WALTER SCOTT, JR.
COLLEGE OF ENGINEERING
COLORADO STATE UNIVERSITY

Department of Systems Engineering

CAMEO SYSTEMS MODELER TUTORIAL

Dr. Mike Borky
Professor of Systems Engineering
Colorado State University

1. Getting Started

A. This tutorial applies to the Cameo Systems Modeler Enterprise Edition™ from No Magic, a wholly owned subsidiary of Dassault Systemes®.

B. The license will either be a one-semester Academic License or a purchased license of the SE Department on a floating license server.

NOTE: this tutorial assumes a basic familiarity with the Systems Modeling Language (SysML). The important features of the language are described and illustrated as the discussion proceeds and in Appendix A. Any modeling effort requires a methodology that describes the sequence of model development, the products to be created, etc. This tutorial is based on the Model-Based System Architecture Process (MBSAP).[1] In the text, terms for model content such as Block and Action are capitalized and key words from SysML are in Arial font.

2. Creating a Model

Demo 1

A. Before doing anything, you must create a model. Run Cameo Systems Modeler, create a new project, select the Systems Engineer role, and make certain the SysML Project icon is highlighted (the default model type is UML, which is only for software architecture).

B. Give your project a name, navigate to the desired location to store your model, and check the box to create a model directory. Fig. 1 shows what this looks like.

3. Cameo Systems Modeler Look and Feel

A. Cameo is an extremely complex and capable tool. Almost any modeling action can be done in multiple ways. However, there is a high level of consistency in the user interface, and there are numerous shortcuts that can save large amounts of time in creating model content. This section is intended to help a new user become familiar with the user interface and begin to develop an intuitive sense of how to use it. As a practical matter, most modelers have one or two preferred ways of doing the most common modeling actions and can usually ignore the alternatives.

[1] Borky, JM and Bradley, TH, *Effective Model-Based Systems Engineering*, Springer, 2019.

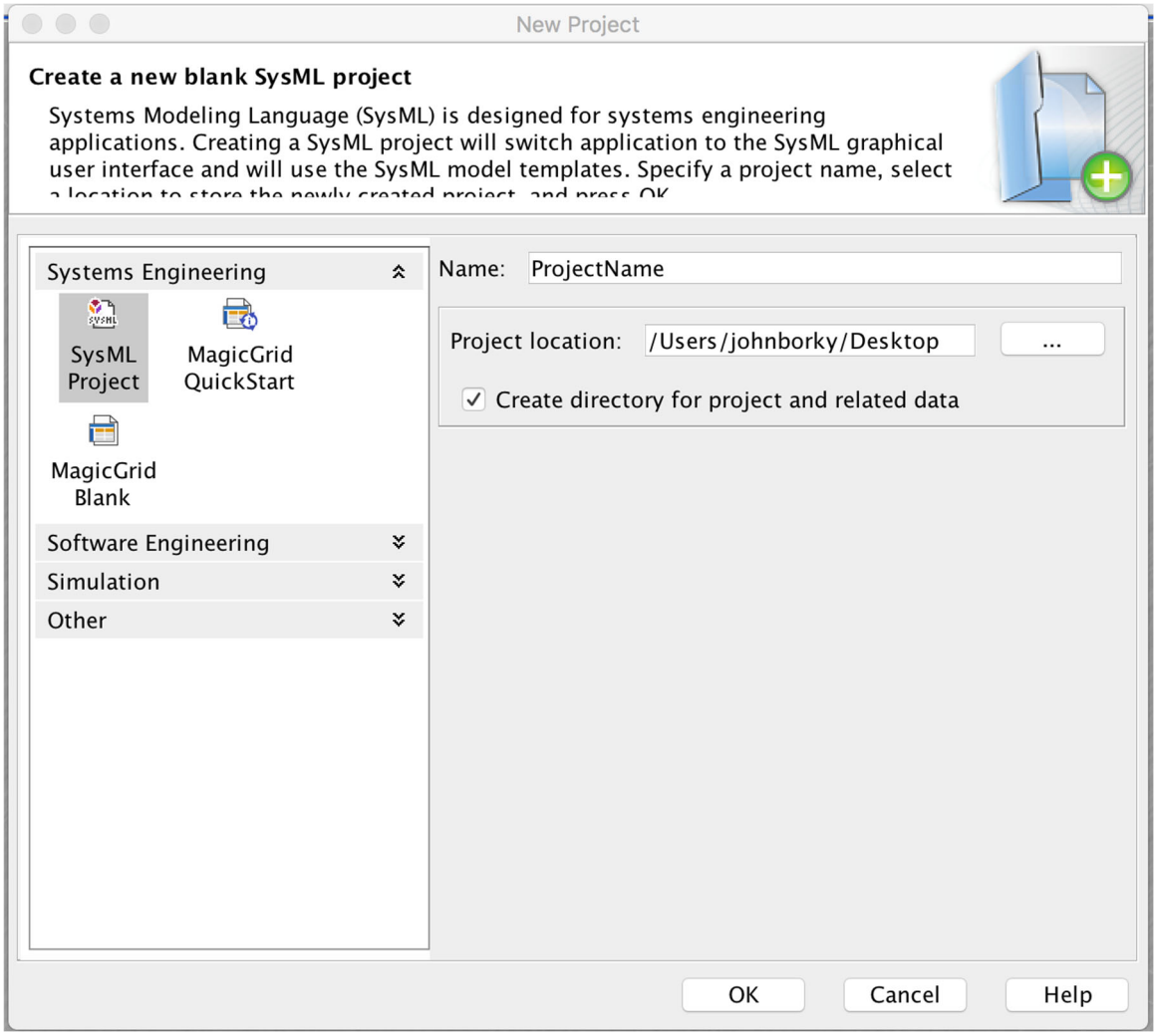


Figure 1. New Project Screen

NOTES:

Cameo is an integrated tool suite built on the MagicDraw™ architecture modeling tool; much of the available documentation has MagicDraw in the title, but the content is valid for Cameo.

Many Cameo windows and menus have multiple "modes:" Standard, Expert, and All; if an expected item doesn't appear in a window or menu, try switching modes.

B. Once a model is open, the screen is partitioned into panes. Fig. 2 is a screenshot from the model of an architecture for a microwave oven that is used as an illustration in this tutorial. The main areas are:

- Main toolbars across the top – manage files, create content, do searches, etc.
- Browser along the left side – basic means of accessing model content.
- Tool Palette next to the Browser – provides the graphical elements to create various diagrams.

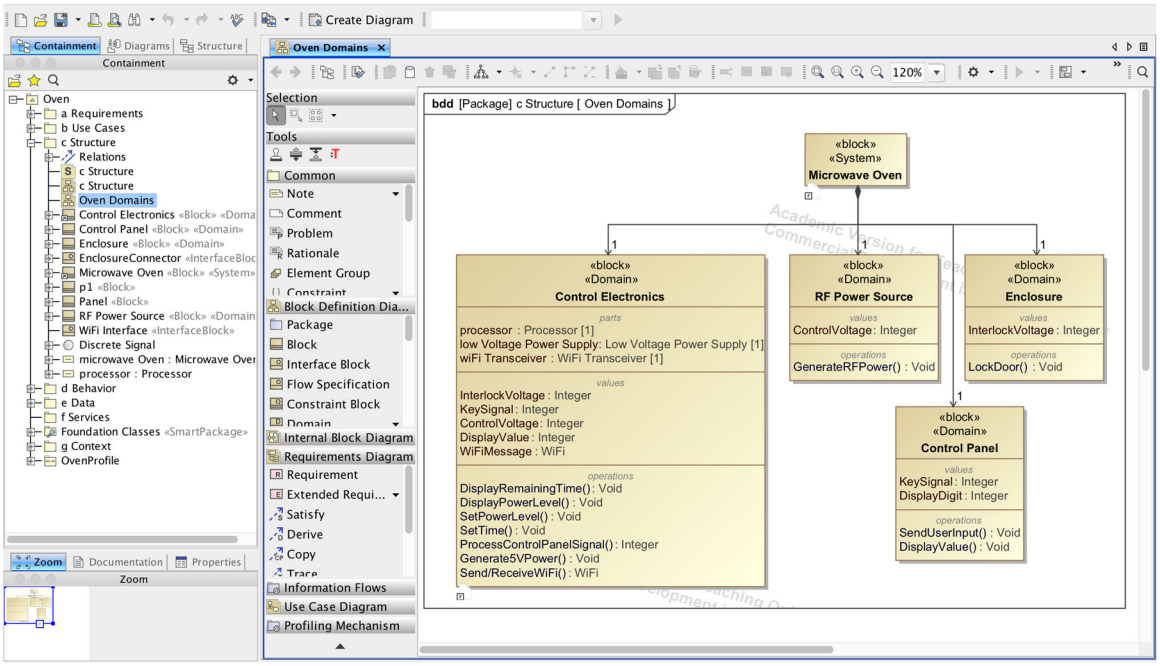


Figure 2. Cameo Desktop.

- Diagram Pane - the work area for diagramming.

All of these are described in more detail as this tutorial proceeds.

C. A few key points about SysML – before getting into the way Cameo implements SysML and allows language-compliant architecture modeling, the following are some fundamentals:

- SysML is a profile of the Unified Modeling Language (UML), which emerged from the Object-Oriented Design (OOD) movement to allow better software engineering. SysML modifies UML in ways that allow System Engineers to perform key tasks like requirements analysis, interface control, performance modeling, etc. However, the basic principles of OOD still apply, and a large part of SysML is identical to UML.
- Cameo does an extremely rigorous job of implementing SysML and enforcing its rules. SysML and Cameo must be mastered together to enjoy the advantages of modern Model-Based Systems Engineering (MBSE).
- SysML can be thought of as a graphical programming language. The basic constructs of a model are Structural and Behavioral Diagrams, and the tool facilitates building these as shown in Fig. 2. Fig. 3 shows the tree of SysML Diagrams.
- The “four pillars” of SysML are Structure, Behavior, Requirements, and Parametrics, with extensive support for creating linkages among model elements in each category. For example, a structural element (called a Block) can be linked to a requirement using a Satisfies

relationship which models the fact that the Block is responsible to satisfy the requirement.

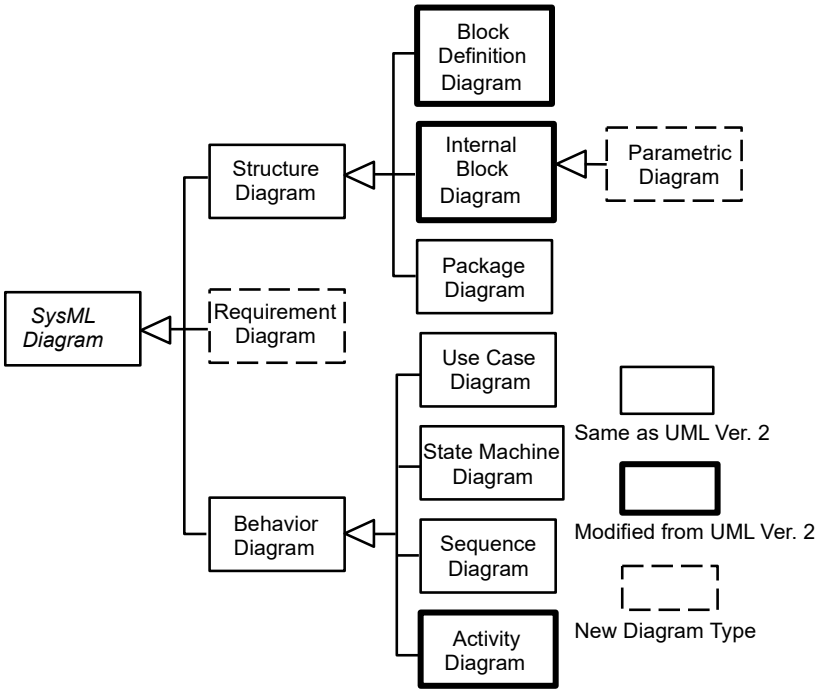


Figure 3. SysML Diagrams with Their Relationships to UML

- SysML elements like Blocks, Parts, Activities, States, Dependencies, Constraints, Guards, and many others give the language its expressive power in modeling, analyzing, communicating, and documenting the structure, behavior, and rules of a complex entity. A Block is the fundamental structural unit of a SysML model; some Blocks can be Parts of other Blocks. Actions, States, Transitions, Messages, and other constructs model various aspects of behavior or function.
- Multiplicity is a UML/SysML construct that accounts for the fact that a model Element, Property, or other entity may be instantiated more than once. The classic example is a car that has a tire as a Part with a multiplicity of four (or five, if you count the spare).
- See Appendix A to this document, Appendices A and B to Borky and Bradley, or the definitive reference by Friedenthal, Moore, and Steiner[2] for more SysML information.

D. Cameo basics - the following are important terms and concepts:
   - Element – a unit of model content, e.g., a Block, Action, or Association.
   - Symbol – the representation of an Element in a diagram:
     - Shape – e.g., the rectangular icon for a Block.

2. Friedenthal S, Moore A, and Steiner R, *A Practical Guide to SysML: the Systems Modeling Language,* 3rd Ed, Morgan Kaufman, 2015.

    - Path – a line, arrow, etc. showing a relationship between shapes.
- Browser – there are upper and lower panes; options for the upper pane include Containment Tree Tab; Hierarchy Tab; Diagrams Tab; Model Extensions Tab; Search Results Tab; and Lock View Tab (shows locked Elements of a server project); of these various ways to display the hierarchy of model content; the two most useful are:
    - Containment Tree - this shows the Packages, Packages within Packages, and Elements within Packages:
        - It can be expanded or compressed with the + and - buttons.
        - It's conventional to give the root package of the structure a name identifying the model.
        - The organization of the model in the Containment Tree is crucial to its usefulness; more on this shortly.
    - Diagrams - sorts all the existing diagrams by type; this is often the fastest way to navigate to a desired diagram.
- The Browser lower pane has a Documentation Tab that displays information about a model Element that has been entered in the model, and a Zoom tab that allows you to zoom a diagram.
- Stereotypes - a stereotype is a special kind of label that can be applied to an Element. The syntax is <<stereotypeName>>. (The double angle brackets are "guillemets.") Liberal use of these (an Element can have more than one) enhances the readability of diagrams by clearly showing important characteristics of an Element.
    - Stereotypes and other language extensions are held in the Containment Tree in a special kind of Package called a Profile.
    - A reasonable number of stereotypes are defined in SysML and provided by the tool; however, it's common to need to define and use additional ones.
    - To create and apply a stereotype:
        - Create a Profile in the Containment Tree by right clicking the root Package, selecting Create Element > Profile,[3] and giving the Profile a name (commonly, just the name of the model).
        - Right click the Profile > Create Element > Stereotype from the list, and give it a name. (You need to be in Standard mode.)
        - Drag a stereotype from the Profile to an Element in a Diagram.
        - You can also apply an existing stereotype to an Element by right clicking the Element, selecting Steotypes from the menu, and selecting the desired stereotypes to apply.

E. Creating model content - the primary options are creating items in the Browser or creating a Diagram and adding content to it:
- To create a new Element in the Containment Tree:
    - Right click on an existing Package or other Element.

[3] The > icon indicates successive menu selections.

- Select Create Element, Create Diagram, or Create Relation.
- Choose the type of content to be created and give it a name.
- The new Element, Diagram, etc. is created inside the selected Element. Very importantly, the containing Element is called the Context for the new content.

- To create a new Diagram:
  - Right click on an Element in the Containment Tree as above and select Create Diagram,

OR

  - Select the owning (Context) Element in the Containment Tree, click the Create Diagram button in the toolbar, and select and name the new diagram.

- To create an Element in a Diagram:
  - Click on the appropriate icon in the Tool Palette for the Diagram, then click in the Diagram frame,

OR

  - Create the Element in the Containment Tree and drag it into the Diagram Frame.

NOTES:

In SysML, it's mandatory for a Diagram to have a Diagram Frame with a header; the tool creates the header automatically. Cameo allows you to turn this off, but that's a bad idea – ensure every diagram has a frame and header.

It's often appropriate to use a given model Element in multiple diagrams; it's *essential* that the Element be created only once, using one of the above methods, then dragged from the Containment Tree to each additional Diagram where it's used.

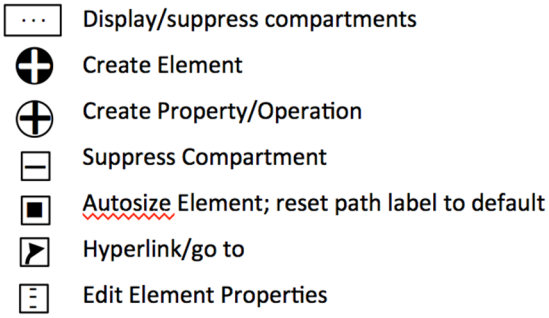
Elements can be dragged to different locations in the Containment Tree.

- To create a Relation between Shapes in a Diagram:
  - Shapes can be connected by Associations, Dependencies, Generalization/Inheritance, and other Relations.
  - It's generally best to create and arrange the Shapes in a diagram, then create the Relations by clicking on an icon in the Tool Palette and dragging from one Shape to the other OR to use the shortcuts of the Smart Manipulator (next topic).

NOTE: it's also possible to create a Relation in the Specification for a Model Element; more on that later.

F. Manipulating Elements in a Diagram:
- Drag things around to get a pleasing arrangement or layout.

- Every Element (Shape or Path) has a Shortcut Menu that gives quick access to key characteristics; right click on the Symbol of an Element to open this menu. The three most useful sub-menus are:
  - Specification – this menu item opens the Element Specification window; this is where the characteristics of the Element are defined and visualized.
  - Symbol Properties – this menu item controls how an Element is displayed in a diagram (colors, fonts, optional display content, etc.)
  - Stereotypes – this menu item opens a list of all the stereotypes in the model and provides an alternative way to apply or remove them.
- Smart Manipulator - this is one of the most powerful productivity features of Cameo; it can save a lot of trips back and forth to the Tool Palette when creating diagrams. There are two parts to a Smart Manipulator, both displayed when a Shape is selected in a Diagram:
  - Buttons: these appear around the frame of a Shape or along a Path:



    - Display/suppress compartments – a SysML Block can show multiple compartments for Parts, – Interfaces, Values, Operations, Constraints, Ports, and many others; this button opens a dialog to select those to display.
    - Create Element – shortcut to create a new Element such as a Property, Operation, or Port within the selected Element.
    - Create Property or Operation – Blocks can have Part Properties, Value Properties, Reference Properties, and Constraint Properties, each shown in its own compartment. It can also have Operations, which are the links that invoke behaviors. This button allows additional Properties or Operations to be created once the corresponding Compartment has been created.
    - Hyperlink – an Element can have a hyperlink to another Element, Diagram, document or other model content; this button invokes that linkage. To create or invoke a hyperlink to an Element, click this button and use the menus.
  - Toolbar - the icons on the Smart Manipulator toolbar provide a fast way to create Relations and other characteristics. The toolbar

is customized to each Element type. For Relations, just click the correct toolbar icon and drag to the Element to which the Relation is being created.

NOTE: well-drawn Diagrams that are easy to follow and understand are essential to good architecture modeling. For example, the Smart Manipulator toolbar for a Path has an icon for rectilinear routing; this lets you drag paths around so they don't cross over shapes. Another useful trick is to press Cntl+W on a Windows machine or Command+W on a Mac, which automatically resizes a diagram to fit in the window. You can then drag content around to minimize wasted white space, place related Elements next to each other, etc.

- The Smart Manipulator toolbar also displays warnings and cautions when something is wrong. Common mistakes are to create a behavior such as an Action that isn't defined anywhere or to put something in a diagram in a place that doesn't match its ownership (context) in the Containment Tree. Often, the toolbar gives you options to automatically correct these errors.
- Refactoring - on an Element Shortcut Menu, the Refactor option allows you to change the Element to something else, generally to convert a generic Element like a Block to something more specific like a Signal. For directional Relations, refactoring includes being able to reverse the direction of an arrow.

G. General Tool Settings – most default settings in the tool will work fine; some things to consider setting or changing include:
   - If the tool runs slowly, consider closing unused windows and increasing allocated memory.
   - It's a good idea to set Auto Recovery – click Options on the top toolbar > Environment > General > Save/Load > enter idle time in minutes.
   - Other project options can also be preset – click Options > Project, and work through the specification menu to select things like fonts and diagram information to be displayed.
   - Generic Numbering – allows customized numbering formats for model elements; to do this:
     - Click Options > Project > General Project Options > Use Element Auto-Numbering > OK.

       OR

     - Options > Project > Symbol Styles > Default > Element Numbering Display Mode <Property>.
     - You can always override in a Diagram with manual numbering and formatting.

H. Search – as a model grows, it can be hard to find specific content; the Search function provides a variety of ways to find things:
   - You can search on a text string, Element type, Property/Property Value, Stereotype, etc.; the search runs as a background task, so work in the model can continue, but it's generally almost instantaneous.
   - Quick Search – Cntl+Alt+F OR click Search button (binoculars icon) on the main toolbar > Quick Find (there are multiple options, which are spelled out in the MagicDraw User Guide).
   - Find Dialog – Click Edit > Find OR Cntl+F OR right click Browser Package > Find; then enter the following:
     - What: type a search phrase or select from the pull down list for Element names or textual properties; you can choose Search in Names or Search All Texts (includes documentation); use the text box shortcut menu to choose options. You can use * and ? wildcards (you must choose Match Exactly for these).
     - Type: select an Element type or leave as <any>.
     - Scope: click the browse button (. . .) and select a Package to be searched.
     - Properties: click the browse button, select Properties and enter Values (you must expand the Options list).
     - Options: click the button to open and select options.
   - Find and Replace Dialog – Edit > Find and Replace OR Cntl+R OR click the Search button on the toolbar > Find and Replace.

I. Favorites - these are used to mark frequently used Elements for easy access; they can be defined by individual modelers who are members of a team:
   - In the Browser, click on an Element > click  at the top of the Containment Tree > Add Selected to Favorites.
   - To manage Favorites:
     - In the Browser, click  at the top of the Containment Tree > Manage Favorites.
     - Add/Remove/Reorder > Add Selected to Favorites OR Select Element, click + or - to add or delete.
     - To move an Element up or down in the list, select the Element and click Up or Down.

J. Some additional useful Cameo tips and tricks are collected in Appendix B.

4. Getting Organized Demo 2

A. Early attention to the structure of a model will both save work later in the modeling effort and make the model easier to navigate and use. Many alternative model organizations have been tried; the approach in this tutorial, which is the core of MBSAP, has consistently yielded good results. MBSAP uses the concept of Viewpoints and Perspectives to organize architecture data in a model as shown in Table 1.

B. When translated into a Browser, this structure organizes the model on the basis of the basic dimensions of a system architecture:
   - Requirements – captured in the model and linked to system components and behaviors.
   - Structure – the way the system is organized, including decomposition of higher-level components into lower level ones, as well as internal and external interfaces.
   - Behavior – the functions exhibited by individual parts of the system and by the system as a whole; this includes Use Cases and Actors.
   - Data – the information (and sometimes other) content of the system.
   - Services – crucial in service-oriented architecture or whenever a services mechanism is employed.
   - Context – any additional (non-semantic) aspects of the system and its architecture, which can include product data, constraints, standards, legal requirements, policies, history, and many others.

   The model structure shown in Fig. 4 is created by right clicking the root Package in the Containment Tree, selecting Create Element > Package, naming the new Package, and repeating to complete the structure. This is a Browser screen shot from the microwave oven example.

   - Details will vary with individual architectures; e.g., there may not be a need for a Services Package if a system doesn't employ services.
   - The Browser in Fig. 4 includes a Profile named OvenProfile to hold stereotypes defined for this system. Dedicated Packages (or, if you prefer, subdirectories) for Requirements, Use Cases, and Data are very helpful in creating an effective model structure.
   - Any model Element must be created only once, placed in the appropriate location in the Containment Tree, and reused from that location as necessary. Generally, Cameo will put things in the right places, but it's occasionally necessary to drag content around in the Containment Tree.

Table 1. MBSAP Viewpoints and Perspectives

Perspective	*Content*	*Model Artifacts*
Operational Viewpoint		
• Structural Perspective	Top level partitioning	Packages/Blocks (Domains)
• Behavioral Perspective	Primary system functions	Use Cases, Activity Diagrams
• Data Perspective	Overall information categories	Conceptual Data Model
• Services Perspective	Domain services	Services taxonomy, Domain specifications
• Contextual Perspective	Supporting information	Documents, graphics, etc.
Logical/Functional Viewpoint		
• Structural Perspective	Functional components/ interfaces; specifications	Block Diagrams
• Behavioral Perspective	Class & Class Collaboration functions	State Machine Diagrams, Sequence Diagrams
• Data Perspective	System information entities; XML schemas	Logical Data Model
• Services Perspective	Specific services (functional)	Services taxonomy, specifications, Sequence Diagrams
• Contextual Perspective	Supporting information	Documents, graphics, etc.
Physical Viewpoint		
• Design Perspective	Hardware/software, component specifications	Class/Block Diagrams
• Standards Perspective	Initial standards profile and forecast	Standards tables; model element specifications
• Data Perspective	Database tables	Physical Data Model
• Services Perspective	Physical service definitions	Service Level Agreements; Service specifications
• Contextual Perspective	Supporting information	Documents, graphics, etc.

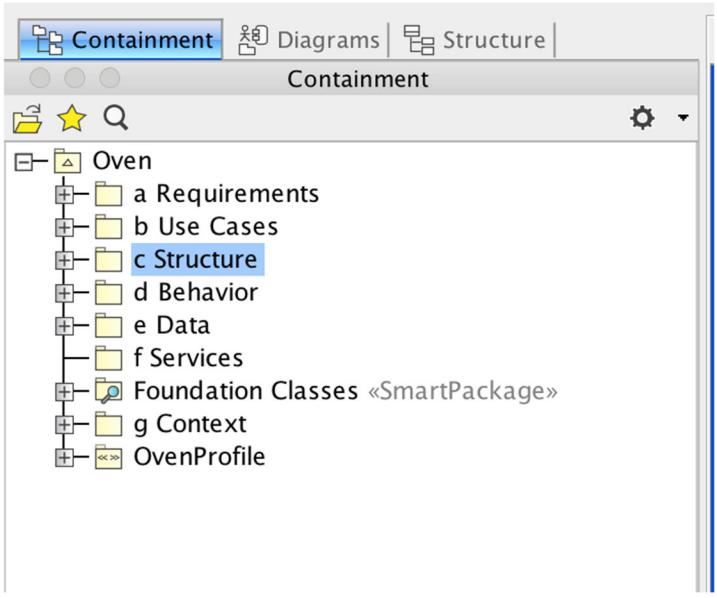


Figure 4. Model Organization for the Microwave Oven Example.

- As a general principle, you should put detailed content inside the correct owning Element (Context) in the Browser; there are several ways to do this:

- Create the new content in the right context in the Containment Tree by selecting the containing Element (context) and using Create Element, Create Diagram, etc.
- Drag the content to the correct place if it's initially elsewhere in the Browser.
- Examples of creating a good hierarchical model structure include:
  - Put a State Machine that defines a Block's behavior under the Block.
  - Put an IBD under the Block whose structure it describes.
  - Put a Scenario Activity Diagram under the Use Case whose scenario it models.

C. Smart Packages - Cameo provides a very useful tool for sorting and grouping model content called a Smart Package. These collect related model Elements and can be used to create catalogs, configuration management files, and for other purposes. A Smart Package can be:

- Manual:
  - Contents are added/deleted manually; right click an Element in the Browser > Create Element > Smart Package > give the Package a name > drag items in the Containment Tree to the new Package.
  - Each added Element becomes an Additional Elements Property Value of the Package.
  - To remove Element, right click it > Delete from Contents.

- Automatic or dynamic:
  - Contents are added and removed by the tool in accordance with a set of specified criteria; these dynamic contents are updated automatically as the model evolves.
  - To create a Dynamic Smart Package, right click a Context > Create Element > Smart Package > give the Package a name > open the Smart Package Specification window > Contents > Query > define criteria > close Spec window > check Package contents for correctness.
  - Properties of a Dynamic Smart Package include Query (search parameters used to create the Package), Additional Elements (manually added), and Excluded Elements (manually removed).
  - To convert Dynamic to Static content: right click the Package > Freeze Contents.

- A very useful way to create a Dynamic Smart Package is based on an Applied Stereotype:
  - Ensure all elements in the category have the appropriate Applied Stereotype.
  - Perform a search with the following parameters:
    - What: * (the wild card symbol).
    - Type: Block (or other Element type).
    - Scope: Whole Project (or lower level Browser Package).

    - Properties: Under General, check Applied Stereotype and select the Stereotype name in the Value Cell.
    - In the Search Results Toolbar, click the Save Search Results button.
    - Select Save Query for a Dynamic Smart Package or Save Results for a Static Smart Package.
    - Open the Package and verify the contents.
  - For example, you could give a structural stereotype like <<subsystem>> to all the Blocks at this level of the system structural hierarchy, then create a Smart Package to group them for easy access and listing. **Demo 3**

D. Element Specifications - every Element has a specification where its characteristics are captured, and where most can be created or changed. The specification is complex and unique to each Element type because it faithfully implements the full detail of SysML. Fig. 5 shows a Block Specification for the microwave oven example.

- The specification has its own mini-browser, with the content laid out hierarchically on the left and various kinds of content displayed in panes on the right. These panes consist of lines with the name of a characteristic on the left and a "Value Field" on the right showing how a characteristic is defined or quantified. Note the field in the upper right corner where you can select Standard, Expert, or All, depending on how much detail you need to see.
- The tool fills in much of this detail automatically as an Element is defined. Some of it is associated with code generation from the model and not needed for general architecture modeling.
- The Properties, Operations, and Inner Elements panes have been expanded to show how these characteristics are displayed.
- The Documentation pane is a free-text field in which anything can be entered; this is a perfect place to put a description of the Element and any specification content that isn't captured elsewhere.
- In some cases, the specification is the most convenient place to create content; an example is to use the Relations pane of a Block Specification to create a <<satisfies>> Dependency linking the structural element to a requirement.
- If the Element has a Classifier Behavior, i.e., a Behavior linked to the Element to define its primary behavior, this would be identified in the Classifier Behavior field of the specification.

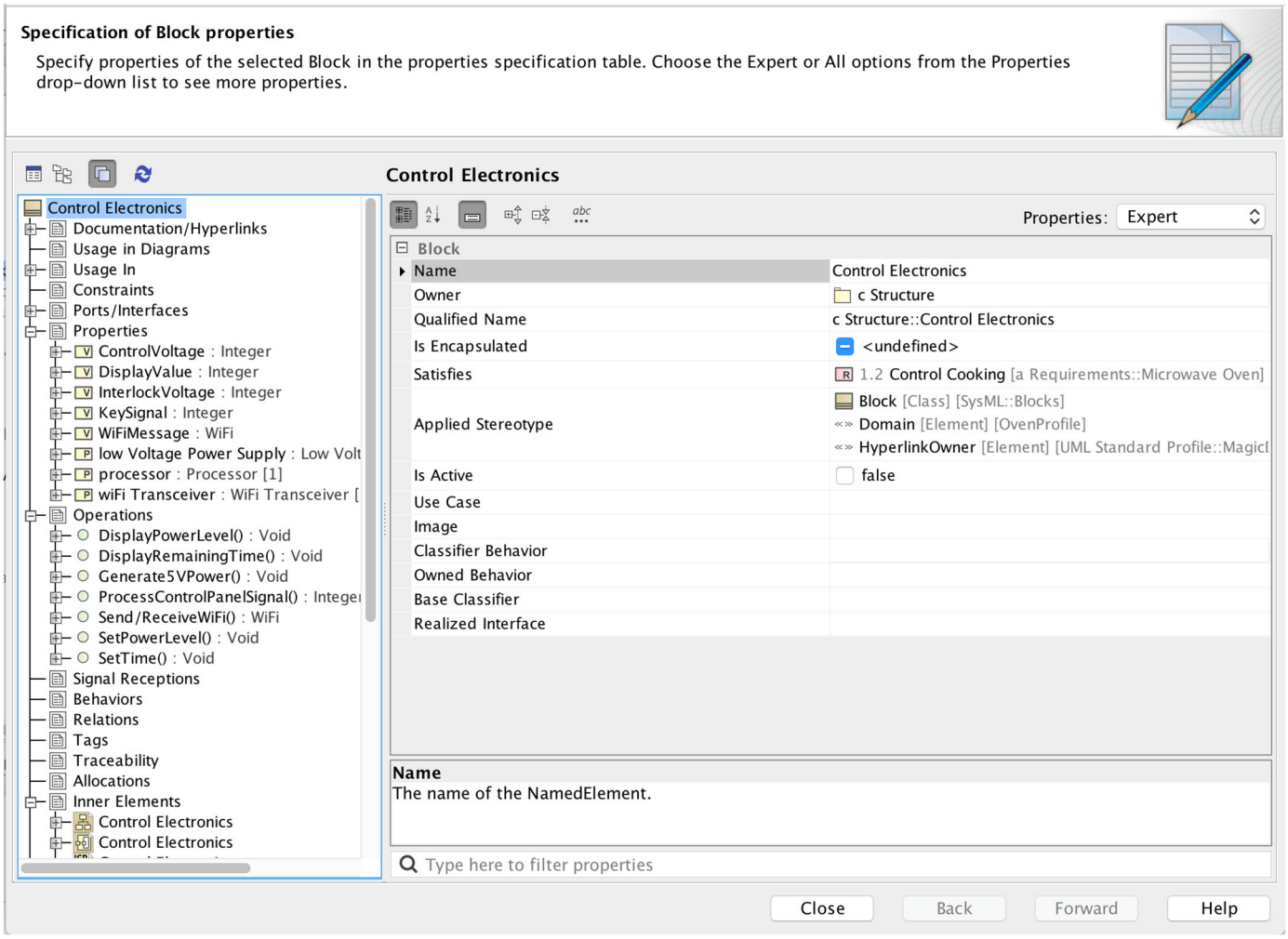


Figure 5. Block Specification for the Microwave Oven Control Electronics

E. Creating Diagrams:

- Decide on and select the correct context (owning Package or other Element) in the Browser.
- Create Diagram > select diagram type > enter diagram name.
- Create content from the Tool Palette or drag Elements from the Containment Tree.
- Use the Smart Manipulator to “wire up” the diagram, set line style, etc.
- Use the Symbol Properties submenu for an Element to define colors and other visualization features.
- Use good drawing technique to achieve a pleasing layout – minimize unneeded white space, avoid Paths crossing Elements (rectilinear style is usually best), etc.

F. Export Model Content:

- Drag to select a diagram frame and content > copy as JPEG > paste into a document.
- Copy and paste text.

5. *The Microwave Oven Example*

A. This fairly trivial architecture has enough complexity to illustrate the use of Cameo without drowning a student in details. The details are entirely invented - this isn't a real kitchen appliance. The oven is assumed to have a cooking chamber fed by a radio frequency power source, an electronics assembly that controls the operation, and a control panel for entering cooking parameters and displaying cooking data.

B. This tutorial will show the creation of the model, but first a bit more detailed look into its structure. Fig. 6 illustrates the principles of placing content correctly in the Containment Tree. The figure shows part of the expansion of the Structure Package; successive levels of indenture show the placement of Elements in their context. Entries in the Browser have icons that identify the Element types; these will become clear as the

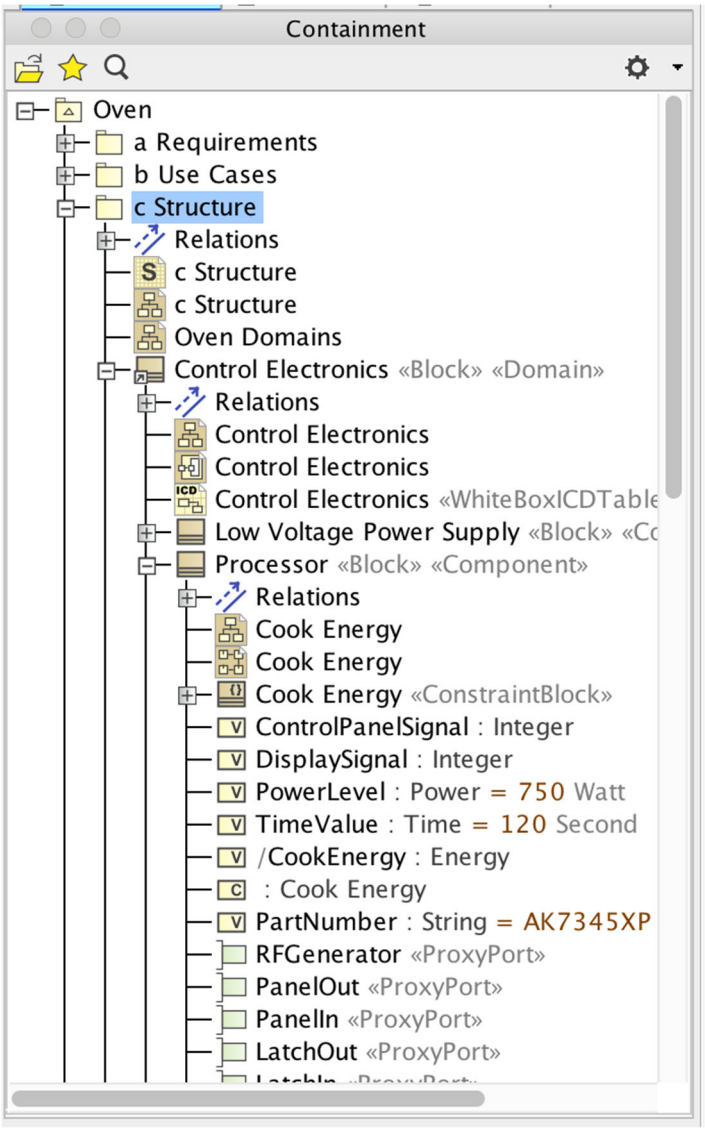


Figure 6. Microwave Oven Example Expanded to Show Containment Tree Organization

model is developed. For example, a simple block diagram graphic denotes a SysML Block Definition Diagram. A rectangle with horizontal lines is a Block, a rectangle with a V is a Value Property, and so forth. As Diagrams are constructed, Cameo builds this detail in the Containment Tree, but, again, the structure can be changed manually by dragging Elements into other contexts.

C. At this point, we can open the Microwave Oven model and explore the Containment Tree more fully. Some organizational details depend on the preference of the modeler; e.g., this model has a Behavior Package holding diagrams for primary or high-level functions that involve multiple parts of the system. Some modelers prefer to put all behavior diagrams in a structural context (under a Block) even if that Block models the entire system.

6. *Modeling Structure with Blocks* **Demo 4**

A. Which dimension of system architecture to work on first is a matter of individual modeler choice and the nature of the system. Many System Engineers prefer to start with requirements, others with primary system behaviors or functions, still others with structure, and, for data-intensive systems, still others start with data modeling. In any event, all these categories of architecture content proceed in parallel with constant interaction as decisions in one part of the model determine or constrain content in other parts. In this tutorial, we'll look first at structure in terms of Block Diagrams, then at Requirements, since a Requirement Diagram is a specialized Block Diagram, and finally at a Data Model which is built with Block Diagrams. The basic principles behind architecture modeling are spelled out in Borky and Bradley. Here, the focus is on creating diagrams and other content in the Cameo tool suite.

B. Block Definition Diagram (BDD)[4] - any unit of structure in SysML is a Block. In general, Blocks are defined in BDDs, including structural decomposition, and given characteristics such as Value Properties and Operations as these are defined. We'll show the procedure by creating the BDD in Fig. 7, which shows the top-level decomposition of the system into four Domains (MBSAP uses Domain for the initial partitioning of a system; some modelers prefer Subsystem or some other term). The steps to create the Diagram are:

- Right click the Structure Package in the Containment Tree > Create Diagram > Block Definition Diagram > enter Oven Domains as the name.
- In the Tool Palette, click/drag a Block into the Diagram; give it the name Microwave Oven.
- Do this four more times for Control Electronics, RF Power Source, Enclosure and Control Panel.

[4] For some reason, the inventors of SysML opted for lower case acronyms. This tutorial conforms to the universal engineering (and literary) practice of writing acronyms in upper case.

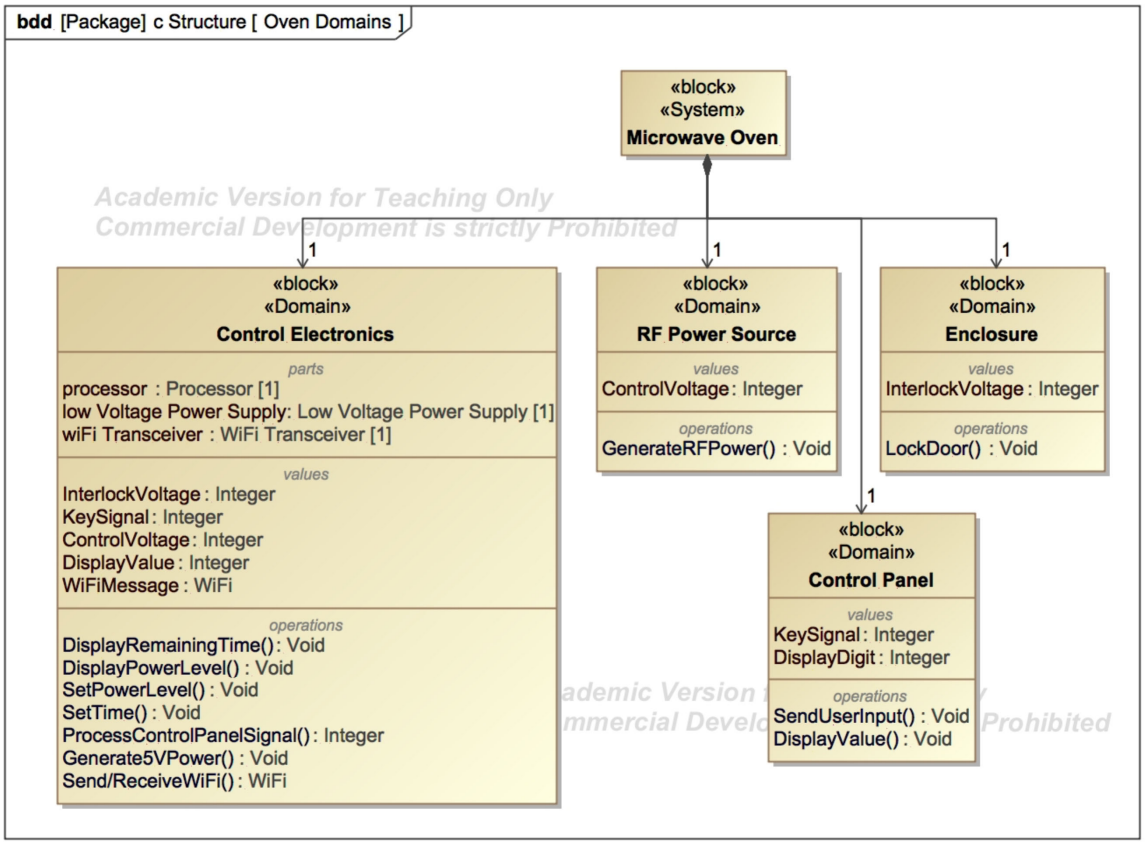


Figure 7. Block Definition Diagram

- Right click the Oven Profile Package in the Containment Tree > Create Element > Stereotype > give it the name System. Do this again for Domain.
- Drag the System stereotype to the Microwave Oven Block and the Domain stereotype to the other Blocks (the tool gives you the Block stereotype automatically, although you can delete it).
- Click the Microwave Oven Block and in the Smart Manipulator toolbar, select the Directed Composition icon (solid black diamond head with a plain arrow shaft) and drag to the Control Electronics Block. Be sure a blue outline appears to show you've got the right Shape. When you release the mouse, a Composition Relationship is drawn between the two blocks. Drag the line segments to get the desired geometry.
- Do this three more times so that all four of the Domain Blocks are defined as Part Properties of Microwave Oven. You can delete the Role labels on these lines - they really just clutter the Diagram. Note that once Part Properties are created, these structural entities appear in the Browser twice: first as Blocks, because that's how all structural units are defined, and second as Parts under the Block that contains them. This isn't double definition - Block definition and Block usage as Parts are distinct aspects.

- It's a very good habit to assign Multiplicities as relationships are created. To do this, open the Specification for each of the Composition arrows in turn to open their Specifications. In the Specification window for each, go down to the Multiplicity property, click on the Unspecified value in the Value Field, and select 1 from the drop down list. You can also do this in the Specification window for the Microwave Oven Block by expanding the Properties category, clicking on each Part Property, and selecting 1 for the Multiplicity. Of course, sometimes the correct Multiplicity is an integer greater than 1, an unspecified number (*), or a range of values (0..5, 2..4, 3..*) etc.
- Now we need to create some initial Value Properties (Values) and Operations for the Domains. This is important to clearly show the nature and responsibilities of each, recognizing that they may be refined as the design matures. This content can be added in the Specifications of the four Domain Blocks, but it's probably simpler to do it directly in the Diagram.
  - Select a Block, and in the upper right corner, click the Create Element icon (white cross in a black circle). From the drop down list, select Value Property. A Values compartment appears in the Block with a Value that has a default name like "value1." Overwrite this with the real name of the Value Property.
  - Next, it's a good idea to give the Value a type. Values are defined using Value Types that incorporate Quantity Kind and Unit characteristics. Many of these are defined in SysML and available in Cameo. When you need additional ones, by convention you create them in the Data Package via the following steps:
    - Right click the Data Package > Create Element > Quantity Kind > name.
    - Use the same procedure to create a Unit.
    - To define a Value Type (Data Type) - right click the Data Package > Create Element > Value Type > enter a name > right click and open the Value Type Spec window > click in the Quantity Kind and Unit fields and navigate to the correct items.
    - You can now apply the Value Type to a Value Property, Operation, etc. by selecting the Type field in the Element Specification and navigating to the Value Type.
  - Right click the new Value and open its Specification. Go down to the Type property and click in the Value Field to get a (long!) list of predefined Value Types. If the one you want is in the list, select it. If not, you need to create it. This simply means right clicking the Data Package in the Containment Tree, selecting Create Element > Value Type > enter the name of the new Value Type. Now go back to the BDD, and you can select the new Value Type for the Value you just created.

- For additional Values, you can simply click the Create Property/Operation icon (black cross in a white circle) on the bottom right of the Values compartment to get additional Values, and you can give them types the same way as the first.
- To add Operations, click the Create Element icon again and select. Give this its correct name, and if it's important to define a return type, you can do that the same way you gave types to your Values.

- Another good modeling practice at this point is to go to the Block Specification, open the Documentation pane, and type in any useful information about the Block. This might include the owner or change authority, safety or security concerns, relevant legacy products, etc. A brief rationale for the Block's nature and usage may be helpful later.
- Ports are discussed under the next topic, Internal Block Diagrams (IBDs), but they can be declared on Blocks in BDDs as well.
- Although not illustrated in Fig. 7, Generalization/Inheritance relationships are very common in BDDs. These will be illustrated in a later Diagram.
- System structure is decomposed, initially with successive BDDs, until the necessary level of detail to support implementation is reached. Fig. 8 shows the Control Electronics Domain decomposed into its Parts. This diagram is created exactly like Fig. 7, starting by dragging the Control Electronics Block in at the top, then creating the new, lower-level Blocks, and giving them Values and Operations.

C. Internal Block Diagrams (IBDs):
- IBDs are derived from the UML Composite Structure Diagram, and they complement BDDs. BDDs define *structure:* Blocks, decomposition, Generalization/Inheritance relationships, etc. IBDs define *interactions:* how the Parts that make up a higher level Block work together to achieve that Block's required capabilities. It's almost always very important to create this content in the right sequence: first define Blocks and their relationships in BDDs, then use the Blocks as Parts in IBDs and in other ways in the model.
- We'll illustrate creating an IBD with the simple example in Fig. 9. This is the partner to Fig. 7 and shows how the Domains of the microwave oven interact. The frame of the IBD corresponds to the boundary of the Block whose internal structure is being modeled, in this case the Microwave Oven Block. The steps to create the diagram are:
  - In the Containment Tree, right click the Microwave Oven Block > Create Diagram > Internal Block Diagram. A blank Diagram appears with the name of the Block. In this example, some of the Values of the Processor Block have been given default values using their Specification windows.

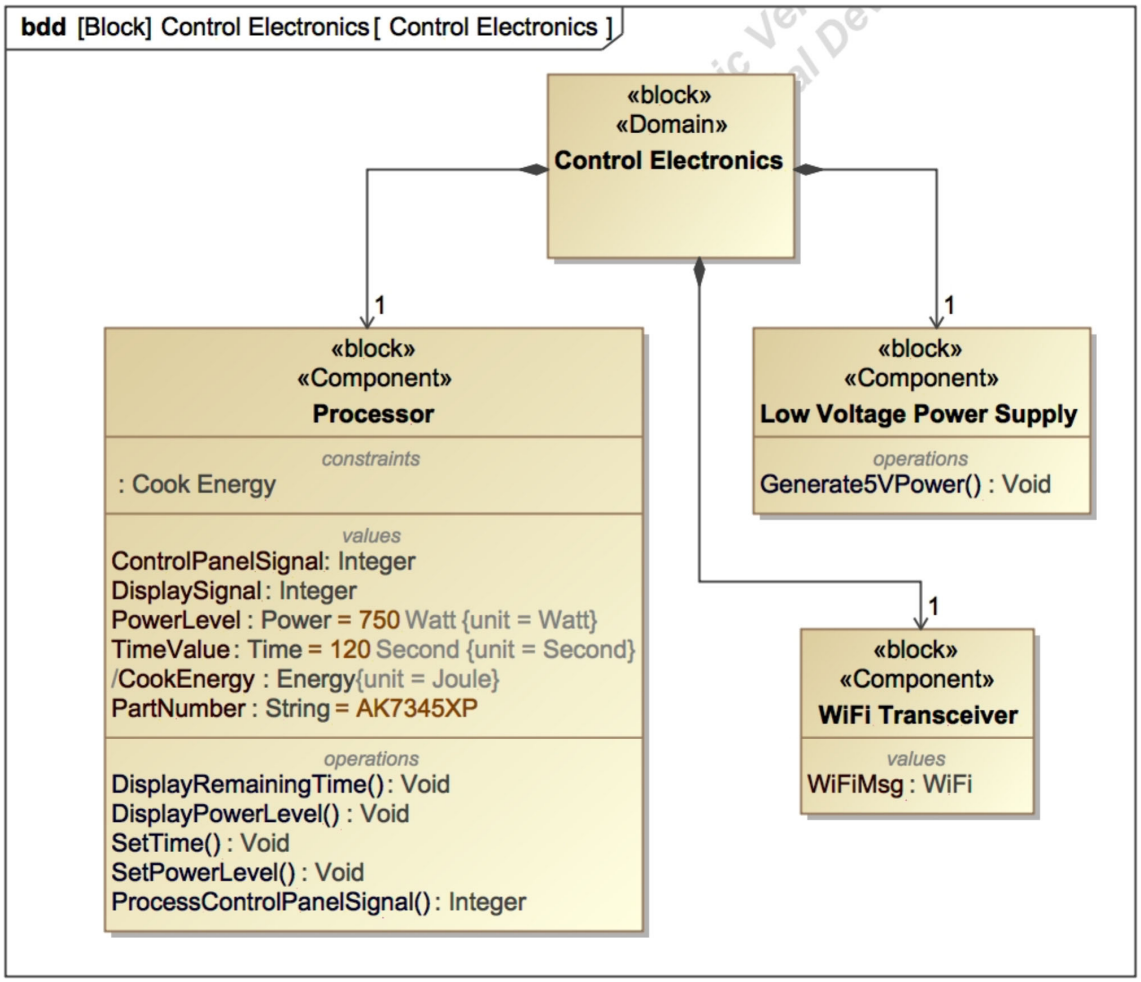


Figure 8. More Detailed Block Definition Diagram

- From below the Microwave Oven Block in the Containment Tree, drag the four Part Properties (small rectangles containing P) into the new Diagram and position them in a logical layout (this can always be adjusted by dragging as the Diagram is created).
- Notice that these Parts have labels consisting of <Part Name>:<Block Name> [n]. A Part (from now on we generally will call a Part Property simply just Part) is an Instance of the Block that it represents. This is how the tool implements the naming convention for an Instance:

<Instance Name>:<Classifier Name>

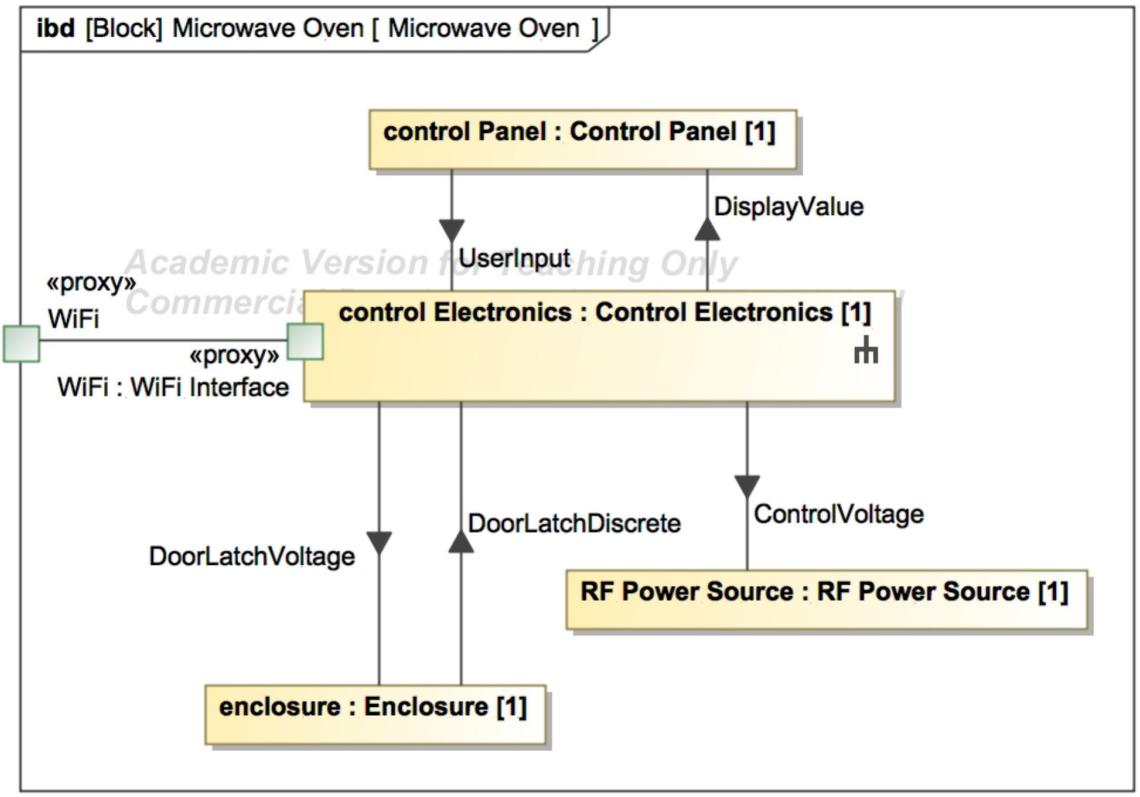


Figure 9. Internal Block Diagram

The Symbol Properties menu allows you to remove either name; a simple name is the name of the Instance, while a leading colon, as in,

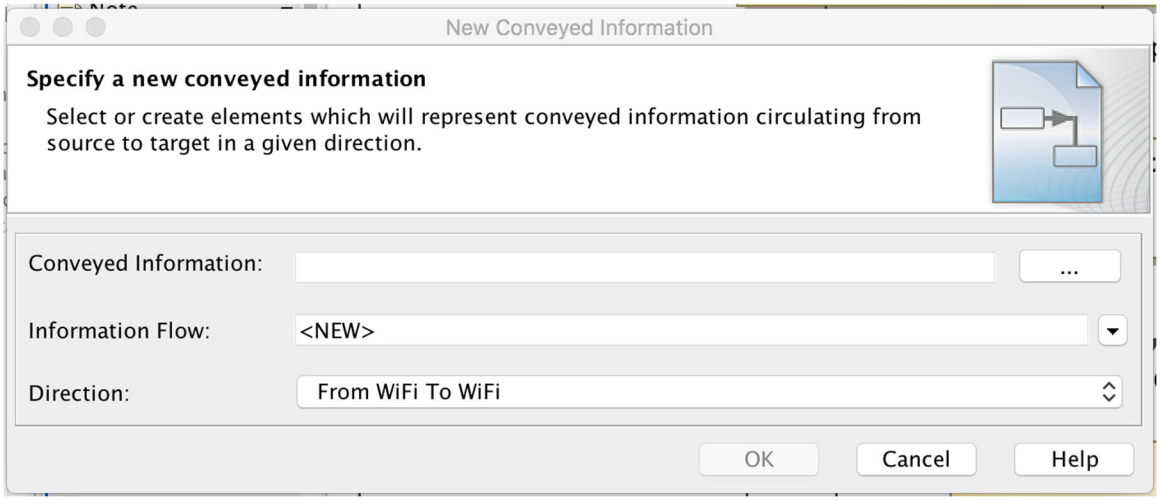
:<Classifier Name>

denotes an unnamed Instance of the Classifier. The number in square brackets is the Multiplicity of the Instance. This naming convention is used in many places, e.g., to identify Instances at the heads of Lifelines in a Sequence Diagram.

- Parts can be displayed with compartments just as Blocks can. By convention, if this content is in a BDD, it needn't be repeated in an IBD. To edit Compartments of a Part, right click to open the Shortcut menu > Edit Compartment > select Compartment > use Compartment Edit dialog > OK.
- The lines in the diagram are Connectors, and a Connector is an Instance of an Association. As usual, you can draw these by clicking on a Part, choosing the Connector icon in the Smart Manipulator toolbar, and dragging to another Part. You can also choose the Connector from the Tool Palette and drag from one Part to the other.
- Connectors support exchanges of content between Parts, and both of these can be accessed via the Smart Manipulator toolbar for a Connector. Before these exchanges can be modeled, the thing to be exchanged needs to be defined in the model. The easiest way to do

this is to create Blocks in the Data Folder of the Containment Tree and give them the names of the things. (You can always go back to these Blocks later and fill in the details.) The simpler kind of exchange is Conveyed Information, and this is what's illustrated in Fig. 9. To add this to a Connector, click the line and select the INFO icon on the Smart Manipulator toolbar. You get a window like Fig. 10. In the top field of this window, click the . . . button and browse to the Block that models the information. You get a window like Fig. 11. Select the proper Block and click the button with a + sign and a little curved arrow to add this to selected items. You can select more than one. Your Conveyed Information should now appear on the Connector in the Diagram pointing from the source to the destination.

Figure 10. New Conveyed Information Window



- The other kind of exchange is a Flow, which is defined by an Item Flow. Flows represent physical quantities (power, fuel, hydraulic fluid, etc.). They are modeled as Flow Specifications (the MBSAP convention puts them in in the Data Package of the Containment Tree) which you can create using the Create Element list for the Data Package in the Containment Tree. Again, details of the Flow can be filled in later. You can add Item Flows to Connectors, but it makes for a clearer Diagram to use the Item Flow arrow that's available on the Tool Palette. At the bottom of the section for IBDs, click Information Flows > Item Flow, and drag from source to destination in the Diagram. You can then open the Specification for the Item Flow and browse to and select a Flow Specification or other entity from the Data Package just as for Conveyed Information.

NOTE: In many architectures, all the exchanges that need to be modeled can be handled with Conveyed Information, but the more physical Item Flow can help make clear the nature of such an interaction.

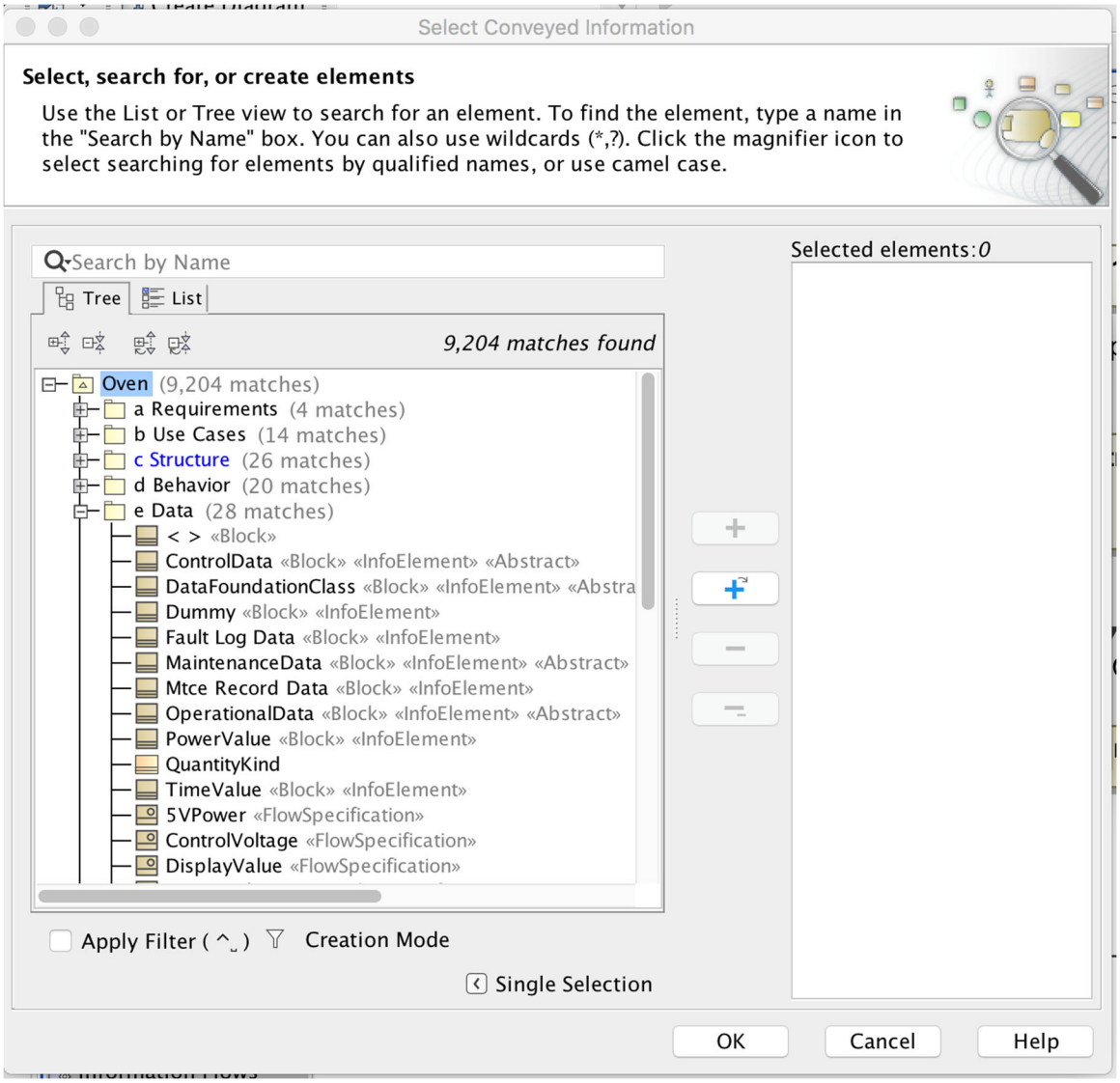


Figure 11. Search Window

- Ports are a vital aspect of structural modeling, but require some explanation:
  - Starting with Version 1.2 of the SysML standard, the original Port types (Standard and Flow) were deprecated and replaced with Full and Proxy Ports. Nevertheless, tools like Cameo still support the old Ports for the sake of backward compatibility. Use of the new Port types is strongly encouraged.
  - A Port models a point of interaction associated with a Block (or Part or Block Instance). As shown in Fig. 8, Connectors can be drawn directly between structural Elements. However, Ports should normally be created when there is sufficient complexity associated with an interaction to warrant capturing those details in the model. This is normally done by defining an Interface and associating it with a Port. An Interface can be Provided, meaning that it exposes content

that the Block or Part is responsible for providing, or Required, meaning that it is the point where the Block or Part brings in content required for it to function correctly.

- A <<proxy>> Port is realized by one or more Parts of a Block; the Port shows where the resources and functions of the Part (or Parts) are exposed. This is the most common choice. A <<proxy>> Port on the boundary of the Block has a Connector to the Part that realizes it. A <<proxy>> Port is typed by an Interface.
- A <<full>> Port, by contrast, is a special kind of Part Property that models both the point of interaction and the resources needed to realize that point. It is typed by a Block that defines the structure and functions of the Port. For example, you could use a <<full>> Port to model a network connection, and the associated Block would have the resources (connector, electronics, etc.) and Operations (transmit Ethernet packet, receive Ethernet packet, etc.) associated with the Port.
- Fig. 9 shows a wireless (WiFi) Port on the microwave oven that makes it an Internet of Things (IoT) device. The <<proxy>> Port on the Block boundary has the name WiFi and is supported by the Control Electronics Part via a Port with the same name. An Interface Block called WiFi Interface types this second Port. The WiFi access point has been modeled this way because the Control Electronics Part has to implement the WiFi Interface and then expose it as an interaction point for the microwave oven as a whole. To create this content:
  - Click on the Proxy Port icon in the Tool Palette and place a Port on the Diagram frame and another on the Control Electronics Part. Give them the selected Port names; these can be different, but in this case it's clear that both Ports are serving to create a WiFi access point for the microwave oven.

NOTE: You can also create the Port on the Control Electronics Part using the Smart Manipulator toolbar.

  - Select one of the Ports, click on the Connector icon in the Smart Manipulator toolbar, and drag to the other Port.
  - No Item Flow is attached to this Connector because the WiFi Interface completely defines what can flow in and out via this Port.
  - A <<full>> Port can be created the same way on the Diagram frame (which, again, is the Block boundary). A Block fully describing the Port should be created and then used to type the Port via the Port Specification.

- Fig. 12 is an IBD that goes with Fig. 8 and illustrates some of these modeling techniques:

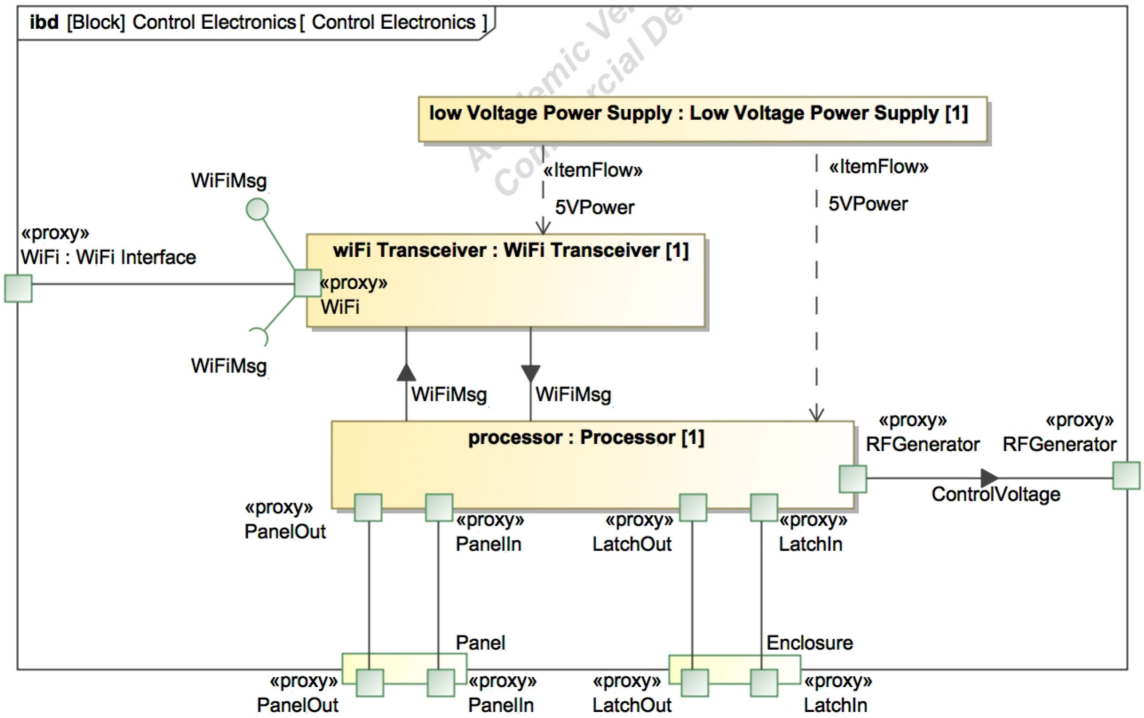


Figure 12. More Detailed Internal Block Diagram

- The WiFi Port on the Control Electronics Block is shown as supported by a WiFi Transceiver Part. The Port on the Block boundary is simply typed by a WiFi Interface. The Port on the WiFi Transceiver shows how Provided and Required Interfaces are represented. The "lollipop" symbol is a Provided Interface, and the "socket" symbol is a Required Interface. They can each have their own interface definitions, but in this case a WiFi Message Interface covers both incoming and outgoing messages.
- Five volt power from the Low Voltage Power Supply Part is shown using Flow arrows with the <<itemFlow>> stereotype.
- The Control Electronics Block has additional Ports for the Control Voltage sent to the RF Power Source, for data to and from the Control Panel, and for the discrete signals associated with the safety latch on the Enclosure door. All of these could have Conveyed Information if desired.
- The WiFi Transceiver and the Processor exchange WiFi Messages; these connections are simple Connectors with WiFi Message as the Conveyed Information. Ports aren't included because the connection is fully described by the Conveyed Information.
- At the bottom of the Diagram frame are examples of nested Ports. These can be useful for modeling an interaction point such as a connector on an electronics box with multiple signal paths. Create an

initial Port, drag it to expand its size, then add the specific Ports to the initial Port.

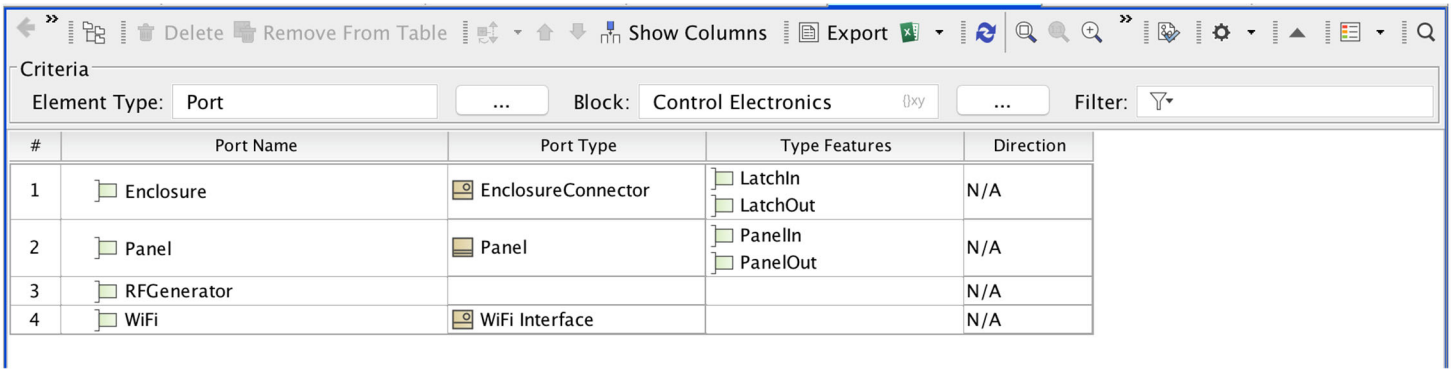
- To create or modify a Provided/Required Interface:
  - Use the Specification dialog of the Port where the Interface is exposed OR display an existing Interface via the Port Shortcut menu.
  - Right click the Port > Specification > Provided/Required Interfaces Group OR use the Smart Manipulator of the Port.
- To type a Port:
  - Do the above to open the Provided/Required Interfaces panel, then > Add > Provided or Required > browse to the Interface Block, Flow Specification, or other Element that defines the Interface.
  - If the Port is already typed and you want to change the type, use the Select Interface dialog > select (or create) Interface and Flow Specifications
  - If the Port is not typed, use Select Port Type menu > Choose:
    - To set a Provided Interface as the Port type > Choose or create the Interface OR
    - Create a dummy Port type automatically OR
    - To select or create a Port type manually > Choose the appropriate Classifier as the Port Type.
- To display an Interface: right click the Port > Show Required or Provided Interfaces OR right click the Port > Related Elements > Display Provided/ Required Interfaces
- You can manage Block Interfaces in the Ports/Interfaces panel of the Block Specification
- An interface is normally defined using an Interface Block, stereotyped as <<interfaceBlock>>
  - This is a specialized Block for typing Proxy Port; it replaces the Flow Specification in earlier versions of SysML.
  - You can define these in the Structure Package, the Data Package, or wherever seems most intuitive.
  - The interface block normally contains a set of Flow Properties in a flow properties compartment with in or out directivity.

NOTE: As usual, content such as the WiFi Interface Block must be created first so it can be used here.

D. Interface Control Documents (ICDs):
   - ICDs are widely used Systems Engineering artifacts, and Cameo automates the creation of both White Box and Black Box ICD tables. A

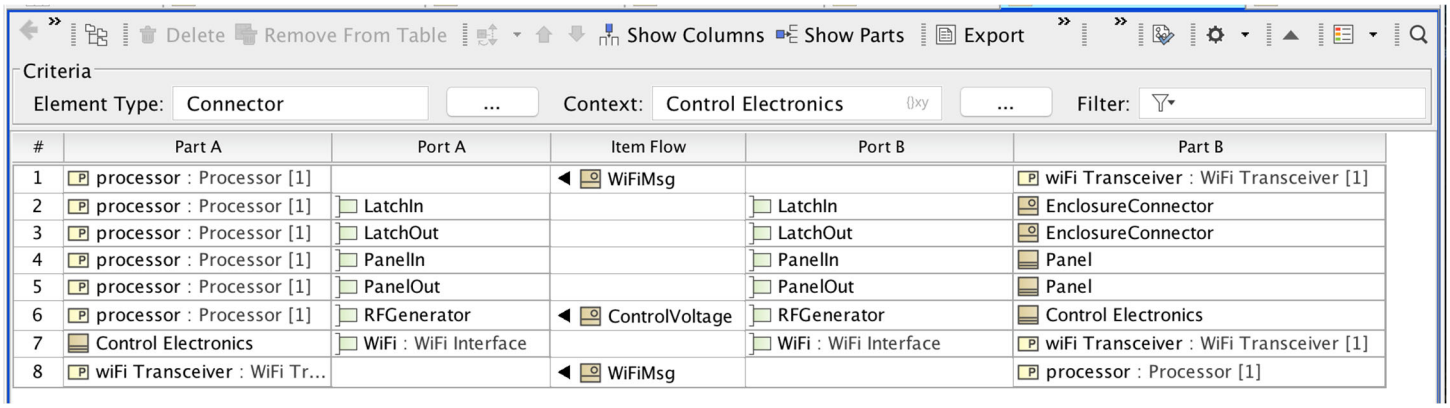
Black Box ICD shows information only at the boundary of a Block while a White Box ICD shows additional internal detail.

- Whitebox ICD Table – show how Parts are connected via Ports/ Interfaces:
  - In the Containment tree or on the diagram pane, select the Block whose structure you want to represent in the Whitebox ICD Table.
  - Do one of the following:
    - From the main menu, select Diagrams > Create Diagram. Type “wh” and press Enter OR .
    - On the main toolbar, click the Create Diagram button. Type “wh” and press Enter OR .
    - Press Ctrl+N. Type “wh” and press Enter OR .
    - Right-click the Block and select Create Diagram > Whitebox ICD Table. The Whitebox ICD Table is created. All Parts, their Ports/interfaces and flows are represented in the table automatically.
  - Type a table name and press Enter.
  - To customize the representation of the table, use the table toolbar buttons. Button commands are described in the Description area under the table. To expand or collapse the description area, click Show/Hide Description on the bottom right on your screen. Use the Show Columns and Show Parts buttons to control content of the ICD table, and drag column widths for a nice looking layout.
- Blackbox ICD Table – represents external Ports/Interfaces of a selected Block:
  - Do one of the following:
    - From the main menu, select Diagrams > Create Diagram. Type “bla” and press Enter OR .
    - On the main toolbar, click the Create Diagram button. Type “bla” and press Enter OR .
    - Press Ctrl+N. Type “bla” and press Enter OR .
    - Right-click the Block and select Create Diagram > Create ICD Table. The Blackbox ICD Table is created. All external Ports/interfaces of the Block are represented in the table automatically.
  - Continue as described under Whitebox ICD.
- Fig. 13 shows a Black Box ICD for the Microwave Oven Block, and Fig. 14 shows a White Box ICD for the Control Electronics Block.



#	Port Name	Port Type	Type Features	Direction
1	Enclosure	EnclosureConnector	LatchIn LatchOut	N/A
2	Panel	Panel	PanelIn PanelOut	N/A
3	RFGenerator			N/A
4	WiFi	WiFi Interface		N/A

Figure 13. Black Box Interface Control Diagram Table



#	Part A	Port A	Item Flow	Port B	Part B
1	processor : Processor [1]		◀ WiFiMsg		wiFi Transceiver : WiFi Transceiver [1]
2	processor : Processor [1]	LatchIn		LatchIn	EnclosureConnector
3	processor : Processor [1]	LatchOut		LatchOut	EnclosureConnector
4	processor : Processor [1]	PanelIn		PanelIn	Panel
5	processor : Processor [1]	PanelOut		PanelOut	Panel
6	processor : Processor [1]	RFGenerator	◀ ControlVoltage	RFGenerator	Control Electronics
7	Control Electronics	WiFi : WiFi Interface		WiFi : WiFi Interface	wiFi Transceiver : WiFi Transceiver [1]
8	wiFi Transceiver : WiFi Tr...		◀ WiFiMsg		processor : Processor [1]

Figure 14. White Box Interface Control Diagram Table

E. Generic Tables:

- This is a good place to introduce another valuable Cameo capability. Virtually any kind of model Elements can be displayed in a Generic Table with selected characteristics. Fig. 15 shows Blocks from the microwave oven model with their Documentation text and Attributes. The steps to build this table are:
  - Open a Generic Table the same way as any other new Diagram, using the Create Diagram button on the main toolbar or right clicking a Package in the Containment Tree (in this case, the Structure Package) and selecting Create Diagram. Give the table a name like Blocks Table.
  - Click the Browse button (. . .) next to Element Type in the table toolbar, scroll down, and select Block.
  - Drag individual Elements (in this case, Blocks) from the Containment Tree to the Scope Field, or browse to a Package containing the elements you want (in this case, the whole Structure Package).
  - The Filter field gives you some refinement options, but you can ignore it for now.
  - In the table toolbar, find the Show Columns button (it's >> followed by a column of dashes), then select Show Columns > Select Columns > click the selection boxes (change False to True) for the ones you want in the table. Your Table should be created.

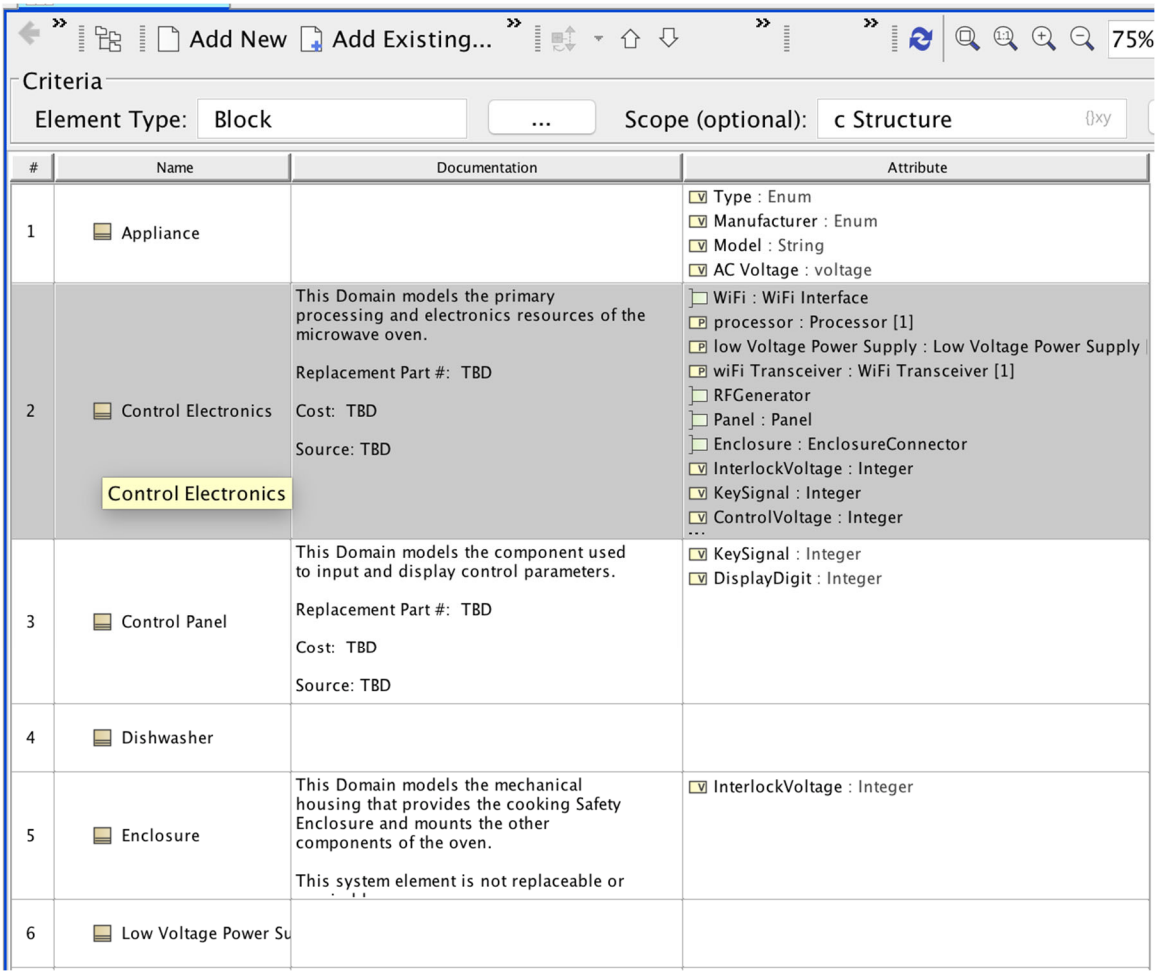


Figure 15. Generic Table with Blocks and Selected Characteristics

- There's a Diagram Wizard that can lead you through table creation, but the process is so easy, it's not really needed. (Cameo provides a lot of Wizards, some of which are very useful indeed.)
- A very important point is that you can edit model content in a table, and sometimes this is the most convenient way to create or update content. For example, the Add New button in the table toolbar in Fig. 15 lets you create new Blocks. Right clicking a cell in the table lets you enter the Block Specification to modify other content.

E. Additional Structural Modeling Techniques:

- It's very common for Blocks to inherit characteristics from more abstract Blocks, meaning that a "child" Block has the same characteristics (Properties, Operations, Constraints, etc.) as the "parent" Block. In fact, a Block can inherit from more than one parent, and specific inherited characteristics can be overridden. The child Block is then given additional, more specific characteristic that make it unique. The parent Block classifies or types the child Block. Fig. 16 shows the microwave oven inheriting from a generic Appliance Block that has Values any

appliance must possess. The diagram is created in the same way as any other BDD except that Generalization relationships are drawn using the appropriate icon in the Smart Manipulator toolbar of the parent Block. In this example, the Appliance Block has the <<Abstract>> stereotype to indicate that it is only a Classifier and is only instantiated through one of its subordinate Blocks.

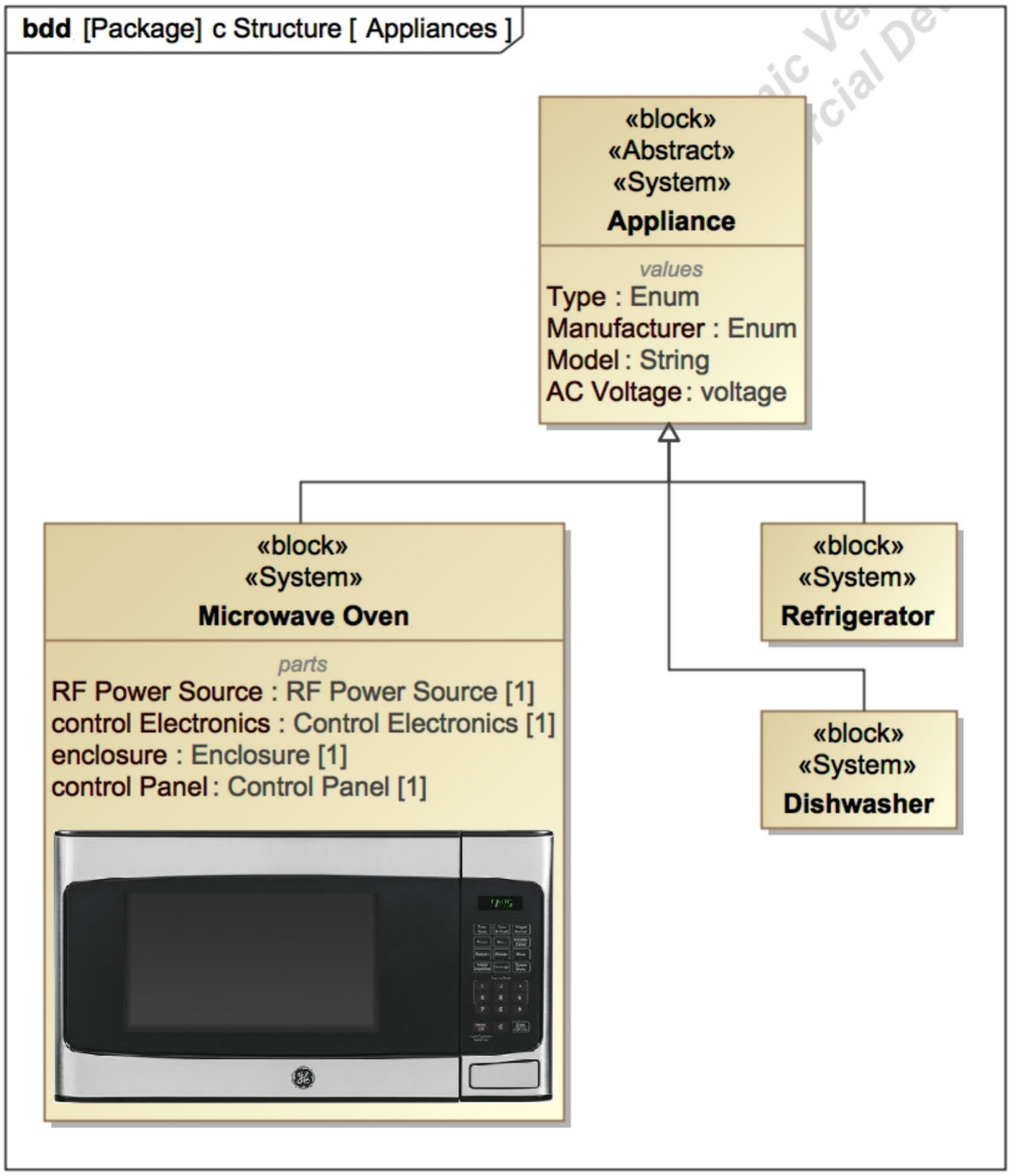


Figure 16. BDD with Generalization Relationships

- Blocks that interact with each other but don't have a structural relationships like one being a Part of the other have a Reference Association. This is created in the same way as a Composition Association except that the Reference icon is selected in a Smart Manipulator toolbar or the Tool Palette. This icon has an empty diamond arrowhead. In some

cases, e.g., if two Blocks are connected by an Association, the tool will automatically create Reference Properties on the Blocks.

- Blocks can be connected by a variety of Dependences, including <<usage>>, <<allocation>>, and <<trace>>.
- Package Diagrams are often very useful in organizing a complicated architecture model. Related model elements (structure, behavior, data entity, etc.) can be dragged into a Package to facilitate browsing and understanding a model. Packages can be nested (this is essentially what happens in the Containment Tree). Packages are commonly connected by Dependencies such as <<import>>, <<usage>>, <<access>>, etc.
- Use graphics in diagrams:
  - For eye appeal, decorate a Diagram, a Block, or another Shape with an image.
  - First, select and copy the image file in the computer file system.
  - In the diagram, press Cntl+V (or Command+V on a Mac) to paste the image. Select Image from the Paste Special menu.
  - Drag the image to a blank area of the Diagram or to a Shape within the Diagram and resize to fit the space. Fig. 16 shows an example.
- Incorporate external information in the model as Attached Files:
  - The Context Package is a logical place to put them - select file in the computer's file system, drag onto the Context Package in the Containment Tree, and select Create Attached File.
  - Alternatively, create hyperlinks to associate external files with model elements: click on an Element > click on the Hyperlink button > Add Hyperlink > use the dialog box to navigate to an external file, another model Element, a URL, etc.

7. Modeling Requirements **Demo 5**

A. A Requirement Diagram (RD) is a modified BDD, and so we covered Blocks in general first, even though many System Engineers would start with requirements. We will also look at a variety of Tables associated with requirements. Fig. 17 shows a simple RD from the microwave oven model.

B. A requirement is modeled by a <<requirement>> Block, which is available from the Requirement Diagram section of the Tool Palette or in the Containment Tree by right clicking the Requirements Package > Create Element > Requirement. Either way, you give the requirement a name, an identifier, and a brief text summary ("shall" statement). This is a bit less cumbersome to do in an RD since you can simply click in the compartments of the requirement Block and enter the information.

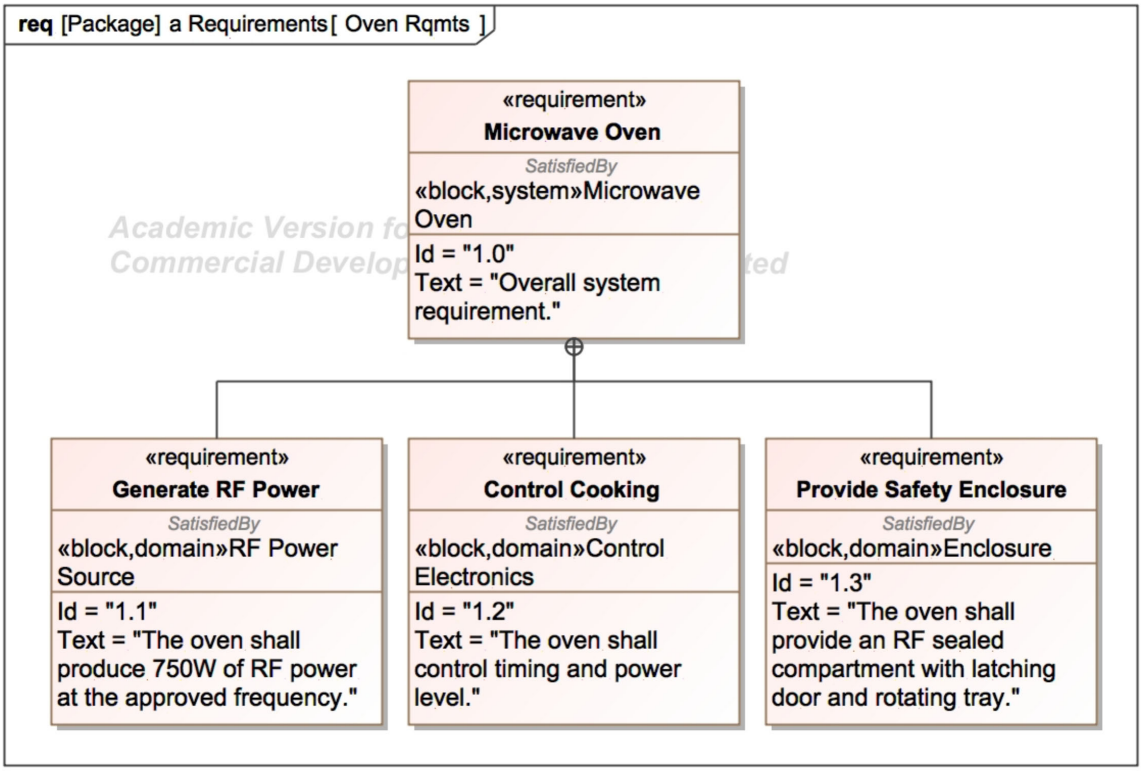


Figure 17. Requirements Diagram

C. Cameo provides a variety of specialized requirement Blocks in the Tool Palette, including Extended, Usability, Business, Functional, Interface, Physical, Performance, and Design Constraint. MBSAP simplifies requirements modeling to use Functional Requirements (of which Performance Requirements are a quantified subset) and Non-Functional Requirements. Functional Requirements are things the system *does*, and Non-Functional Requirements are characteristics the system *has*. But SysML allows pretty much any requirements categories - you can create new ones by defining and applying the appropriate stereotypes.

D. The steps to create the Diagram in Fig. 17 are:

- Right click the Requirements Package of the Containment Tree and choose Create Diagram > Requirement Diagram; name it Oven Rqmts.
- Select Requirement in the Tool Palette; hold down Shift and click four times in the Diagram; drag the Blocks to the proper places.
- Click in the various compartments of each Block and enter the Name, ID, and Text. The ID is an alphanumeric string based on the requirements numbering scheme adopted for the system. The tool will try valiantly to automatically number requirements, and you generally have to overwrite these default numbers.
- There are two kinds of requirement decomposition: Containment relationships and <<deriveRqmt>> relationships. The first, used in Fig. 17,

is appropriate when subordinate requirements are subsets of the higher requirement. The second is used when lower level requirements are created by analysis of higher-level requirements. For this example, select the Microwave Oven Requirement Block and on the Smart Manipulator toolbar, select the *second* Containment icon and drag to the Generate RF Power Block. Repeat for the other two requirements. To make the lines rectilinear, click on each line and click the rectilinear icon (this is a toolbar with only one choice). Drag the line segments to get the desired layout.

- The final step is to link requirements to the structural Blocks that satisfy them using Satisfy Dependencies. That's another reason to define these Blocks in BDDs before working on Requirements Diagrams, but you can always add this content later when the structure is defined. The easiest way to create these links is in the Requirements Block Specification. Open the Specification and select the Relations area. You get a window like Fig. 18. Click the Create Incoming button (a Satisfy Dependency points from the structural Block to the Requirement Block). From the list that opens, select Satisfies. You get a search window like Fig. 19, which is essentially the entire model structure. Expand the Structure Package and click the desired Block. The new Satisfy relationship (Dependency) appears in the Relations window. Click Close.

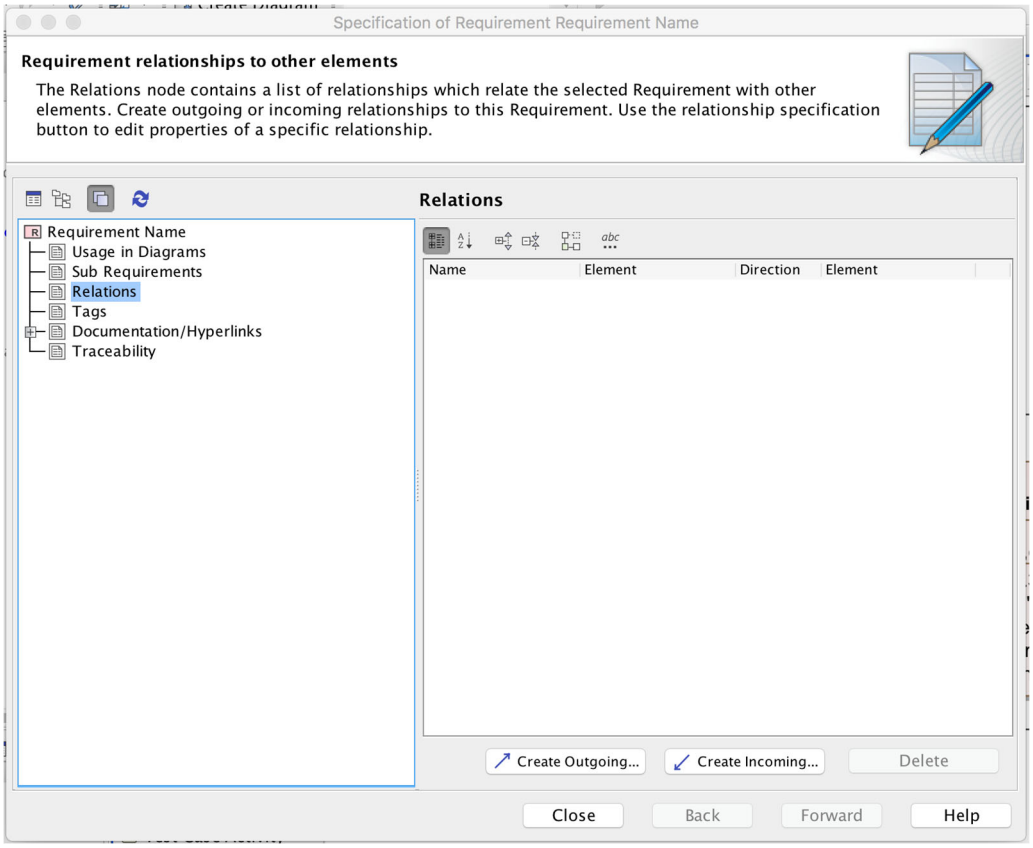


Figure 18. Relations Area of a Requirement Block Specification

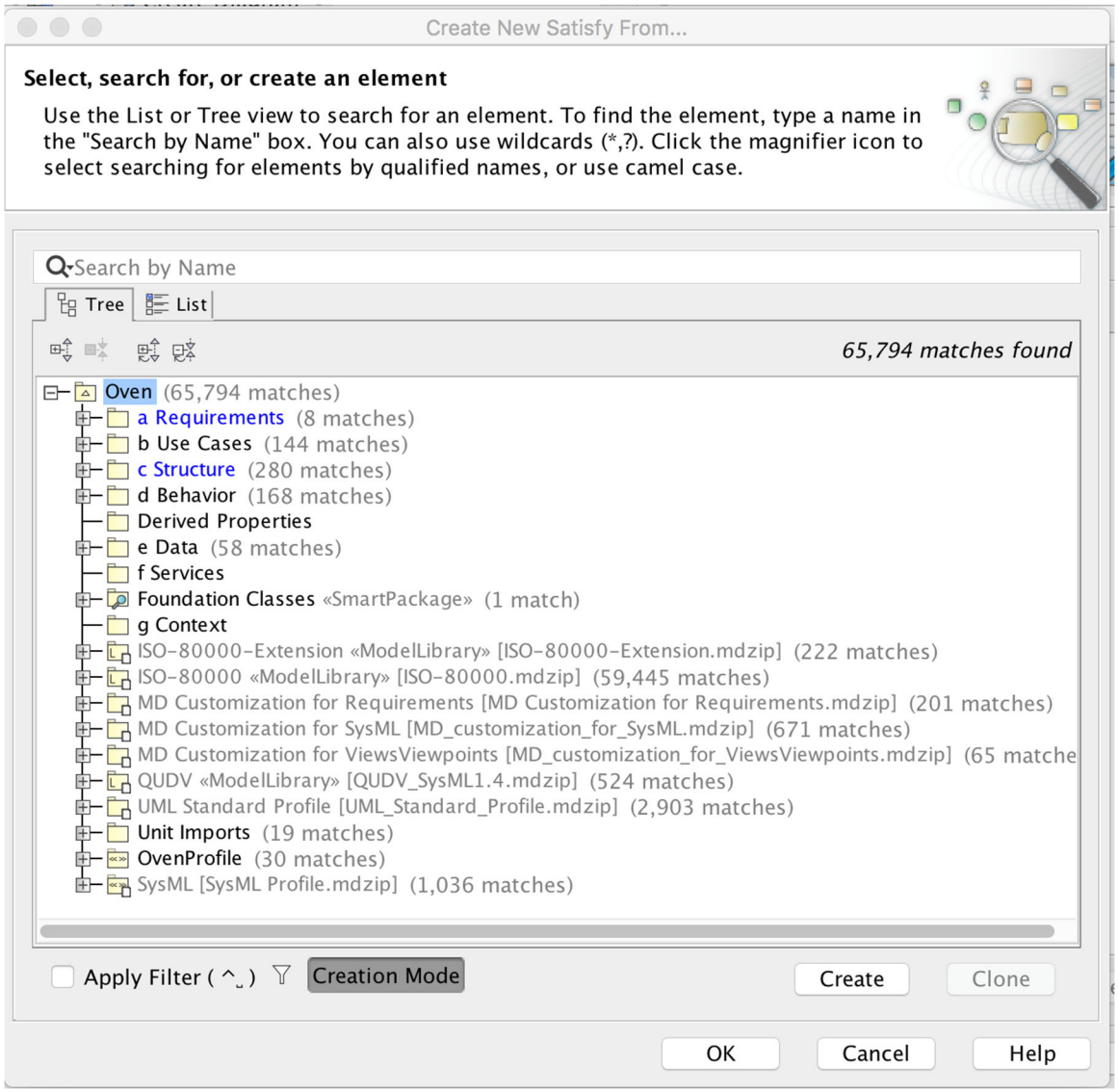


Figure 19. Search Window for a New Satisfy Relationship

- Finally, you have to display the Satisfied By compartment on the Requirements Blocks in your Diagram. On a Requirement Block, click the Compartments icon (three dots in a square) in the upper left corner, select the down arrow and select Edit Compartments. You get a window like Fig. 20. This window is complicated because there are a lot of compartments in SysML; the following steps will produce the desired Diagram:
  - Click the Element Properties tab and scroll down and click on Satisfied By.
  - Click on the right-pointing arrowhead to put this property in the Selected window on the right side.
  - Click OK, and check the Requirements Diagram to verify that the correct Block is shown in a Satisfied By compartment.

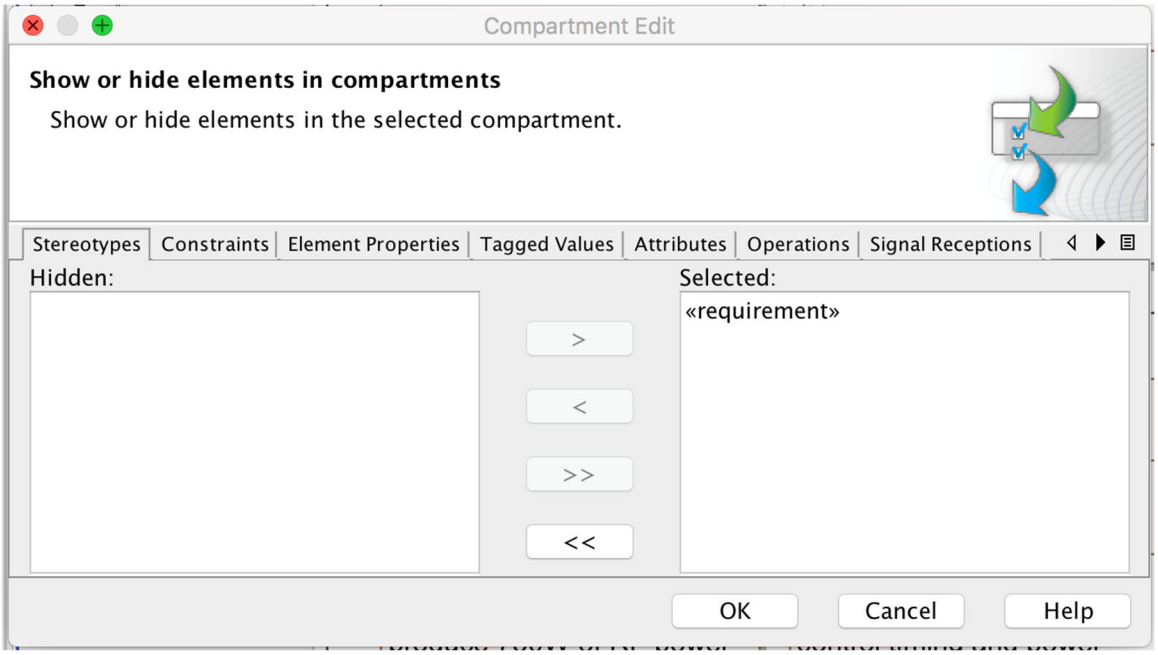


Figure 20. Edit Compartments Window

E. SysML defines a table format for requirements as well as the RD. This is a good illustration of Cameo's capability to easily create a wide variety of useful tables. Fig. 21 shows a Requirements Table for the microwave oven. The content is exactly the same as in the RD created earlier. The steps to create the table are:

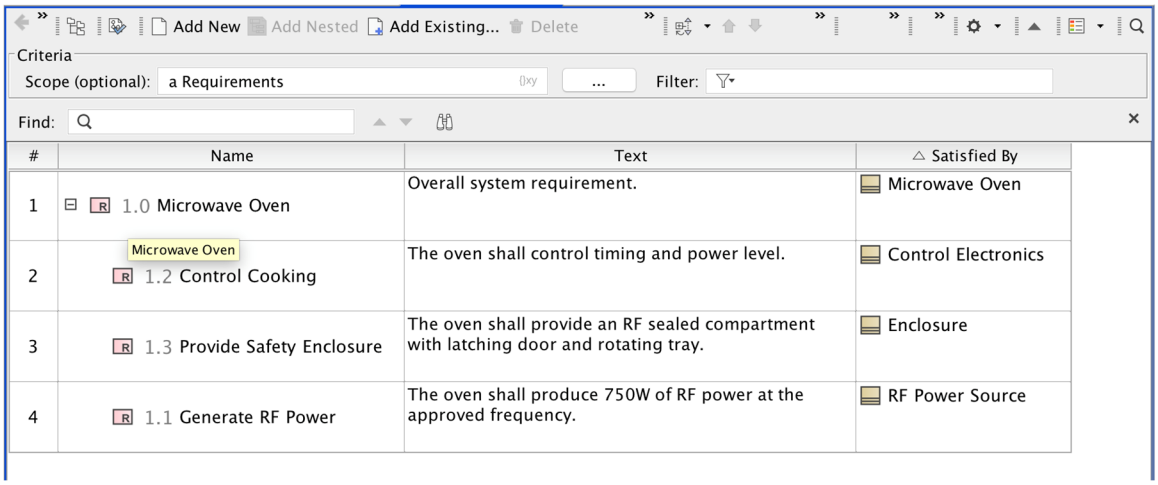


Figure 21. Requirement Table

- Create requirements, either directly in the Containment Tree using Create Element or in an RD. In the Containment Tree, a requirement created under an existing requirement will automatically have a Containment Relationship to that higher-level requirement.
- Open a Requirement Table by right clicking the Requirement Package in the Containment Tree > Create Diagram > Requirement Table

OR

In the toolbar at the top of the Cameo desktop, click Create Diagram and scroll down and select Requirement Table (this also works for all the other kinds of Diagrams, Tables, and Matrices). Give the Table a name or accept the default.

- Drag a Package containing requirements from the Containment Tree to the Scope window above the Table. In this case, the entire Requirements Package gets dragged, but with a large requirements model, you might want to subdivide into lower-level requirements Packages and use one of these.
- Any Property that's been defined on Requirement Blocks is available as a column in the Table. There's a VERY non-obvious button on the Table toolbar (one of the ones with >> and a column of dashes) that opens the list and allows you to select which properties to include.
- The Requirement Table toolbar gives you great flexibility in creating the desired tabular display:
  - You can edit model content directly in a Table. You can change the name, text, etc. of a requirement right in the table. In this case, the Add New button allows you to create new requirements right in the table. These changes become part of the model database and will show up in an RD or other display.
  - The Add Existing button lets you add requirements that you've defined but didn't initially include in the scope.
  - Other buttons allow you to export the Table as an Excel spreadsheet (very useful in processes that prefer to manage requirements in this form), to search for specific content, etc. A bit of experimentation will pay dividends.

F. Cameo also provides a variety of Matrices that show associations between groups of model Elements. A very useful example is a Satisfy Matrix for Requirements. Fig. 22 shows this for the microwave oven. The steps to create the Matrix are:
   - Use Create Diagram in the Containment Tree or the desktop toolbar to open a Satisfy Requirements Matrix, and give it a name or accept the default.
   - Fill in the Row and Column parameters in the Matrix header. This can be done by dragging items from the Containment Tree, but it may be more intuitive to use the browse buttons (. . .) to the right of each field. Select Requirement and Block for the Row and Column types since this is to show Satisfy relationships between requirements and structural Blocks. Select the Requirements Package and the Structure Package as the Row and Column scopes. You should already have Satisfy as the Dependency Criterion, but if not, browse and select it.

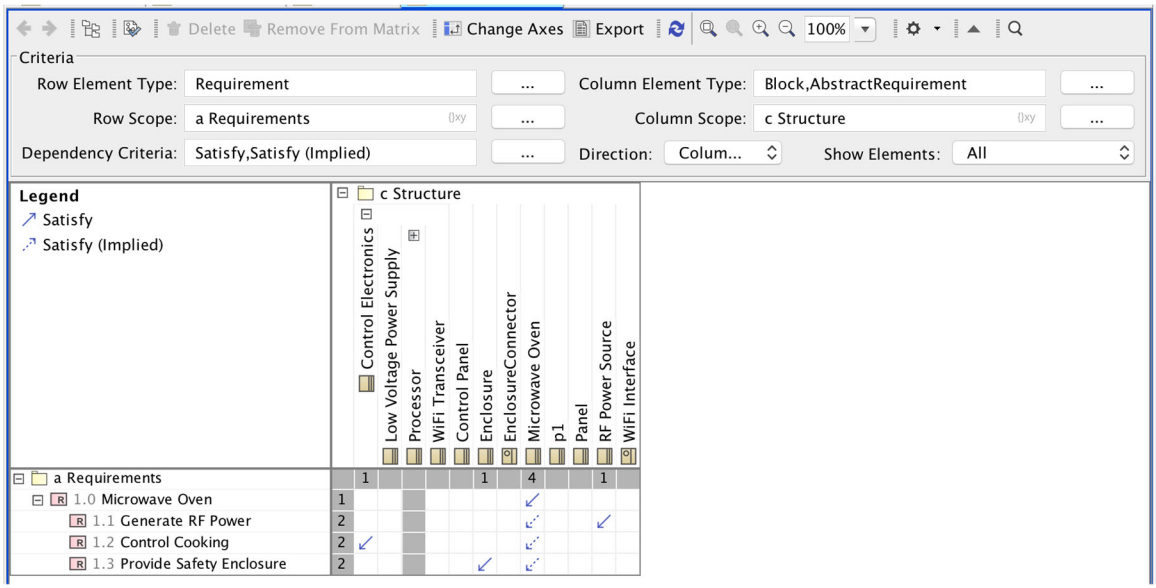


Figure 22. Satisfy Requirements Matrix

- The Satisfy relationships in the model show up as arrows. If a satisfying Block is Part of a higher-level Block, an Implied Satisfy is displayed.
- You can create new Satisfy Dependencies directly in the Matrix by double clicking in the appropriate cells. This might be the most efficient method of all to do this, especially with a large requirements model.
- This Matrix provides a convenient way to check that every requirement has been allocated to one or more structural entities that must satisfy it.

G. Appendix A and the books cited earlier have more information on modeling requirements in SysML.

8. *Modeling Data* Demo 6

A. Block Diagrams are also the basis for building a data model under the MBSAP methodology. The data content of a system is sometimes specified in advance and usually discovered or refined as the architecture development proceeds. The data model, in the Data Package of the Containment Tree, represents each information entity as a Block with a stereotype such as <<infoElement>>, and places these in BDDs to show grouping and inheritance. This approach starts with a Conceptual Data Model (CDM) that defines overall categories of content using <<Abstract>> Foundation Class Blocks, then fleshes these out into actual information items used by the system in a Logical Data Model (LDM). Finally, the details of data storage and management are documented in a Physical Data Model (PDM).

B. Fig. 23 illustrates this with a very simple CDM from the microwave oven model. All of these Blocks are stereotyped as <<Abstract>> since they are never directly instantiated. At the root is a completely general Data Foundation Class that defines the Values and Operations needed by all Foundation Classes. Three more specific Foundation Classes inherit from this

root and could, in general, add Values and Operations as appropriate to the information categories they monitor. Except for the stereotypes, this diagram is built exactly like any other BDD.

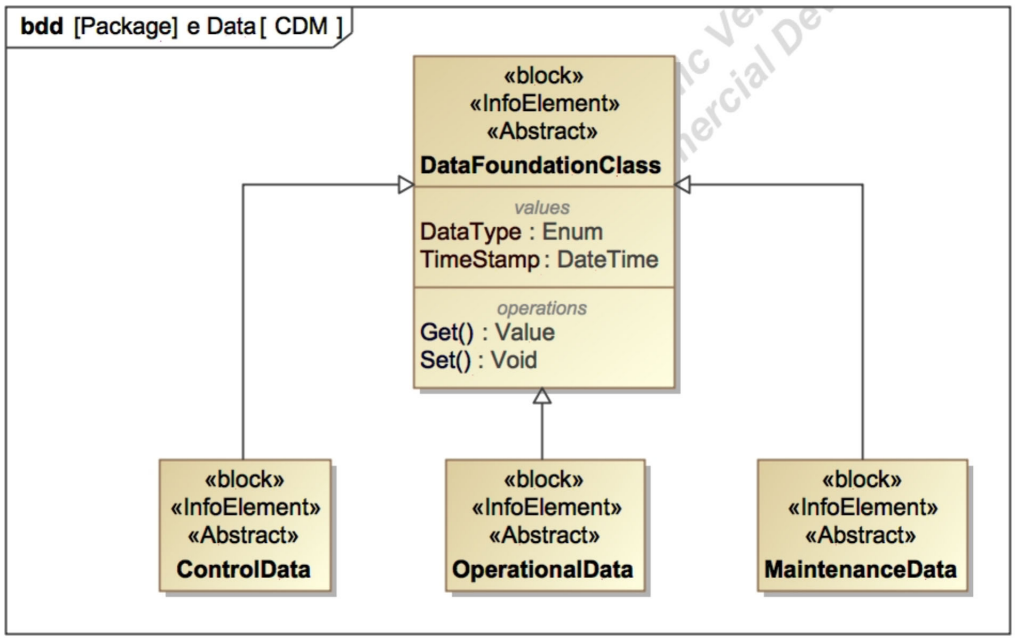


Figure 23. Conceptual Data Model with Foundation Classes

C. Fig. 24 gives an LDM example in which the Control Data Foundation Class is specialized into the two specific parameters that control oven operation: power level and cooking time. These are defined as Value Properties (Values) and given default numbers, which can be changed in operation.

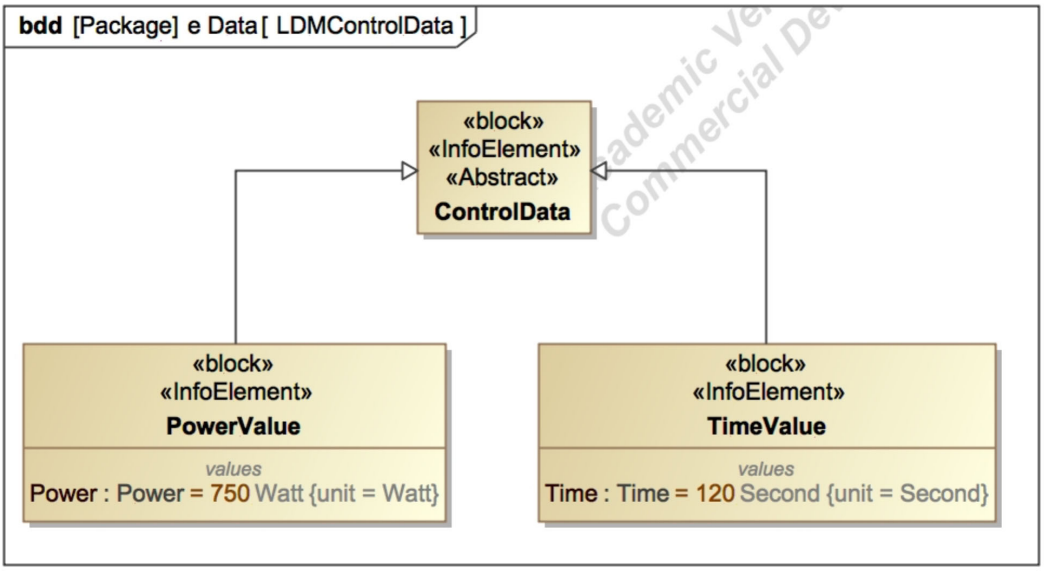


Figure 24. Logical Data Model with System Data Classes (Blocks)

D. The details of the PDM depend on the system and might include things like a memory map and the algorithms used to store and retrieve data.

E. Data Types - a data type is a type whose values have no identity; that is, they are pure values. It is a classifier and inherits the general features of the classifier: visibility, generalizable element properties, and operations. MagicDraw provides the following predefined data types: boolean, byte, char, date, double, float, int, Integer, Real, long, short, void, and String. You can also create Enumeration or Primitive data types.

   To create a data type: open a Package (usually, the Data Package), a subsystem, or the model Shortcut menu > Create Element > Data Type, Enumeration, or Primitive OR open Element Spec window > Inner Elements > Create > select Data Type THEN use the Data Type Spec window to define properties.

   Enumeration is a user-defined data type whose instances (range) are a set of user-specified and named *enumeration literals*; a variable with type enum can take on any of the literals. A simple procedure to create an Enumeration is:

   - Create a BDD in the Data Package of the Containment Tree.
   - In the Tool Palette, click on Enumeration and click in the diagram; give the new Enumeration a name.
   - To create the literals: Open the Enumeration Block Spec window > Enumeration Literals > Create > enter new literal OR in the Containment tree, right click an Enumeration Block > Create Element > Enumeration Literal OR in a diagram, click the Enumeration and use the Smart Manipulator

9. *Modeling Use Cases and Actors* Demo 7

A. Behavioral modeling starts with Use Cases (UCs) and their associated Actors. A UC models a unit of system behavior which has defined start and end conditions (called Pre- and Post-Conditions), usually some kind of trigger that causes the behavior to execute, and one or more scenarios that spell out the order of the behavior. UCs can be thought of as a "table of contents" of system functions, and collectively, they offer a way to portray a Concept of Operations (CONOPS). They can be a very useful tool, especially early in an architecture development, to think through system functions and associated requirements for completeness and consistency.

B. Actors model anything external to the system with which the system interacts. This may include people, networks, other systems, controlled machinery, external communications, etc. Associating Actors with UCs in a UC Diagram is a good way to check on the basic behavioral structure of an architecture – ideally, there should be clean and relatively simple relationships between Actors and the UCs they interact with.

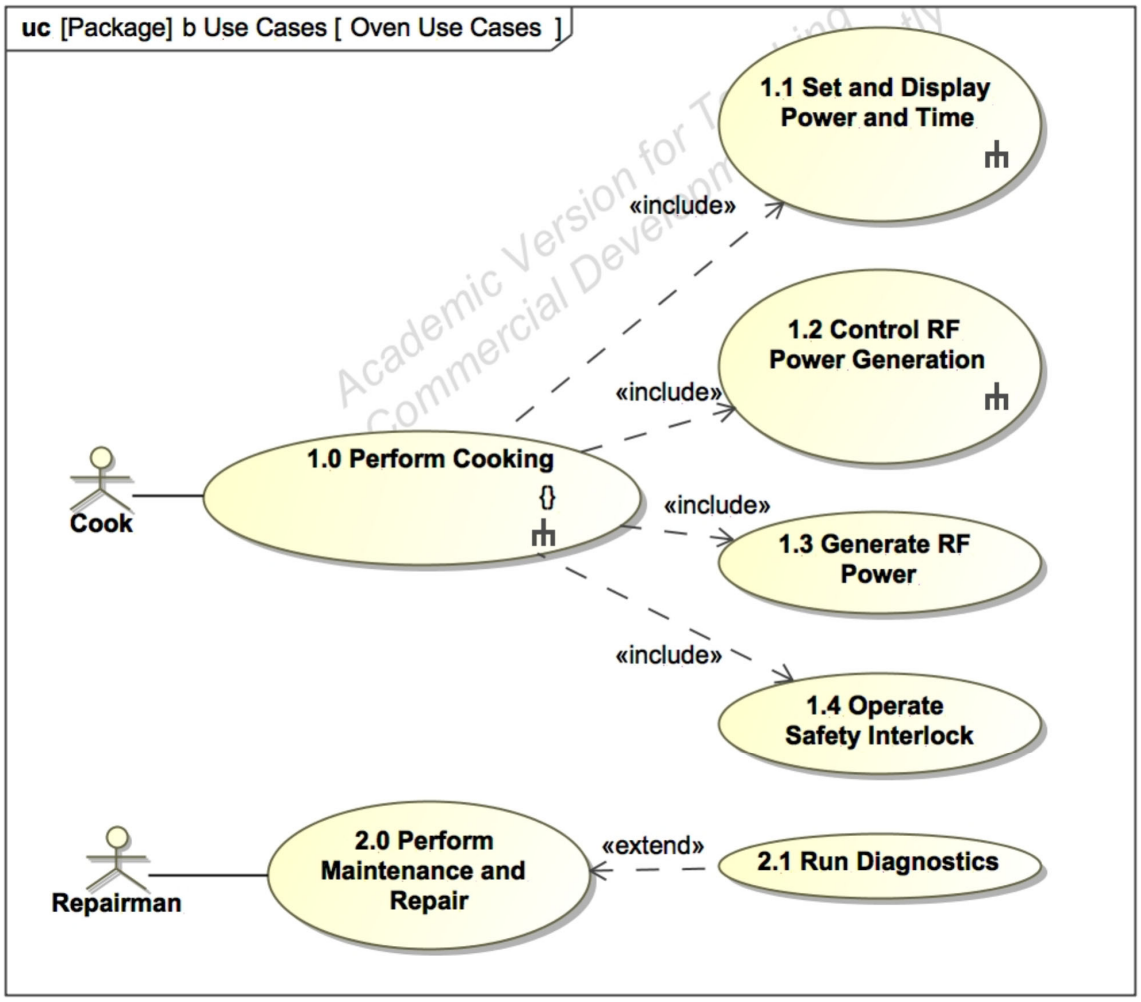


Figure 25. Use Case Diagram

C. Fig. 25 shows some UCs from the microwave oven model. The steps to create the diagram are:

- Open a new UC Diagram using Create Diagram in the Use Case Package of the Containment Tree. Give it an appropriate name.
- In the UC area of the Tool Palette, select the UC icon, hold down the Shift key, and click multiple times in the diagram. Drag the UCs to create a pleasing layout and give each UC a name. It's a good idea to use a numbering scheme like the one shown in Fig. 25.
- Next select the Actor icon in the Tool Palette and create the Cook and Repairman Actors in the Diagram. The tool will put the new UCs and Actors in the Containment Tree for you.
- Select the Association icon and draw the relationships between Actors and primary UCs.
- SysML provides two main kinds of Dependencies for UC Diagrams. The <<include>> Dependency is used to decompose a higher-level UC into

more basic UCs. Select the Include icon in the Tool Palette or from the Smart Manipulator toolbar and draw the relationships between 1.0 Perform Cooking and the four subordinate UCs.

- The other primary Dependency is <<extend>>, which is used to show that a specific behavior modeled as a Use Case is an optional addition to the behavior in another UC. In Fig. 25, the Repairman might choose to run some embedded diagnostics that are built into the oven. Note that this Dependency arrow points *towards* the parent UC. Create this with the Include icon from the Tool Palette or the Smart Manipulator toolbar.
- It's also permissible to use <<usage>> and other Dependencies among UCs. In fact, you can show Generalization/Inheritance among UCs although that would be unusual with well-defined UCs.
- The MBSAP methodology calls for three kinds of related model content to fully define a UC: (a) the UC itself, in a UC Diagram, (b) an Element Specification for the UC, and (c) a behavior Diagram showing the UC scenario(s). Fig. 26 shows a UC specification following a standard template from Borky and Bradley that is entered in the Documentation window of the Specification for UC 1.0. Some of this content is optional, but collectively this is a full description of the UC behavior from which system designers can proceed to implementation. Having all this in the UC Specification simplifies generating a report from the model that documents the UCs. It's also a good idea to explicitly identify the requirements that this behavior satisfies. Right click the UC, select Specification, open the Documentation window, and enter the information.

NOTE: Strictly speaking, pre- and post-conditions are Constraints on a UC and can be created in the Constraints pane of the Spec window. That may be important in sophisticated simulations involving UCs, but simply using the Documentation text is usually sufficient.

- Several of the UCs in Fig. 25 have "rake" symbols in them. This identifies the fact that there are embedded diagrams, in this case Activity Diagrams showing the UC scenarios, as called for in (c) above. Double clicking UC 1.0 opens the Diagram as shown in Fig. 27. Activity Diagrams are discussed more completely in the next section.

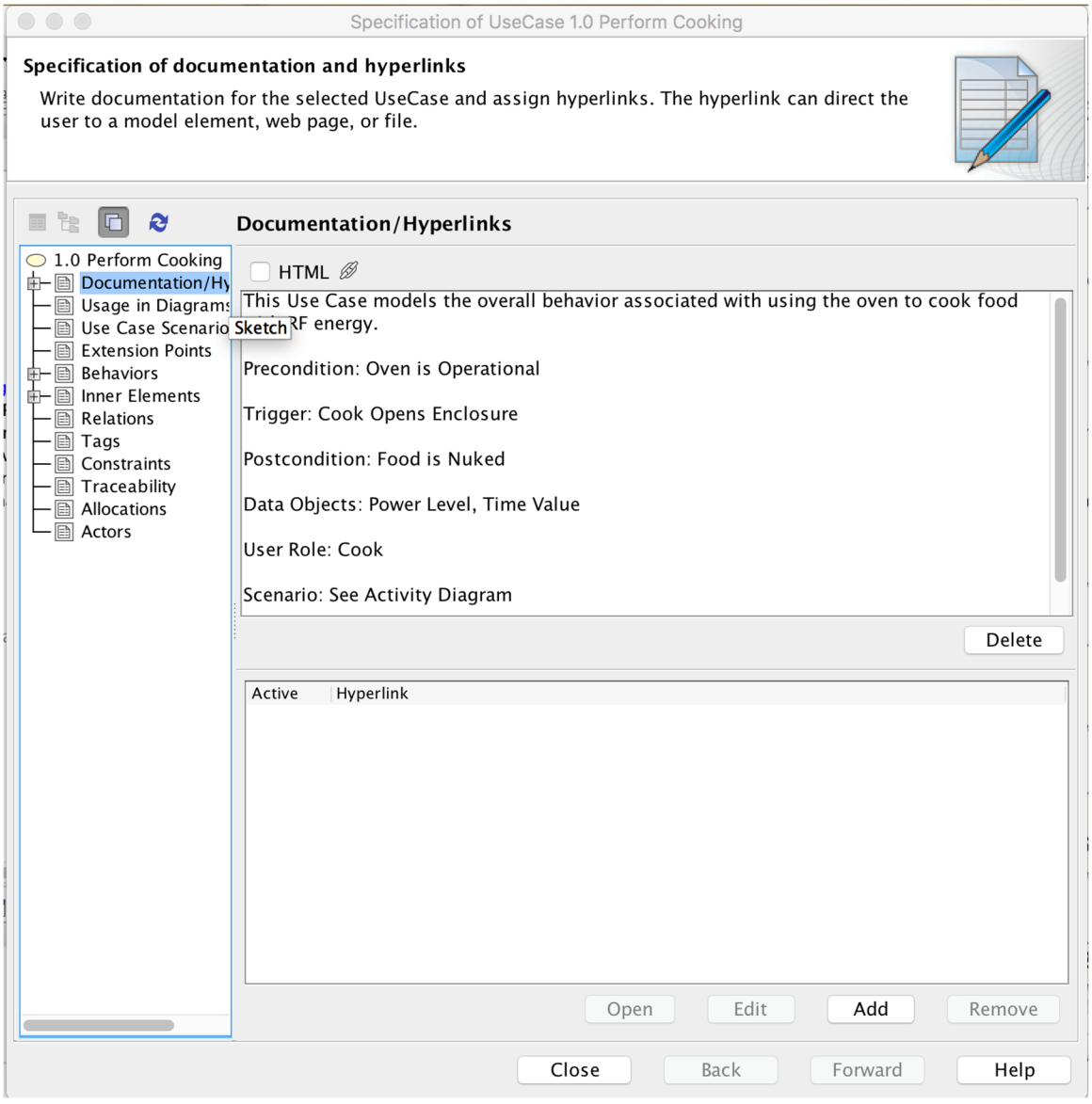


Figure 26. Use Case Specification

- Fig. 27 illustrates a common modeling pattern that applies when a higher-level UC is decomposed. It shows how the lower level UCs, modeled here as Actions, operate in sequence to implement the primary behavior. Additional Actions, as well as other Activity Diagram content that will be discussed shortly, could also be included. One of the Actions also has a rake symbol, and double clicking it yields a lower level Activity Diagram as in Fig. 28. Behavioral Elements can be decomposed down to whatever level of detail is desired

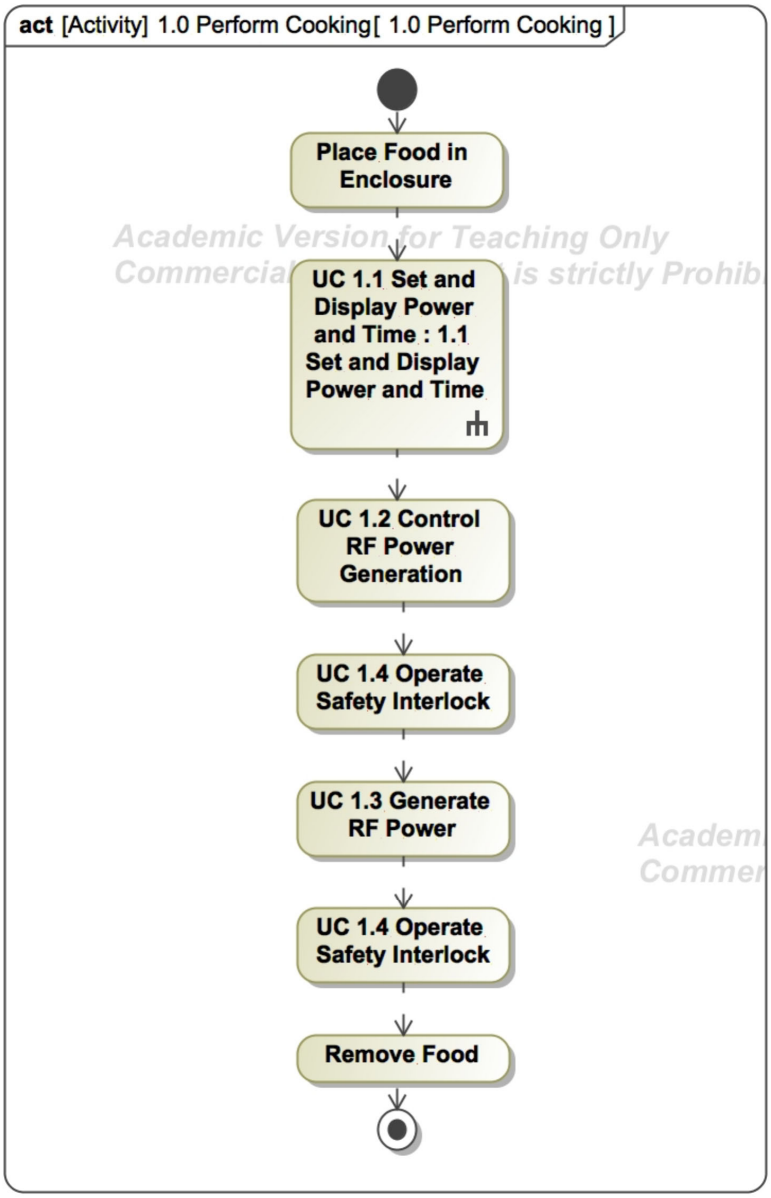


Figure 27. Use Case Scenario Activity Diagram with Subordinate Use Cases

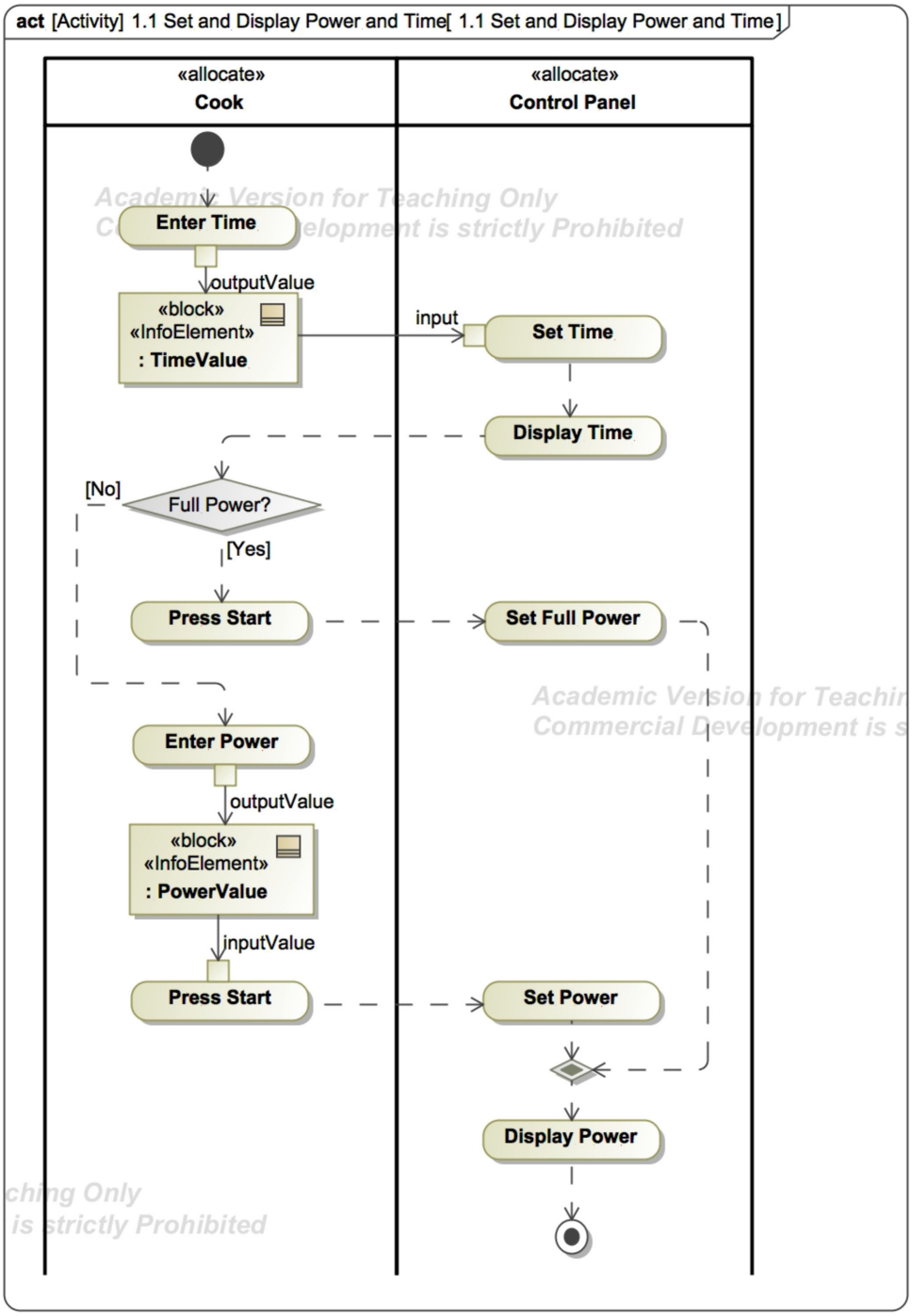


Figure 28. Lower Level Scenario Activity Diagram.

- To complete this part of the model, each UC should have a Specification filled in and a scenario Diagram created within the UC.

- MBSAP recommends creating a special kind of UC Diagram called a User Roles Diagram. It contains only Actors that model User Roles, i.e., the interactions of system users with the system. In a complex system with many kinds of users, this is a good way to capture characteristics such as required training and to group related User Roles. It's often very convenient to have a hierarchy of User Roles with specific roles inheriting the shared characteristics of more general roles. Fig. 29 is a very simple example from the microwave oven model. It's created like any other UC Diagram except that Generalization relationships are used to show that Cook and Repairman inherit from a general Oven User Role.

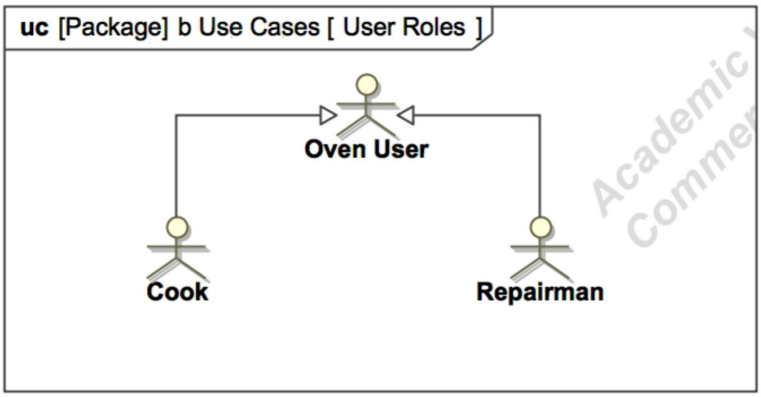


Figure 29. User Roles for the Microwave Oven

10. Modeling Activities

A. SysML defines three primary kinds of behaviors, Activities, Interactions, and States/State Transitions, each of which has a diagram type. Activities are modeled in Activity Diagrams, Interactions in Sequence Diagrams, and stateful behavior in State Machines. Each has characteristics that suit it for particular kinds of behavior, e.g., a State Machine is the common way to model the behavior of a Block. As a rule of thumb, Activity Diagrams (ADs) are best for higher-level behaviors, which is why they're the most common way to show UC scenarios, while Sequence and State Machine Diagrams (SDs, STMs) are used to model more detailed behaviors. Any of the three can be made the Classifier Behavior of a Block. An AD looks a lot like a traditional flowchart. Mathematically, it's a variant of Graph Theory, and SysML brings in the features of an Enhanced Functional Flow Block diagram (EFFBD).

B. An Activity is made up of specific units of behavior or function called Actions. An Action can be atomic, meaning that it can't be decomposed to a lower level, or non-atomic, meaning that it can be shown in finer detail by embedding a lower-level AD or other behavioral Diagram within it. Actions are linked by two kinds of Flows: Control Flows and Object Flows. A Control Flow simply models the fact that when one Action completes, another starts; strictly speaking, this consists of passing a Control Token. An Object Flow also passes control, but in addition it passes a unit of content such as a data item; it passes a Control Token. For example, an Action allocated to one part

of the system might create a plan, which is then passed to an Action in another part for execution. SysML defines a variety of Actions, including:

- Accept Event - this Action waits for the occurrence of an event generated elsewhere in the system that meets specified conditions.
- Call Behavior - this is the most common and causes a behavior to be executed; accordingly, the Action must be allocated to a separately defined behavior, which can simply be another AD. If you haven't provided the required behavior, the tool raises a caution (a yellow outline around the Action). You can then do one of two things:
  - Change the Action to an Opaque Action that names a behavior but doesn't specify what it is.
  - Open the Smart Manipulator for the Action, click the caution icon at the top of the toolbar, and choose the option to let the tool create a behavior and allocate the Action to it. The result will be an empty AD in the model, and you can go back later and fill in the details to define the behavior.
- Call Operation - this is similar to a Call Behavior, except it invokes a specific Operation defined on a Block.
- Opaque - this can be used as a placeholder until the details of an Action are known.
- Send Signal - creates a Signal that can be detected and acted upon elsewhere in the model; this is a common way to synchronize simultaneous Activities.

C. SysML has two ways to show Object Flows. One uses Object Nodes to model the entity that is passed by one Action to another. An Object Node can be typed by a Block that models the entity and is usually defined in the Data Package as part of the system data model. The other way to model an Object Flow uses Pins on the Actions that are connected by a Control Flow path. The Pins define the exchanged entity. Fig. 28, above, illustrates the first of these. The steps to create the diagram are:

- Create an AD in the usual way. If this is a UC scenario, right click the UC and select Create Diagram. If it's associated with a Block, right click the Block. If it's an overall system behavior, right click the Behavior Package in the Containment Tree.
- One of the most powerful features of an AD is its ability to link behavior to structure by allocating Actions to the structural Elements that perform them. This is accomplished by organizing the Diagram with Activity Partitions and then allocating each partition to a Block, Actor, Interface, or other structural Element. Activity Partitions are universally called "Swimlanes" from their obvious resemblance to a swimming pool.

NOTE: It's a very good habit to make a simple pencil sketch of the AD before building it in the tool, as this will help get number of Swimlanes and the order in which they should occur right the first time.

- A simple AD showing the behavior of a single Block or of the entire system can simply contain Actions (and perhaps other content) wired up with Flows as illustrated in Fig. 27. In this case, the entire AD is a single Swimlane. More commonly, the Activity involves more than one Block, Actor, Interface, etc. each of which is represented by a Swimlane. For Fig. 28, the first thing to do is create Swimlanes by selecting Vertical Swimlane in the Tool Palette and clicking in the Diagram. If more than two Swimlanes are needed, right click in an existing Swimlane and select the desired option. Do this as many times as needed to get the required number of Swimlanes. As the Diagram develops, it's always possible to insert additional Swimlanes or delete them as needed.

NOTE: Activity Partitions can also be horizontal if desired, and they can be hierarchical (nested). In fact, you can use both in the same AD.

- Allocate Swimlanes to structural Elements by dragging the Elements from the Containment Tree to the headers of the Swimlanes. The name of the Element and an <<allocated>> stereotype should appear in each header.
- A legal AD must have both an Initial Node that shows where the behavior begins and an Activity Final Node that shows where it ends. If the AD models a UC scenario, these are the places where Pre- and Post-Condition invariants apply. There may also be conditions (e.g., a failure or an abort condition) when the behavior halts prematurely. These are modeled using the Flow Final Node. Start by clicking the Initial Node icon in the Tool Palette and clicking in the Diagram at the top of the Swimlane where the Activity begins.
- You then build up the Diagram from the Tool Palette by adding Actions, Object Nodes, Decisions, Forks, Joins, Merges, etc. and "wiring them up" with Control and Object Flows. Click an icon in the Tool Palette and either click or click-drag (for a Flow Path) to put the Symbol (Shape or Path) in the Diagram. Any Flow involving the transfer of an Object must be an Object Flow. Finish the Diagram with an Activity Final Node, and include any Flow Final Nodes you need. Several important details to pay attention to include:
  - There can't be any "orphan" or "dangling" Actions or Objects. Every one of these must have incoming and outgoing Flow paths. In other words, once the Activity is executed, it must be able to run either to completion at an Activity Final Node or to an intermediate Flow Final Node.
  - Making the Paths rectilinear almost always yields the most attractive Diagram layout. If a Path has another style, you can simply select it and click the rectilinear icon on the Smart Manipulator toolbar. Drag

Paths around as necessary to avoid crossing other Symbols and to get the desired layout.
    - Make sure every Flow leaving a Decision has a Guard. Click the Flow path, open the Specification, and enter the Guard in the indicated field. Remember that a Guard is Boolean (it must evaluate to True or False) and only one Flow path leaving a decision can have a True Guard at any given time.
    - A Flow path can be divided (i.e., go to more than one Action) using a Fork, which can be Vertical or Horizontal depending on the orientation of the Swimlanes. A Join brings two or more Flow paths back together synchronously, i.e., all the incoming Paths must be active before the behavior continues past the Join. In Fig. 28, just before the Display Power Action, there is a Merge. This is an asynchronous rejoining of Paths, i.e., as soon as any incoming Path is active, the behavior continues past the Merge.

- One further advantage of this AD style is that it supports the key process of data discovery. It's often not completely clear at the start of an architecture development what all the information and other content of the system will turn out to be. In creating ADs, you can frequently identify the outputs and required inputs of various Actions – these are the exchanged Objects whose existence is demanded by the functioning of the system. If a needed Object hasn't yet been defined, a very good modeling practice is to first create it as a Block in the Data Package, then use that Block to type an Object Node in the AD. A good start on the system data model will be gradually built up in the course of this behavior analysis and modeling. If the full details of a given Object aren't initially known, the Block can be created as a placeholder and later refined and completed.

D. Fig. 30 illustrates the other style of Object Flow in an AD using Pins. Some key points about this diagram include:
    - The Object Flow from the Compute RF Power Source Control Voltage Action to the Apply Control Voltage Action is shown as an arrow (with a solid shaft) between Pins on the two Actions. If you select Object Flow in the Tool Palette and drag from one Action to the other, this arrow is created along with the output and input Pins. To show the Object being exchanged, first drag the appropriate Element from the Containment Tree to the Flow arrow. In this case, a Flow Specification called ControlVoltage is dragged to the arrow. Then you can right click each Pin, open the Symbol Properties menu (also called a "dialog box" in Cameo), and click the box for Show Type to display the name of the Object being flowed.

    - In the upper left corner of the Diagram are two rectangles that cross the Diagram frame. These are Activity Parameter Nodes, and they provide a way to bring in values that are generated elsewhere in the model, in this case, as data from the Control Panel of the microwave oven. For each of

these inputs, select Activity Parameter Node in the Tool Palette, click at the desired spot on the Diagram frame, type the node by dragging the appropriate Element from the Containment Tree, and draw an Object Flow to the Action that uses the parameter. You can also create an output Activity Parameter Node to export a value created within an AD.

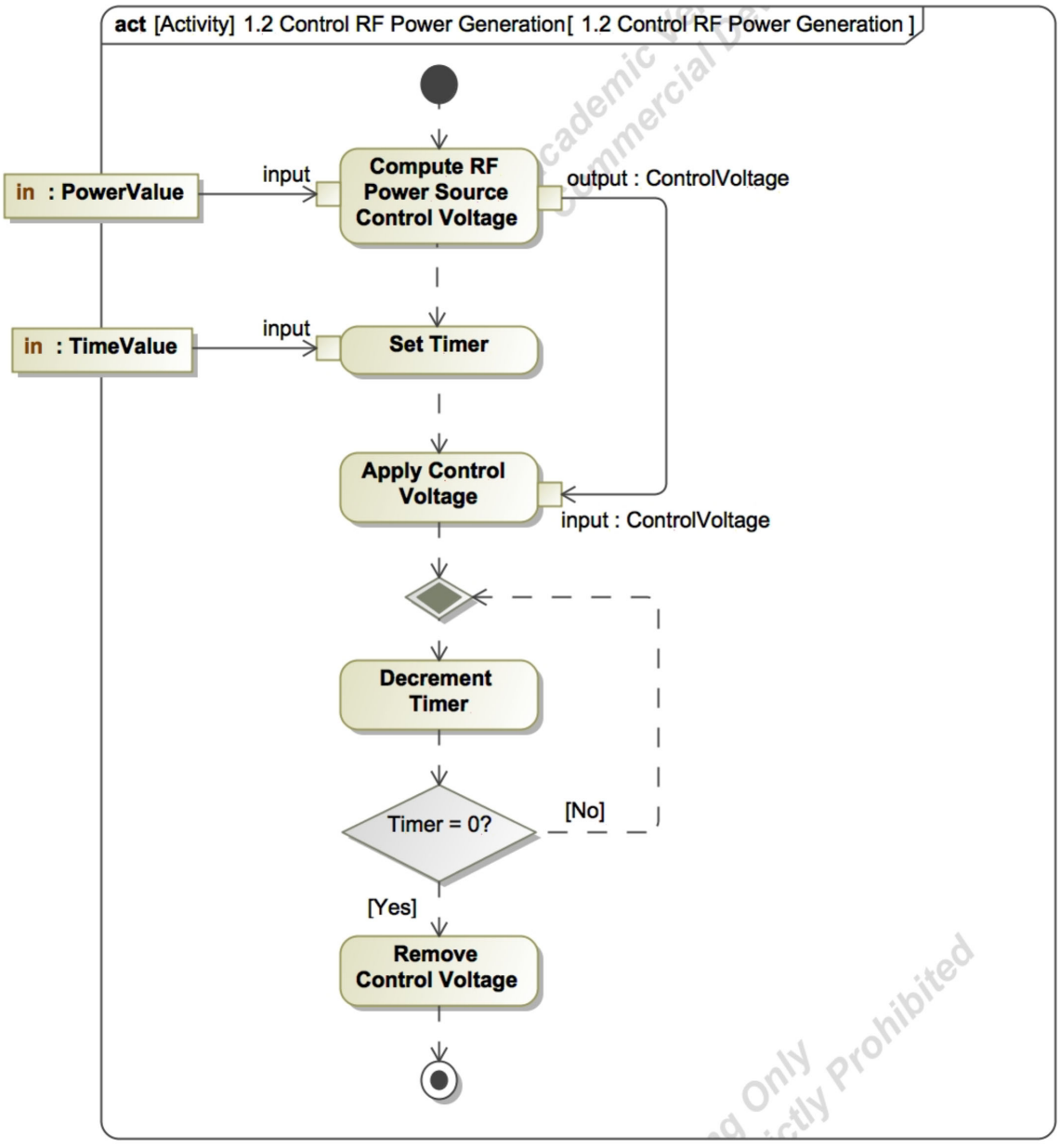


Figure 30. Activity Diagram Using Control Flow Between Action Pins

- This diagram can be simulated as a simple example of model execution.

E. Fig. 31 shows a slightly more complex AD that illustrates another common modeling practice. This is the use of an AD to show the overall, normal behavior of a system, sometimes referred to as a "day-in-the-life." In Fig. 31, the Domains of the microwave oven are shown executing the series of

behaviors associated with setting the controls and cooking the food. Some simple graphics, in the form of shaded rectangles, are used to associate parts of the diagram with Use Cases. These are created using the Common area of the Tool Palette, specifically a tool with choices for horizontal separator (line), vertical separator, and rectangle. Right clicking one of these in a diagram and selecting Symbol Properties allows you to set line styles and colors, and these graphical elements can be moved backward or forward so as not to obscure other diagram content.

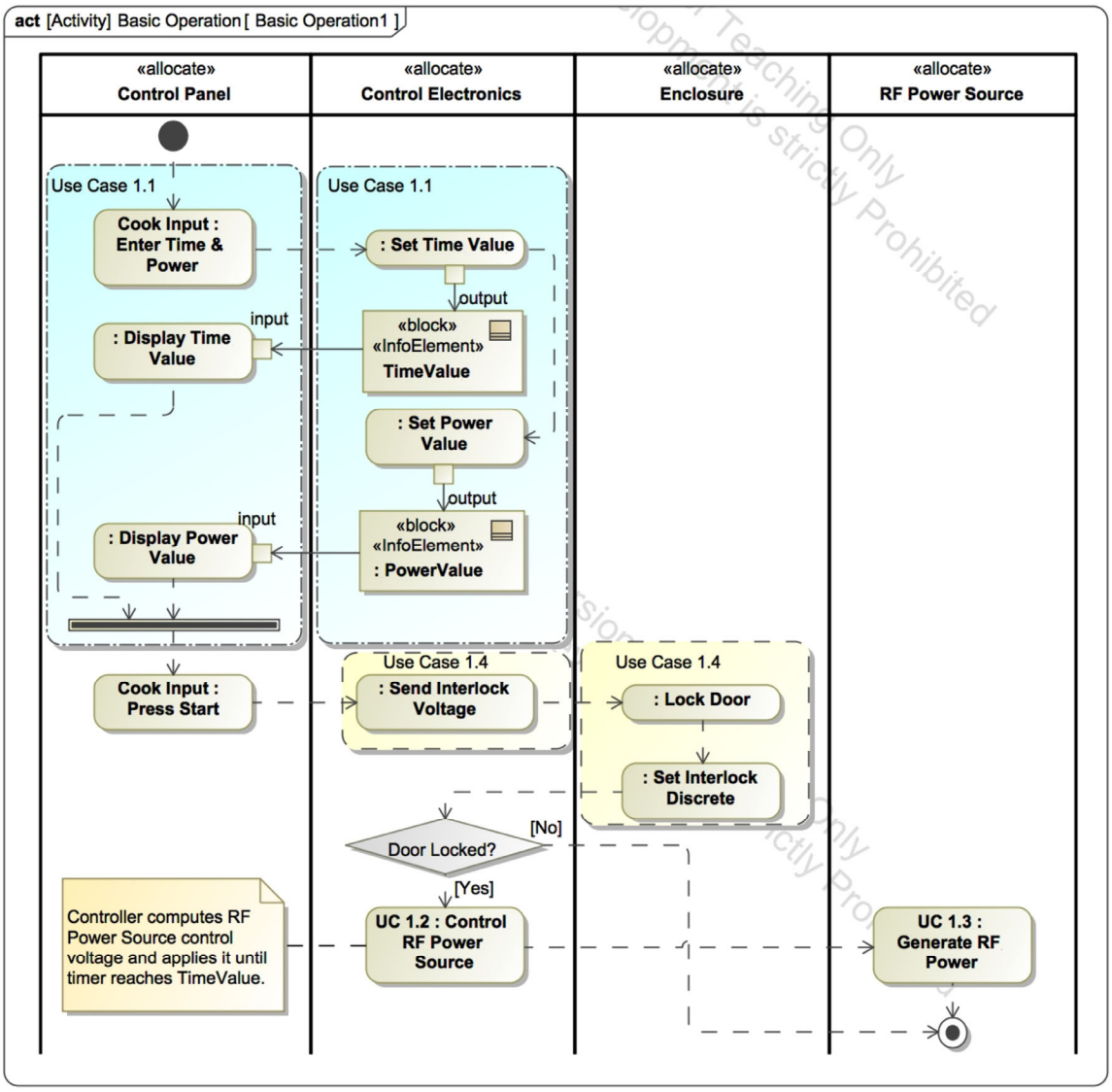


Figure 31. "Day-in-the-Life" Basic Operation of the Microwave Oven.

11. *Modeling Stateful Behavior* Demo 8

A. A State is a condition during the lifetime of an object or an interaction during which the object meets certain conditions, performs an action, or waits for an event. Another way to say this is that a State is a behavioral configuration of an entity such as Off, In Test, In Maintenance, or In Operation. The system or

other entity can transition from one State to another in response to events, commands, messages, or other circumstances. A State commonly prescribes certain Actions, parameter values, responses to events, etc. as well as limiting the other States to which the system can transition next. The key point is that the current State is determined by the system's history. All this constitutes "stateful" behavior. By contrast, when what the system does next doesn't depend on its history, the behavior is "stateless." In general, transitions are initiated by triggers, which are modeled as Events, and so this is also referred to as "event-driven" behavior.

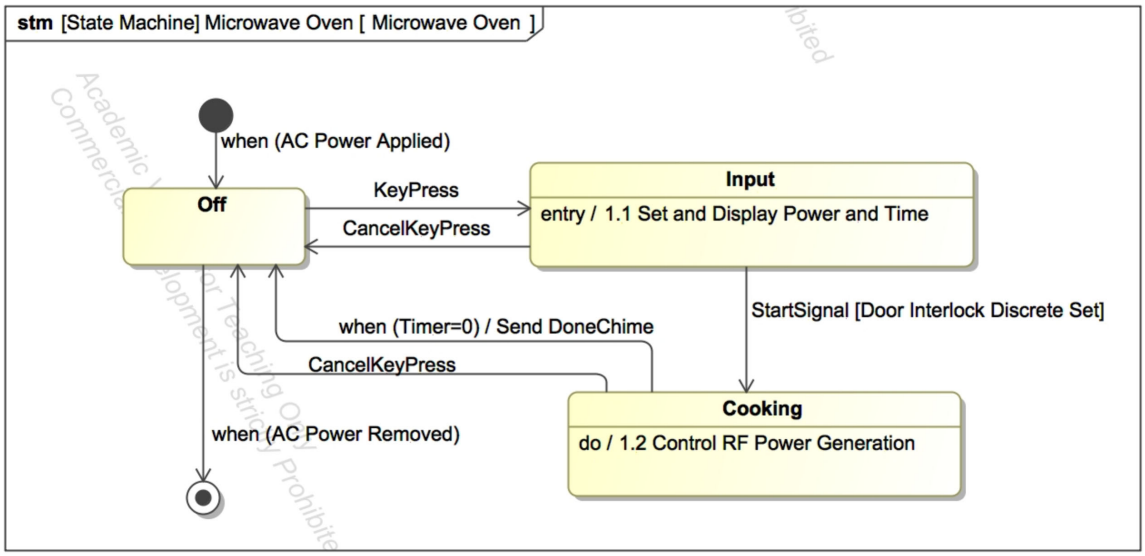


Figure 32. State Machine Diagram

B. Stateful Behavior is modeled using State Machine Diagrams (STMs) and is often associated with individual Blocks or a group of interacting Blocks. In fact, an STM is frequently the Classifier Behavior of a Block. Fig. 32 is a simple STM for the microwave oven. The steps to create this diagram are:

- Right click the Microwave Oven Block in the Containment Tree > Create Diagram > State Machine Diagram; the tool gives the new diagram a default name from the context Block, which you can change if desired.
- Build the Diagram from the Tool Palette:
  - Start by putting States in the Diagram from the Tool Palette, naming them, and defining their activities. In this example, the oven is initially in the Off State, transitions to an Input State when any key on the Control Panel is pressed, transitions to a Cooking State when the Start key is pressed, and transitions back to Off when the timer counts down to zero.
    - A state can have an Entry Behavior, which executes as soon as the State is entered; a Do Behavior, which executes in the State; and an Exit Behavior that executes just before the system transitions to another State. These are all optional, and they are defined in the

State Specification. The most common behavior is an Activity, but there are a number of other types.

- For basic architecture modeling, it's sufficient to select the types of behaviors and give them names to display in the Diagram. Right click a State > Specification, and scroll down to the Entry, Do, and Exit areas. For each behavior you want to define, click in the Value Field for Behavior Type, and make a selection; the most common choice is Activity. Then click the Name field and type in the name.
- A State Machine done this way shows the essential information, but won't simulate because behaviors are named but not actually defined. A more rigorous approach is to link State behaviors to defined behaviors, most commonly ADs. First ensure all these ADs have been created in the model. Then in the Name field, enter the exact name of the AD.
- For this example, the Input State models the Cook entering a time and power level, which is precisely Use Case 1.1. Enter the name of the Scenario AD for this UC in the Name field for the Entry behavior. No Do or Exit Behaviors are defined. The Diagram shows the AD hyperlinked to the State, and double clicking on the State displays the AD. Pressing the Cancel key takes the oven back to off.
- The Cooking State aligns with both UC 1.2, Control RF Power Generation, and UC 1.3, Generate RF Power. It's sufficient to create a Do Behavior linked to the Scenario AD for UC 1.2 since all the RF Power Source cares about is whether a voltage is applied to activate it. In many STMs, State Behaviors don't align this cleanly with UCs, and some, especially Do Behaviors, may be quite complex. In these cases, you can create an AD that spells out the details and link it to the State Behavior. Once again, the AD is hyperlinked and can be opened by double clicking on the State. The oven returns to Off when the timer expires or the Cancel key is pressed.

- Next, add the Transitions:
  - Any Transition can have the following syntax:

    <Trigger>[Guard]/<Action>

    where the trigger is an event that calls for the Transition, [Guard] is a Boolean expression that must evaluate to True for the Transition to occur, and Action is one or more behaviors that execute in conjunction with the Transition. SysML defines a variety of Events for use in STMs, including:

    - Time Event - this can be after( ) to model a duration or at( ) for a specified clock time,
    - Signal Event - models a Transition responding to receiving a signal,

        - Change Event - when (<expression>models a Transition responding to a change in some condition, and
        - Call Event - call models a Transition responding to the calling of an operation of the STM's owning Block by another model Element.
    - All these details are entered in the Transition Specification.
    - An STM must have an Initial Pseudostate Node, whose Symbol is a solid dot. This is one of a variety of pseudostates defined by SysML. Other pseudostates include Final, Choice (Decision), Junction, History, and Terminate - see the references for details. The Initial Node tells the STM where to "wake up" when the behavior of its owning Block is initiated. Click Initial in the Tool Palette, click in the Diagram, then click Transition in the Tool Palette and drag from the Initial Node to the Off State. This Transition fires when AC Power is applied to the oven.
    - Similarly create a Final Node with a Transition from the Off State and the Change Event when(AC Power Removed).

C. STMs have many more semantic features, including Composite States, decisions, branches, and joins that may be useful in modeling complex behaviors. See Appendix B and the references for more details.

NOTE: The creators of SysML adopted the semantics of an existing state machine that was created by David Harel and is called a Harel Statechart.

12. *Modeling Stateless Behavior*

A. The remaining primary Behavioral Diagram is the Sequence Diagram (SD), which models the collaborative functioning of a group of model Elements. It is also referred to as an Interaction. Both ADs and SDs commonly model stateless behavior, although state information can be incorporated, e.g., to mark the state a model element is in at a particular point in the flow of a behavior. An SD can be a Classifier behavior but is more commonly associated with a group ("collaboration") of Blocks interacting to create a behavior. Key features of the SD include:
    - It portrays a set of model Elements that interact in some time-ordered sequence to create a behavior. Interactions are modeled as Messages, but these can represent virtually any kind of functionality or exchange. Time increases monotonically from top to bottom of the Diagram, but the time scale is not, in general, linear – vertical distance isn't necessarily proportional to elapsed time.
    - The Elements in the SD are Instances of various Classifiers. Their labels usually follow the :<Classifier Name> naming convention introduced earlier in this tutorial. Typical examples are Block Instances, Actor Instances, and Interface Instances. In the Diagram, these Instances are

ranged across the top with “Lifelines” modeling the times when they are active.

- Behavior occurring within an Element, e.g., a processor performing a computation, is modeled with a Self-Message to account for the time that the function consumes. Collectively, the characteristics of an SD allow at least an initial timing analysis of system behavior to be performed.
- SysML defines multiple Message types, including:
  - Basic Message – models the sending of something from one Instance Lifeline to another.
  - Call message – models the invocation of an Operation on the receiving Instance.
  - Return – models the sending of the result of a call to the requesting Instance.
  - Send or Call message – another way to model sending a request for an Operation.
  - Create message – models the creation of a specific Operation; in a software model, a Create message shows the instantiation of a software Block in memory so it can be used.
  - Destroy message – models the deletion of an Instance or Operation that is no longer required.
  - Synchronous and Asynchronous messages – distinguish between messages that halt the execution pending a reply from those that don’t, asynchronous messages are often used to model a return from a service request.
- An SD can be decomposed into smaller increments of behavior using Combined Fragments and Interaction Occurrences:
  - Both of these are modeled using frames within the main Diagram frame, and both have a header in the top left corner to show the type of Fragment or usage.
  - A Combined Fragment is like an SD within an SD. It spans some or all of the Lifelines in the main Diagram and some vertical distance representing time to capture a subset of the behavior. Combined Fragments can have a number of Operators that control if and how the sub-behavior is executed, including:
    - alt — Alternative Fragments, each with a Boolean guard; the Fragment with guard = True is the one that executes. executes.
    - opt — Optional; the Fragment executes if its guard condition = True.
    - par — Parallel; shows Fragments that run in parallel.
    - loop — The Fragment repeats as long as its guard = True.
    - region — Critical region; this Fragment has one thread executing at once.
    - neg — Negative; this Fragment shows an invalid interaction.

- sd Sequence diagram; this is the Operator for the entire diagram.
- Others include ignore, consider, strict, assert and critical; these are seldom used.

- Together, these Operators allow quite sophisticated decision logic to be modeled (and simulated) in an SD.
- An Interaction Occurrence (or Use) is a placeholder for an Interaction that is defined in a separate SD. It is marked by a ref Operator. It is labeled with the name of the separate, defining SD. As the behavior modeled by the main Diagram executes, when it comes to a ref frame, it invokes the defining SD and executes it.
- An Interaction Occurrence has two primary uses:
  - A sub-behavior that occurs in multiple higher-level behaviors can be defined once and then inserted wherever it's needed.
  - A complex SD can be broken up for easier reading and use by composing it of a series of Interaction Occurrences.

NOTE: Although a set of interacting model elements is sometimes referred to as a "collaboration," this is NOT a Collaboration Diagram from UML.

B. Fig. 33 shows a basic SD from the microwave oven model, showing the interaction among Domains in performing the basic cooking function. The steps to create this diagram are:
   - Create a new SD in the Behavior Package of the Containment Tree and name it Control Sequence.
   - From the Containment Tree, drag the four Domain Blocks into the Diagram; notice that they are converted to Instances. You can create an Instance and its Lifeline directly from the Tool Palette, but it's usually better to define the structure of the system and then use the Blocks in behavior Diagrams.
   - Add Messages by selecting Lifelines in turn and using the Message or Message to Self icons in the Smart Manipulator toolbar (or by selecting them in the Tool Palette) and then dragging from one Lifeline to another. Note that the origin and destination Lifelines are highlighted in blue if a valid message is being created. The tool numbers the messages, and you can fill in the names.
   - For now, it's enough to use basic Messages for interactions between timelines and the Message to Self Element for functions within a single Lifeline. SysML defines, and Cameo supports, a variety of specialized Messages which you can select in the Tool Palette. You can drag messages vertically to get a pleasing layout.
   - If the sending of a message is not assumed to be instantaneous, e.g., due to network latency, it may be important to account the time from sending

to receipt. This can be modeled with a Diagonal Message, which slants down from the source end to the destination end. The lower end can be dragged vertically to get a roughly proportionate interval on the time axis.

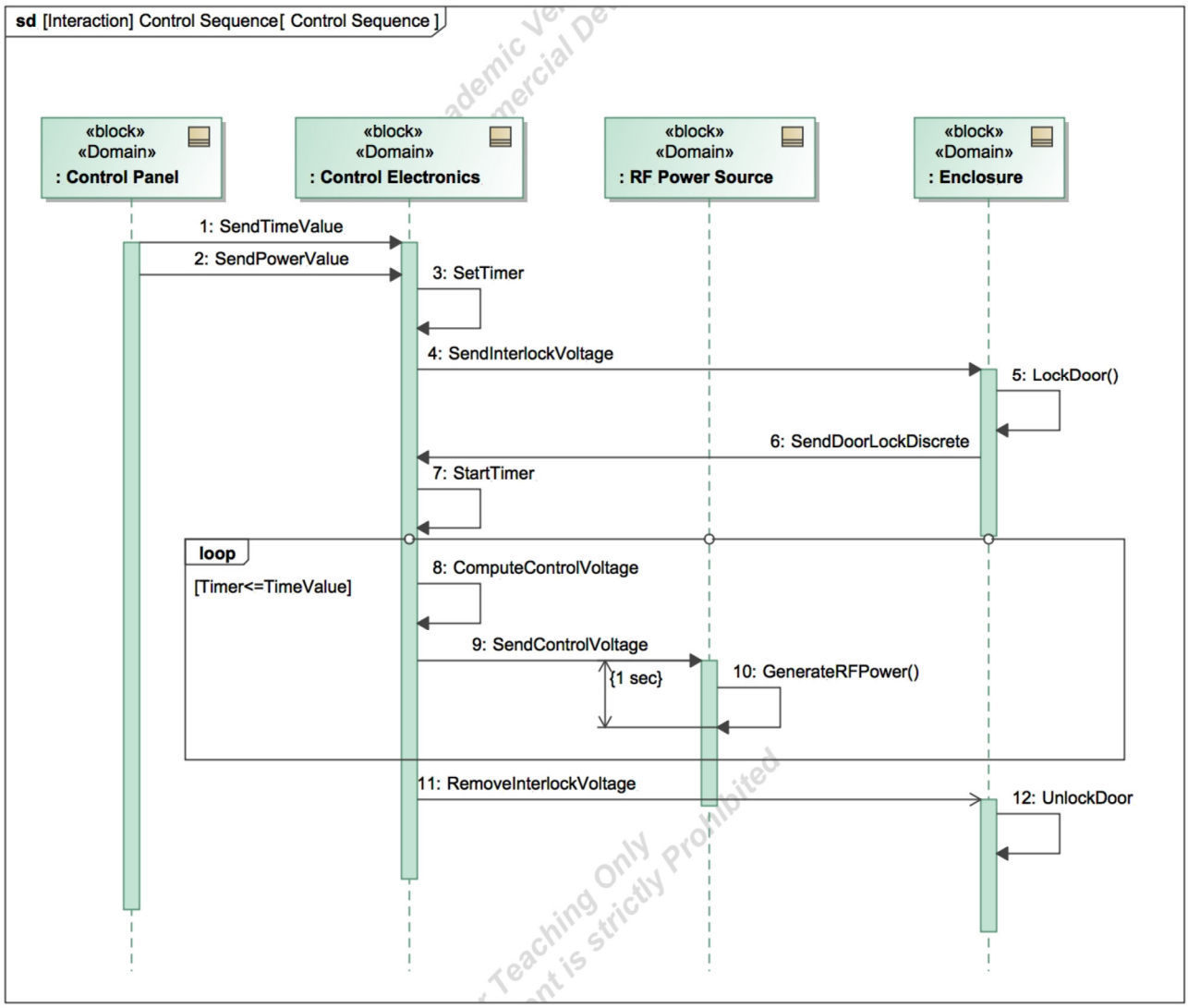


Figure 33. Sequence Diagram

- Sometimes it's useful to show the behavior in an SD being triggered by a message from outside. This is modeled with a Found Message that usually comes in at the top of the first Lifeline to become active.
- The tool creates Activation Bars (the vertical rectangles on Lifelines) to indicate that the Instance for that Lifeline is active. You can drag the bottom edges to get the duration you want.
- Once the basic SD is created, you can add Combined Fragments. Fig. 34 illustrates a Loop Fragment. These are available at the top of the Sequence Diagram area of the Tool Palette. Select a Fragment with the

desired Operator (you may have to use the down arrow to open a list), then drag in the main Diagram to enclose the Lifelines and Messages you want to be contained in the Fragment. Many Fragments require guard conditions, which are known as Operands. You can click in the space between square brackets and enter the expression (in this case, Timer<=Time Value), or you can right click the Fragment, open the Specification, and enter the expression in the Operand Field. You can also give the Fragment a name, but that's not usually necessary for a readable Diagram.

- An Alt Fragment chooses between two alternative behaviors. For a larger number of alternatives, you can use a sequence of Opt Fragments, and as the Interaction steps through them, the one whose Guard/Operand evaluates to True will execute.
- Because SDs are very useful for at least initial timing analysis, SysML defines Time and Duration Constraints. These are found near the bottom of the Sequence Diagram area of the Tool Palette. A Time Constraint defines minimum and maximum values of absolute time (e.g., system clock time) at a point in the behavior. The more useful Duration Constraint defines the time value between two points in the behavior, i.e., two instants on the time axis. You can select these in the Tool Palette and drag in the Diagram to create them.

13. Modeling Parametrics **Demo 9**

A. Parametrics allow you to model mathematical relationships among system/model elements (see Friedenthal, Moore and Steiner, Chap 8). You can:
   - Express relationships (functions) among variables,
   - Evaluate functions,
   - Measure effectiveness vs. criteria (e.g., system mass vs. allowable mass), and
   - Relate analysis to design by binding parameters of design equations to Properties of system elements.

B. Creating Parametrics is a two-step process:
   - First, define Constraints in one or more BDDs; these are reusable, e.g., in a Model Library.
   - Second, use Constraint Blocks as Constraint Properties in Parametric Diagram(s) with Binding Connectors to Parameters/Variables.
   - It simplifies things considerably to define Constraints and create Parametric Diagrams in a Block that contains all the independent (input) and dependent (computed) parameters as Values; you can then let the Cameo diagram wizard simplify the process of choosing the parameters and creating the bindings. For this example, everything happens in the

Processor Block of the Control Electronics Domain, so that's where everything will be created.

C. Fig. 34 is a Constraint Block from the microwave oven model that defines Cook Energy as the product of Power Level and Time Value (the cooking time). To create the Diagram:
   - Right click the Processor Block in the Containment Tree > Create Diagram> Block Definition Diagram > name it Cook Energy.
   - From the Tool Palette, select Constraint Block (a Block icon with curly brackets inside), and click in the Diagram. Give the Block a Name, Cook Energy.
   - Click in the Constraints Compartment and type in the equation.

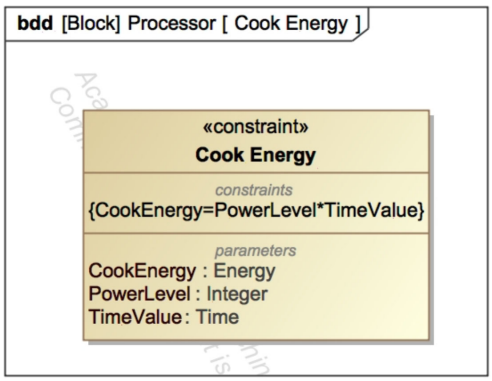


Figure 34. Constraint Block

   - Select the Block, click the Add Element button (white cross on a block dot) > Constraint Parameter and enter the first parameter, CookEnergy. If you haven't already defined a Value Type for Energy, create that in the Data Package. Open the Parameter Specification and in the Type field select Energy. Now you have both Constraints and Parameters Compartments on your Block.
   - Click the Add Constraint Parameter button (black cross on white dot) to add the other two Parameters. Give them the appropriate types. All of these are Value Properties of the Processor Block and will be given numerical values, either assigned as Default Values for the inputs or computed for the output.

D. Fig. 35 is a Parametric Diagram that has the new Constraint Block as a Constraint Property. You can build this Diagram manually:
   - Drag the Constraint Block and the Value Properties that are used as Constraint Parameters from the Containment Tree. When you bring in the Constraint Block, it becomes a Constraint Property.
   - Click on the Constraint Parameter icon in the Tool Palette (a small square with part of a Block boundary indicated) and click in the Constraint Property. Drag the square to the desired point on the boundary of the Constraint Property. Note that unlike a Port, it doesn't straddle the

boundary, but sits just inside it. Give it the name of the associated Constraint Parameter.

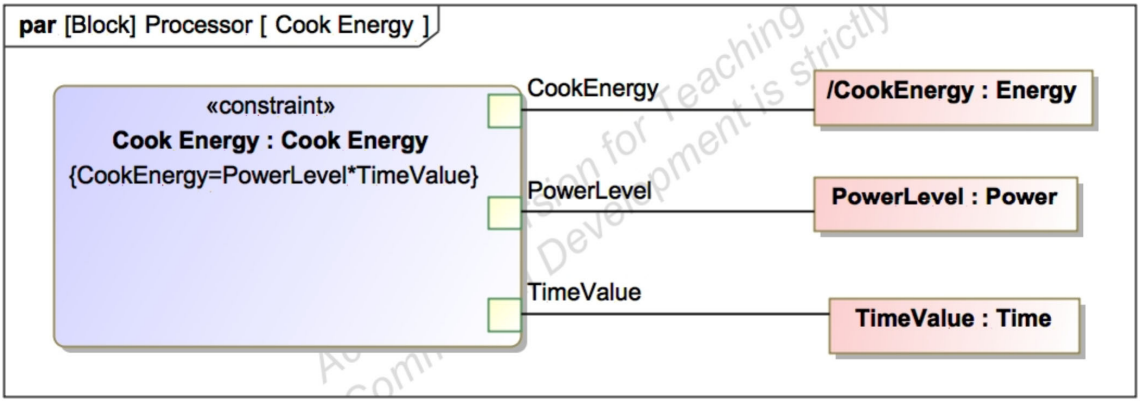


Figure 35. Parametric Diagram

- Wire up the Diagram with Binding Connectors from the Tool Palette. A Binding Connector guarantees the same numerical value on both ends.
- Notice that because Cook Energy is a Derived Parameter, the tool precedes the name with a forward slash.

E. Cameo provides a Parametric Equation Wizard that can automate the generation of a Parametric Diagram from a Constraint Block. Right click the Constraint Block > Tools > Parametric Equation Wizard, and use the window that opens to bind the Constraint Parameters to the Constraint Property.

F. To evaluate the function, give the input Parameters default values, Power Level = 750 and Time Value = 120, then simulate the Parametric Diagram to get the value for Cook Energy. Figure 36 shows the simulation window with the computed value of 90000 for Cook Energy.

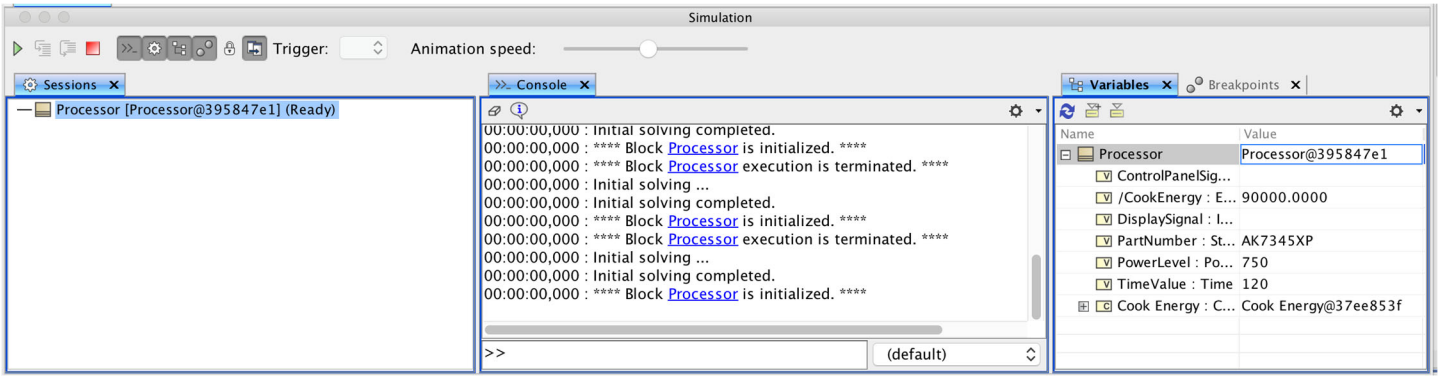


Figure 36. Computed Value

14. *Modeling Physical Architecture* Demo 10

A. The simplest way to incorporate physical ("point design") data in a model is to add it to the specifications of existing model Elements. Figure 37 shows this for the Microwave Oven Block; whatever detail is required to document each item of the actual system gets included or referenced.

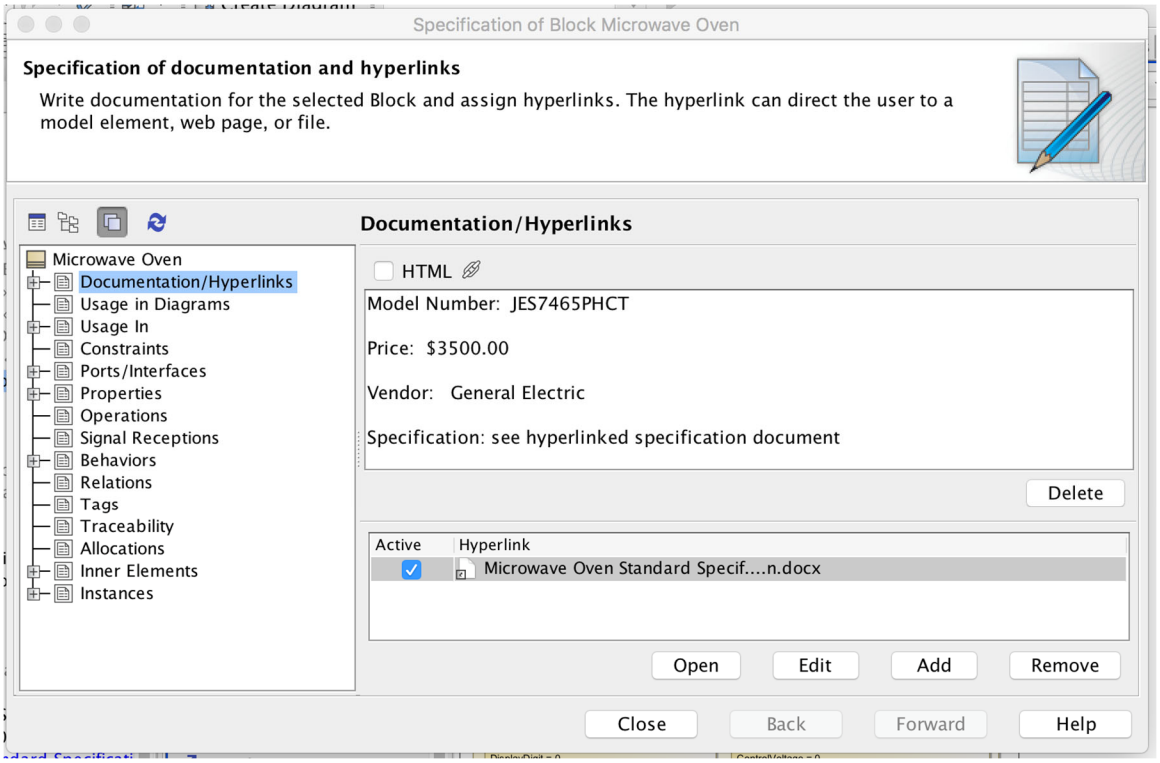


Figure 37. Physical Specification Data

B. Product and other data can also be incorporated as Attached Files with hyperlinks to Blocks or other model elements. The microwave oven example has a specification linked to the Microwave Oven Block.

C. You can also create a Block Instance:

- Select Block > right click > Tools > Create Instance
- Check/uncheck Values to be included > Next
- Select Package to hold Instance > Finish

Figure 38 shows an Instance of the Microwave Oven Block, with structure shown by nesting of Parts.

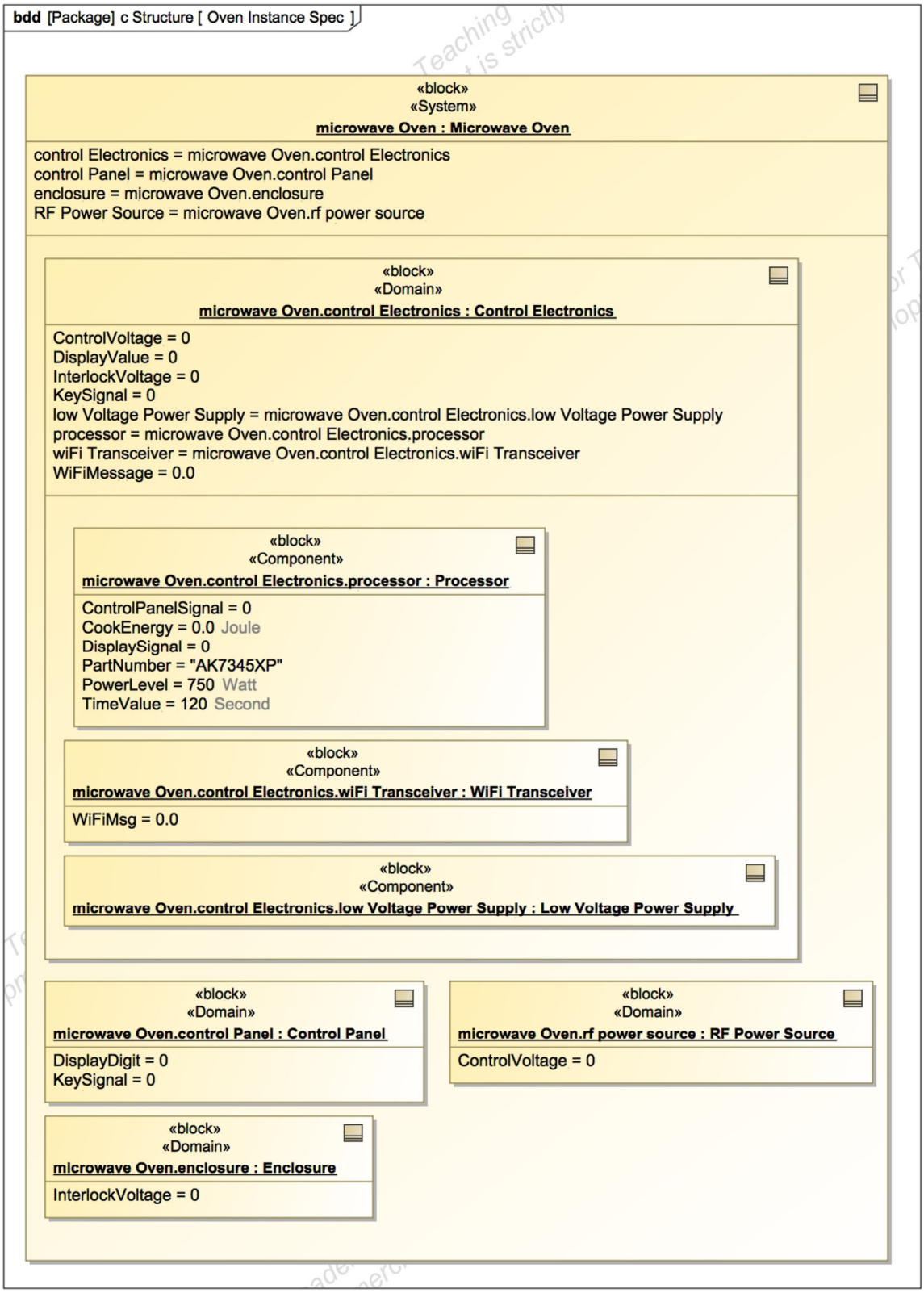


Figure 38. Block Instance

D. Performance Data can be added in several places in a model:

- Add detailed timing to SDs; in a real time system, this can include annotations from the Modeling and Analysis of Real Time Embedded (MARTE) profile of UML.
- Use Parametrics with Simulation to show compute/display performance parameters.
- Add {Constraints} in various places to model performance requirements.

Appendix A: Notes on the Systems Modeling Language (SysML)

A. Basic principles of Object-Orientation (OO):

- *Abstraction* – capture the essential, shared characteristics of an entity or group of entities
- *Hierarchy* – rank or order abstractions to capture common and unique features among entities
- *Modularity* – decompose complexity to enable understanding; localize effects of changes
- *Generalization/Inheritance* – define elements such as Classes whose characteristics apply to multiple more specialized elements; child elements inherit the characteristics of the parent
- *Encapsulation* – conceal information that does not need to be exposed to an outside entity; bundle related information and functionality within an entity
- *Interfaces* – clearly define interactions among entities and with the external environment; allow entities to interact by exchanging messages or other information across controlled boundaries
- *Polymorphism* – hide details of implementation behind an interface; allow the same service invocation to yield different results depending on circumstances (e.g., the data type of a passed parameter)

B. Classifier - any model Element that abstracts (or, equivalently, types) a category of system content and can type another Element by describing its structural and behavioral characteristics; this is a UML concept continued in SysML. Classifiers include Block, Interface, Actor, Use Case, Signal, et al. When an actual entity in the category typed by a Classifier is created, it is an Instance of the Classifier. This is the Type/Instance dichotomy that lies at the heart of OO theory. A Classifier can have an associated Classifier Behavior, which defines the normal or default functions that Instances of the Classifier can exhibit. A Behavior can also be "owned" by a Classifier when the Classifier initiates the Behavior but other Elements are also involved.

C. Behavior - SysML diagrams and supporting semantics are basically divided into methods for modeling *structure*, *behavior,* and *constraints*. Behavior models what a system does using Activity, Sequence, State Machine, and Parametric Diagrams[5].
   - Behaviors come in a variety of types:
     - Activity - general definition of system functioning in an Activity Diagram.

[5] Parametric Diagrams are officially Structural in SysML, but because they can be used for computation of behavioral outcomes, they are also, in a sense, Behavioral.

- Opaque Behavior - sometimes called a "black box" behavior, this conceals the specific of the Behavior; it's represented as a textual description in some language (other than SysML).
- Function Behavior - similar to an Opaque Behavior, but is constrained to not alter the state of an owning Block and to communicate only using parameters; it's often used to model a mathematical expression.
- Protocol State Machine - this specifies what behaviors can be invoked and when; it's generally used to model a network or other protocol.
- State Machine - this models stateful behavior where the system at any time is in a behavioral configuration called a State that depends on preceding Behaviors.

- As noted in B above a Behavior can an be Classifier or Owned.

D. Constraint - this is a kind of language extension that attaches additional semantic information to an element, which a correct design must satisfy; a Constraint expression is shown between { } curly brackets. These are modeled as Constraint Properties of a Block or in a specialized Constraint Block for use in a Parametric Diagram or Constraint Note to attach to an Element in a Diagram. Some practical aspects of Constraints include:
   - Expressed in natural language, Object Constraint Language (OCL), C++, etc.
   - Time Constraint - specifies min/max values of a time interval, shown as {t_1 . . t_2}, {t_1 . . t_1 + 3}; in a Sequence Diagram, the time interval icon is a vertical vertical two headed arrow between horizontal lines.
   - Constraints can be defined in the Constraints Property Group in an Element Specification; you can also edit in the Constraint Spec window. To display Constraints on a Diagram, set Show Constraint to true in Symbol Properties.
   - To create a Constraint, open an Element Shortcut menu in the Browser > Create Element > Constraint OR open Element Spec Window > Inner Elements > Create (define Constraint).
   - To apply a Constraint, open an Element Spec window > Constraints > Apply > in Selected Elements, choose existing Constraint from All Data > Add > OK.

E. Data Type- this is a special kind of Classifier whose Values have no identity; it defines the characteristics of information entities (data) within the system.
   - Predefined Data Types are provided by the tool, and include Boolean, byte, char, date, double, float, int, Integer, Real, long, short, void, and String; there are also Enumeration and Primitive types.
   - It's generally necessary to declare additional Data Types to model specific content of a system.

F. Event - this is an occurrence that may trigger effects, such as a transition between States of a Block or System.

- In an Activity Diagram, an Event is associated with an Accept Event Action; in a State Machine, it's associated with a Transition trigger. Events are also used in Protocol State Machines.
- An Event can be a Call Event, Change Event, Signal Event, or Time Event.
- The timing of a Time Event can be absolute or relative to the instant a trigger becomes active. To define a trigger in Cameo, open a Transition or Accept Event Action Spec window > Trigger > define the trigger.
- For an absolute Time Event in Cameo, open a Transition or Accept Event Action Spec window > set Is Relative to False > enter time value.
- For a relative Time Event, set Is Relative to True.

G. Flow - this models the exchange of something between Blocks; it's tied to an Association that models the connection. There are two key concepts:
   - A Flow Property models the *character* of a flow (information, energy, fluid, etc.).
   - A Flow Item models a specific thing that flows; it represents a specific usage of a Flow Property.

H. Interface - this specifies the externally-visible operations of a classifier without a specification of the internal structure [NOTE: Interfaces and Interface Blocks are not the same and are used differently in typing model elements.]
   - An Interface can be Provided (models the exposed operations of a Block) or Required (models external inputs required by a Block).
   - A Block is responsible for realizing its Provided Interfaces.
   - Most commonly, Interfaces are associated with Ports that model the places where Blocks interact with other Elements.

I. Profile - this is a construct that can be used to extend the basic SysML Metamodel by adding semantic content.
   - A Profile is a Package with Package Properties; its most common use is to hold model-unique Stereotypes
   - A profile is created at the root level (entire model) of the Containment Tree by right clicking the root Package > Create Element > Profile > give the Profile a suitable name, which can be the model name.

J. Stereotype - this is a way to extend an existing metaclass in the SysML Metamodel. It's represented as a label within guillemets, as <<text>>.
   - To set/edit the properties of a Stereotype, open the Stereotype Spec window > set the Property Values.
   - Tags - Stereotype Properties are Tag Definitions, which apply to an Element when the Stereotype is applied; these are used to define new meta attributes of the extended metaclass. they can be defined once, then applied everywhere the Stereotype is applied. They are used as regular Block Values:

- An Instance of a Tag Definition is a Tagged Value, or simply, a Tag; it holds extra information such as a Use Case precondition, project status, or code generation constraints.
- A Tagged Value consists of two parts: name and value (e.g.: Author = Joe).
- To create a Tagged Value, open the Stereotype Spec window > Tag Definitions > Create > add new Tag Definition > Close
- To edit a Tagged Value, open the Element Spec window > Tags > select existing Tag Definition > Create Value > set the tag value using drag and drop; you can drag an element from the Containment Tree and drop in the Specification window tag value area.

K. Dependency - a relationship that models the fact that one Element (the client or dependent Element) needs something from another Element (the provider or independent Element) in order to function correctly. the dependence can be on resources, functions, content and many other things.
- A Dependency is shown in a Diagram as an arrow with a Vee arrowhead on the provider end and a dashed shank.
- There are many kinds of Dependencies, denoted using Stereotypes; they include <<access>>, <<import>>, <<use>>, <<allocate>>, <<refine>>, and many others. A special set of Dependencies is defined in SysML for use with Requirements, while <<include>> and <<extend>> Dependencies are used to show hierarchy or decomposition in a Use Case Diagram.
- Realization is a special kind of Dependency It is a specialized kind of abstraction relationship that models a situation where the client realizes or implements a specification defined by the provider. There are three flavors:
  - Interface Realization (most common) shown as - - -▷ is where a Classifier implements an Interface that conforms to the Interface Spec in the provider.
  - Realization just a solid line representing a Classifier-Interface relationship.
  - Substitution is marked by a <<substitute>> Dependency to show a Classifier-Classifier relationship in which runtime instances of the substituting Classifier are substitutable where those of the Contract Classifier are expected.

L. Dependencies involving Requirements - although System Engineers pay great attention to allocating requirements to design, SysML explicitly does NOT use the <<allocate>> Dependency with requirements. Instead, a set of more specialized Dependencies are used:
- <<Satisfy>> - establishes a relationship between a requirement and the system Element(s) that are responsible for satisfying it.
- <<Copy>> - create a duplicate for use elsewhere.

- <<Refine>> - documents the rationale for using a source of information to improve or clarify a requirement.
- <<deriveRqmt>> - documents the rationale for creating a new (usually lower-level requirement by analysis of a higher-level requirement.
- <<Trace>> - documents one or more aspects of the architecture that result in a requirement being defined.
- <<Verify>> - establishes a relationship between a requirement and a Test Case whose results will be use to verify satisfaction of the requirement .

Appendix B: Additional Cameo System Modeler Tips and Tricks

A. The Tool Palette in Cameo adapts to the Diagram type being created or modified. A very complete set of icons is provided for each Diagram. Some of these have variations, indicated by simple down arrowheads to the right of the icon. Examples include directed vs. non-directed Associations and a list of specialized Requirements Blocks.

B. Creating early model content:
   - Key concept - early in model creation, define (ONCE!) things that will be widely used, then use them as often as needed.
   - Typical examples include:
     - Define all structural elements first as Blocks in BDDs, then use them as Parts in IBDs, etc. (NOTE: the SAME system piece will appear in the Browser both as a Block and as a Part Property).
     - Define Actors for User Roles and for External Entities; a good early diagram to create is a BDD showing the context of the system, including explicitly declaring the system boundary and making Associations to the external entities with which the system interacts.
     - Define at least high level Use Cases as early as system behaviors can be identified.
     - Define Data Types, Value Types, and Units.
     - Define Foundation Classes in a Conceptual Data Model (CDM), then define System Data Classes in Logical Data Model (LDM) Diagrams as these entities are discovered.
     - Similarly model Flow Items, Interfaces, etc. as they are discovered.

C. Enhancing Diagrams - many simple techniques can add greatly to communicative power and understandability of a Diagram, including:
   - Use the Symbol Properties menu to adjust line and fill colors, fonts, etc.
   - Use Notes and Comments to convey key information about a Diagram and its content.
   - Use graphical annotations from the Common area of the Tool Palette; things like separators, outlines, and text boxes can visually organize a Diagram.
   - Use Legends to define styles (e.g., colors) for various kinds of Diagram content (e.g., a line representing power flow vs. one representing network traffic); these can be defined once and then used in many diagrams.

D. Attaching Files to a Model - select a file in the file system, drag onto an Element in the Containment Tree (usually a Package, but any Element will work), select Create Attached File. A good general practice when using MBSAP is to create an Attached Files Package at the root level of the model, put the attachments there, then hyperlink them to the model Elements they relate to.

E. Displaying Multiple Diagrams - open the Diagrams, right click the diagram tab > New Horizontal Group or New Vertical Group.

F. Inserting Multiple Elements of the same type in a diagram - select the desired shape shape in the tool palette, hold down shift while clicking multiple times in diagram OR press Z to make the button "sticky," insert the desired number of Elements, and press Z again to cancel.

G. Deleting Diagram Content:
   - Delete - pressing the Delete key or selecting Delete from a menu deletes the item from the model; generally, the tool asks you to confirm you want to do this.
   - Select the Element and click the Edit menu > Delete from Diagram; this eliminates the Element from the Diagram but not from the model.

   NOTE: ANY deletion in an Activity or State Machine Diagram deletes the Element from the model

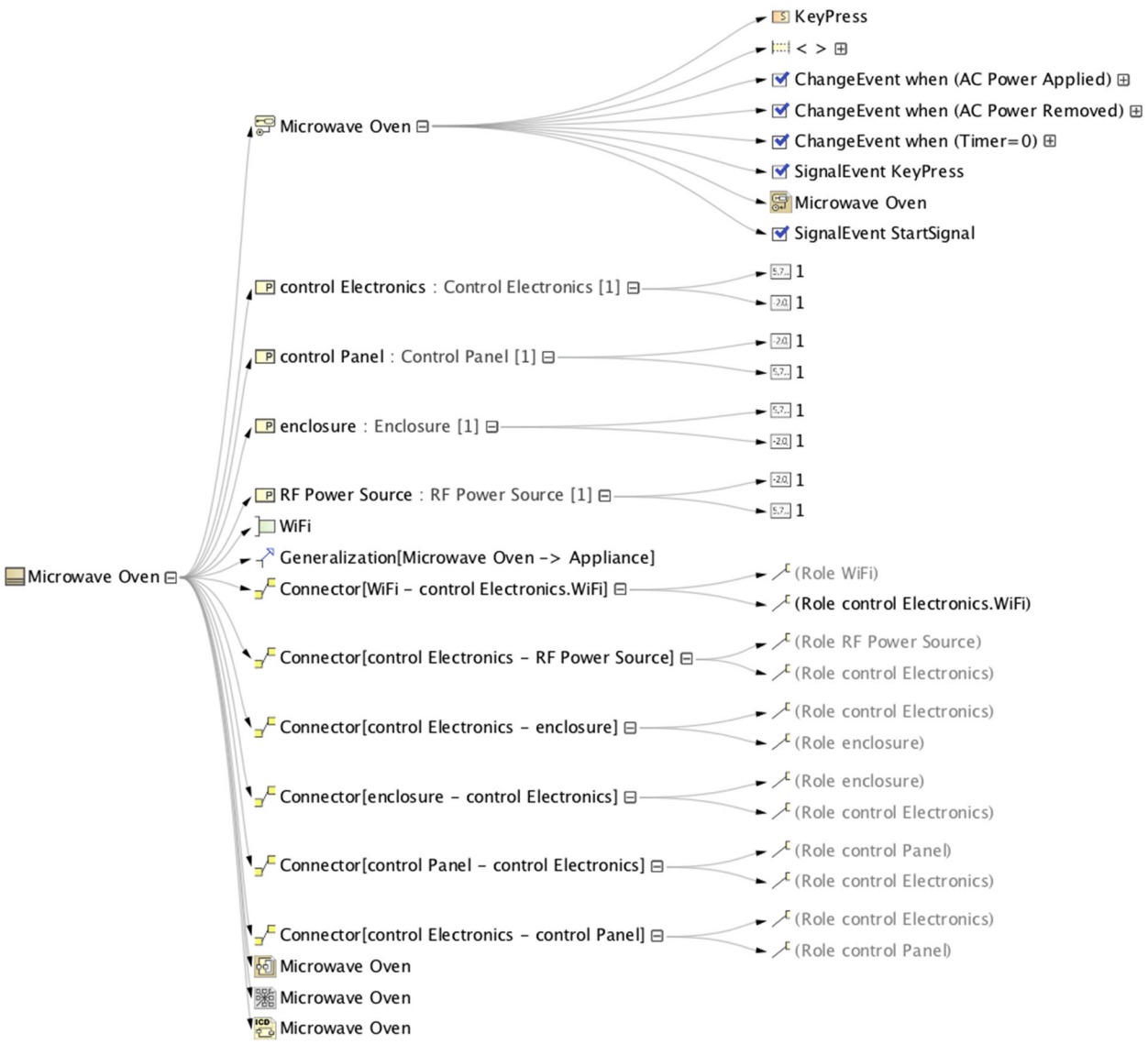
   - To delete a Path, select the Path and press the Delete key to remove it from the Diagram; press Cntl+D to remove from the model.

H. Creating a Path:
   - Select the desired Path in the tool palette, click on the source Shape in the Diagram, and drag to the destination Shape OR

     Select the Path in the Smart Manipulator for a Shape and drag to the destination.

   - To auto-route a path rectilinearly, select the Path, and click  in the Smart Manipulator tool bar; you can then drag segments of the Path around to create a pleasing layout and avoid Paths crossing Shapes.
   - To chose the style of Path corners, right click the Path > Symbol Properties > Select/Deselect Rounded Corners.

G. Creating a Relation Map - a Relation Map summarizes the structure of a model or selected model Element:
   - Select a Block in the Containment Tree that this will be the Context of the diagram.
   - On the toolbar, click Create Diagram button and select Relation Map.
   - In the Relation Criteria field, click . . . and scroll down to select Owned Element.
   - The following figure shows the result for the microwave oven Block.
   - This can also be used with requirements and other kinds of model Elements.



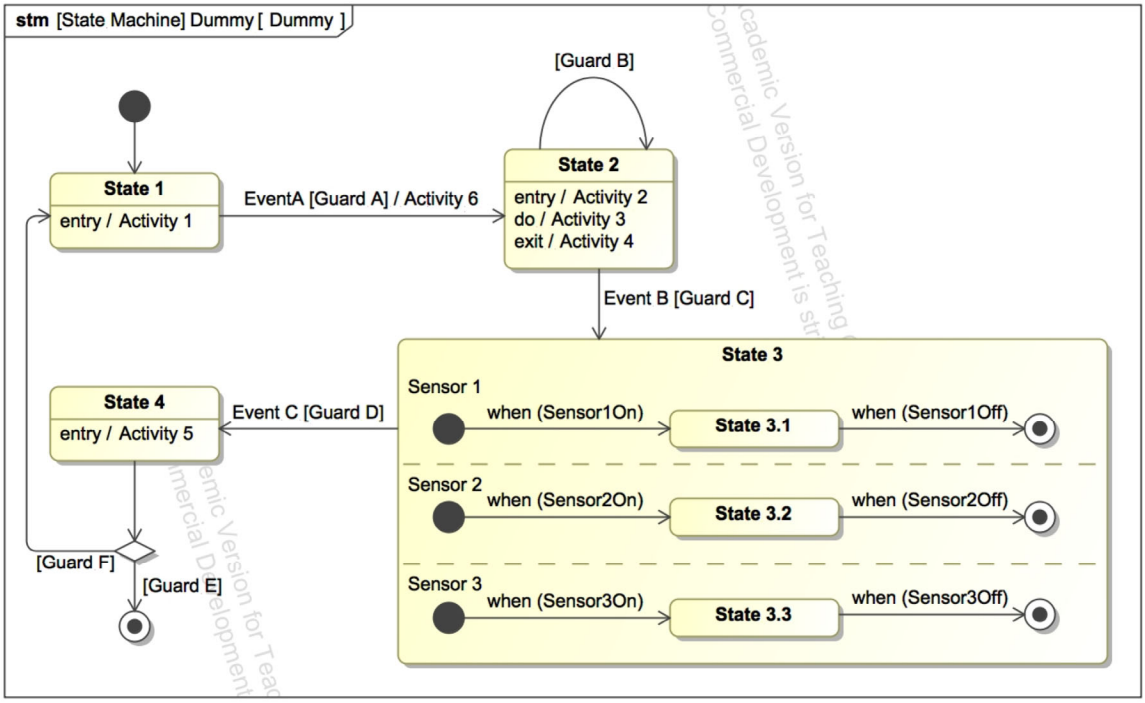
H. Additional State Machine Semantics: the following figure shows some more complex behaviors modeled in a state machine. The key points are:

- Transitions are controlled in the usual way by labels with
  <Trigger>[<Guard>]/Effect

- State 2 has a Transition to Self with Guard B. This creates a loop. Upon completion of its Actions, the State repeats, starting with its Entry Action, as long as Guard B evaluates to True.

- A simple State models a system condition with associated Operations/ behaviors and Transitions. A Composite State can have one or more Regions containing sub-States; these can only be active when the enclosing State is active, so a Transition out of the enclosing State "turns off" all the sub-States. A Composite State can have multiple "Orthogonal" Regions, each of which has

an active State that is independent of the others. The enclosing State can also have Entry, Do, and Exit behaviors that execute independently of any sub-States. State 3 is partitioned into Regions, labeled Sensor 1, Sensor 2, and Sensor 3. In these Regions, the associated sub-States become active when the Guards on the Initial Transitions are true and inactive when they change to the opposite condition.



- The Transition out of State 4 goes to a Decision, and the behavior either terminates on a Final State Node (the bullseye) or goes back to State 1. The Initial Node, the Decision or Choice Node, and the Final Node are examples of Pseudostates which can be used to model complex behaviors. They include Junctions (separate flows recombine), Joins (output transitions are evaluated when all input transitions are present), Forks (all output Transitions are executed when the single input transition is present), History (returns a Region to the State it was last in), Terminate (terminates the behavior), and others.

- To create the content of the figure:
  - To associate an Effect/Activity with a transition, first create an Activity Diagram, then drag the Activity from the Browser to the Transition.
  - To associate behaviors with a State, first create the Activity Diagrams, then drag an Activity to the symbol for the State. Select Entry, Do, or Exit to set which kind of behavior this is.
  - To create Orthogonal Regions in a Composite State, first create and name the Composite State from the Tool Palette. (Note that you can also choose Orthogonal State and Submachine State; these are beyond the scope of this tutorial.) Right click the State and select Add New Region from the

Shortcut Menu. If you want to label the regions, first go to the Symbol Properties menu of the enclosing State and click Show Region Name to make it true. Then in the Containment Tree, expand the enclosing State and type in Region labels (the icon is two little dashed lines between two vertical solid lines).
    - Create the content of each Region like any other State Machine.
    - For a Decision /Choice or other Pseudostate, select in the Tool Palette and click in the diagram.

I. Diagram Legend:
- This is a great time saver while achieving common graphical conventions across multiple diagrams. You can specify line and fill colors, path styles, etc. then apply these selectively to Symbols in any diagram.
- A Legend is itself a model Element, located in the Containment Tree. Once created, it can be dragged into any diagram and used to quickly set graphical properties of Symbols.
- See the MagicDraw User Guide for the steps to create and apply a legend.

J. Using Miscellaneous Cameo Tricks:
- To select Line Jumps, right click the Diagram > Diagram Properties > change the Add Line Jumps To property; you can make horizontal or vertical path crossings have jumps which can make a crowded diagram easier to read.
- To control contents of compartments, right click an Element > Symbol Properties > scroll to the bottom and edit the Compartments section OR

  Select a Shape and click the  icon in the upper left corner, then use the menu to edit compartments.
- To select multiple Elements in a diagram:
    - To select all Elements, press Cntl+A.
    - To select all Elements of the same type, press Alt + click on Windows or Option + click on a Mac.
    - To select a group of specific Elements, press Shift click on each Element.
    - To select adjacent Elements, click in an open are of the Diagram and drag the selection box across the Elements.
- To copy text or an image to a Diagram:
    - Select the item in the computer file system > copy (Cntl+C or Command+C)) > go to Cameo and open the Diagram > paste (Cntl+V or Command V) > choose Paste Special option.
    - To associate an image with a Shape in a Diagram, select the image > drag to the Shape (this creates a value for the Image Property of the shape) OR

      Start in the Image Property Value Cell of the Shape Specification window and click . . . to browse to a desired image.

- An image or icon can be displayed for certain Types, including Call Behavior Action, Swimlane, Object Node Activity Parameter Node, Collaboration Use, Instance Specification, Part, Lifeline; in any case right click the Shape > Symbol Properties.

- Drag/Drop/Copy/Cut/Paste:
  - Click drag one or more Symbols to move it/them to a new location.
  - Drag/Drop items from the Browser to a Diagram.
  - Copy/Paste a Symbol (usually from one Diagram to another).
  - Drag/drop files from the file system to a Browser Element or Symbol in a Diagram (this creates a hyperlink; double click the file to open it).
  - Drag an Element to a property in an Element Specification window; e.g., drag a Block to the Type property in the Specification window of an Operation to define the return type of the Operation.
  - Similarly, drag property values from a Specification window to a blank space in a Diagram (creates a Shape), to an existing Shape (creates a Value), or to an Element in the Browser (creates a new Element).
  - Relationships can be drag/drop added to a diagram.
  - Drag an Element to a Note or Text Box; again, this creates a hyperlink.
  - In a Sequence Diagram, drag an Operation from the Browser to a Message (creates a Call Message).
  - In a State Machine Diagram, drag an Event from the Browser to a Transition (creates a Trigger).
  - In an Activity Diagram:
    - Drag an Activity, Interaction, or State Machine to an Activity diagram to create a Call Behavior Action.
    - Drag a Signal to an Activity diagram to create a Send Signal Action.
    - Drag a Signal to a Send Signal Action to set or change the Signal.
    - Drag a Signal to an Accept Event Action to set or change the Signal.
    - Drag an Event to an Activity diagram to create an Accept Event Action.Drag an Event to an Accept Event Action to set the Event.
    - Drag a Signal Event to an Activity diagram to create an Accept Event Action.
    - Drag a Property, Actor, Class or Instance Specification to an Activity diagram to create a Swimlane; Elements will be set as representative Elements.
    - Drag a Property, Actor, Class or Instance Specification to an Activity diagram by using the right mouse button; from the shortcut menu, make a selection to create a Swimlane, Central Buffer Node, or available Actions.
- To create graphics from a Diagram (this is very useful in exporting specific content from a model to a document):
  - To copy a diagram as an image, resize the Diagram so all of it is visible, drag a selection box across the diagram to select the frame and all its contents, then click Edit > Copy as BMP/EMF/JPG/PNG, and paste into external document.

- You can also select specific diagram Elements to copy as graphics and paste as external graphics.
- More elaborate image options are available under File > Save As Image.

- To Zoom a Diagram:
  - To size the Diagram to the screen, click View > Fit In Window OR press Cntl+W on Windows or Command+W on a Mac.
  - To enlarge or shrink the diagram, click View > Zoom In/Out OR use the Zoom tab in the lower pane of the Browser.
  - To restore the original size, click View > Zoom 1:1

Appendix C: Modeling Services

A. In a Service-Oriented Architecture (SOA), system capabilities are provided and consumed using a services mechanism through which a capability can be discovered, invoked, and delivered from a "server" to a "client." This is especially valuable when dealing with an enterprise or system-of-systems in which a set of individually heterogeneous systems and other resources must interact smoothly to achieve the objectives of the enterprise. By wrapping functions and content as services that are accessed at known locations via fully defined and standards-compliant service interfaces, each participating system can conform to a set of enterprise-level standards, conventions, operating modes, user interfaces, etc. and thereby achieve the required enterprise integration.

B. The Object Management Group (OMG) supports a SOA Modeling Language Profile (SoaML) of UML that enables effective SOA development. SoaML extends UML semantics primarily with service-related stereotypes and a set of SOA design patterns. The following paragraphs provide a quick introduction to services modeling in SoaML.

- The most useful SoaML constructs include:
  - <<Participant>> - a service provider or consumer; it is modeled as a Block (or Component) that can have specialized Ports called Service Points and Request Points. A Participant can have a ParticipantArchitecture.
  - <<Agent>> - a specialized Participant that can interact with and adapt to its environment; instances of an Agent share features, constraints and semantics.
  - <<MessageType>> and <<Attachment>> - these model service interactions using messaging (e.g., SOAP).
  - <<Capability>> - this can denote a general capability of a Participant, or a specific ability to provide a service.
  - <<RequestPoint>> - this models the use of a service by a Participant and defines the connection point through which a Participant makes requests and uses or consumes services.
  - <<ServicePoint>> - this is the complement of a Request Point and models a point where a service is offered by one Participant to others using well-defined terms, conditions and interfaces.
  - <<ServiceChannel>> - communication path between Service and Request Points.
  - <<ServiceContract>> - this applies to a specialized Block that formalizes a binding exchange of information, goods, or obligations between parties; in effect, it defines a service in an implementable and auditable way.
  - <<ServiceInterface>> - this defines the Interface to a Service Point or Request Point. A Service Interface types a role in a Service Contract. It can be bidirectional and can incorporate a Protocol.

- In basic situations, a Simple Service may be all that's required:

- Define an <<Interface, ServiceInterface>> Block for each service endpoint.
- Create a BDD with Ports on a <<Participant>> or other Block.
- Type a Port with an Interface.
- Create Required/Provided Interface and type with a Flow Specification.
- Right click Port > Display > Display Provided/Required Interfaces.
- Fig. D1 shows an example.

- A Complex Service Interface is commonly composed of multiple Interfaces and typically incorporates a Protocol. Fig. D2 shows an example.
- Once Participants and Interfaces are defined, they can be used to model the use of services in a SOA. Fig. D3 shows a very basic situation in which Provider and Client Participants exchange service invocation and response. The Collaboration Symbol (dashed oval) models the fact that these Participants operate within a Service Contract to exchange Service A.
- In Fig. D4, the simple client/provider relationship is expanded to suggest how multiple Participants can invoke services and receive the responses. Each Connection modeling a Service Channel should be decorated with roles such as Client or Provider to show how a given Participant is involved in the use of a given service.
- Service behavior is most commonly modeled in an SD because service transactions are generally message-based. Fig. D5 shows an example.

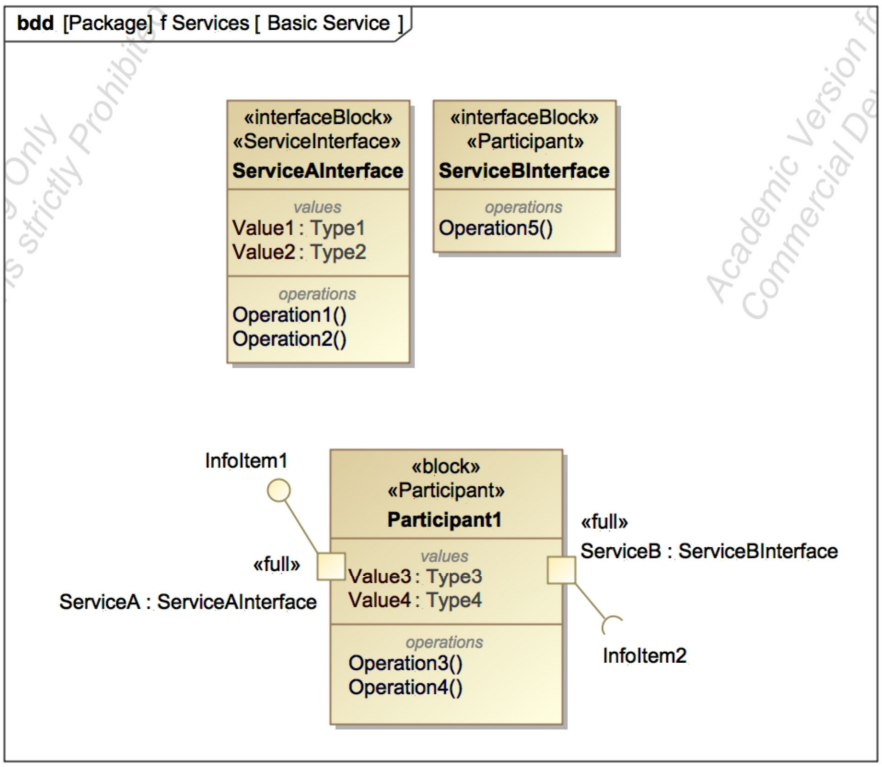


Figure D1. Simple Service

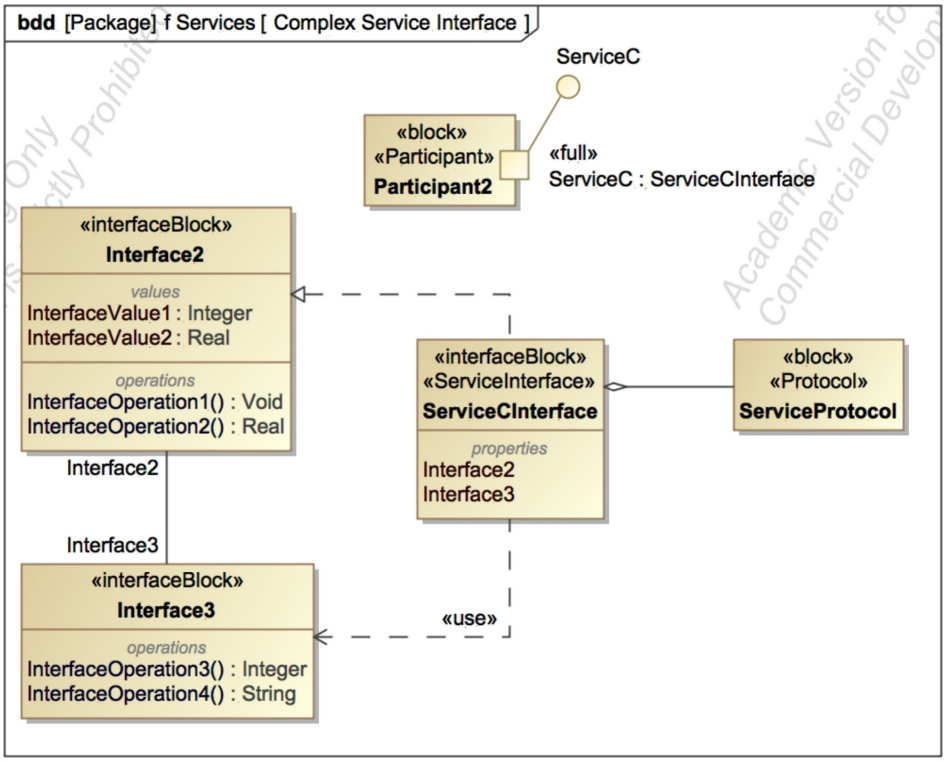


Figure D2. Complex Service

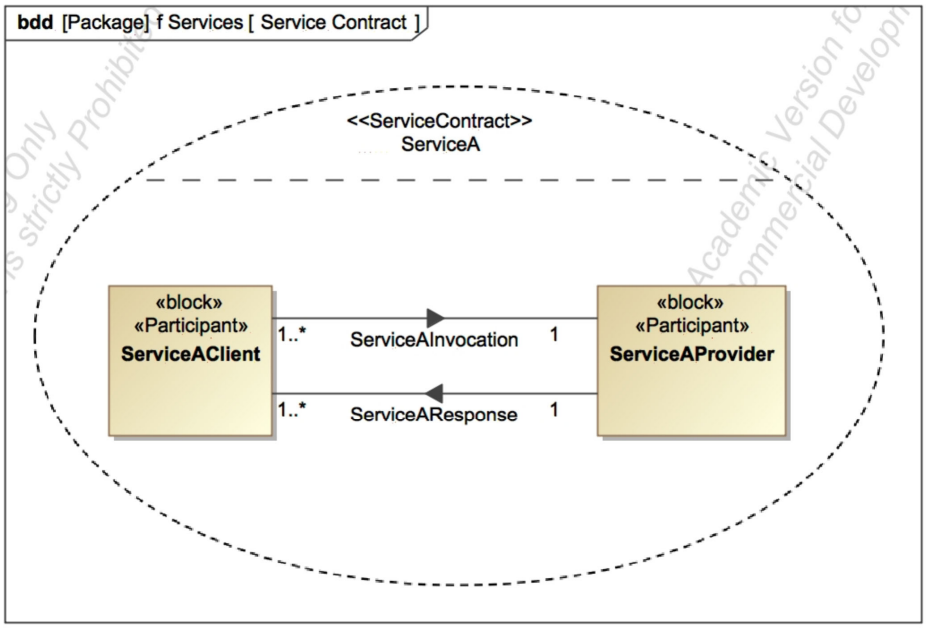


Figure D3. Simple Provider/Client Service Contract

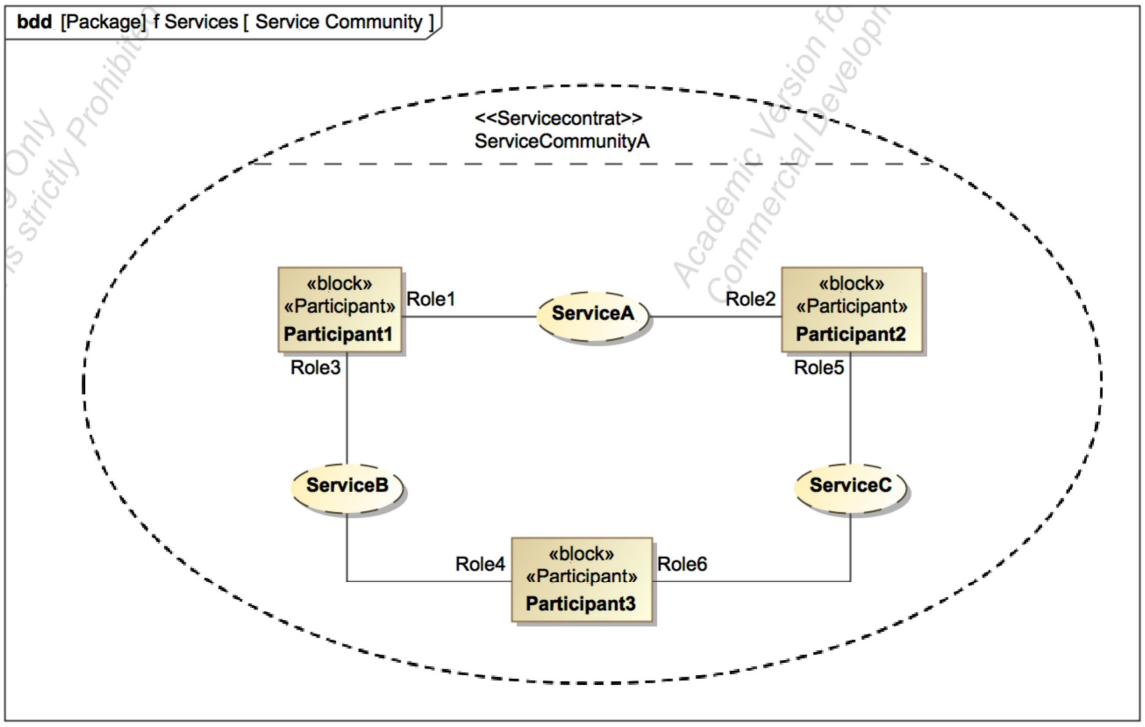


Figure D4. Service Community with Multiple Participants and Services

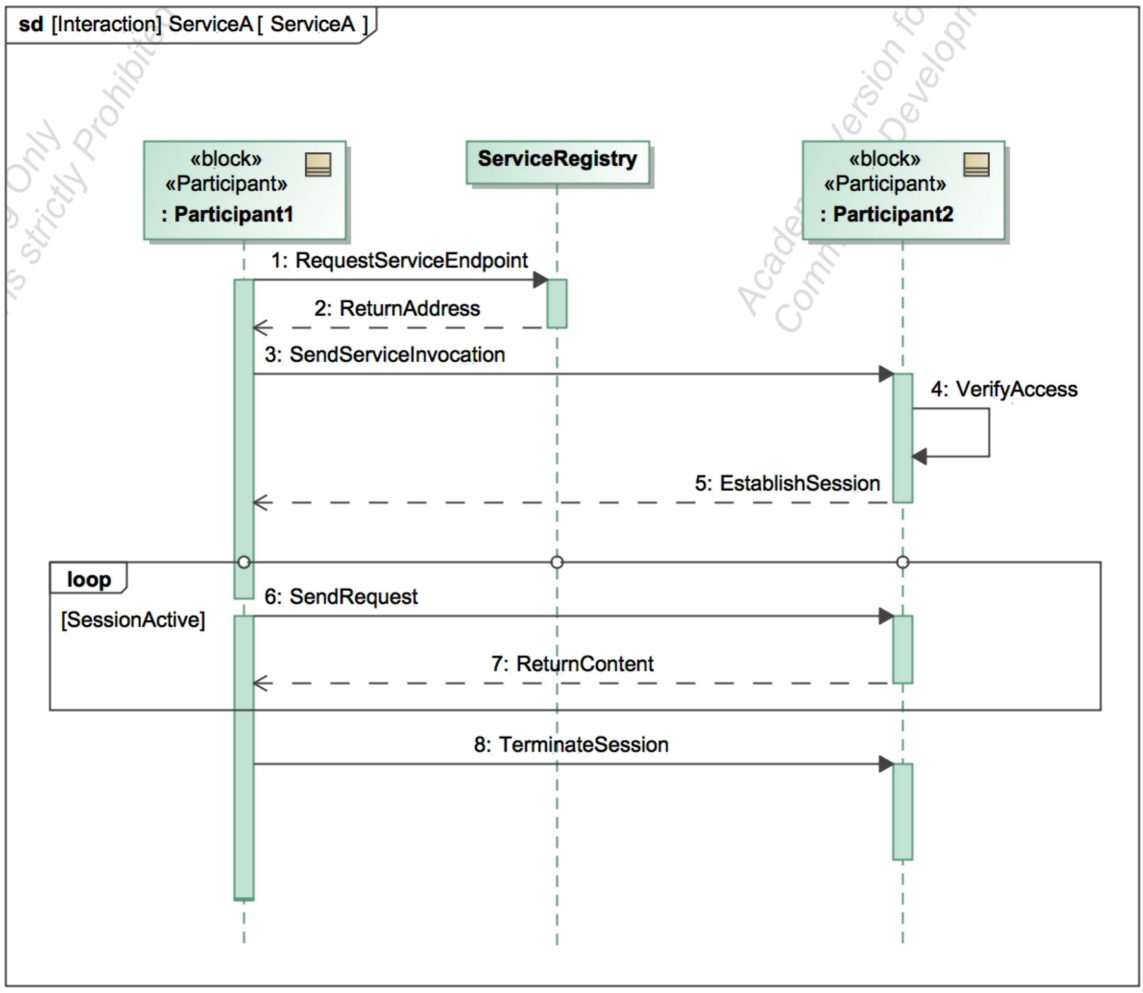


Figure D5. Service Transaction Modeled in a Sequence Diagram