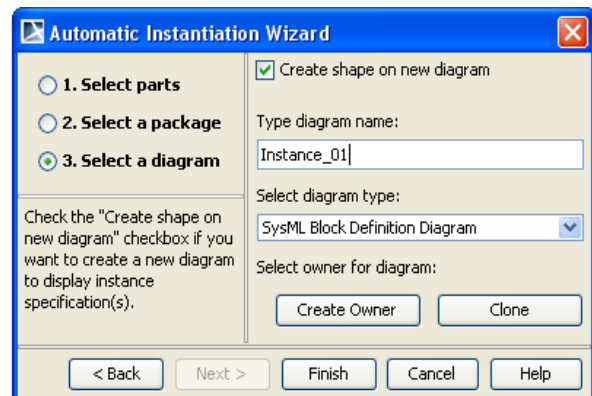


Figure 3.7c Automatic Instance Generation, Part 3

- v. Click the “Create shape on new diagram” checkbox (see Figure 3.7c).
 - vi. Diagram name = Instance_01, Type = SysML Block Definition Diagram.
 - vii. Click Finish.
- b. The diagram Instance_01 looks similar to Figure 3.8

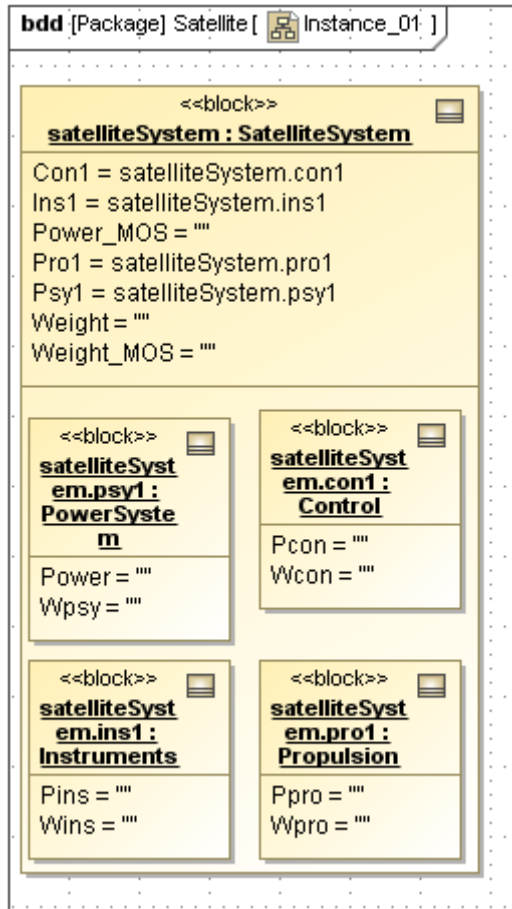


Figure 3.8 Satellite Instance_01 Diagram, after wizard

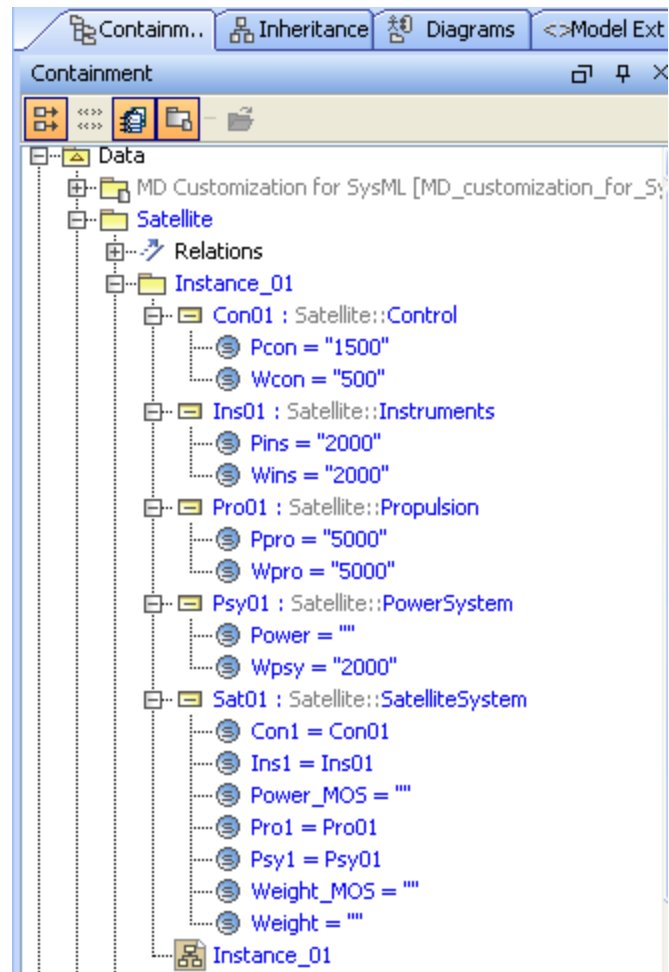


Figure 3.9a Satellite Instance_01 Tree, after revision

- c. Modify the instance diagram to prepare for parametric solving by assigning inputs. Some of the suggested changes make the diagram better organized and easier to read.
 - i. The four part properties instances are dragged outside the satelliteSystem instance block.
 - ii. Double-click on satelliteSystem.pro1:Propulsion
 1. Rename the block as Pro01 in the Instance Specification window
 2. Click on Slots
 3. Assign a value of 5000 to the Ppro slot.
 4. Assign a value of 5000 to the Wpro slot.
 - iii. Double-click on satelliteSystem.con1:Control
 1. Rename the block as Con01 in the Instance Specification window
 2. Click on Slots
 3. Assign a value of 1500 to the Pcon slot.
 4. Assign a value of 500 to the Wcon slot.

- iv. Double-click on satelliteSystem.ins1:Instruments
 1. Rename the block as Ins01 in the Instance Specification window
 2. Click on Slots
 3. Assign a value of 2000 to the Pins slot.
 4. Assign a value of 2000 to the Wins slot.
 - v. Double-click on satelliteSystem.psy1:PowerSystem
 1. Rename the block as Pow01 in the Instance Specification window
 2. Click on Slots
 3. Assign a value of 2000 to the Wpsy slot.
 - vi. Double-click on satelliteSystem.SatelliteSystem
 1. Rename the block as Sat01 in the Instance Specification window
 - vii. Drag the Instance_01 diagram into the Instance_01 package in the containment tree.
- d. The instance package should appear as in Figure 3.9a and the instance diagram should appear similar to Figure 3.9b.

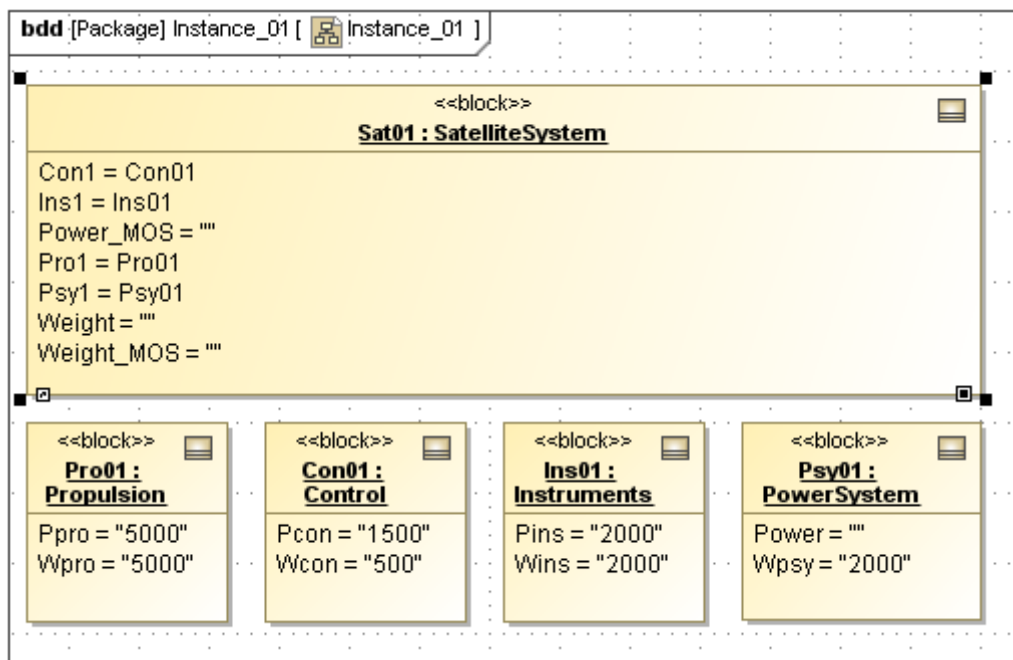


Figure 3.9b Satellite Instance_01 Diagram

11. Create a CXI_heading, which contains a string that uniquely identifies the file generated in the following step
 - a. RC Instance_01
 - i. Select ParaMagic→Util→Create CXI_heading
 - ii. Assign an identifying string such as "1.0", to the version,
 - iii. Click OK.

12. Assign causality to all variables.
 - a. RC on the package Instance_01
 - i. Select ParaMagic→Util→Assign default causalities. See discussion of Causality below.
 - b. Change causality for Power_MOS and Weight_MOS to *target*
 - i. Double-click on Sat01,
 - ii. Expand Slots (click on + box next to Slots),
 - iii. Double-click on Weight_MOS,
 - iv. select Tags,
 - v. DC on <<causality>> type,
 - vi. Select *target* from dropdown list and Close.
 - vii. Repeat for Power_MOS.

Discussion – Causality

The Assign default causalities utility automatically assigns *given* causality to slots (parameters) with assigned real values and *undefined* causality to empty slots, which will be calculated by ParaMagic. However, at least one slot must be assigned *target* causality (as in 13b above) for ParaMagic to seek a solution. Typically, these will be the final results, not intermediate results. One parameter in each independent system of equations should be a target. For example, in this tutorial, if Weight_MOS is assigned as a target and Power_MOS as undefined, no value for Power_MOS will be calculated since that equation is independent from the weight calculation.

Step VIII Solve the Instance

13. Run the parametric solver
 - a. RC the Instance_01 package.
 - b. Select ParaMagic→Browse.

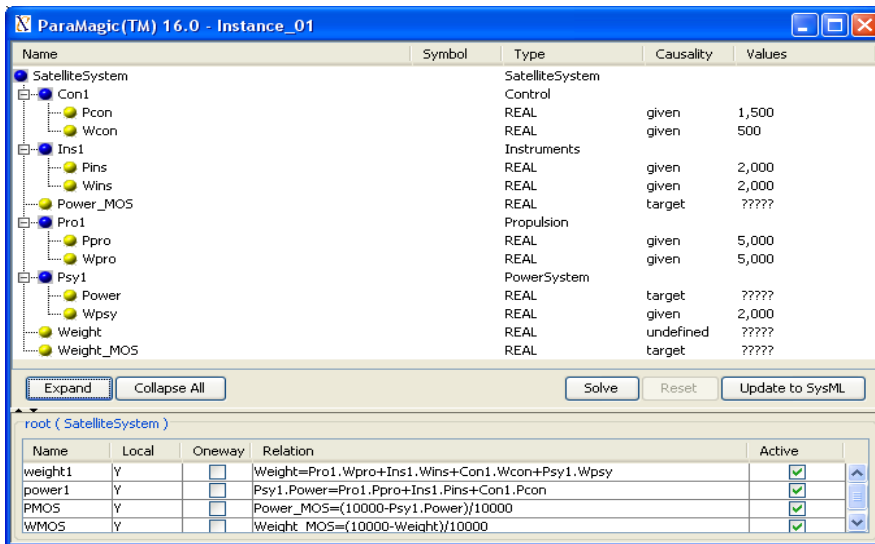


Figure 3.10 ParaMagic Browser for Instance_01, before solution

- c. Press “Solve”. The ????? symbols in the *target* variables should change to their calculated values.
 - i. The results in Figure 3.11 show positive values of Weight_MOS and Power_MOS. In other words, both requirements have been successfully met with positives margins of safety.
 - ii. After solution, parameters that began with *undefined* causality now have calculated values and *ancillary* causality. Slots for which no values were calculated remain as *undefined*. This may be caused by an error in building the parametric model, by a failure of Mathematica to solve the parametric equations, or by these slots not being in the path required to calculate the *target* values.
- d. Press “Update to SysML”. Then you should see the new calculated values in the Instance diagram.

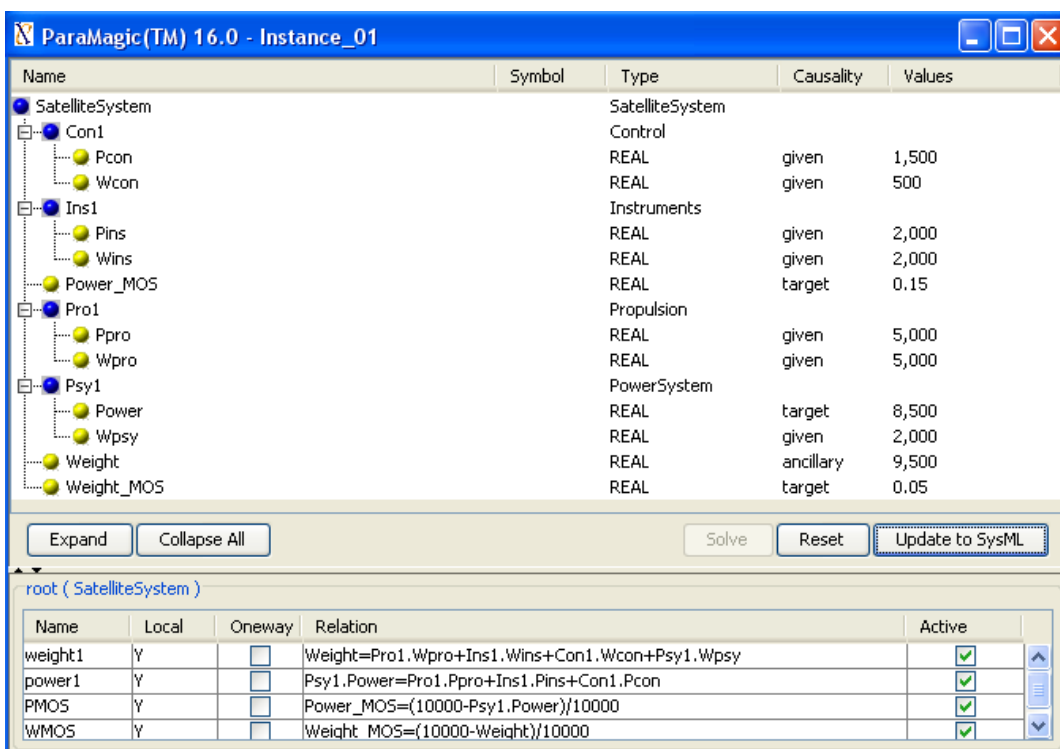


Figure 3.11 ParaMagic Browser for Instance_01, after solution

14. It is possible to manipulate the values inside the browser window to repeat the calculation with a new set of given values, or even change the choice of given and target values. Selecting the Causality type for a parameter in the Browser opens a pull-down list with three choices: given, undefined, and target.
 - a. To reset a target value in the browser window, change the causality for that value from target to undefined, then back to target. The value will reset to ?????.
 - b. To change a given value, click on that value in the browser window and edit. This will reset the solution

- c. To change a target value to a given, or a given value to a target, change the causality for that parameter. In Figure 3.12, the problem has been inverted. “Given the weight budget (10,000 kg) and power budget (10,000 kW) and the properties of the other subsystems, how much weight and power can be allocated to the Instruments subsystem?”. In this example, we set the margins of safety equal to zero and calculate Wins and Pins as targets. The solution is shown in Figure 3.13.

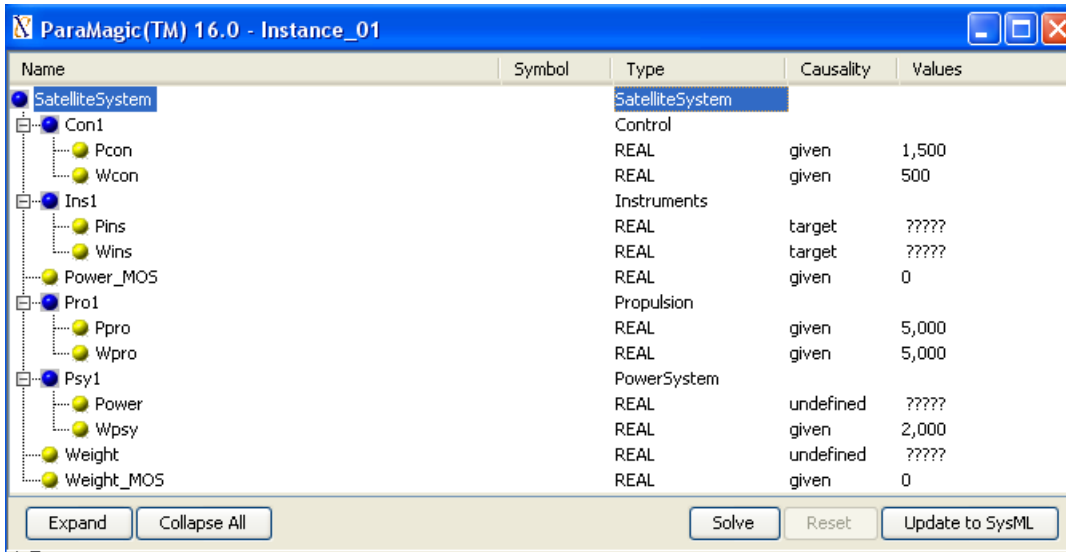


Figure 3.12 ParaMagic Browser for Instance_01, before solution, with changes in causality

Discussion – Causality

Mathematica is a non-causal solver, that is, it can solve many equations in any direction. The user can take advantage of this to use the same model to answer different kinds of questions. However, not all solvers are non-causal (e.g. Microsoft Excel formulas work in one direction only) and not all functions work in multiple directions (e.g. $A = \text{MINIMUM}(B,C,D)$ cannot be solved for D given A, B and C). Keeping track of causality can require some effort on the part of the modeler.

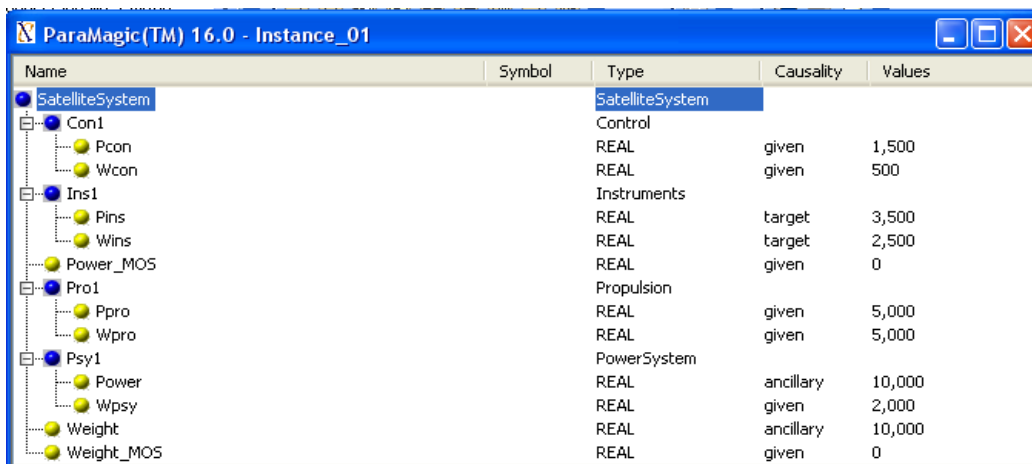
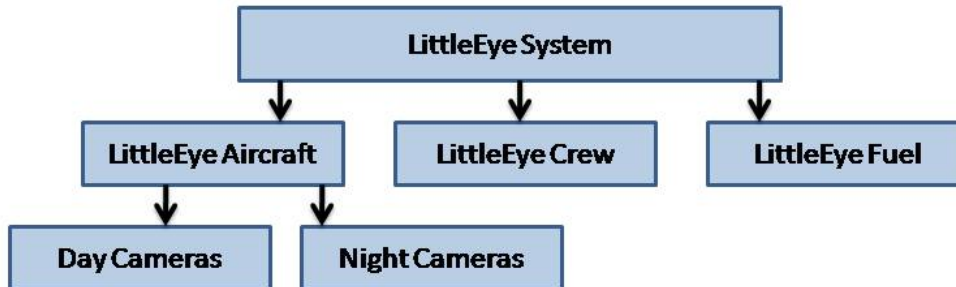


Figure 3.11 ParaMagic Browser for Instance_01, after solution

4 SYSML PARAMETRICS TUTORIAL - LITTLE EYE

4.1 Objective

The third tutorial concerns using SysML to determine operational performance using a sequence of equations. The system is the LittleEye unmanned aerial vehicle (UAV) which is used to provide reconnaissance. The objective is to calculate how many miles of road can be scanned per 24 hours, which will be determined by the number and duty cycle of aircraft, the number and duty cycle of crews, and the availability of fuel.



$\text{NumberMilesScannedPer24Hours} = \text{NumberAvailablePlanes} \times \text{MilesScannedPerHour} \times 24$

$\text{NumberAvailableSystems} = \text{MINIMUM}(\text{NumberAvailablePlanes}, \text{NumberAvailableCrews}, \text{NumberAvailableFuelLoads})$

$\text{NumberAvailablePlanes} = 0.5 \times (\text{NumberAvailablePlanesByDay} + \text{NumberAvailablePlanesByNight}) \times \text{DutyCyclePlane}$

$\text{DutyCyclePlane} = (1 - \text{DutyCycleTurnaround}) \times (1 - \text{DutyCycleCameraRefit}) \times (1 - \text{DutyCycleMaintenance})$

$\text{NumberAvailablePlanesByDay} = \text{MINIMUM}(\text{NumberPlanes}, \text{NumberDayCameras})$

$\text{NumberAvailablePlanesByNight} = \text{MINIMUM}(\text{NumberPlanes}, \text{NumberNightCameras})$

$\text{NumberAvailableCrews} = \text{NumberCrews} \times \text{CrewTimeOn}$

$\text{NumberAvailableFuelLoads} = \text{FuelSupplyPerDay} / \text{DailyFuelLoadPerPlane}$

Information Provided

$\text{MilesScannedPerHour} = 40 \text{ mph}$

$\text{CrewTimeOn} = .42$ (120 hours on over 12 day period)

$\text{DutyCycleTurnaround} = .23$ (30 hours turnaround per 100 flight minutes)

$\text{DutyCycleCameraRefit} = .02$ (15 minutes refit per 12 hour period)

$\text{DutyCycleMaintenance} = .09$ (3 hours maintenance per 30 flight hours)

$\text{DailyFuelLoadPerPlane} = 50$ gallons

Figure 4.1 Outline of Objective

The complexity of the model, with six elements, eight constraints and more than twenty parameters, makes it a good place to introduce “object-oriented” modeling techniques, which allow a complex model to be built from simple, independent and potentially reusable subsystems, tied together only at the highest levels

What the User Will Learn

- Applying an “object-oriented” approach to SysML parametrics
- Embedding constraints and parametric diagrams within multiple blocks in the model
- Using standard functions, e.g. Minimum
- Using simulation to explore a model with “what-if scenarios”

4.2 Step-by-Step Tutorial

Step I Create Project

1. Create new SysML project, Name = LittleEye
2. Create new Package, Name = LittleEye

Step II Create Infrastructure

3. Install ParaMagic Profile module, same as in the Addition tutorial.

Step III Create Structural Model

4. Create a Block Definition Diagram containing the four structural elements with attributes as shown in Figure 4.2

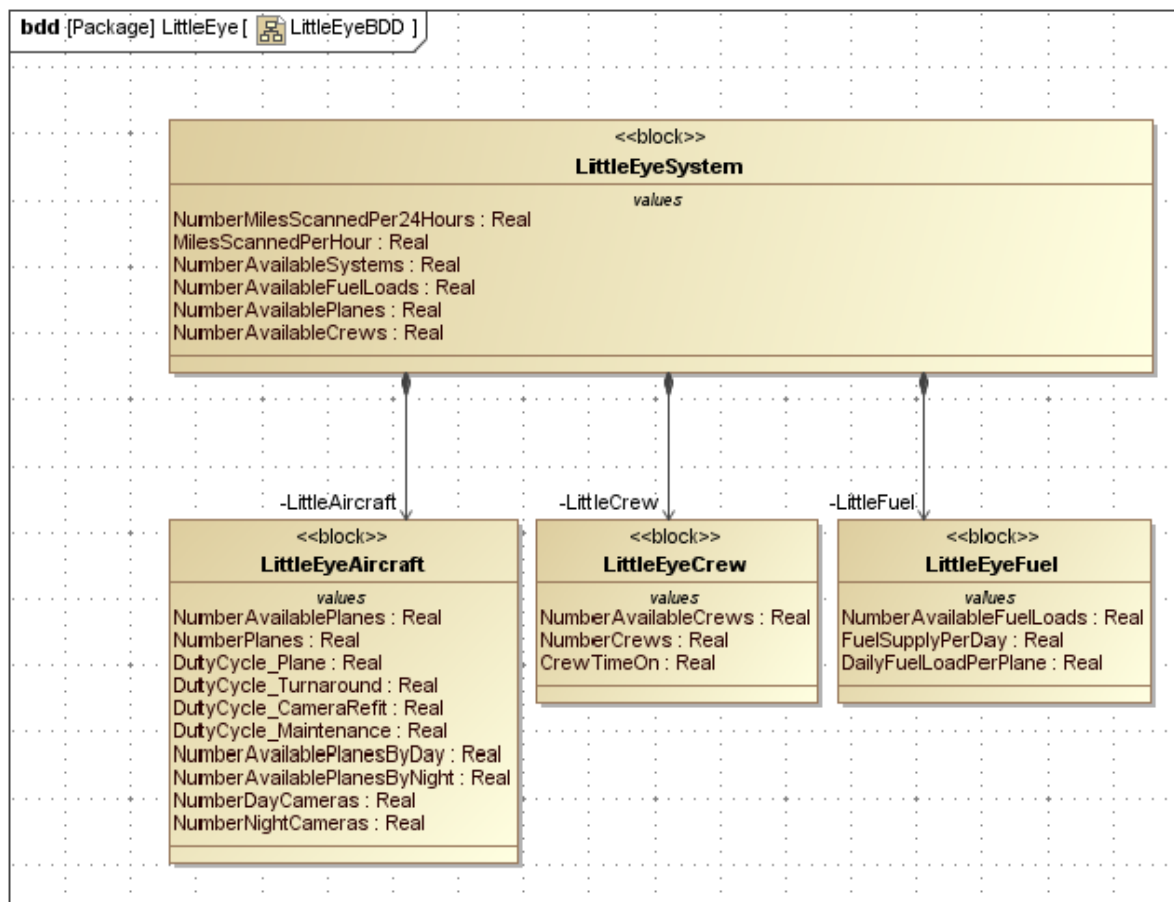


Figure 4.2 Structural Elements in Block Definition Diagram LittleEyeBDD

- a. Create the Directed Composition arrows as demonstrated, to show that Aircraft, Crew and Fuel are parts of System.
- b. Name the Part Properties created inside LittleEyeSystem by the Directed Composition arrows (LittleAircraft , LittleCrew, and LittleFuel in this example).

Step IV Create Constraints

5. Inside LittleEye, create eight constraint blocks, which contain the eight mathematical relationships appearing in Figure 4.1, using the procedure described in the earlier tutorials. Figure 4.3 shows the eight constraint blocks displayed on the right side of the LittleEye Block Definition Diagram. The parameter list has been hidden for the constraint blocks by selecting Presentation Options→Suppress Parameters, Operations, Attributes on each block.

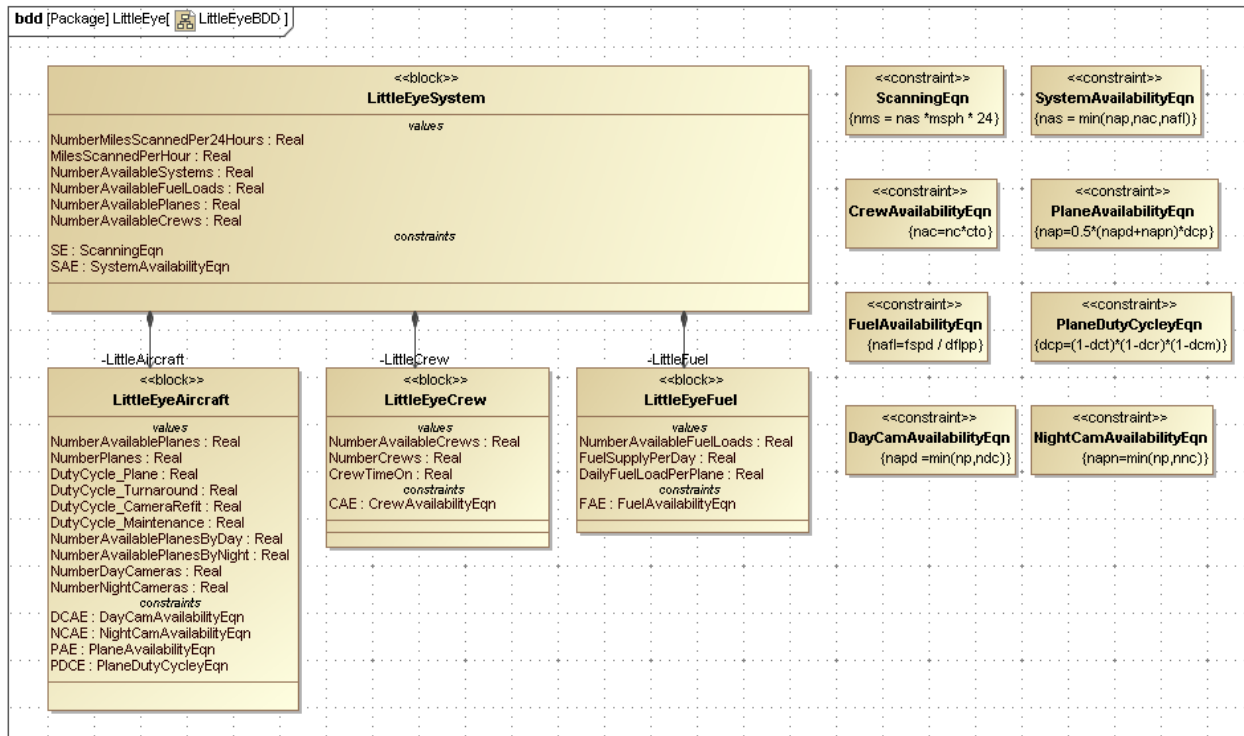


Figure 4.3 Block Definition Diagram with Constraints and Model added

Step V Create Parametrics Model

6. Using an “object-oriented” programming approach, the four elements of Figure 4.2 will be converted into independent submodels containing internal constraints and relationships. Each sub-model could be modified without damaging the larger system model. It could also be copied and reused in a different project.
 - a. The submodel LittleEyeSystem will contain the six values listed in Figure 4.3 plus two constraints that connect them, the Scanning Equation and the System Availability Equation at the top right of Figure 4.2. See Figure 4.4.
 - i. In the Containment window, drag the Scanning Equation and the System Availability Equation inside the LittleEyeSystem block.
 - ii. In LittleEyeSystem, create a New Diagram→SysML Parametrics Diagram, name = LittleEyeSystem
 - iii. Use the Select Parts window to choose the six Value Properties inside LittleEyeSystem to appear on parametrics diagram.

- iv. Create a new Constraint Property by dragging the Constraint Block ScanningEqn into the parametrics diagram, Name = SE. Use the Display Parameters icon to show nms, msph, and nas inside SE.
- v. Repeat step iv. for a second constraint property, name = SAE, Type = SystemAvailabilityEqn.
- vi. Using connectors, wire the value properties and constraint parameters as shown in Figure 4.4.

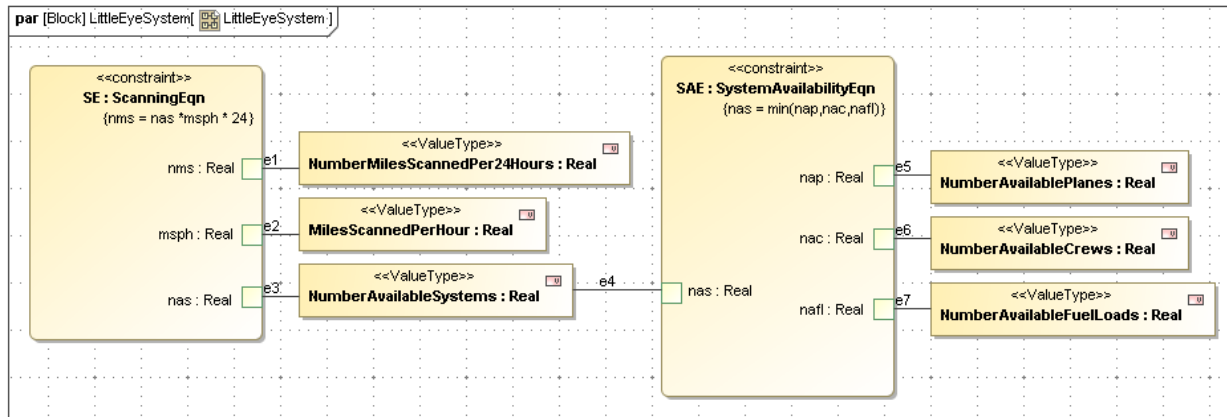


Figure 4.4 LittleEyeModel Parametrics Diagram

- b. Create submodels for the Aircraft, Crew, and Fuel blocks using the same procedures, ending with the parametrics diagrams shown in Figures 4.5, 4.6 and 4.7

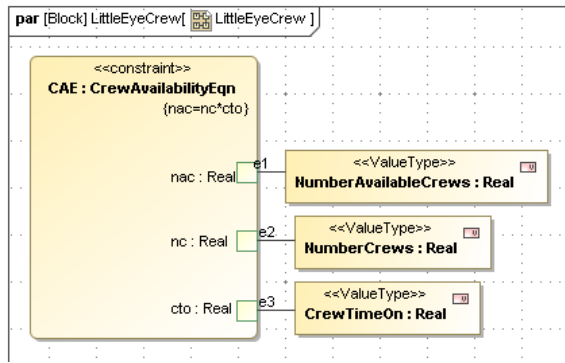


Figure 4.5 LittleEyeCrew Parametrics Diagram

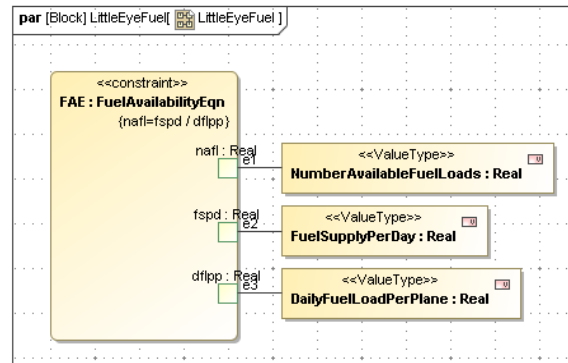


Figure 4.6 LittleEyeFuel Parametrics Diagram

Discussion: Adding Constraint Properties to Parametric Diagrams

In the first three tutorials, we have used slightly different methods to incorporate constraints into parametric diagrams. In the first, we dragged the Constraint Block from the Containment tree into the parametric diagram, creating a Constraint Property of the same type. In the second, we dragged a constraint property icon from the toolbar into the parametric diagram and used the Select Classifier window to identify it with an existing Constraint Block. In this example, we placed the Constraint Blocks inside the submodels, before dragging them into the parametric diagrams to create individual Constraint Properties of the same type. In each case, we have had to assign the

Constraint Property created a unique name.

All three methods work. The third implies an exclusive ownership of the constraint by a particular submodel, which may be misleading if the constraint block is to be re-used by other parts of model. Assigning ownership may be useful in a collaborative environment, but it is also OK to keep Constraint Blocks anywhere in the project package and drag them into specific parametric diagrams as needed.

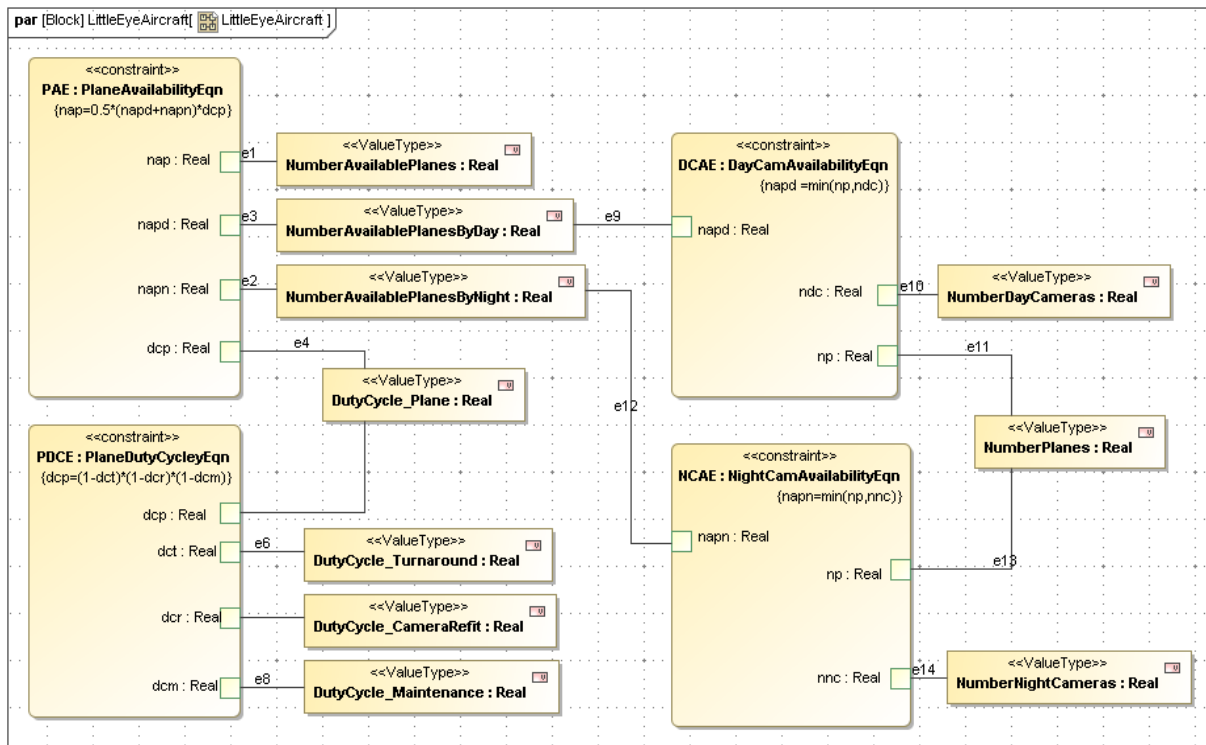


Figure 4.7 LittleEyeAircraft Parametrics Diagram

7. It is also necessary to create an overall parametric diagram to connect the different submodels. This is an objective of “object-oriented” programming, to create independent modules and link them in the simplest possible way. Within the top-level block, LittleEye System, several properties have been created with the same names as properties in the lower-level models, e.g. NumberAvailablePlanes. In the following steps, we use connectors to link models together through these duplicate parameters. (Alternate: the duplicate parameters could also be linked using explicit equality constraint relationships, created as an Equality constraint block in Step 8 and used three times within this parametric diagram. However, this involves a lot of extra work. ParaMagic interprets direct connections between value properties as equality relationships).
 - a. Create a second SysML Parametrics diagram inside LittleEyeSystem called LittleEyeSystem_2.
 - b. Use the Select Parts window to select the Part Properties LittleAircraft, LittleCrew and LittleFuel, to appear in the parametrics diagram.

- c. Drag the three value properties, NumberAvailablePlanes, NumberAvailableCrews and NumberAvailableFuelLoads into the parametric diagram.
- d. Using Edit Compartment and Presentation Options, display the internal structure of the three submodels as shown in Figure 4.8. Note that only the parameters necessary for wiring between the high-level model and the submodels are shown; all other parameters can remain hidden.
- e. Draw connectors between model properties as shown in Figure 4.8.

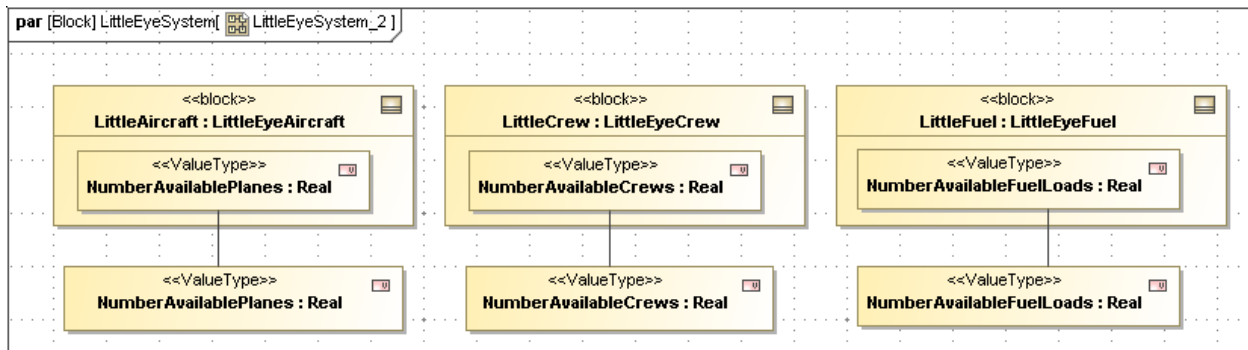


Figure 4.8 LittleEyeSystem_2 Parametric Diagram

Step VI Validate Parametrics Model

8. To create the CXS_heading and validate the model schema,
 - a. RC on LittleEyeSystem in the Containment tree and select ParaMagic→Util→Create CXS_heading
 - b. RC on LittleEye and select ParaMagic→Validate.

Step VII Create an Instance

9. Create an instance of the model. The steps are similar to previous tutorials, however, some short-cuts are used. For example, the values of the givens are assigned in step d below and most causality tags are set automatically using the “Assign default causalities” utility in step f. Figure 4.9 is an example of an instance. The values for several of the givens are taken from Figure 4.1. Others, such as the Number of Planes, are user-set trial values, which will be varied during the solving operation.
 - a. Create a package called LittleEyeInstance01 inside the LittleEye package.
 - b. Create a block definition diagram for the instance.
 - c. Create instances of System01, Aircraft01, Crew01 and Fuel01.
 - d. To display parameters inside each instance, DC on the instance, select Slots, DC on each parameter to be displayed, assign a value if the parameter is a given, and close the window when all parameters in that instance are complete.
 - i. When inputting decimal values less than one, it is necessary to use the notation 0.42 instead of .42 or the ParaMagic browser window will not open.
 - e. Link the Aircraft01, Fuel01 and Crew01 blocks to System01 block using the Link connector on the floating toolbar (click on the block to see toolbar).

- f. Create a CXI_heading element inside LittleEyeInstance01
 - i. Right-click LittleEyeInstance01 in the Containment tree
 - ii. Select ParaMagic→Util→Create CXI_heading
 - iii. Approve over-write, if necessary
 - iv. Provide a version number and press OK.
- g. Assign causalities.
 - i. Right-click LittleEyeInstance01.
 - ii. Select ParaMagic →Util→Assign default causalities. All parameters that have been assigned a numerical value are now assigned as *givens*. All parameters that were assigned an empty value are now assigned as *undefined* (to be calculated by the solver).
 - iii. Assign NumberMilesScannedPer24Hours *target* causality in this example (see Step 12b in Tutorial Satellite).

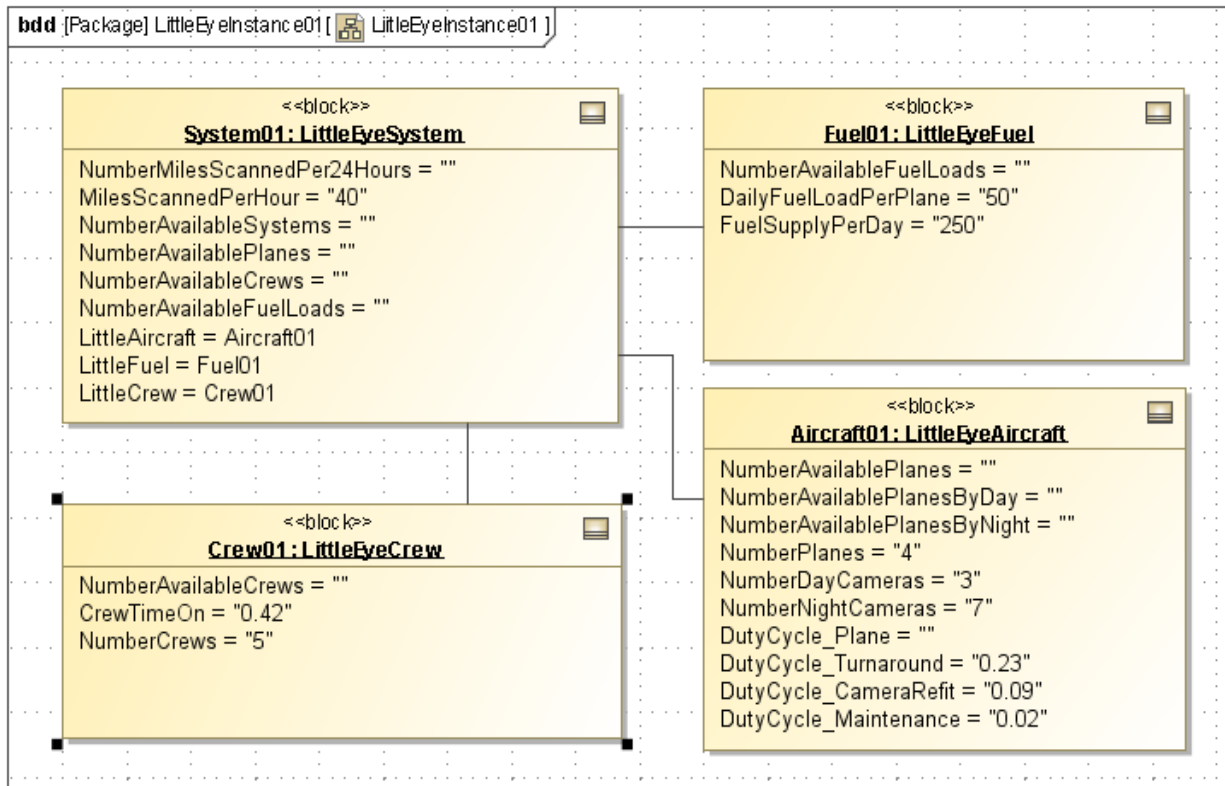


Figure 4.9 Creating an Instance of the LittleEye model

Note: The Assign default causalities utility can be applied to the Instance package only after the CXI_heading has been created, but may be applied to the individual instance blocks after step d.

Step VIII Solve the Instance

The trial parameters appear in the browser window as shown in Figure 4.10. After solution, the results are shown in Figure 4.11. The target result, NumberMilesScannedPer24Hours, is 2, 016 miles. Note that the number of available systems is 2.1, on average, and is limited by the number of available

crews. The number of available aircraft, 2.4, and fuel loads, 5, are not the limiting factors in keeping UAVs in the air.

Name	Symbol	Type	Causality	Values
root		LittleEyeSystem		
LittleAircraft		LittleEyeAircraft		
DutyCycle_CameraRefit		REAL	given	0.09
DutyCycle_Maintenance		REAL	given	0.02
DutyCycle_Plane		REAL	undefined	?????
DutyCycle_Turnaround		REAL	given	0.23
NumberAvailablePlanes		REAL	undefined	?????
NumberAvailablePlanesByDay		REAL	undefined	?????
NumberAvailablePlanesByNight		REAL	undefined	?????
NumberDayCameras		REAL	given	3
NumberNightCameras		REAL	given	7
NumberPlanes		REAL	given	4
LittleCrew		LittleEyeCrew		
CrewTimeOn		REAL	given	0.42
NumberAvailableCrews		REAL	undefined	?????
NumberCrews		REAL	given	5
LittleFuel		LittleEyeFuel		
DailyFuelLoadPerPlane		REAL	given	50
FuelSupplyPerDay		REAL	given	250
NumberAvailableFuelLoads		REAL	undefined	?????
MilesScannedPerHour		REAL	given	40
NumberAvailableCrews		REAL	target	?????
NumberAvailableFuelLoads		REAL	target	?????
NumberAvailablePlanes		REAL	target	?????
NumberAvailableSystems		REAL	undefined	?????
NumberMilesScannedPer24Hours		REAL	target	?????

Figure 4.10 Browser for First Instance of the LittleEye model, before Solution

In Figure 4.11, several parameters are labeled “ancillary” after solution. This implies that they were calculated during the solution process, and were used in further calculations. For example, the number of available crews was calculated from the number of crews and the crew duty cycle, and is used in calculating the number of available systems.

Name	Symbol	Type	Causality	Values
root		LittleEyeSystem		
LittleAircraft		LittleEyeAircraft		
DutyCycle_CameraRefit		REAL	given	0.09
DutyCycle_Maintenance		REAL	given	0.02
DutyCycle_Plane		REAL	ancillary	0.686686
DutyCycle_Turnaround		REAL	given	0.23
NumberAvailablePlanes		REAL	ancillary	2.403401
NumberAvailablePlanesByDay		REAL	ancillary	3
NumberAvailablePlanesByNight		REAL	ancillary	4
NumberDayCameras		REAL	given	3
NumberNightCameras		REAL	given	7
NumberPlanes		REAL	given	4
LittleCrew		LittleEyeCrew		
CrewTimeOn		REAL	given	0.42
NumberAvailableCrews		REAL	ancillary	2.1
NumberCrews		REAL	given	5
LittleFuel		LittleEyeFuel		
DailyFuelLoadPerPlane		REAL	given	50
FuelSupplyPerDay		REAL	given	250
NumberAvailableFuelLoads		REAL	ancillary	5
MilesScannedPerHour		REAL	given	40
NumberAvailableCrews		REAL	target	2.1
NumberAvailableFuelLoads		REAL	target	5
NumberAvailablePlanes		REAL	target	2.403401
NumberAvailableSystems		REAL	ancillary	2.1
NumberMilesScannedPer24Hours		REAL	target	2,016

Figure 4.11 Browser for First Instance of the LittleEye model, after solution

In Figure 4.12, the number of crews assigned to the base is increased by one, from 5 to 6. After solution, the average number of available systems has increased to 2.4, now limited by the number of available planes, and the number of miles scanned per 24 hours has increased to 2307. Increasing the number of crews by 20% has increased the number of miles scanned by only 14% because a different resource became the limiting factor. Further experimentation would show how the numbers of planes, day cameras, and night cameras affects total miles scanned.

In some cases, the user will want to create and save multiple instances to record the effect of changing input variables. One way to do this is to make multiple copies of the original instance within the model and modify each one as desired, with a different set of input parameters, for example.

Name	Symbol	Type	Causality	Values
root		LittleEyeSystem		
LittleAircraft		LittleEyeAircraft		
DutyCycle_CameraRefit		REAL	given	0.09
DutyCycle_Maintenance		REAL	given	0.02
DutyCycle_Plane		REAL	ancillary	0.686686
DutyCycle_Turnaround		REAL	given	0.23
NumberAvailablePlanes		REAL	ancillary	2.403401
NumberAvailablePlanesByDay		REAL	ancillary	3
NumberAvailablePlanesByNight		REAL	ancillary	4
NumberDayCameras		REAL	given	3
NumberNightCameras		REAL	given	7
NumberPlanes		REAL	given	4
LittleCrew		LittleEyeCrew		
CrewTimeOn		REAL	given	0.42
NumberAvailableCrews		REAL	ancillary	2.52
NumberCrews		REAL	given	6
LittleFuel		LittleEyeFuel		
DailyFuelLoadPerPlane		REAL	given	50
FuelSupplyPerDay		REAL	given	250
NumberAvailableFuelLoads		REAL	ancillary	5
MilesScannedPerHour		REAL	given	40
NumberAvailableCrews		REAL	target	2.52
NumberAvailableFuelLoads		REAL	target	5
NumberAvailablePlanes		REAL	target	2.403401
NumberAvailableSystems		REAL	ancillary	2.403401
NumberMilesScannedPer24Hours		REAL	target	2,307.26496

Figure 4.12 Browser for Second Instance of the LittleEye model, NumberCrews increased by one, after solution

5 SYSML PARAMETRICS TUTORIAL - COMMNETWORK

5.1 Objective

The fourth tutorial uses SysML to simulate a simple communication network. The objective is to calculate the output of the network given the input and the loss in the individual channels between stations..

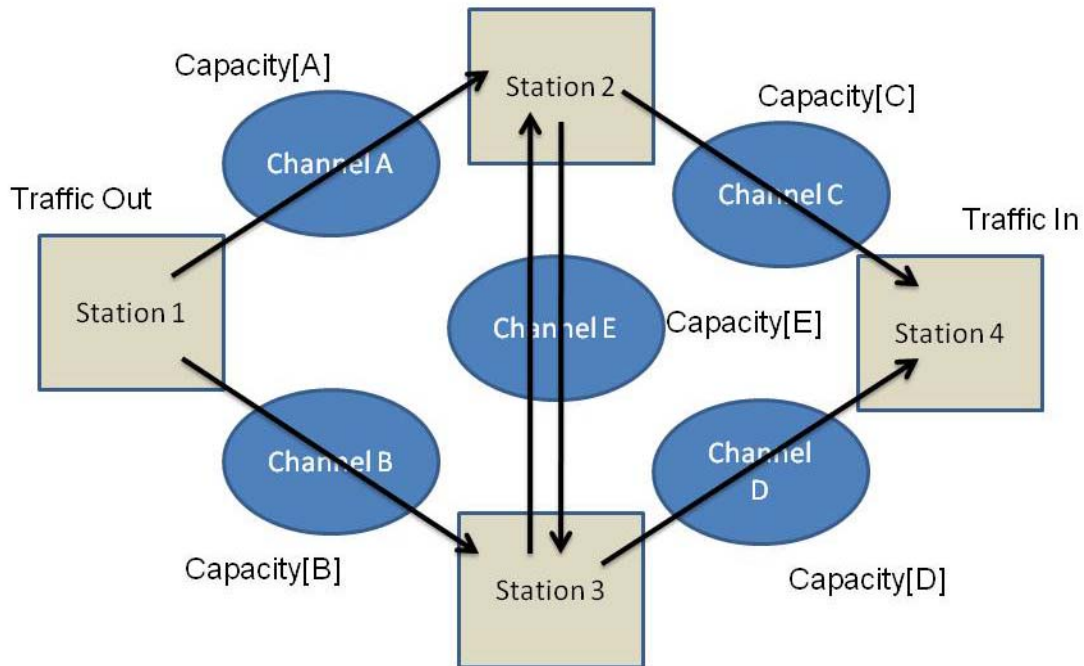


Figure 5.1 Outline of Network

The focus of this tutorial is working with limited number of standard elements to build up more complex structures. In this example, there are only two standard elements, stations (nodes) and channels. Each element contains constraint relations describing its behavior. An Internal Block Diagram is created to assist in completing the parametric diagrams correctly. Parametric constraints are also used at a higher level to define interactions between elements.

What the User Will Learn

- Building complex structures from multiple usages of simple structures
- Using internal block diagrams

5.2 Step-by-Step Tutorial

Step I Create Project

1. Create new SysML project and Package, Name = CommNetwork

Step II Create Infrastructure

2. Install ParaMagic Profile module, same as in first tutorial.

Step III Create Structural Model

3. Create three blocks: Network, Node and Channel.
 - a. Network will be the top level (root) block, but contains no value properties.
 - b. Node represents a station which can receive and transmit messages. It has two inputs, two outputs, and the ability to redistribute the message traffic depending on the capacity of the transmission channels. To build the node model, we will use flowports, value properties and parametric diagrams.
 - i. Each node has four flowports, two for receiving message traffic and two for transmitting (see Figure 5.2). To create a flowport
 1. RC on Node, create New Element→Port
 2. Double-click on the new port, Name = Rec1, AppliedStereotype = Flowport, Direction = in
 3. Repeat for other three flowports. Note that Direction is out for Tr1 and Tr2

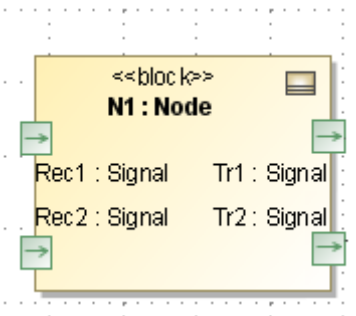


Figure 5.2 Node with Ports

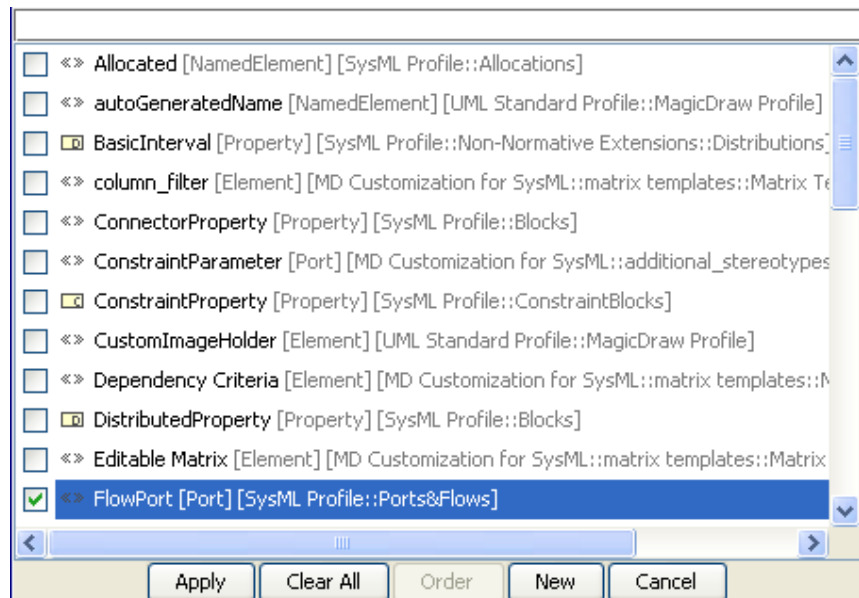


Figure 5.3 Select Applied Stereotype as Flowport

- ii. The Node block has nine Value Properties, all Real. Six represent levels of message traffic at input and output, units unspecified: R1, R2, R, T1, T2, and T. Two represent the capacities of the upstream channels: C1 and C2. The final attribute, D, represents the redistribution factor in splitting outgoing message traffic between the Tr1 and Tr2 ports.
 - c. Channel represents a two-way communication link between two stations. The throughput of each line, the output signal divided by the input, is a function of the channel capacity C and the total signal traffic level. It has two inputs and two outputs, but may only use one I/O pair in a particular instance. To build the channel model, we will use flowports, value properties and parametric diagrams.

- i. Each channel has four flowports, two for input message traffic and two for output (see Channel blocks in Figure 5.6). To create a flowport
 - 1. RC on Channel, create New Element→Port
 - 2. Double-click on the new port, Name = In1, AppliedStereotype = Flowport, Direction = in
 - 3. Repeat for other three flowports. Note that Direction is out for Out1 and Out2.
 - ii. The Channel block has six Value Properties, all Real. Five represent levels of message traffic at input and output, units unspecified: IT1, IT2, IT, OT1, and OT2. C represents the intrinsic capacity of the channel.
4. Create a Block Definition Diagram and an Internal Block Diagram
- a. RC on CommNetwork package, create New Diagram→SysML Diagrams→SysML Block Definition Diagram, Name = Network_BDD
 - i. Drag Network, Node and Channel blocks into the diagram
 - ii. Using the Direct Composition connector from the floating toolbar, connect as shown in Figure 5.4. Note that four connections are made from Network to a single Node block, and the resulting four Part Properties are named N1, N2, N3 and N4. Similarly, five connections are made from Network to a single Channel block, named as ChA thru ChE.

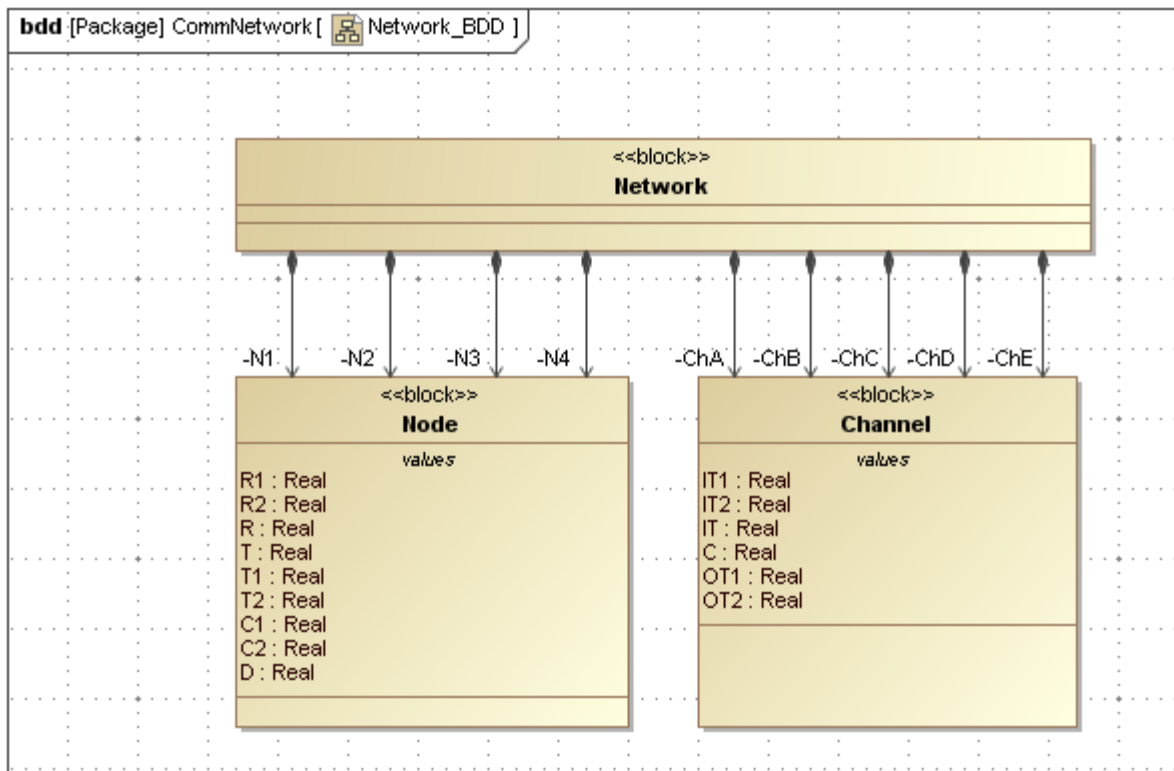


Figure 5.4 CommNetwork Block Definition Diagram

- b. RC on Network block in Containment tree, create New Diagram→SysML Diagrams→SysML Internal Block Diagram, Name = Network.
 - i. Drag all nine part properties in Network (four node and five channels) into the diagram and arrange as shown in Figure 5.6.
 - ii. Click on each block and use the Display Ports icon on the floating toolbar (Figure 5.5a) to show the ports. Arrange the ports as shown in Figure 5.6.
 - iii. Click on a port and use the Connector icon on the floating toolbar (Figure 5.5b) to create a connector, which can be dragged to the matching port on another block. Arrange the connectors as shown in Figure 5.6.

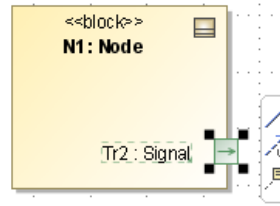
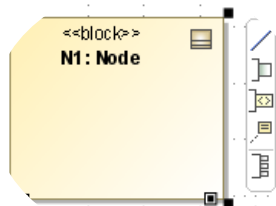


Figure 5.5a Display Ports icon Figure 5.5b Connector icon

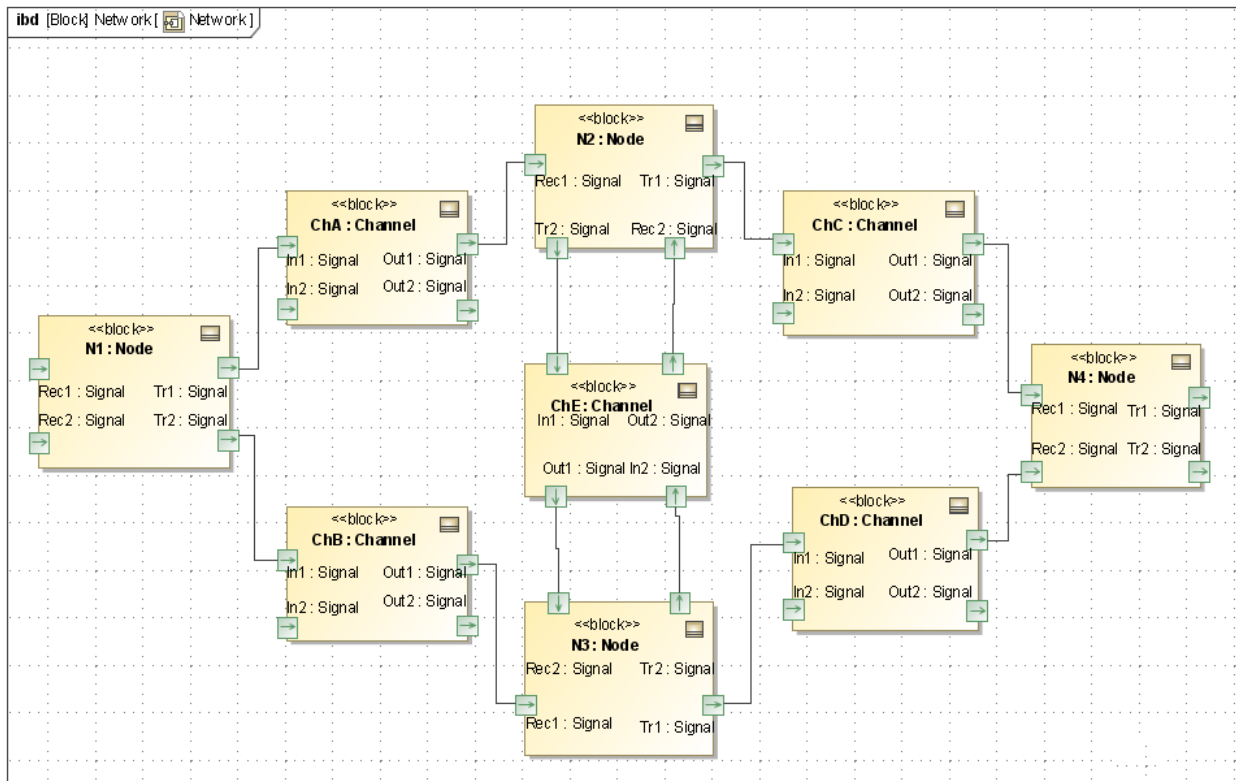


Figure 5.6 CommNetwork Internal Block Diagram

Step IV Create Constraints

5. Create the five constraint blocks required by the model: ErrorRate, Fraction, Product, Product1 and Sum. Usage of these constraint blocks are shown in the parametric diagrams in Figure 5.7 and 5.8, which show the constraints and constraint parameters for each.

Step V Create Parametrics Model(s)

6. Create a parametric model for CommNetwork. There are standard relationships internal to Node and Channel elements and there are equalities tying the Nodes and Channels together into a network.
 - a. Create a parametrics diagram within the Node block, Name = Node. This diagram, shown in Figure 5.7, contains four separate constraint properties, with the constraints shown, the nine internal value properties, and the nested connectors tying them together. In this example, the objective of the equations to redistribute the incoming signal to the two outgoing channels, based on their relative capacities. No claim is made about the realism of this model.

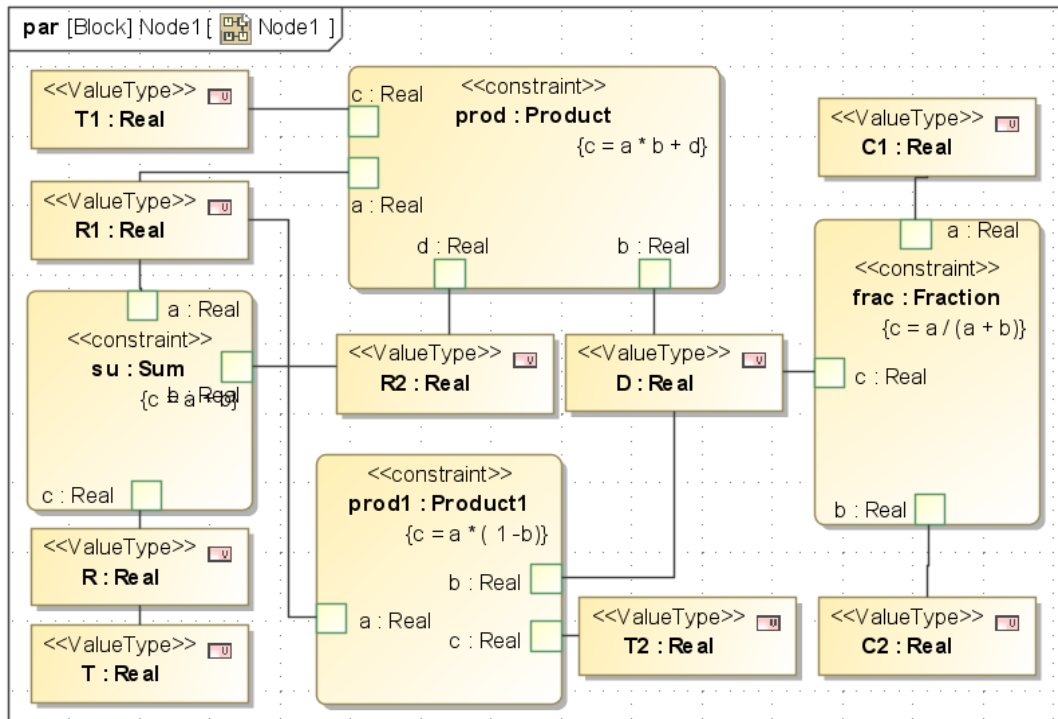


Figure 5.7 Node parametric diagram

- b. Create a parametrics diagram within the Channel block, Name = Channel. This diagram, shown in Figure 5.8, contains three constraint properties (including two usages of the ErrorRate constraint), with the constraints shown, the six internal value properties, and the nested connectors tying them together. In this example, the objective of the equations to reduce the signal throughput at the output of each line relative to the input by an exponential factor based on Capacity C and total incoming traffic. No claim is made about the realism of this model.

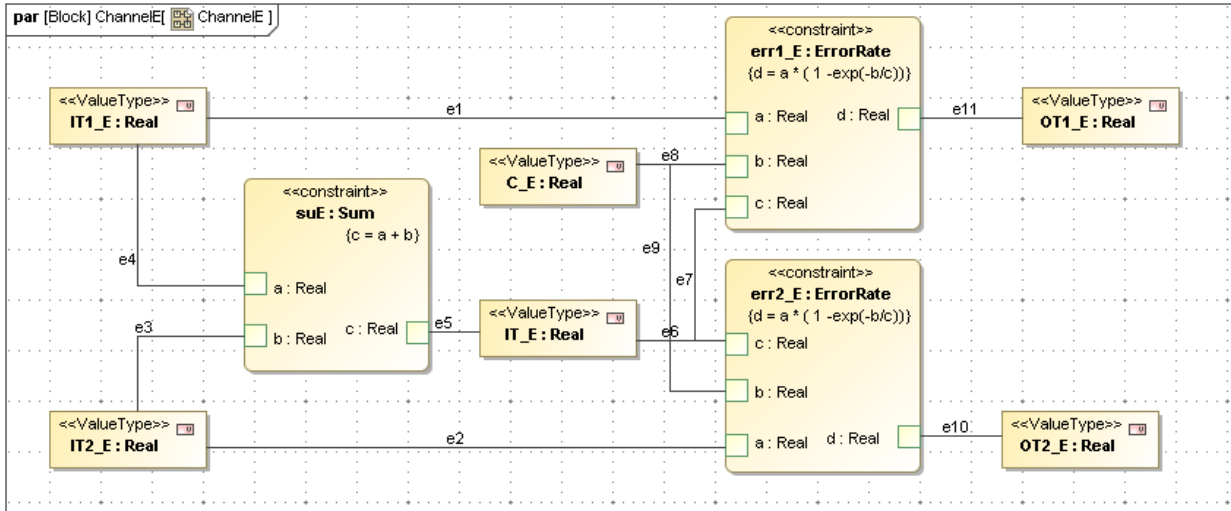


Figure 5.8 Channel parametric diagram

- c. Create parametric diagrams connecting the channels and nodes through equalities as in Figure 5.9a-d. Use the internal block diagram in Figure 5.8 to keep track of which blocks and ports connect. In this example, we have chosen to divide the entire set of high level parametric relationships into four separate diagrams in order to keep each individual diagram simpler with fewer elements. This illustrates the combined use of internal block diagrams for a high-level qualitative perspective and narrower parametric diagrams to make the actual quantitative connections.

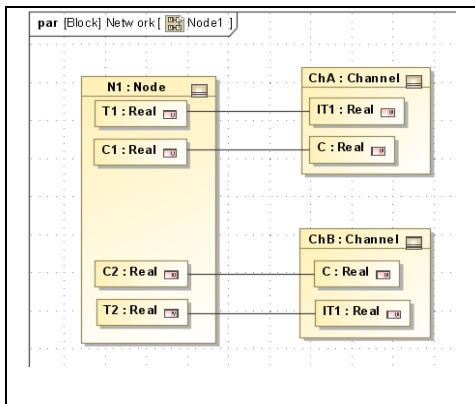


Figure 5.9a Node 1 parametric diagram

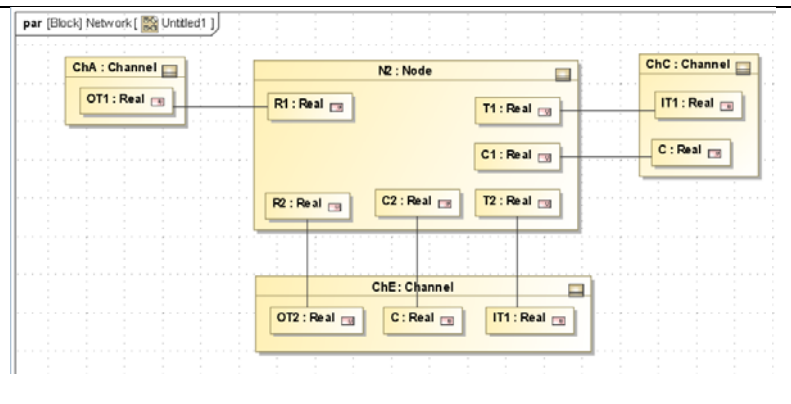


Figure 5.9b Node 2 parametric diagram

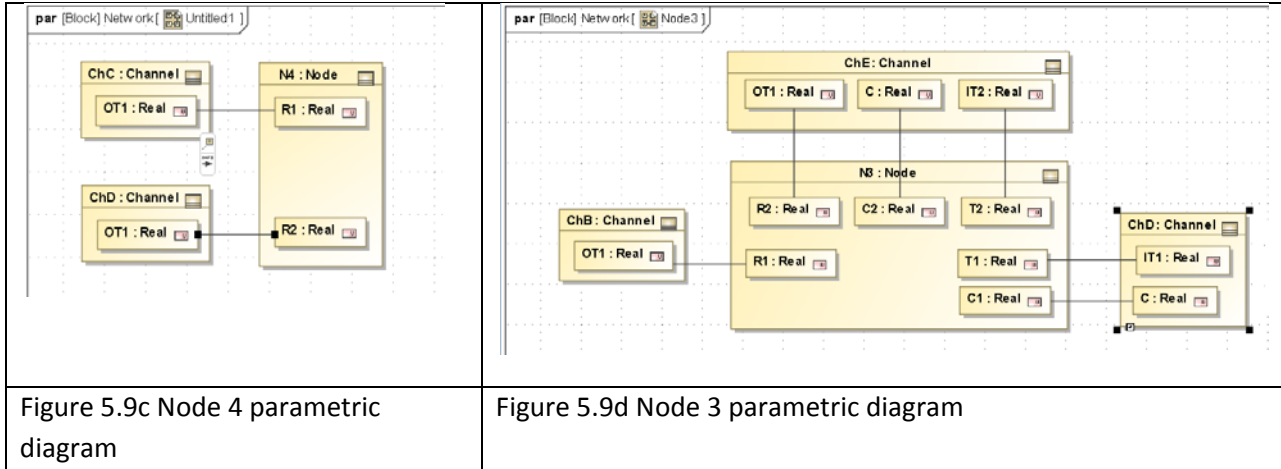


Figure 5.9c Node 4 parametric diagram

Figure 5.9d Node 3 parametric diagram

Step VI Validate Parametrics Model

7. To create the CXS_heading and validate the model schema,
 - a. RC on Network in the Containment tree and select ParaMagic→Util→Create CXS_heading
 - b. RC on the CommNetwork package and select ParaMagic→Validate.

Step VII Create an Instance

8. We will create an instance of CommNetwork using a library of standard instances, rather than creating each instance individually. The ability to create a library in a separate package from the main instance is new to ParaMagic since 16.5.
 - a. Create a library of standard instances of system element.
 - i. RC on CommNetwork and select New Element→Package, Name = InstanceLibrary
 - ii. RC on InstanceLibrary and select New Element→Instance Specification, Name = StartNode, Classifier = Node
 - iii. Double-click on StartNode to open Instance Specification window (Figure 5.10a).

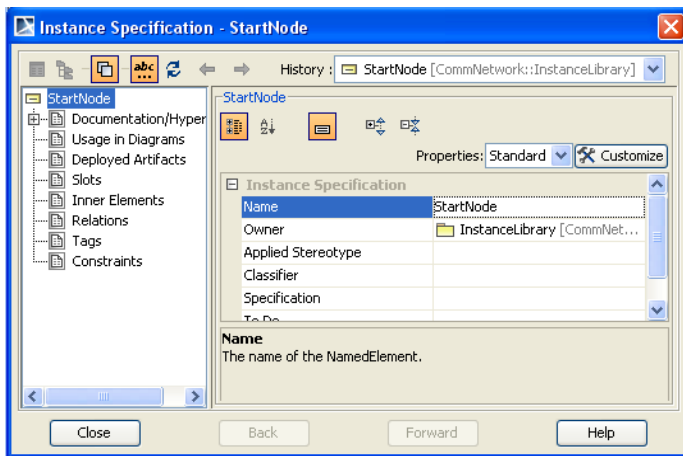


Figure 5.10a Instance Specification window

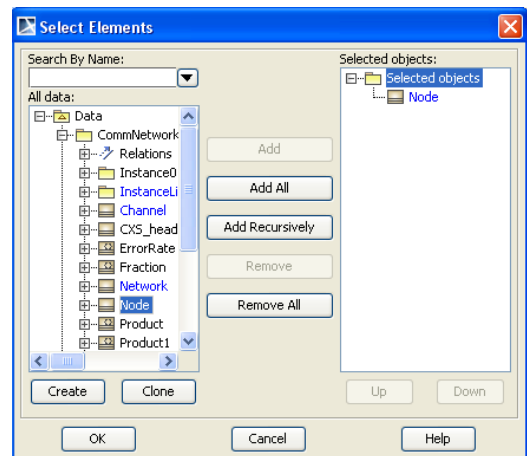


Figure 5.10b Select Elements window

- iv. Select Classifier and click Edit icon to open window in Figure 5.10b
- v. Double-click on Node to set StartNode as an instance of Node and click OK.
- vi. Click Slots in Instance Specification window (Figure 5.10a) and double-click on each of the value properties to activate them. For the StartNode instance, we assume a given input signal level by assigning R1 as 10 (arbitrary units) and R2 as 0. All the remaining slots are left empty. Click Close at the end.
- vii. Repeat this process for four more standard instances:
 1. ThruNode (stations which both receive from and transmit to other stations in the network). All slots are activated, but left empty.
 2. EndNode (stations which only receive, i.e. final destination). Assign C1 and C2 values = 1 or any non-zero fixed value because there are no “downstream” channels to supply these values.
 3. SingleChannel (channels in which only one I/O path is used). Classifier = Channel. Assign C = 5 (capacity in arbitrary units) and IT2 = 0 because input to second I/O path is not used.
 4. DoubleChannel (channels in which both I/O paths are used). Classifier = Channel. Assign C = 5 (capacity in arbitrary units).

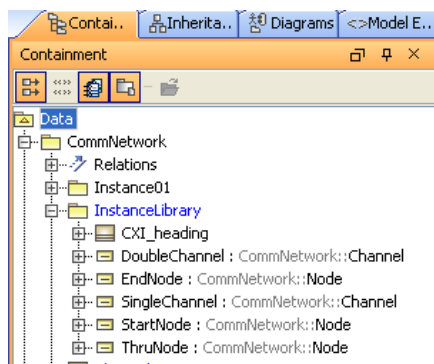


Figure 5.11 InstanceLibrary contents

viii. Assign causality to all standard instances by right-clicking on InstanceLibrary and selecting ParaMagic→Util→Assign default causalities.

ix. Create a CXI heading in InstanceLibrary by selecting ParaMagic→Util→Create CXI_heading. The final contents of InstanceLibrary should appear as in Figure 5.11. Any of these instances can be dragged into an instance diagram in another package and used without further modification.

- b. Create a specific instance package for the network proposed
 - i. RC on CommNetwork and select New Element→Package, Name = Instance01
 - ii. RC on Instance01 and select New Diagram→SysML Diagrams→SysML Block Definition Diagram, Name = Instance01.

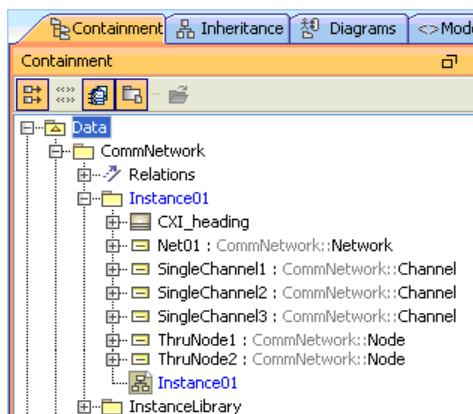


Figure 5.12 Instance01 contents

- iii. We need four separate copies of SingleChannel in our network, to represent Channels A, B, C, and D. In this example, we will copy SingleChannel and paste three copies into Instance01, named SingleChannel1, SingleChannel2 and SingleChannel3.
- iv. We need two separate copies of ThruNode in our network, to represent Nodes 2 and 3. In this example, we will copy ThruNode and paste two copies into Instance01, named ThruNode1.

- v. Create an instance of the Network block inside Instance01, Name = Net01. The final contents of Instance01 should appear as in Figure 5.12
- vi. Drag all the instances in InstanceLibrary and Instance01 into the Instance01 diagram.
- vii. Using the connector icon on the floating toolbar, create link from Net01 to StartNode. A Select Association window appears, as in Figure 5.13. Check the first line to assign the StartNode instance to the N1 part property.
- viii. Draw connectors from Net01 to the remaining instances. In each case, assign an appropriate instance to each part property, as shown in Figure 5.14.
- ix. Assign *target* causality to slot R in EndNode.
- x. Create a CXI heading in package Instance01 by selecting ParaMagic→Util→Create CXI_heading. The final contents of Instance01 diagram should appear as in Figure 5.14.

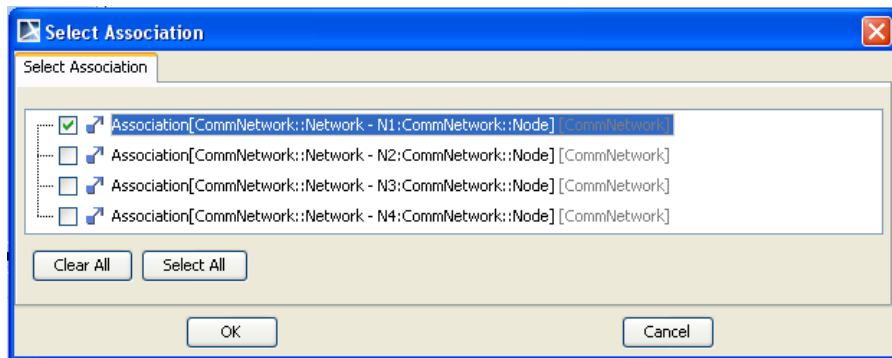


Figure 5.13 Select Association window assigning StartNode instance to N1

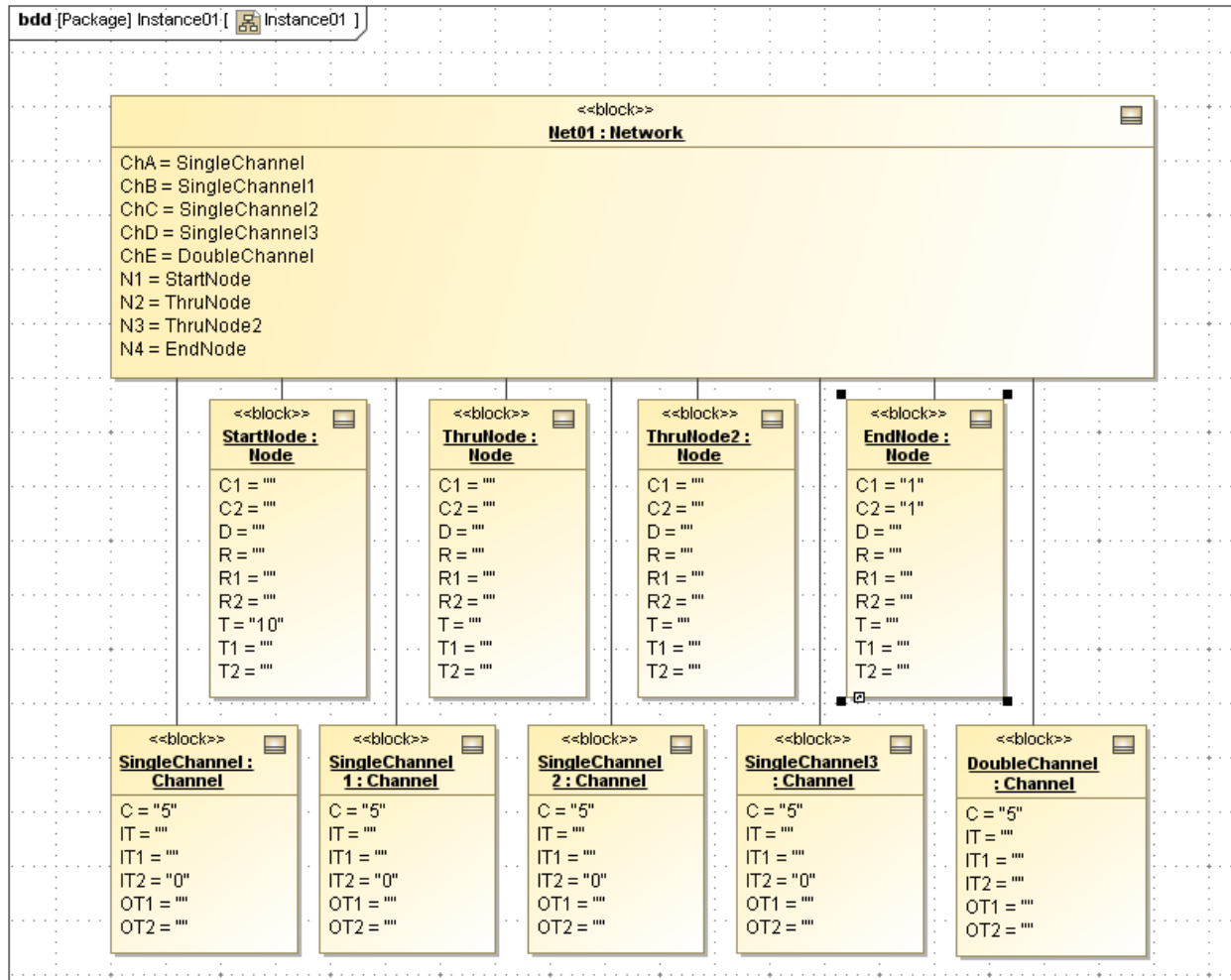


Figure 5.14 Instance01 diagram complete

Discussion – Instance Generation

We have used four different methods for creating instance in our first four tutorials,

- Creating individual instances from the central toolbar
- Creating individual instances using menu commands
- Using the Instance Generation Wizard
- Copying and pasting existing instances from a standard instance library

Similarly, there are at least two ways of linking instances,

- Draw a link from the higher block to the lower and use the Select Association window to assign the instance to an existing part property.
- Double-click on the higher block, click on Slots, double-click on a part property, and assign it to an existing instance. In this approach, no visible line connects the instances on the instance diagram, but the higher-level instance box displays the association, e.g. ChE = DoubleChannel.

Which method to choose depends on the situation. Being familiar with multiple approaches is advantageous.

Step VIII Solve the Instance

- In Figures 5.15 and 5.16, only a few of the output variables are expanded to display. All of the values are available, however.

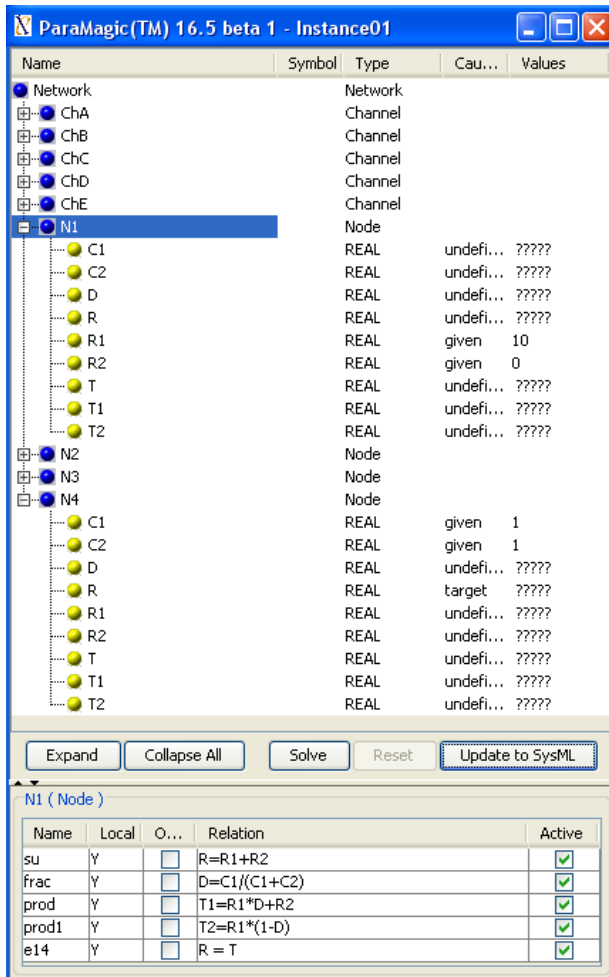


Figure 5.15 Browser for Instance01 before solution

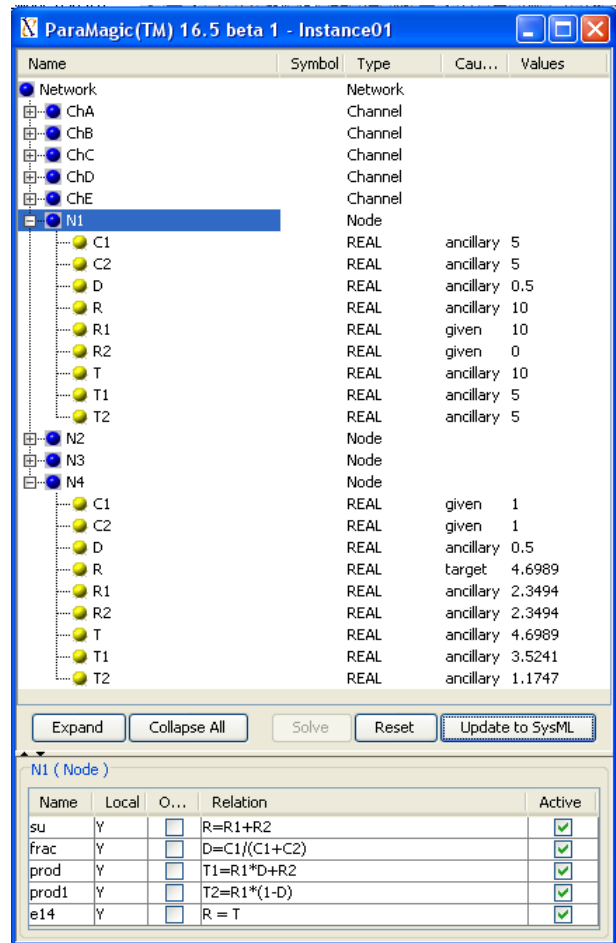


Figure 5.16 Browser for Instance01 after solution

6. SYSML PARAMETRICS TUTORIAL - ORBITAL

6.1 Objective

Create a SysML project combining an orbital mechanics subsystem and a spacecraft subsystem. The orbital mechanics section is linked to a spreadsheet which calculated solar power levels and data transmission levels for ten sections in the spacecraft's orbit around Mars, based on the geometry between the spacecraft, Mars, Earth and the Sun. See Figure 6.1. Note that the writer claims no expertise in this field.

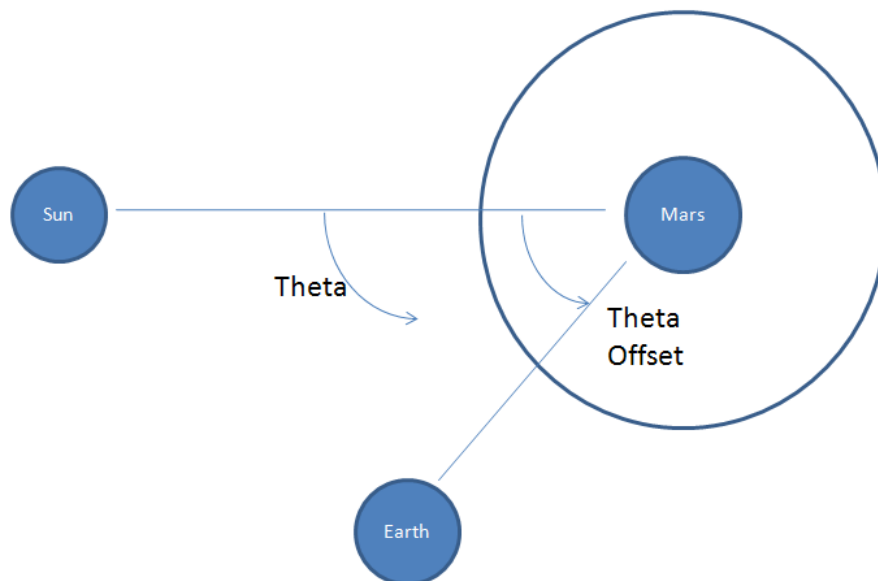


Figure 5.1 Outline of Objective

The spacecraft subsystem is tied to a second spreadsheet which contains the power and data transmission requirements for the spacecraft by subsystems, three instrument packages, a transmitter, and a power system. Information from both spreadsheets is combined at the Mission level using parametric constraints solved by Mathematica and final results are written to a third spreadsheet.

What the User Will Learn

- Read and write between Microsoft Excel spreadsheets and MagicDraw SysML instance blocks
- Working with aggregates (parameters that can contain a list of values)
- Plotting aggregates with standard Mathematica functions

Aggregates are parameters with multiple values, for example, $A = [A_1, A_2, A_3, A_4]$, and are valuable in multiple ways. They can express vectors, time series, and other sets of numbers that will be handled collectively or repetitively. In this tutorial, the individual elements of the aggregates represent values for ten separate sections of the orbital cycle. The specific number of elements does not need to be defined until the instance stage.

Mathematica has extensive graphics capability. Graphs can be created and saved during ParaMagic execution, using several ParaMagic-provided standard functions. User-created Mathematica functions, described in the ParaMagic Users Guide, provide even more flexibility.

Spreadsheets, such as Microsoft Excel, could be envisioned either as a means of loading data into a specific instance from an existing table or database, a means of reporting and organizing results from a parametric simulation, or a mathematical solver for parametric relationships that are part of the model. Currently, ParaMagic only supports the first two functions.

System Requirements

- Microsoft Excel 2003 or 2007. Note: Excel is not required to execute the ParaMagic-Excel interface. It is required to create, edit and view the spreadsheets used in the example.
- In order to use standard Mathematica functions that produce a graphics file, Mathematica must be loaded on the user's local machine or server (no web services) and the ICAX standard function library must be installed for autoloading.

6.2 Step-by Step Tutorial

Step I Create Project

1. Create new project
 - d. Name = Orbital
2. Create a package within the project
 - a. RC (Right-click) on Data folder in Containment tree (left column)
 - b. Choose New Element→Package
 - c. Enter Name = Orbital

Step II Create Infrastructure

3. Install ParaMagic Profile module, same as in Tutorial 1.

Step III Create Structural Model

4. Create elements in model
 - a. Inside the Orbital package, create blocks call Mission, Orbital, Spacecraft, InPkg1, InPkg2, InPkg3, Transmitter, and PowerSystem.

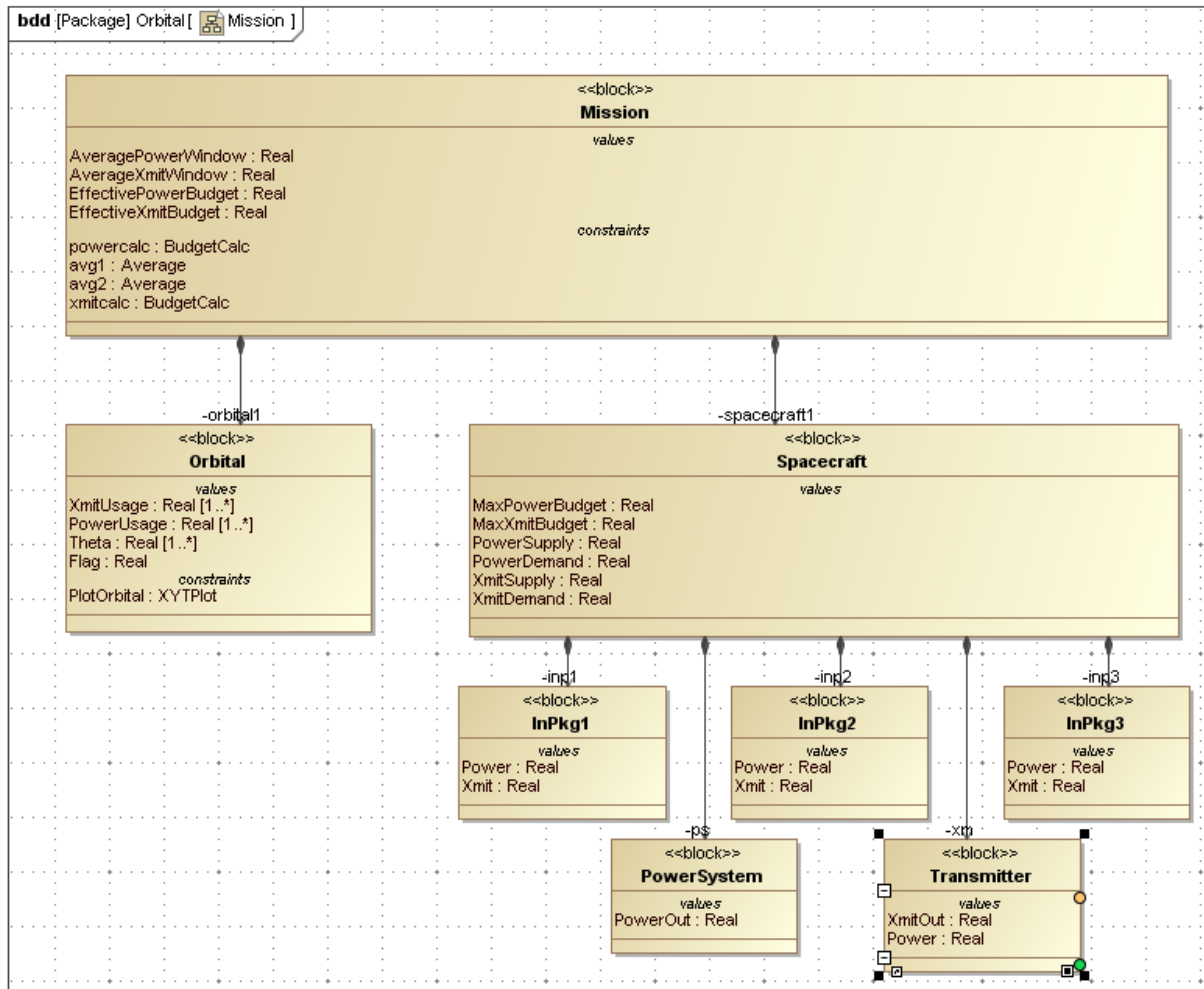


Figure 6.2 Block Definition Diagram for Orbital model

- a. Create a block definition diagram, Name = Mission, and drag the Mission elements into the diagram. See Figure 6.2. Use Directed Composition connectors to link the blocks in the hierarchy shown.
- b. Inside Mission, create a value property called AveragePowerWindow and assign it a Type as Real. Repeat this procedure for three additional properties in Mission: AverageXmitWindow, EffectivePowerBudget, and EffectiveXmitBudget.
- c. Referring to Figure 6.2, create value properties as shown in the Spacecraft, InPkg1, InPkg2, InPkg3, Transmitter, and PowerSystem blocks, all type Real.
- d. Inside Orbital, create the value properties as shown in Figure 6.2.
 - i. Create Theta, Type Real
 - ii. Double click on Theta (in Containment tree or on block definition diagram) and open the Value Property dialog box (see Fig. 6.3).
 - iii. Choose 1...* from the dropdown list next to Multiplicity. This allows the variable Theta to contain any number of separate values.
 - iv. Repeat this procedure for PowerUsage and XmitUsage.

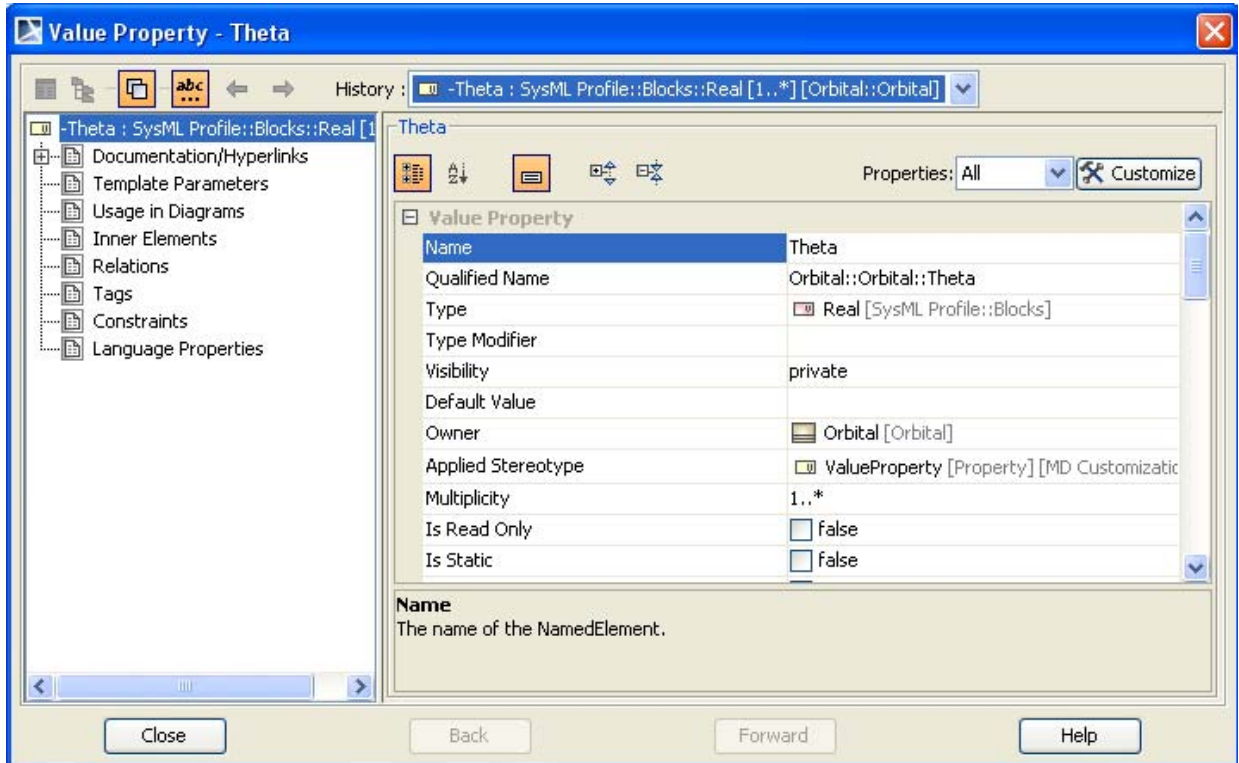


Figure 6.3 Setting the value property Theta multiplicity to 1..*

Step IV Create Constraints

5. Create two constraint blocks, Average and BudgetCalc, as shown in Figure 6.4. Note that Average uses list, a constraint parameter with multiplicity larger than one. To change the multiplicity of a constraint parameter, RC the constraint block in the BDD diagram. Average calculates the average of all the values contained in list and returns a single value result, mean. The ParaMagic User Guide provides a list of Mathematica functions that act on arguments with multiplicity ≥ 1 .

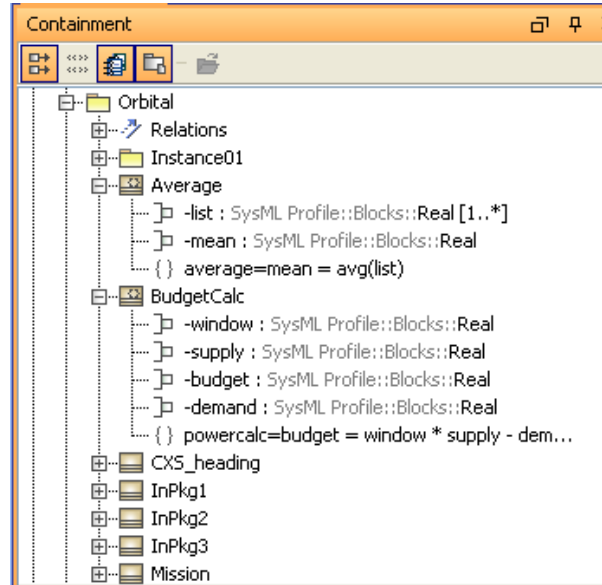


Fig. 6.4 Constraint Blocks

6. Create a constraint block for plotting the power and transmission availability versus Theta.
 - a. See User Guide for a list of standard Mathematica graphics and statistical functions that are provided with ParaMagic. These should be configured to automatically load when Mathematica is started.

- b. Create a constraint block, XYTPlot, with four constraint parameters, x, y, t and c, where x, y, and t have multiplicity ≥ 1 .
- c. Add a constraint to XYTPlot.
c = cMathematica(ICAXPlotXYT,t,x,y,"Availability","Sector","Power")
 ICAXPlotXYT is a standard function for graphing two lists of values against a third list (x, y, and t, respectively), with the three final string arguments (in quotes) determine graph title, x axis label, and y axis label.
- d. Assign the destination for the XYTPlot graphics file (new for ParaMagic since 16.5)
 - i. Double-click on the constraint and open the Constraint window (see Figure 6.5)
 - ii. Select Tags
 - iii. Scroll down to <<External Models>> section
 - iv. Double-click on working_dir (or select and click Create Value)
 - v. Enter a destination directory for any plots generated, e.g. C:\\MD-temp. Note that this directory must already exist when the ParaMagic model is executed (it will not be created automatically) and a double back-slash character is used in the pathname.

Note for Mac Users: The destination directory needs to be phrased using the path notation common to Mac operating systems. For example /Users/JohnDoe/Documents/Orbital

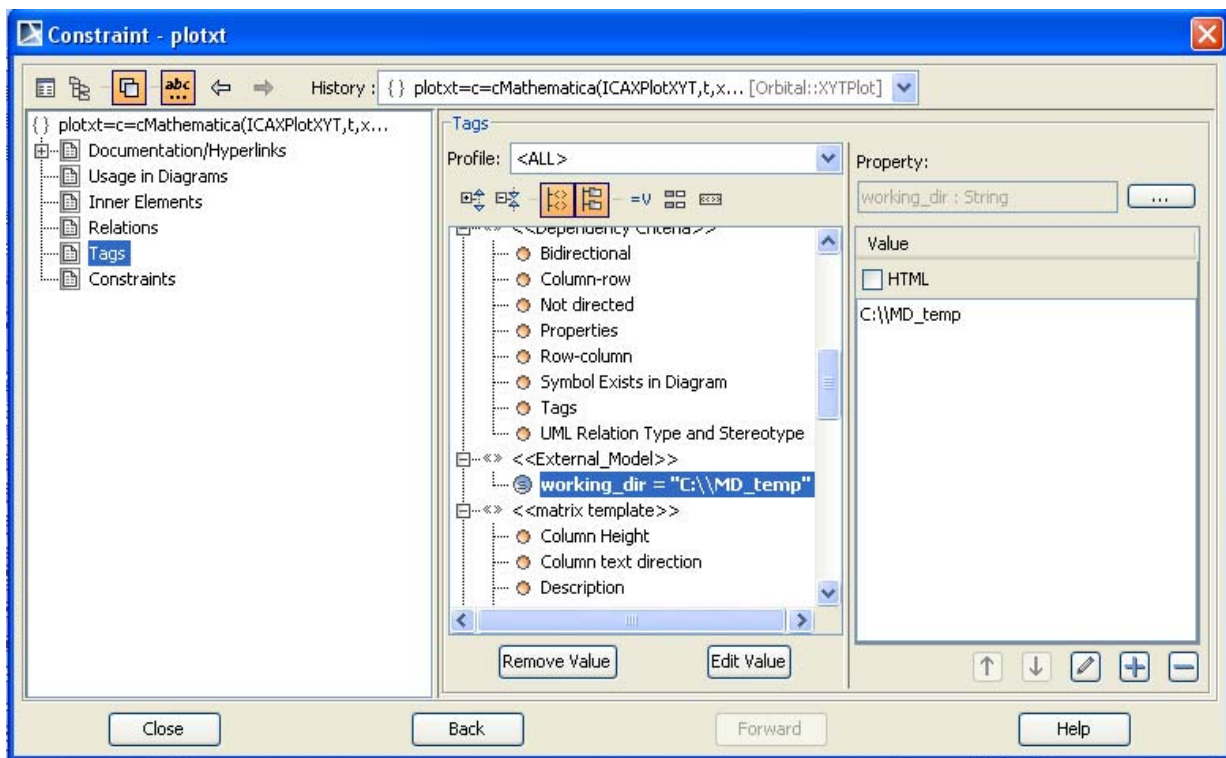


Figure 6.5 Constraint window for defining graph output destination

Step V Create Parametrics Model

7. Create the parametrics diagram shown in Figure 6.6 inside the block Mission. There are two usages of each of the constraint blocks defined in Step 5.

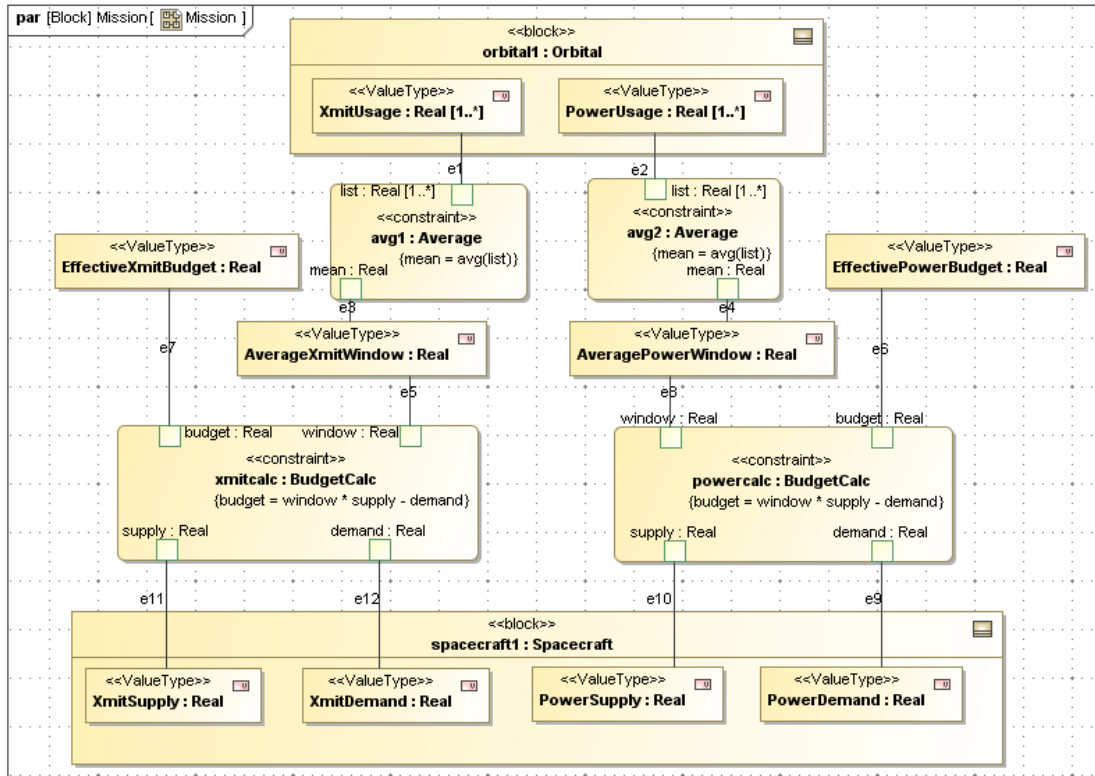


Figure 6.6 Parametric diagram for Mission

8. Create a second parametrics diagram inside Orbital, as shown in Figure 6.7. This diagram exists to plot the values of the PowerUsage and XmitUsage against Theta during ParaMagic execution.
 - a. Use Select Parts window to select PowerUsage, XmitUsage, Theta and Flag to appear in the diagram. ICAXPlotXYT will return a value of 1 into the parameter Flag when the Mathematica function is successfully executed.
 - b. Drag the constraint XYTPlot into the diagram and connect as shown.

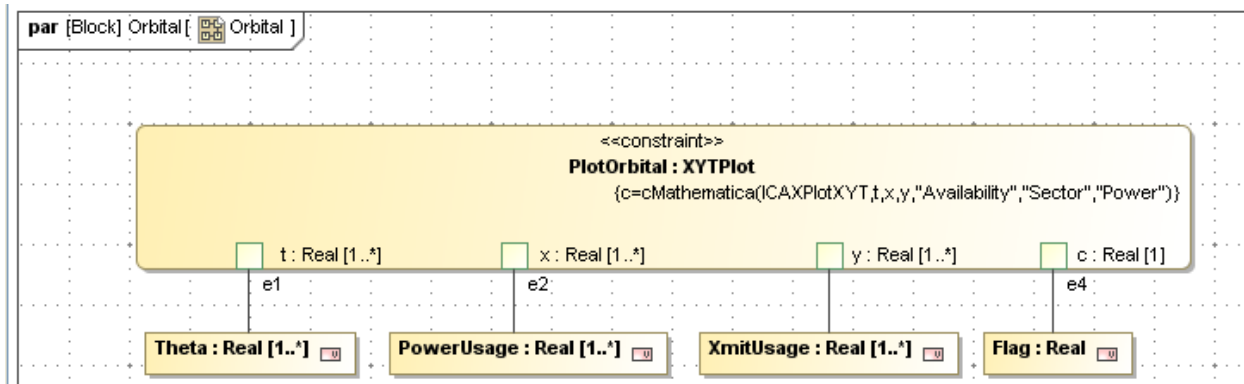


Figure 6.7 Parametric diagram for Orbital

Step VI Validate Parametrics Model

9. To create the CXS_heading and validate the model schema, follow the instructions in Step VI of previous tutorials

Step VII Create an Instance

In this example, we will create an instance of the SysML model and three separate Microsoft Excel spreadsheets that will link to this instance and allow data to be written from Excel to MagicDraw SysML and from MagicDraw SysML to Excel. This supports the use of spreadsheets to load data into a model and to report results from parametric simulation of the model. In normal circumstances, some or all of these spreadsheets may exist prior to the SysML model, but in this tutorial, we will create them at this stage.

10. Create three workbooks in Microsoft Excel that instances of the model will link to.
 - a. Open Excel
 - b. Create New
 - c. Build and save the workbook *spacecraft* shown in Figure 6.8 (in worksheet Sheet1).

	A	B	C	E
1	Spacecraft Model			
2		Power (Watts)	Xmit (Mb/sec)	
3	Instrumentation Pkg 1	100	20	
4	Instrumentation Pkg 2	100	50	
5	Instrumentation Pkg 3	150	100	
6	Power Supply Output	500		
7	Transmitter Output	300	300	
8				
9	Max Budget	-150	130	
10	Supply	500	300	
11	Demand	650	170	
12				

Figure 6.8a Workbook spacecraft, showing values

	A	B	C
1	Spacecraft Model		
2		Power (Watts)	Xmit (Mb/sec)
3	Instrumentation Pkg 1	100	20
4	Instrumentation Pkg 2	100	50
5	Instrumentation Pkg 3	150	100
6	Power Supply Output	500	
7	Transmitter Output	300	300
8			
9	Max Budget	=B6-B3-B4-B5-B7	=C7-C3-C4-C5
10	Supply	=B6	=C7
11	Demand	=B3+B4+B5+B7	=C3+C4+C5
12			

Figure 6.8b Workbook spacecraft, showing formulas

- d. Build and save the spreadsheet *orbital* shown in Figure 6.9 (in worksheet Sheet1).

	A	B	C	D
1	Orbital Mechanics Profile	Author		Date
2				
3		ThetaOffset	0.5236	
4				
5	Time Period	Theta	Power	Data
6	0	0.00	1.00	0.97
7	1	0.63	0.90	1.00
8	2	1.26	0.65	0.93
9	3	1.88	0.35	0.78
10	4	2.51	0.10	0.55
11	5	3.14	0.00	0.26
12	6	3.77	0.10	0.05
13	7	4.40	0.35	0.36
14	8	5.03	0.66	0.63
15	9	5.65	0.90	0.84

Figure 6.9a Workbook orbital, showing values

	A	B	C	D
1	Orbital Mechanics Profile	Author		Date
2				
3		ThetaOffset	0.5236	
4				
5	Time Period	Theta	Power	Data
6	0	=ROUND(A6*PI()/5,2)	=ROUND(COS(B6/2)^2,2)	=ROUND(ABS(COS((B6-\$C\$3)/2)),2)
7	1	=ROUND(A7*PI()/5,2)	=ROUND(COS(B7/2)^2,2)	=ROUND(ABS(COS((B7-\$C\$3)/2)),2)
8	2	=ROUND(A8*PI()/5,2)	=ROUND(COS(B8/2)^2,2)	=ROUND(ABS(COS((B8-\$C\$3)/2)),2)
9	3	=ROUND(A9*PI()/5,2)	=ROUND(COS(B9/2)^2,2)	=ROUND(ABS(COS((B9-\$C\$3)/2)),2)
10	4	=ROUND(A10*PI()/5,2)	=ROUND(COS(B10/2)^2,2)	=ROUND(ABS(COS((B10-\$C\$3)/2)),2)
11	5	=ROUND(A11*PI()/5,2)	=ROUND(COS(B11/2)^2,2)	=ROUND(ABS(COS((B11-\$C\$3)/2)),2)
12	6	=ROUND(A12*PI()/5,2)	=ROUND(COS(B12/2)^2,2)	=ROUND(ABS(COS((B12-\$C\$3)/2)),2)
13	7	=ROUND(A13*PI()/5,2)	=ROUND(COS(B13/2)^2,2)	=ROUND(ABS(COS((B13-\$C\$3)/2)),2)
14	8	=ROUND(A14*PI()/5,2)	=ROUND(COS(B14/2)^2,2)	=ROUND(ABS(COS((B14-\$C\$3)/2)),2)
15	9	=ROUND(A15*PI()/5,2)	=ROUND(COS(B15/2)^2,2)	=ROUND(ABS(COS((B15-\$C\$3)/2)),2)

Figure 6.9b Workbook orbital, showing formulas

- e. Build and save the spreadsheet *mission* shown in Figure 6.10 (in worksheet Sheet1). This will be used to report the results of the parametric simulation, so the non-title cells are currently empty.

	A	B
1	Mission Estimates	
2		
3		
4	Average Power Window	
5	Average Xmit Window	
6	Effective Power Budget	
7	Effective Xmit Budget	
8		

Fig. 6.10 Workbook mission, showing titles

11. Create an instance. Review Step VII in the previous tutorials for the detailed procedures.
 - a. Create a package, Instance01, within the package Orbital.
 - b. Create a block definition diagram, Instance01, within the package Instance01.
 - c. Create the instances of the model elements as shown in Figure 6.11 by dragging Instance from the toolbar, selecting the block it will be an instance of from the Select Classifiers window, and assigning the instance block a unique name.
 - d. Create linkages between mis:Mission and orb1:Orbital, and between mis:Mission and spa01:Spacecraft. Note that no linkages have been created between spa01:Spacecraft and its subsystems. These are not required because there are no parametric calculations linking these blocks in SysML parametric diagrams. They are linked inside the spreadsheet spacecraft.xlsx.
 - e. Activate the slots as in previous tutorials, by double-clicking on the blocks, selecting Slots, and double-clicking on each slot to be activated. In this example, no slot has values assigned at this stage. Given values will be loaded from the spreadsheets.

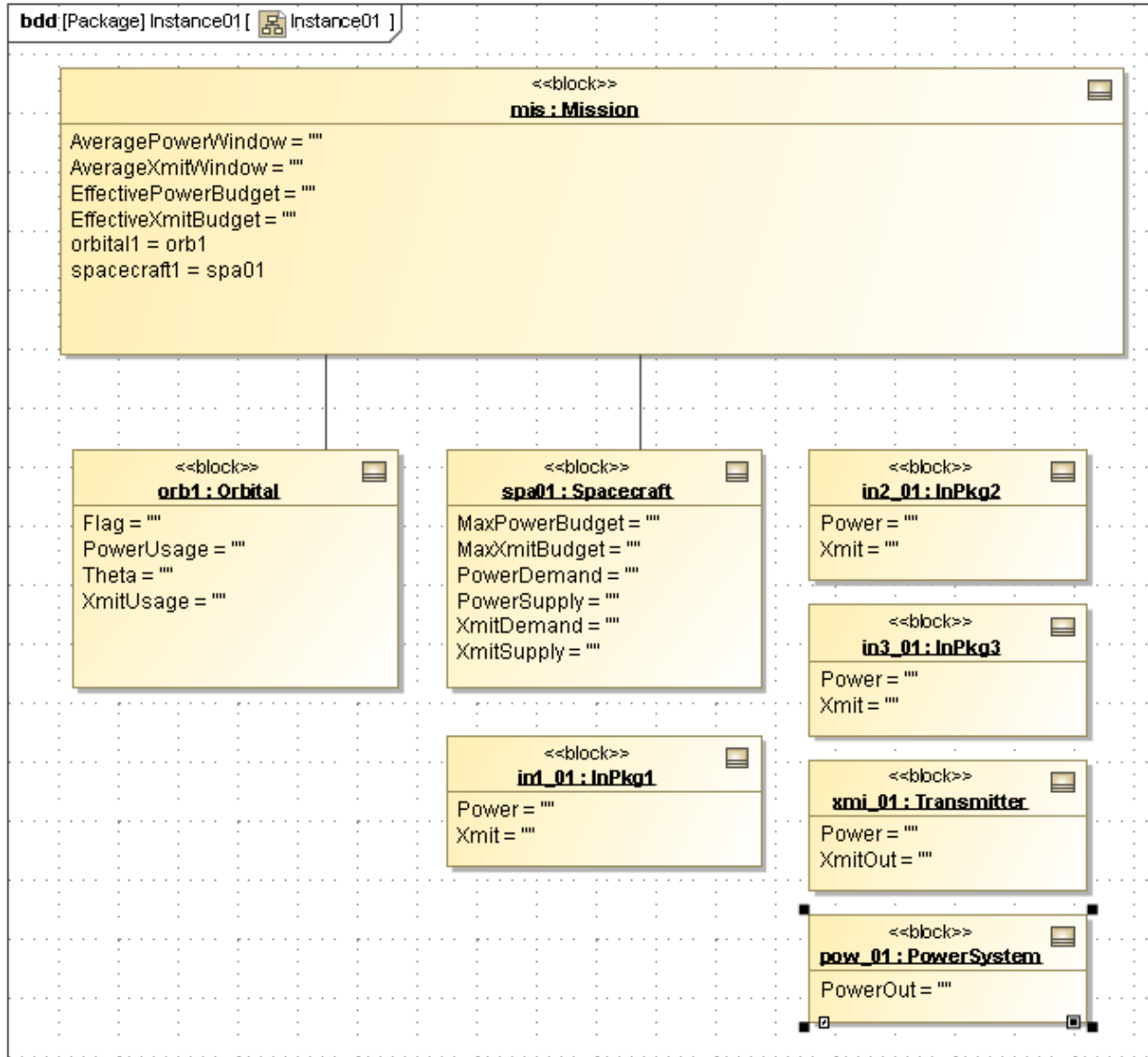


Figure 6.11 Instance01 Diagram

Discussion – Aggregates

There are two methods for creating Slots with multiple values (aggregates) in instances. The first method, which we use here, is to activate the slot as a single value and then read multiple values into it (e.g., from a range of cells in a spreadsheet). Multiple slot values are automatically created. The second method is to create room for multiple values when we activate the slots. For example, if we were to activate Theta by double-clicking it in the Instance Specification window as usual (see Figure 6.12), we could add additional empty slots by clicking the Plus button on the bottom right side of the Instance Specification window nine times to create a total of ten empty values. We could enter ten numbers in a similar way, using the Plus button after each value was entered.

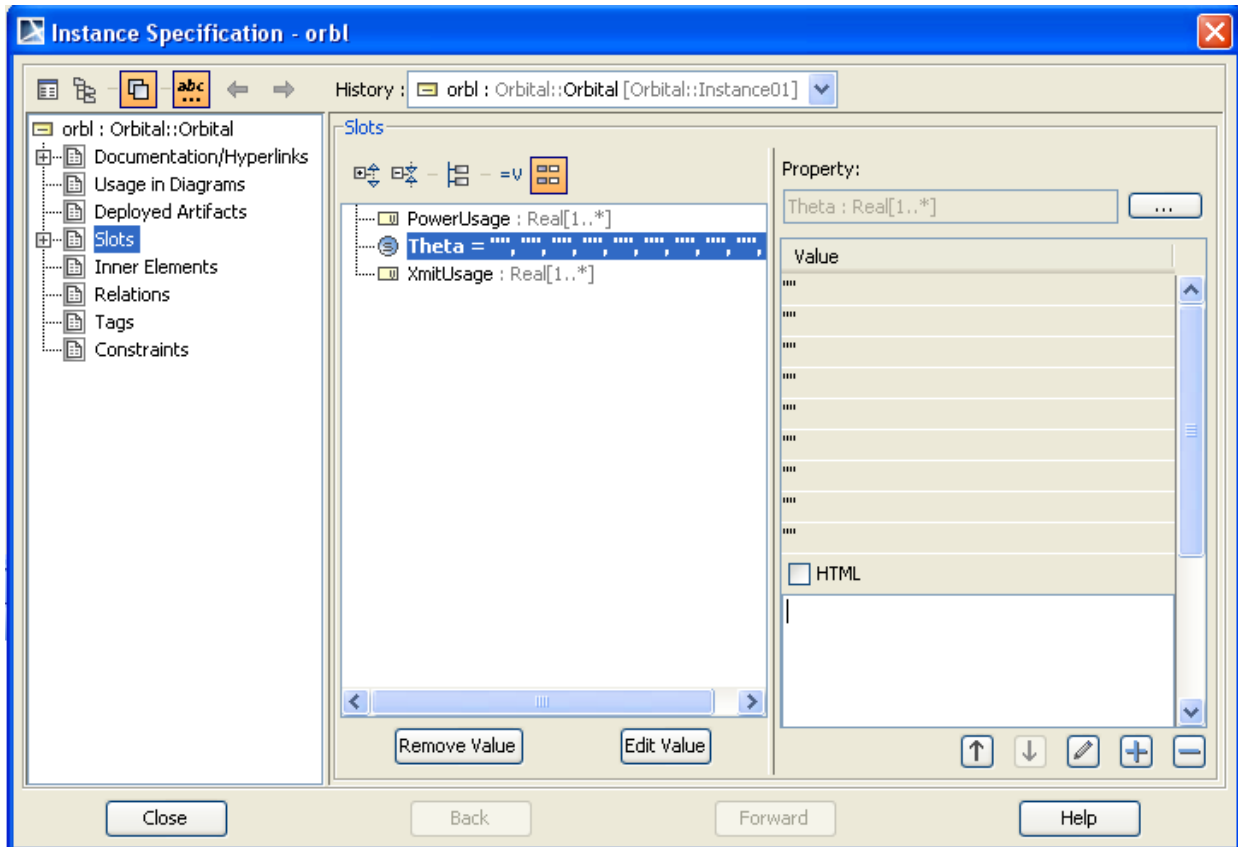


Figure 6.12 Assigning ten empty values to the Theta parameter in orb1

12. Define the ParaMagic causality of each parameter. Each slot value in the instance can be assigned two causalities, one for ParaMagic/Mathematica calculations and one for Excel export/import actions. We will set the Excel causality type for each parameter in the XIF-Excel Browser windows in the following steps. We still need to set ParaMagic causality type for all parameters used in Mathematica calculations.
 - a. Right-click on Instance01 and select ParaMagic→Util→Create CXI_heading.
 - b. Right-click on Instance01 and select ParaMagic→Util→Assign default causalities.
 - c. Assign Target causality to EffectivePowerBudget, EffectiveXmitBudget and Flag using the procedure described in previous tutorials.

13. Create the linkage between the Spacecraft instance blocks and spreadsheet *spacecraft.xlsx*.
 - a. In the Containment window, right-click on the instance block In1_01 and select ParaMagic→Excel→Setup.
 - b. The ParaMagic-Excel Setup window will appear as in Figure 6.13a. The spreadsheet to be linked can be identified in two ways
 - i. Browse to the desired file. This will enter the full path name in the Workbook file text box, or
 - ii. Type in the filename, e.g. *spacecraft.xlsx* and click Refresh. If no path is typed, it will assume spreadsheet is in the same folder as the model (.mdzip

file). This may be more flexible if the model and spreadsheet are moved around between different systems.

- c. After the Workbook name is entered, the available worksheets within that file will appear as a drop-down list by the Worksheet name label. Select the desired worksheet, Sheet1. See Figure 6.13b.
- d. Click OK.

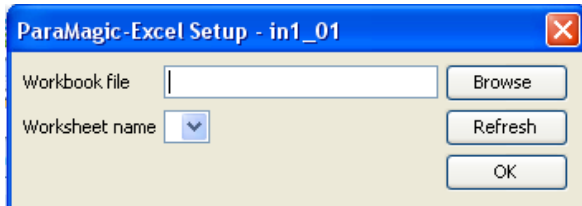


Figure 6.13a ParaMagic- Excel Set-up for instance

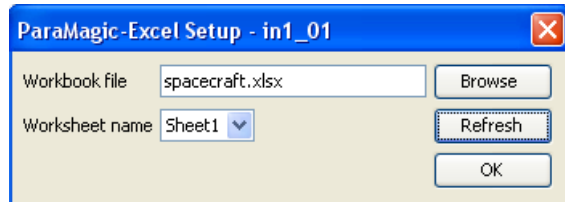


Figure 6.13b ParaMagic- Excel Set-up for instance after completion

- e. In the Containment window, right-click on the Power slot under instance block in1_01 and select ParaMagic→Excel→Setup. See Figure 6.14.
 - i. Under Cell range, enter the cell coordinates on the spreadsheet for the parameter desired, B3.
 - ii. Under Access Mode, select Read_from_Excel (the value in the spreadsheet cell will be copied to the instance, though only after we transfer the data in step 14).
 - iii. Click OK

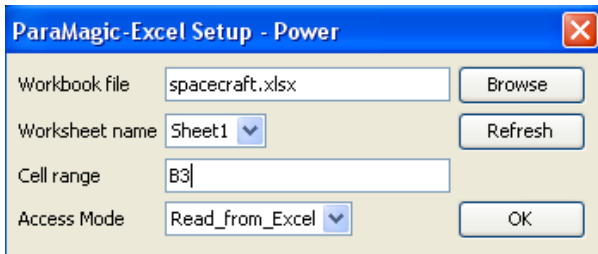


Figure 6.14 ParaMagic- Excel Set-up for slot

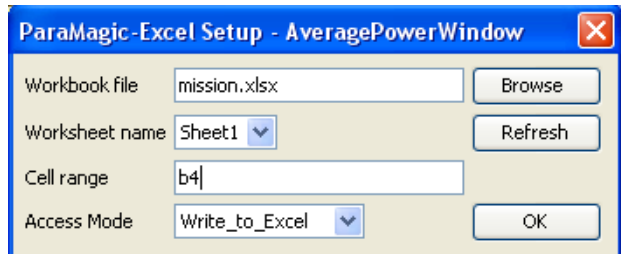


Figure 6.15 ParaMagic- Excel Set-up for slot

- f. Repeat for the Xmit slot under instance block in1_01. The corresponding Cell range is C3.
- g. Repeat steps d, e, and f for the remainder of the instance blocks in the diagram.

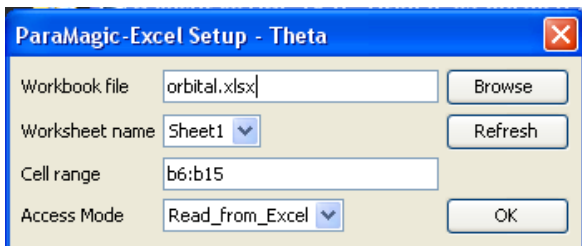


Figure 6.16 ParaMagic- Excel Set-up for aggregate slot

- i. For slots where we will write the result from the SysML model to the mission.xlsx spreadsheet, set the Access Mode to “Write_to_Excel”, as in Figure 6.15.
- ii. For slots where we will read in multiple values from a spreadsheet, Cell range will contain

a row or column of cells, as in Figure 6.16.

Step VIII Solve the Instance

14. Run the Excel→SysML data transfer

- a. Right-click the Instance01 package.
- b. Select ParaMagic→Excel→Read from Excel
- c. During execution, all target values are read from all spreadsheets and written to the SysML instance diagram. See Figure 6.17.

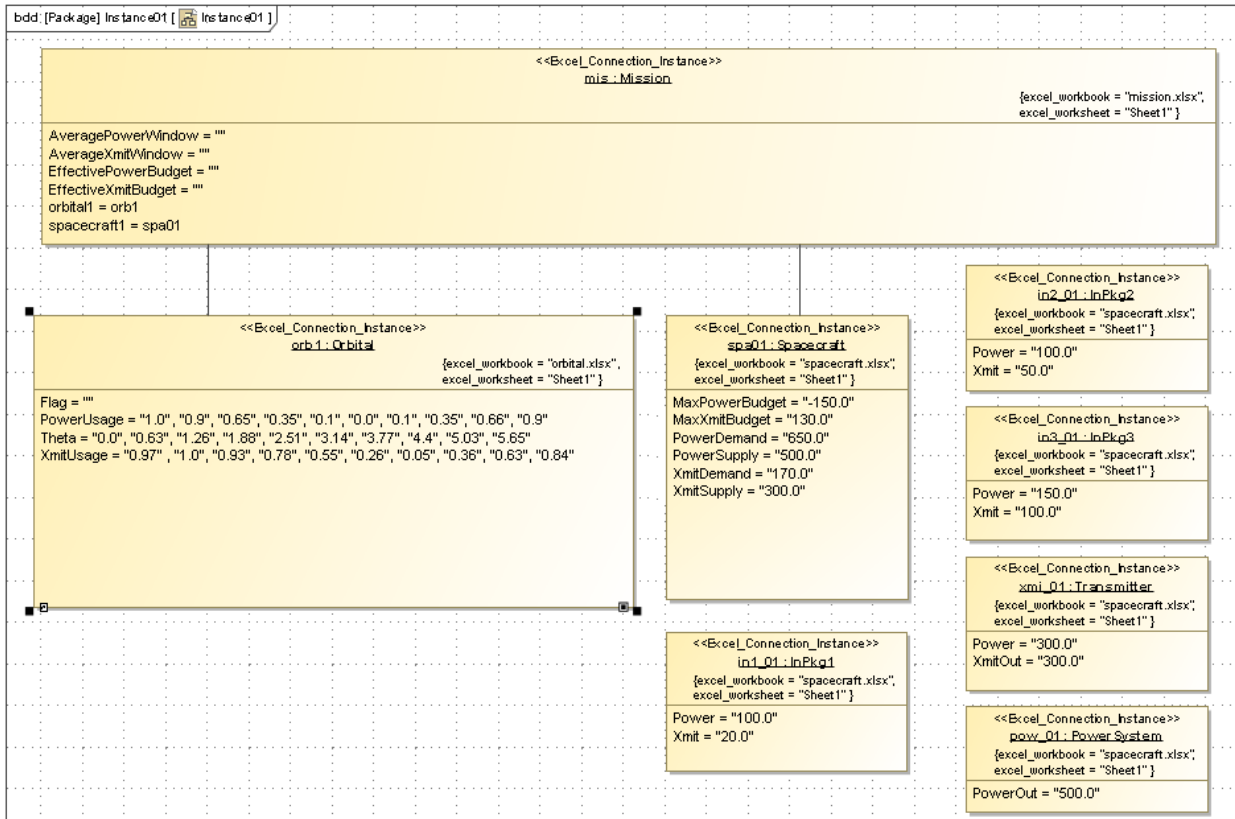


Figure 6.17 Values displayed in Instance01 diagram after Read from Excel

15. Run the ParaMagic/Mathematica parametric simulation.

- a. Right-click the Instance01 package.
- b. Select ParaMagic→Browse
- c. Click Solve. Browser before and after solution is shown in Figure 6.18. Note that Flag has a value of 1 after solution, indicating completion of the graph.

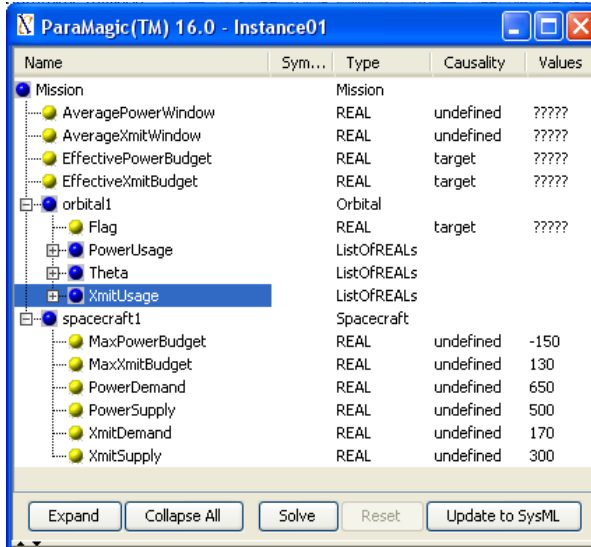


Figure 6.18a Browser before execution.

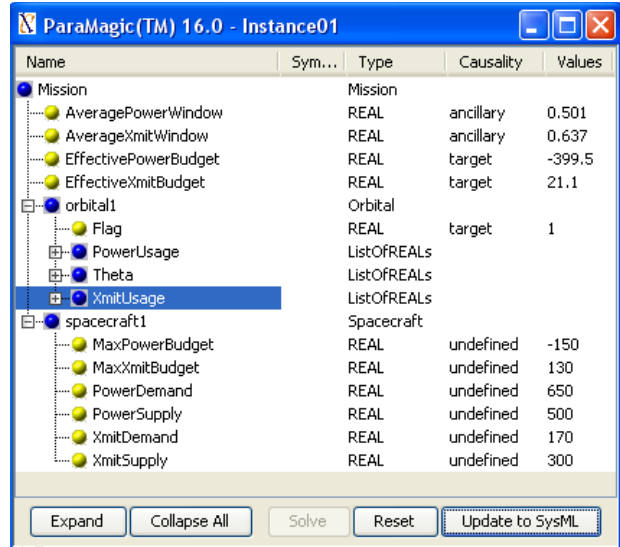


Figure 6.18b Browser after execution.

- d. Click Update to SysML. Mis:Mission slot values appear as in Figure 6.19.

16. Run the SysML→Excel data transfer

- a. Right-click the Instance01 package.
- b. Select ParaMagic→Excel→Write to Excel
- c. During execution, all given values are read from the SysML instance diagram and written to the spreadsheet *mission.xlsx*. See Figure 6.20.

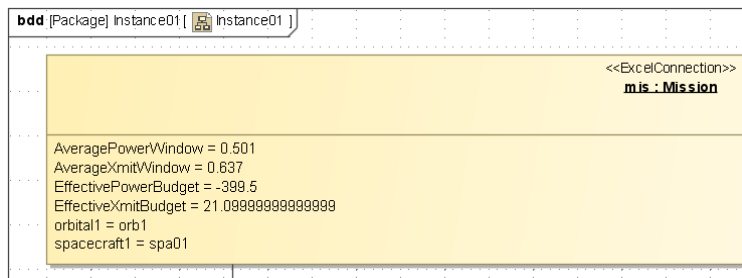


Figure 6.19 Browser before execution.

	A	B
1	Mission Estimates	
2		
3		
4	Average Power Window	0.501
5	Average Xmit Window	0.637
6	Effective Power Budget	-399.5
7	Effective Xmit Budget	21.1
8		

Figure 6.20 mission.xlsx after write.

17. View the graph of PowerUsage and XmitUsage vs Theta created by ParaMagic.

- a. The ICAXPlotXYT function always returns a graphical file named **XYTlineplot.jpg** and saves it in the directory designated in step 6.d.
- b. Create a hyperlink from the model to the graph so that it can be called up quickly after the simulation is complete.
 - i. Right-click on the package Orbital→New Element→Hyperlink
 - ii. A dialog box will appear, as shown in Figure 6.21. Use the browser (button with three dots) to point to the graphics file. Alternately, type in the name or select it from the list.
- c. Double-click on the hyperlink in the Containment Tree to see graph, as in Figure 6.22.

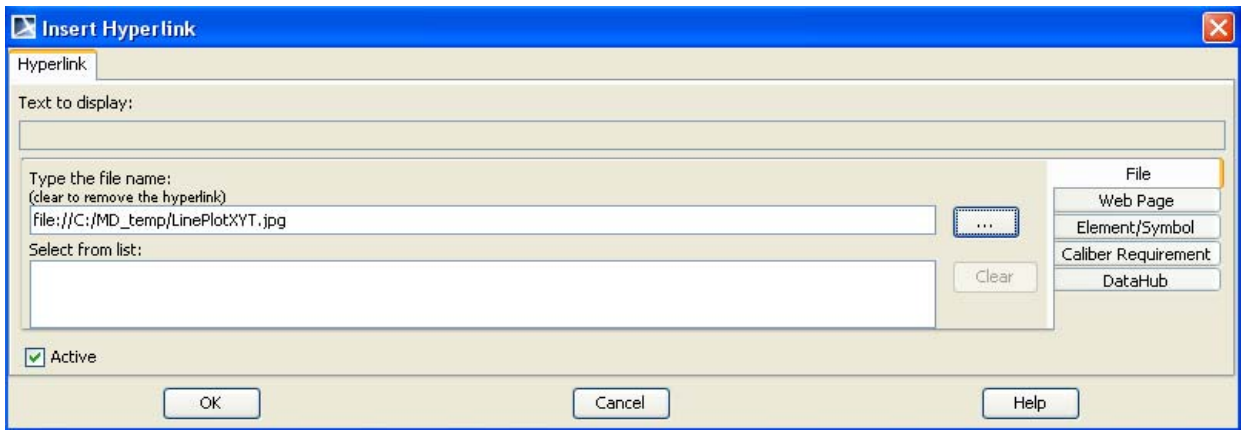


Figure 6.21 Hyperlink

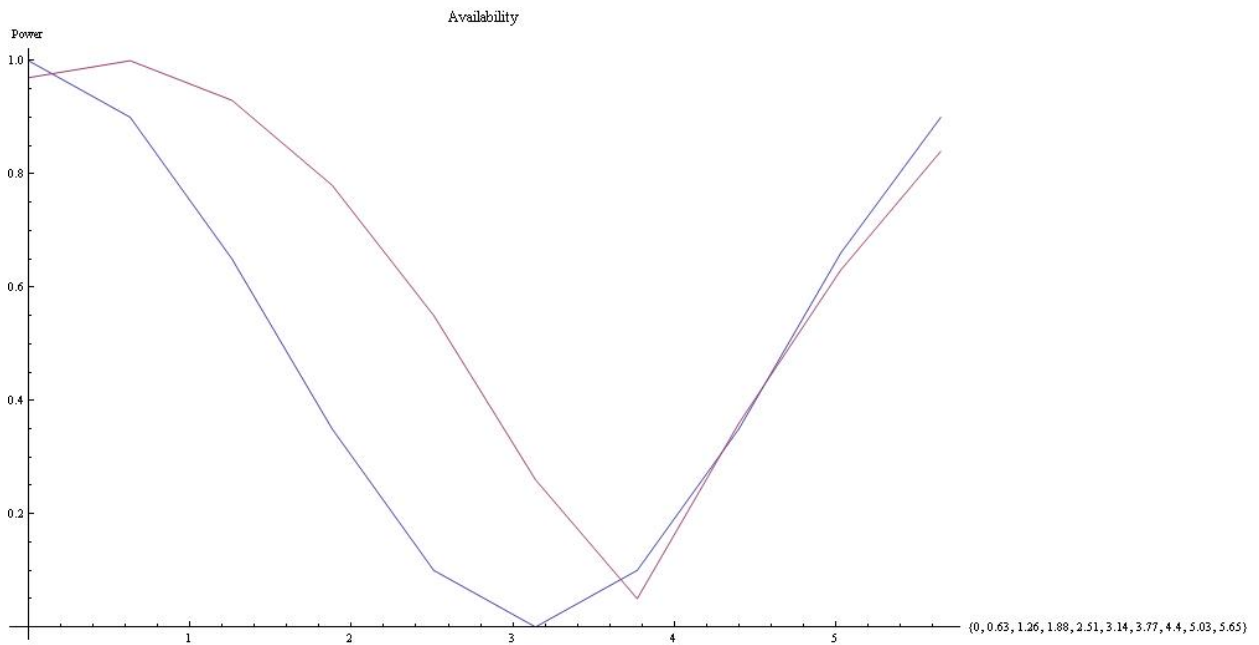


Figure 6.22 Graph created during ParaMagic execution

7. SYSML PARAMETRICS TUTORIAL - HOMEHEATING

7.1 Objective

Create a SysML project incorporating a Simulink simulation model of a home heating system and a MATLAB function that predicts outside temperature. The Simulink model, a minor modification of a standard demonstration model from MathWorks, is shown in Figure 7.1. Input 1 is the outside temperature, which varies periodically through the day and through the year. Outputs 1, 2 and 3 are the cumulative cost, indoor temperature, and outdoor temperature as a function of time. The objective is to estimate the cost per day of heating the home.

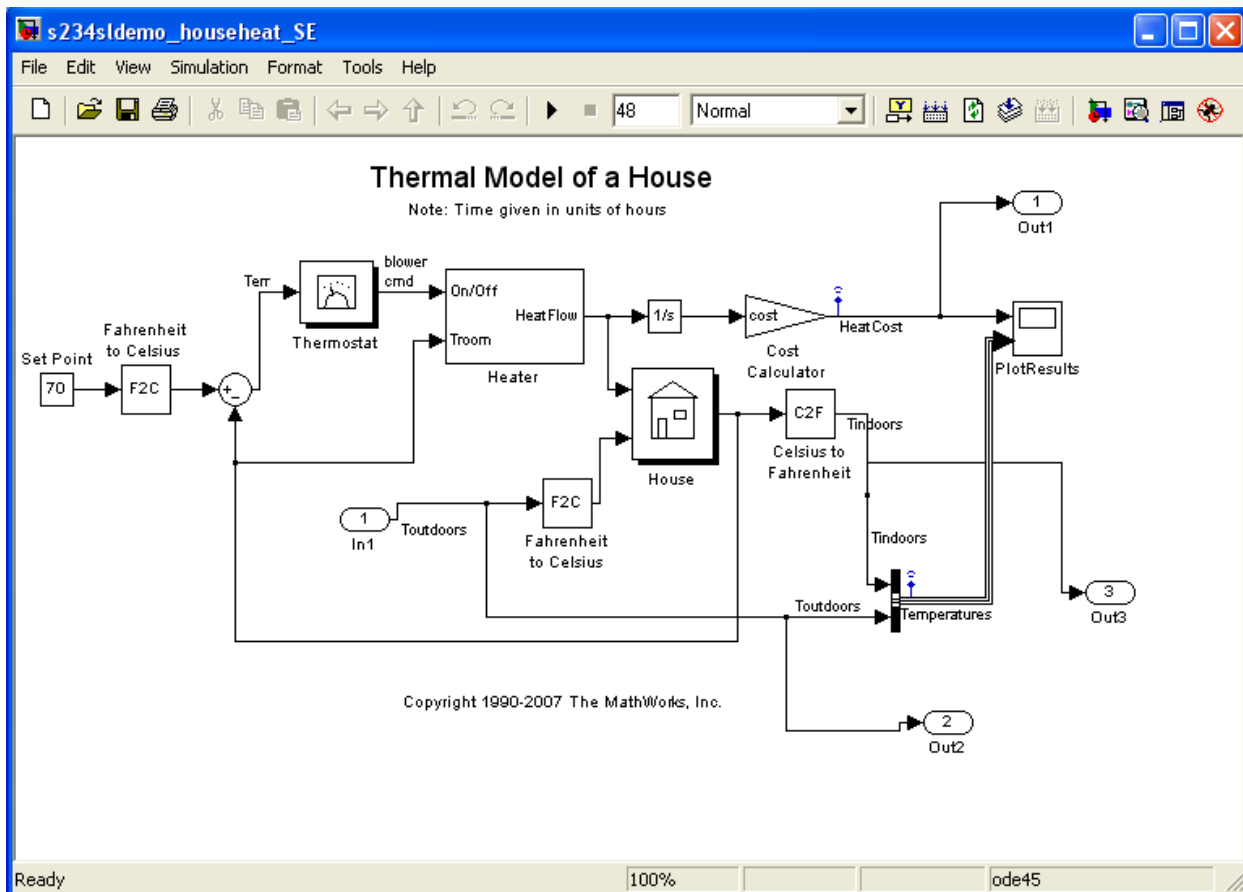


Figure 7.1 Outline of Objective

What the User Will Learn

- Connect to MATLAB functions and scripts through SysML constraint blocks
- Connect to Simulink models through SysML constraint blocks

A large body of simulation models have been developed and validated using MATLAB and Simulink (from The MathWorks, Inc.). Being able to incorporate these existing models into larger SysML models can greatly accelerate system development. ParaMagic allows Simulink/MATLAB

models to be treated as “black box” constraint blocks within MagicDraw parametric models. ParaMagic does not convert SysML models into Simulink models or the reverse.

We assume in this tutorial that the reader is familiar with MATLAB and Simulink and focus completely on the steps necessary to interface existing MATLAB/Simulink elements to SysML parametric diagrams. For further help, see the ParaMagic Users Guide and the user documentation for MagicDraw and MATLAB.

7.2 Step-by Step Tutorial

Step I Create Project

1. Create new project
 - a. Name = HomeHeating
2. Create a package within the project
 - a. RC (Right-click) on Data folder in Containment tree (left column)
 - b. Choose New Element→Package
 - c. Enter Name = HomeHeating

Step II Create Infrastructure

3. Install ParaMagic Profile module.

Step III Create Structural Model

4. Create elements in model
 - a. Inside the HomeHeating package, create blocks HomeHeatingSystem, Home and Outdoors.

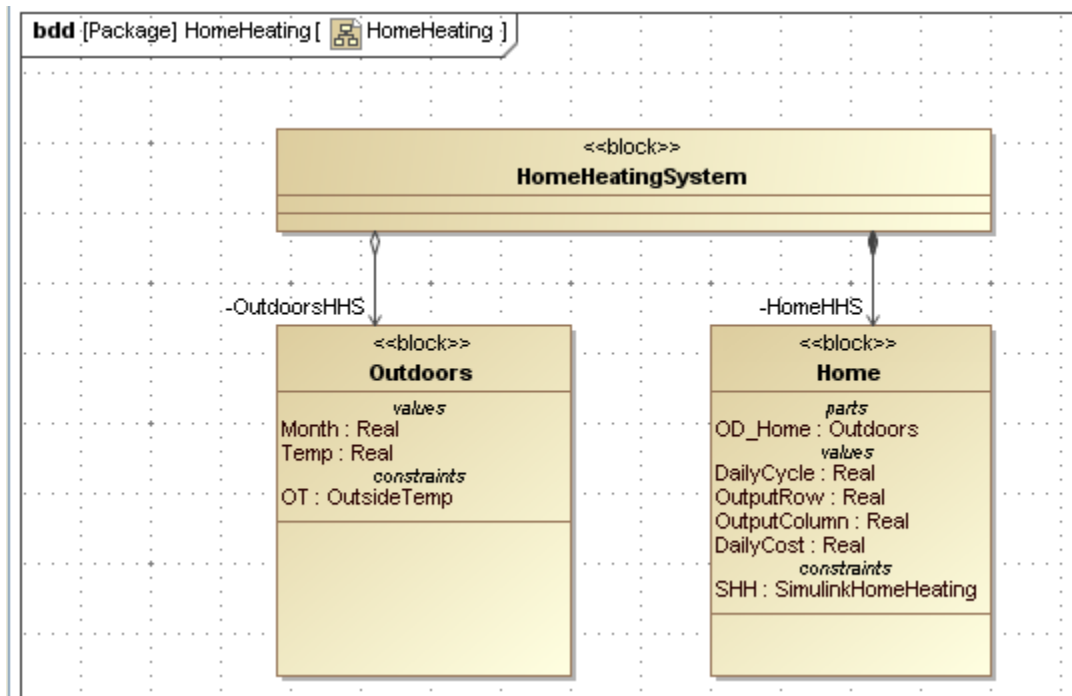


Figure 7.2 Block Definition Diagram for HomeHeating model

- e. Create a block definition diagram, Name = HomeHeating, and drag the blocks into the diagram. See Figure 7.2. Note that we use a Directed Composition connector to link HomeHeatingSystem to Home but reference the Outdoors block with a Directed Aggregation arrow.
- f. Create the Value Properties shown in Figure 7.2. All are Real with a multiplicity of 1.

Step IV Create Constraints

5. Create a constraint block using the MATLAB function annualcycle,

```
function t = annualcycle(m)
% returns t, average temperature, based on m, month from 1 to 12
t = 70 + 20 * sin(2 * pi() / 12 * (m - 4));
```

annualcycle takes a month in the form of a number from 1 to 12 and returns t, the average daily temperature (degrees F) for that month. We assume, for this tutorial, that the function already exists. To use it with ParaMagic,

- a. Add the following final two lines to the function, using the MATLAB editor or some other text editor.

```
save('output.txt','t','-ASCII');
exit
```

These lines cause the value of t to be saved in an ascii file, output.txt, and retrieved by ParaMagic after MATLAB has exited.

- b. Save the complete function as annualcycle.m in the directory set up for MATLAB files in ParaMagic.ini.
- c. Create a constraint block, OutsideTemp, in the HomeHeating package, with two constraint parameters, m and t, and the constraint, $t = \text{xfwExternal}(\text{matlab}, \text{function}, \text{annualcycle}, m)$. The arguments of the function xfwExternal are
 - i. external solver being called, here MATLAB
 - ii. type of element called, function or script, here function
 - iii. the name of the function or script, here annualcycle
 - iv. the input arguments, here m. All input arguments must have multiplicity 1. There may be more than one input argument.
6. Create a constraint block using the MATLAB script demoscRIPTasciisimulink, which launches the Simulink model s234sldemo_househeat_SE.mdl. We assume that the Simulink model and MATLAB script has been previously developed and tested.
 - a. Add the following initial and final lines to the script file

```
inSel= load('input.txt');

o1=inSel(1);
o2=inSel(2);
TempOutside=inSel(3);
Amplitude=inSel(4);
```


(Body of existing script)

```
a=yout(o1,o2);
save('output.txt','a','-ASCII');
exit
```

The initial lines open a text file, input.txt, created by ParaMagic, and extract the four input parameters for use in the Simulink model. The final lines extract a single value for the 720 x 3 element matrix yout created by the Simulink model and save it to another text file to be read by ParaMagic.

- b. Save the modified script as demoscryptasciisimulink.m in the directory set up for MATLAB files in ParaMagic.ini.
- c. Create a constraint block, SimulinkHomeHeating, in the HomeHeating package, with five constraint parameters and the constraint, `cost=xfwExternal(matlab,scriptascii,demoscriptasciisimulink,row,col,outtemp,daycyc)`. The arguments of the function `xfwExternal` are
 - i. the external solver being called, here MATLAB
 - ii. type of element called, function or script, here `scriptascii`
 - iii. the name of the function or script, here `demoscriptasciisimulink`
 - iv. the first input argument, here `row`, the row of the matrix element to be returned (the 360th element is at time 24 hours in this Simulink model)
 - v. the second input argument, here `col`, the column of the matrix element to be returned (the 3rd element is the cumulative cost in this Simulink model).
 - vi. the third input argument, here `outtemp`, the outside temperature.
 - vii. the fourth input argument, here `daycyc`, the amplitude of the daily temperature variation.
 - viii. The value returned is `cost`, the cumulative cost after one day of heating, which corresponds to element (360,3) of the output matrix calculated by the Simulink model.

Step V Create Parametrics Model

7. Create the parametrics diagram shown in Figure 7.3 inside the block Outdoors. The constraint block OY:OutsideTemp contains the MATLAB function `annualcycle`.

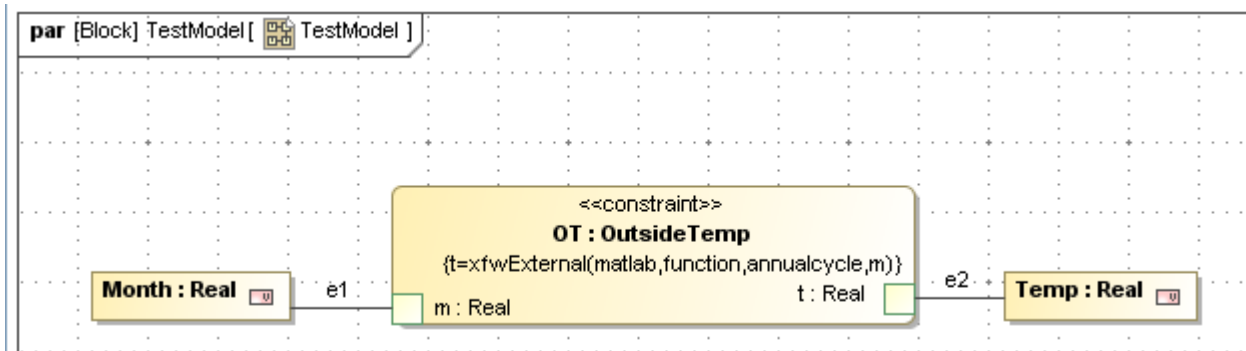


Figure 7.3 Parametric diagram for Outdoors

8. Create the parametric diagram shown in Figure 7.4 inside the block Home. The constraint block contains the MATLAB script *demoscRIPTasciisimulink*, which launches the Simulink model *s234sldemo_househeat_SE.mdl*. Note that the value Temp calculated by the MATLAB function *annualcycle* is one of the inputs to the Simulink model.

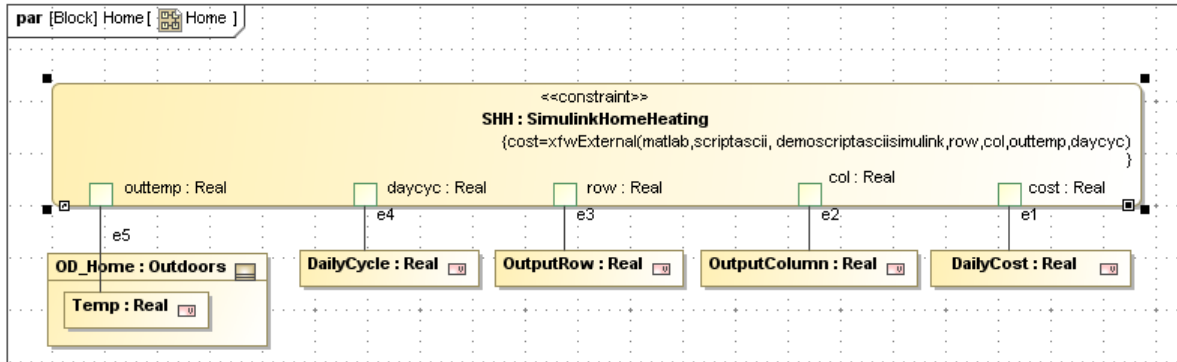


Figure 7.4 Parametric diagram for Home

Step VI Validate Parametrics Model

9. To create the CXS_heading and validate the model schema, follow the instructions in Step VI of previous tutorials

Step VII Create an Instance

10. Create an instance as shown in Figure 7.5

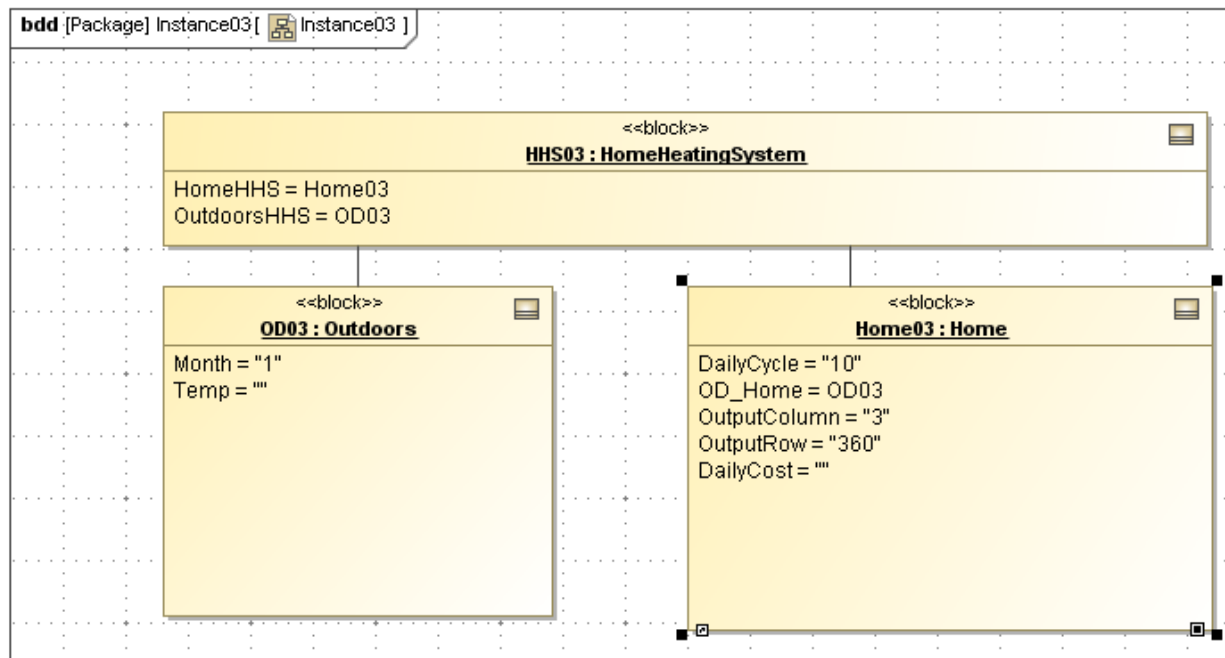


Figure 7.5 Instance of HomeHeatingSystem

- a. Point the Part Property OD_Home inside Home03 to the instance OD03:Outdoors. Note that this assignment is not made automatically in creating the linkages, as in previous examples.
 - i. Open the Instance Specification for Home03 (Figure 7.6)
 - ii. Select Slots
 - iii. Click on OD_Home:Outdoors
 - iv. Click Create Value button

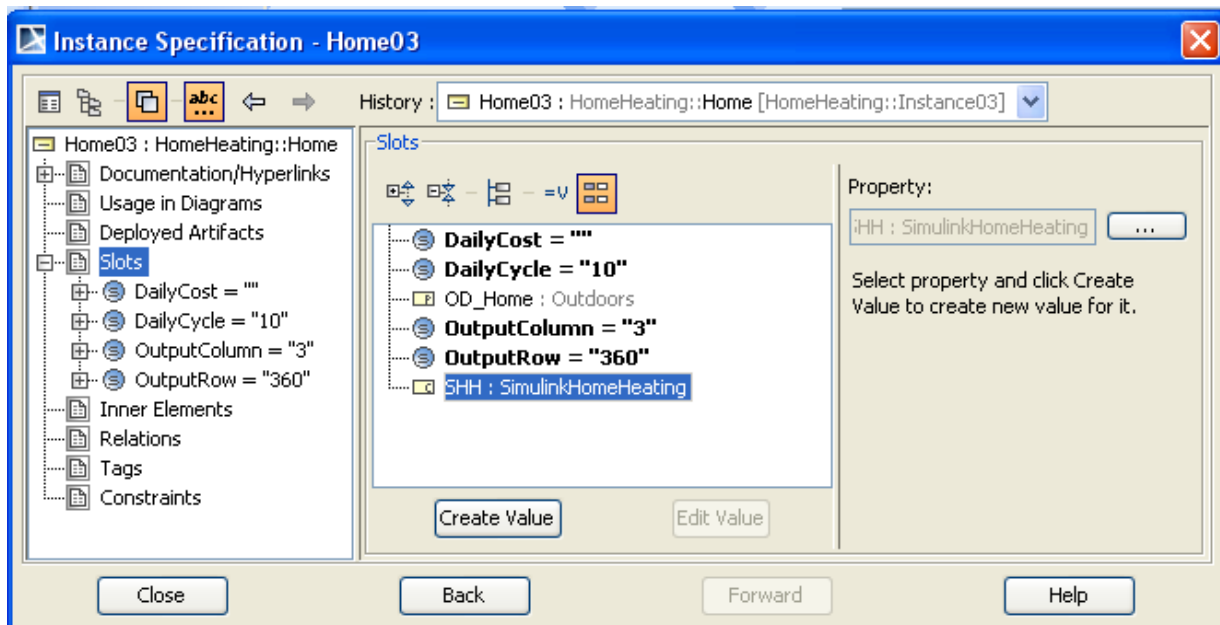


Figure 7.6 Instance Specification for Home03

- v. Select OD03:HomeHeating:Outdoors in the Select Elements window (see Figure 7.7)
- vi. Click Add, OK, and Close

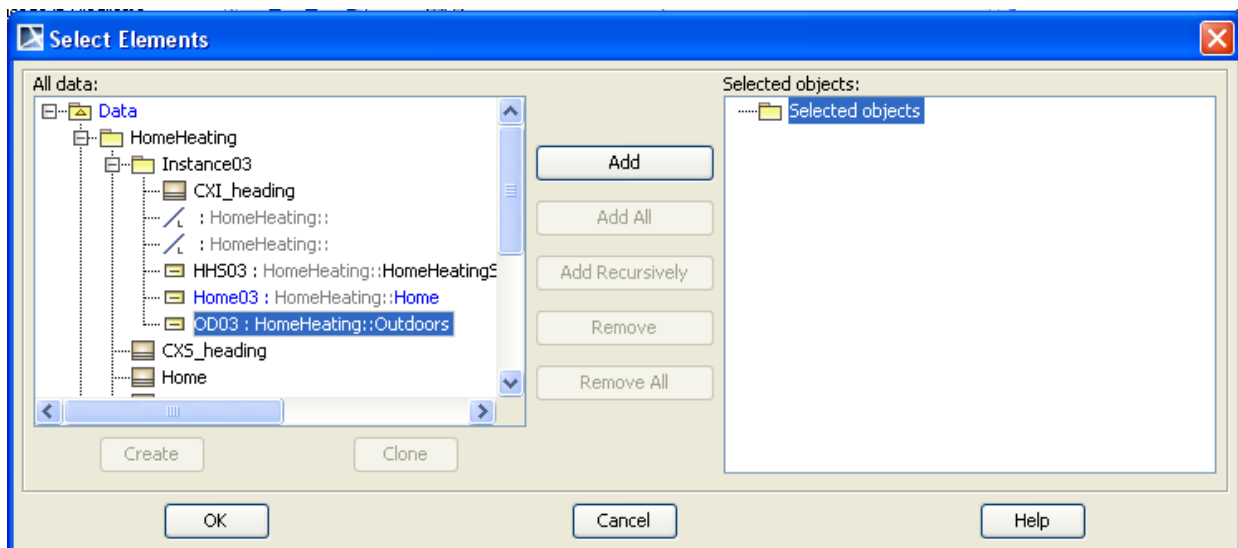


Figure 7.7 Select Elements to link OD03 to OD_Home

- b. Create the links and populate the slots as shown in previous tutorials. The givens are
 - i. Month = 1 (January)
 - ii. DailyCycle = 10 (temperature varies +/- 10 daily)
 - iii. Row = 360 (one day as Simulink model timescale is set)
 - iv. Col = 3 (cumulative cost as Simulink model output array is configured)
- c. Create a CXI_Heading in the instance folder and assign causalities to slots using the Add default causalities utility. Change causality of DailyCost to *target*.

Step VIII Solve the Instance

11. Open the Browser (Figure 7.8) and click Solve.

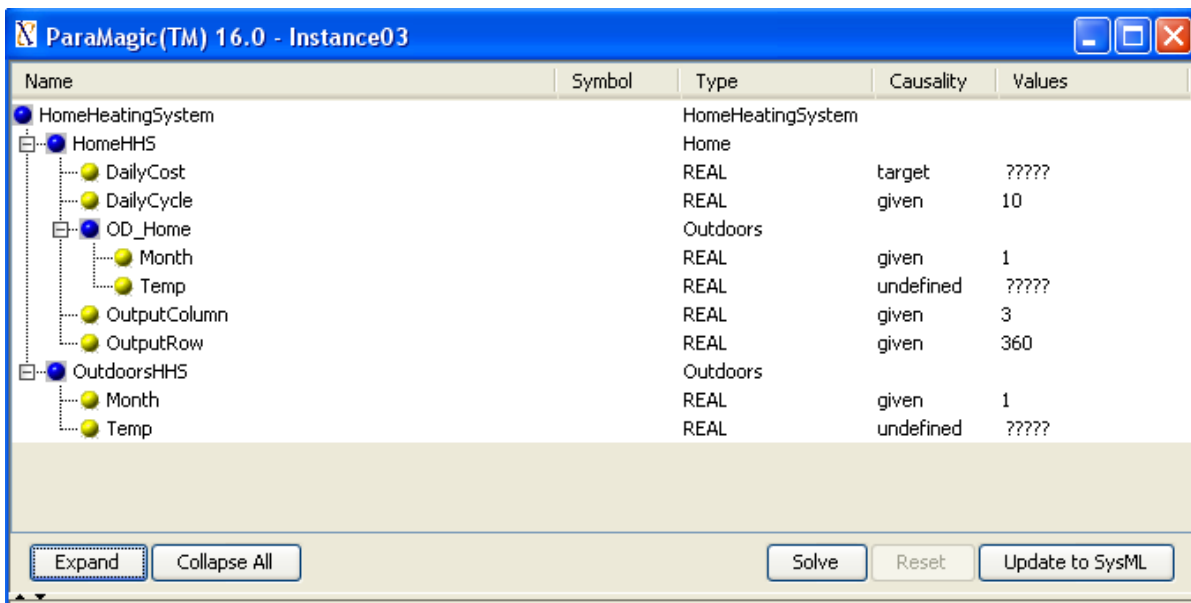


Figure 7.8 Browser before solution

- a. During ParaMagic execution, MATLAB will launch and several figures will appear briefly on your screen, including the Simulink model diagram from Figure 7.1 and the PlotResults graph shown in Figure 7.9. The PlotResults window shows the cumulative cost in the upper window over a two day period. The lower window shows the outdoor temperature (violet) and the indoor temperature (yellow) on the same time scale. The target indoor temperature is a constant, 70 F, set inside the Simulink model (although the model could be modified to treat this as an input from SysML, as well), but the actual indoor temperature varies as the thermostat cycles on and off.

The appearance of these figures is set in the Simulink model, not in the SysML model or in the MATLAB script. These windows will disappear when the the simulation is complete and MATLAB exits.

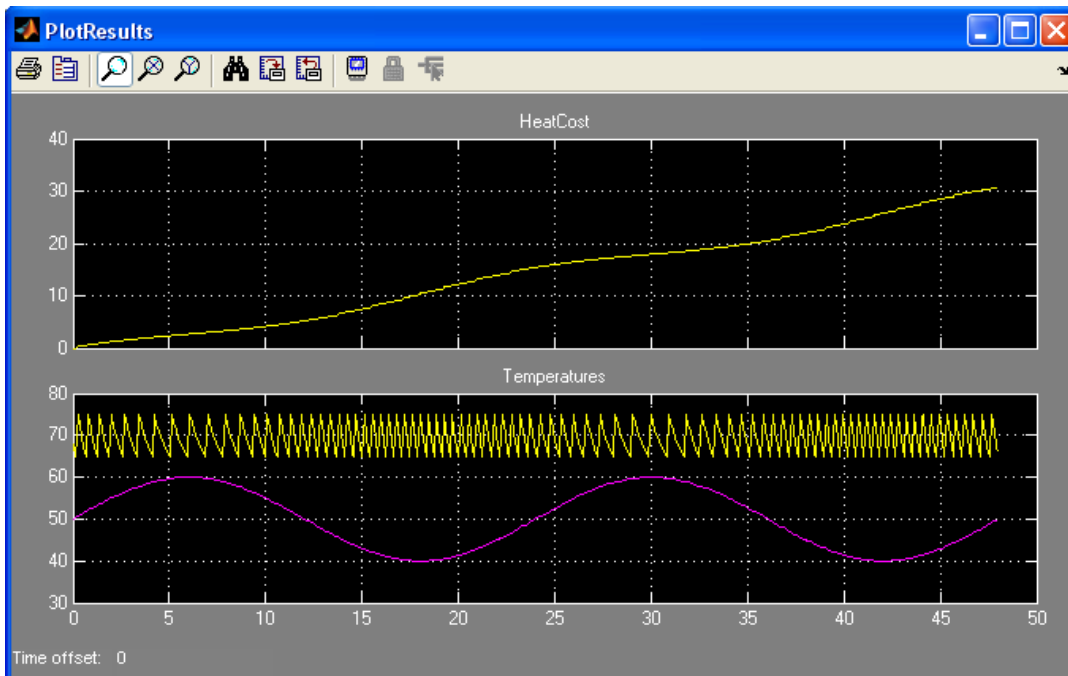


Figure 7.9 Plot Results window during Simulink execution

e. The final browser results are shown in Figure 7.10.

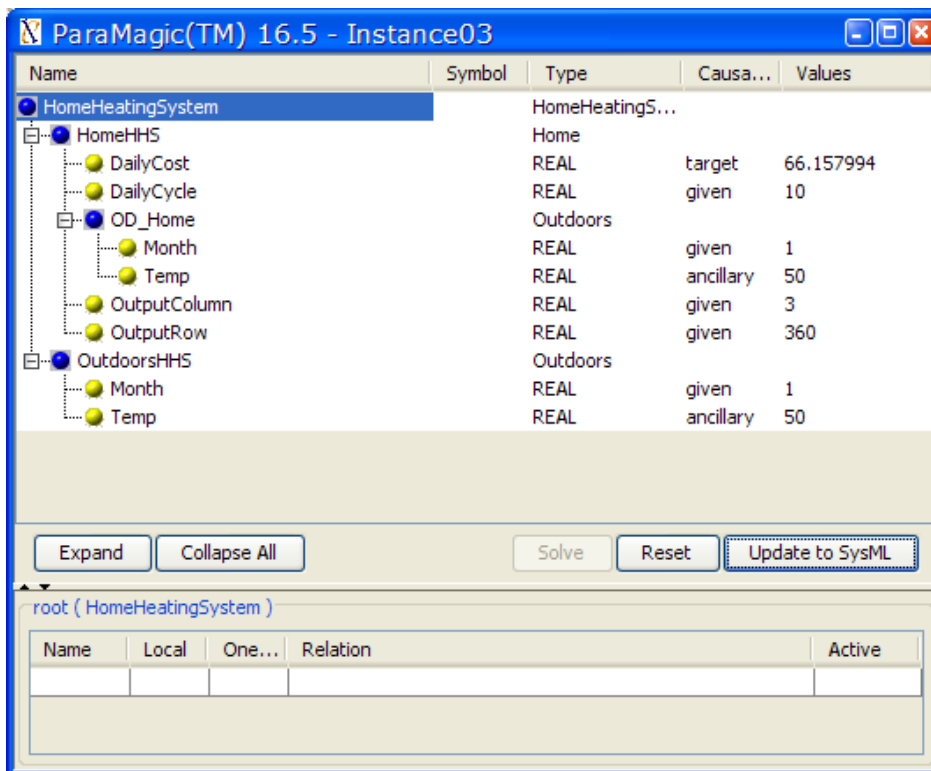


Figure 7.10 Browser after execution.