Acme Lab 2:
Defining Styles in AcmeStudio

Introduction

By now, you should be familiar with the basic features of AcmeStudio to manipulate component-and-connector architectural designs. This lab will introduce you to some of the advanced features of AcmeStudio and in particular how to modify and create a family, edit visualizations, edit rules and properties, and check rules.

Example: A Three-Tiered Family

For this lab, you will augment an existing three-tiered family[1]. We will add a couple of new types, edit visualizations, and add several properties and rules.

Importing a Project from Zip archive

1. Download AcmeLab2.zip from blackboard.
2. In AcmeStudio, choose File→ Import.
3. Select Existing Projects into Workspace from the General category of the Import dialog box and click Next.
4. Select Select archive file and Browse to the location of the Zip file distributed with this tutorial and click Finish.

Modifying an existing Family

From the project AcmeLab2, open the three-tiered family under families/ThreeTieredFam.acme. The editor opens the family in the Family editor. We will work within this family editor for most of the lab.[2]

Browse types

Before we begin making changes, let's inspect the types defined in this family. First, click on the ThreeTieredFam (Press to select) button. This selects the whole family. In the lower-right quadrant of AcmeStudio, make sure that the Properties View is activated. In the Types tab you can see the inheritance tree for this family. It combines three families: RemoteCallReturnFam, LocalCallReturnFam, and RepositoryFam. This is because three-tiered systems combine two styles of call-return (local and remote) and the data-centered style (repositories).

[1] Note that AcmeStudio can work with built-in families (read-only) and locally created ones. You will be manipulating a local one that we provide in the AcmeLab2 project that uses some built-in ones.

[2] By convention, AcmeStudio looks for Acme families in the families/ directory of the project. This directory can have structure underneath, so that you can organize the families for the project. However, if the families are not located in this directory, they may not appear in the dialog boxes when you come to try to create instances of the families.

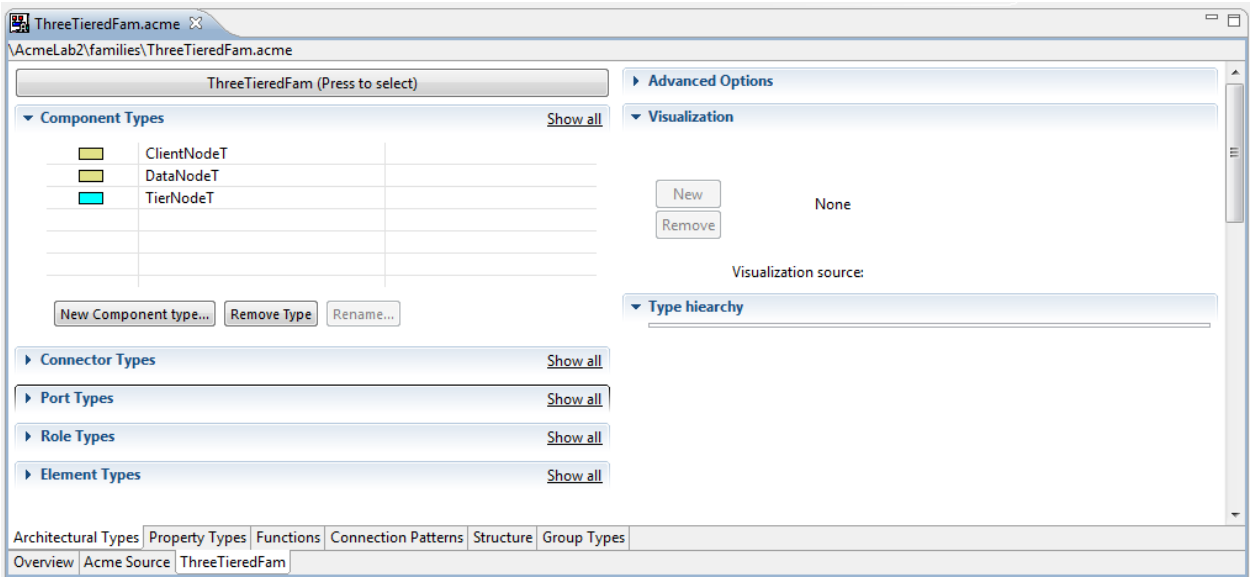


Figure 1: AcmeStudio showing the ThreeTieredFam opened in the Family editor.

Inspecting types in families is similar to inspecting instances, as described in AcmeLab 1. One way to do this is to via the Properties View. To browse a type, click on the type in the Family editor (e.g., DataNodeT) and look through its properties, rules, structure, types, and other attributes available in Properties View. Notice the Show all option for each type list. Clicking it will show all types inherited from other families. You will not be able to edit these types, but it is useful to know what other types are available, and whether they can be reused directly in your family (for example, we will use LocalCallPortT from LocalCallReturnFam without modification in this lab).

Another way to browse types is to use the Inspect type function, useful in cases when you cannot edit the family, such as a built-in family:

1. From the menu bar, Choose Family → Inspect Type…. The dialog box that opens shows a list of component, connector, port, role, and property types defined in the family.
2. Double-click on DataNodeT. Another dialog box appears showing different kinds of information about this component type, such as its properties, rules (constraints), sub-structure, inherited types, and Acme source.
3. Explore some of the other types in this family.
4. Click OK to exit this dialog.

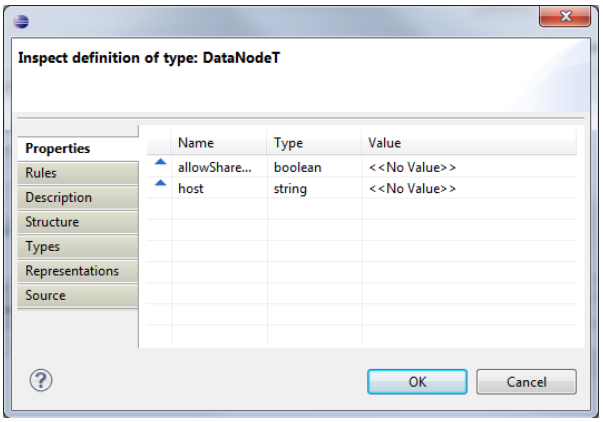


Figure 2: Dialog box for inspecting element types.

Change visualization

Presentation is an important aspect of architectural documentation. AcmeStudio has considerable flexibility in specifying the way a family should look. For example, you can choose geometric shapes, colors, icon decorations, label fonts and alignments, and port alignment policy for components and role alignment policy for connectors.

For this tutorial, the first thing we will do is to change the shape of the DataNodeT component type. Right now it looks the same as ClientNodeT, which is not good for documenting. We will change DataNodeT to a repository shape and its color to blue fill, orange outline, and white label.

5. Select DataNodeT. In the right column of the editor, the visualization preview will change to be the current visualization of the type. Click the Modify… button in the Visualization section to the right of the type. This will open the Visualization Editor dialog. Note that there are five tabs to this dialog: Shape, Label, Variants, Representations, and Port Policy. The first three tabs are standard tabs that appear in the visualization dialog of every visualized element type (components, connectors, ports, roles). The fourth appears for component and connector types. The fifth is specific to the element type and defines layout policies.
6. Let's change the shape. Back under the Shape tab, look for the Basic shape section and find the Stock image dropdown menu. Pull down the selection and choose Repository. Notice that the Preview shape has changed.
7. Next we will change the fill color to blue. Under the Color/Line Properties section, click on the button  . A color palette dialog box appears (On Mac, you can click the "Color Palette" tab). Select the blue color and click OK. Notice the change in Preview.
8. Next we will change the outline color to orange. Click on the button  under the Properties box. A color palette dialog box appears. Select the orange color and click OK. Notice the change in Preview.
9. Lastly we will change the color of the label. Click on the Label tab and in the Font Settings section click the button  . Choose the white color on the palette and press OK. The figure below shows what the result looks like if you return to the Shape tab.

10. Press OK on the visualization dialog box. The visualization for DataNodeT is now changed.

At this point, you might try experimenting with other forms of customizations for this and other element types.

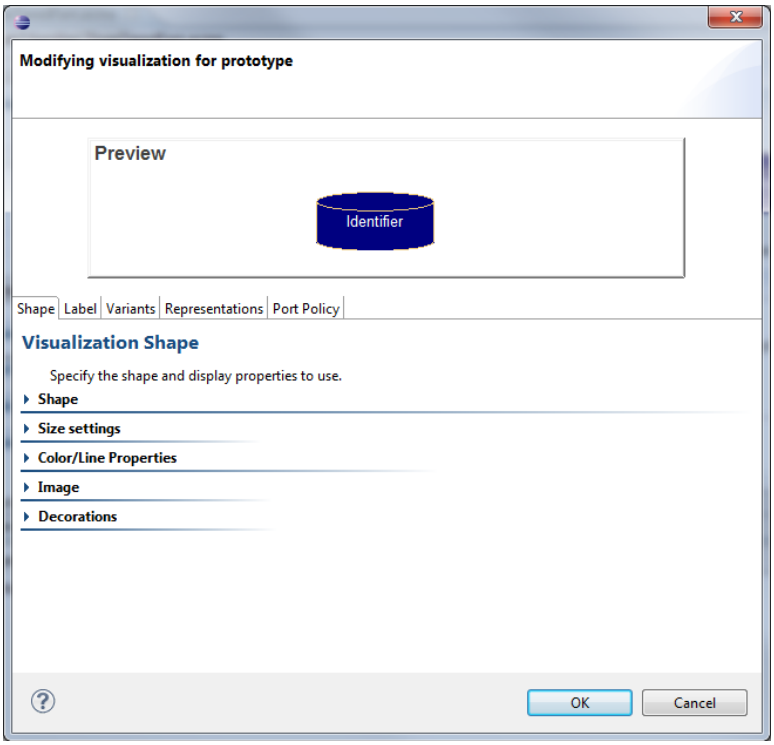


Figure 3. A possible result of DataNodeT visualization change. The exact colors are not important.

Defining Connection Patterns

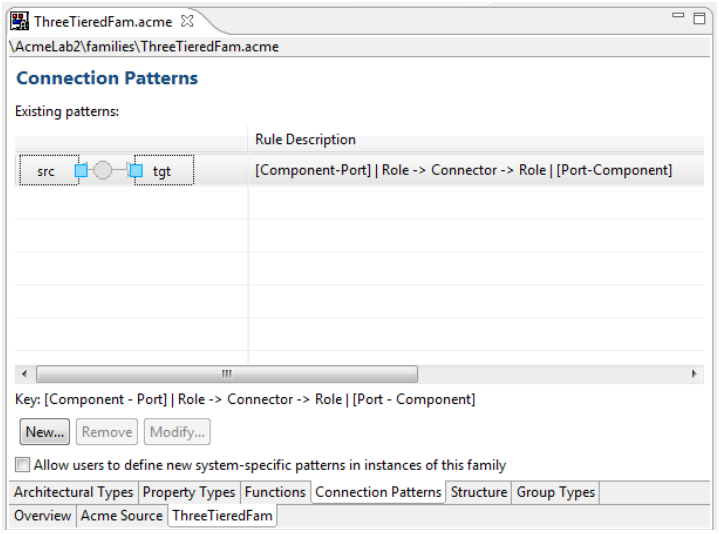


Figure 4. The Connection Pattern editor.

Connection patterns define shortcuts for connecting components using the Connect tool in a diagram. In AcmeLab 1, we used connection patterns to simply connect two filters

(without having to create the connector and attach the roles as separate steps). Definition of these patterns is done in the Connection Patterns part of the Family editor.

Notice that our family has three types of connectors: LocalCallReturnConnT, RemoteCallReturnConnT (both inheriting from an abstract CallReturnConnT), and DataAccessConnT. All of these connector types are inherited from other families.

We will construct a connection pattern that connects using LocalCallReturnConnT.

1. Select the Connection Patterns editor page in the Family editor (see Figure 4). There is currently only the default connection pattern involving Acme's most basic types like Component and Port.
2. Press the New… button to open the defining dialog box.
3. This dialog box allows us to define the connection pattern by selecting different types that will be instantiated by the pattern. We want to define a rule that creates a port of LocalRequestPortT at caller end, a port of LocalResponsePortT at the responder end, creates a LocalCallReturnConnT to connect them through appropriate roles. We want this rule to work on any component, so we will choose Any as the source and target component types. Do this by selecting from the combo boxes in the dialog, to make it look like Figure 5.

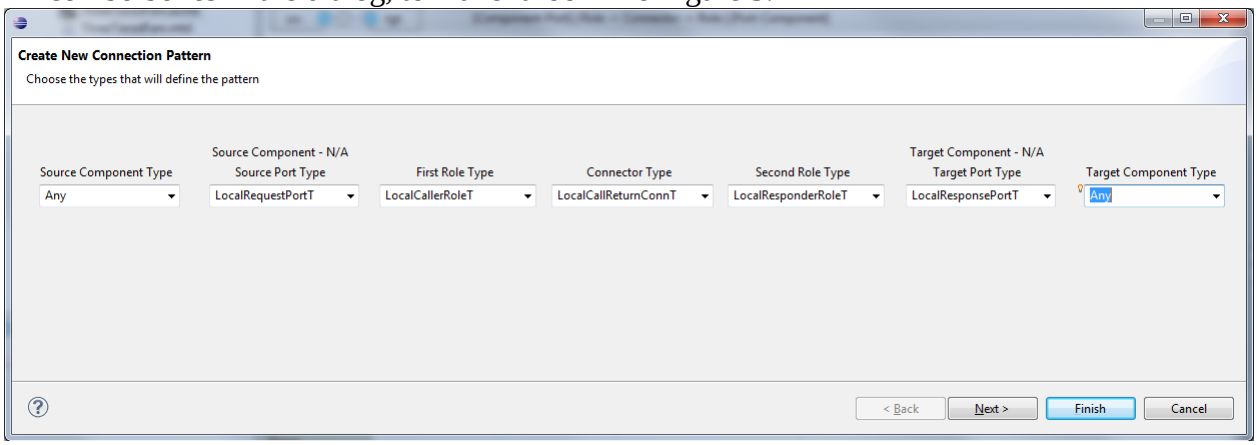


Figure 5. Defining a new Connection Pattern.

4. Select Finish. Note that you can assign patterns to the names of the elements created using this pattern. Here, we will just use the default, which uses the type name as the basis for names of instances. The new connection pattern can be seen in Figure 6. Note that there are two connection patterns actually created, where one with use as the source, the other with provide as the source.

You may also analogously define a connection pattern to connect the RemoteCallReturnConnT using remote ports and roles.

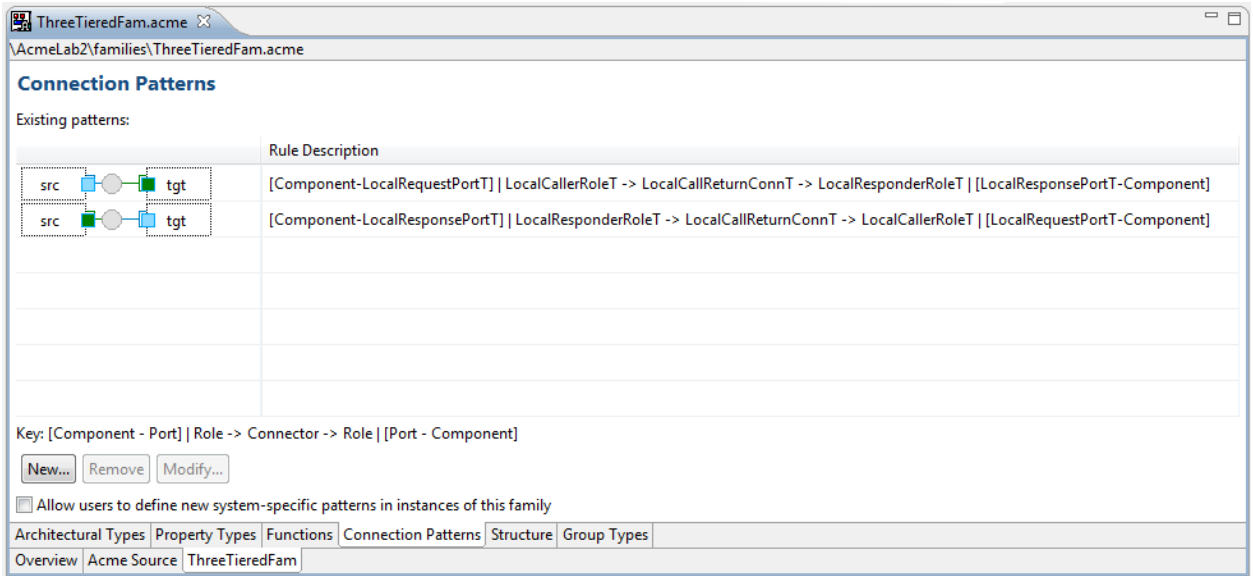


Figure 6. The newly defined Connection Patterns.

Add a new element type

You might notice that although this family is a three-tiered family, it is actually missing the component type definition for one of the tiers – the business logic tier. Let's add a LogicNodeT component type.

11. In the Architectural Types editor, select the New Component type… button in the Component Types section. A type entry dialog box appears.
12. Enter `LogicNodeT` for the Type Name.
13. Hold Ctrl (Cmd on Mac) and select the following supertypes: `TierNodeT`, `DataAccessorCompT`, `CallerCompT`, and `ResponderCompT`. We do this because a business logic node will be a tier node, will access data, as well as call other components as reply to calls. Different types let us partition these responsibilities.
14. Notice that you can click Next a few times to enter ports, properties, and rules. This time just click Finish.[3]
15. Notice that the newly added component type now appears in the Component type table.
16. Change the visualization of the LogicNodeT so that it is light-green filled with a black outline of size 2, everything else remaining the same. The outcome is shown in Figure 7.

[3] Note that if you make a mistake in defining the type, you can correct it (or make additions) using the Properties View.

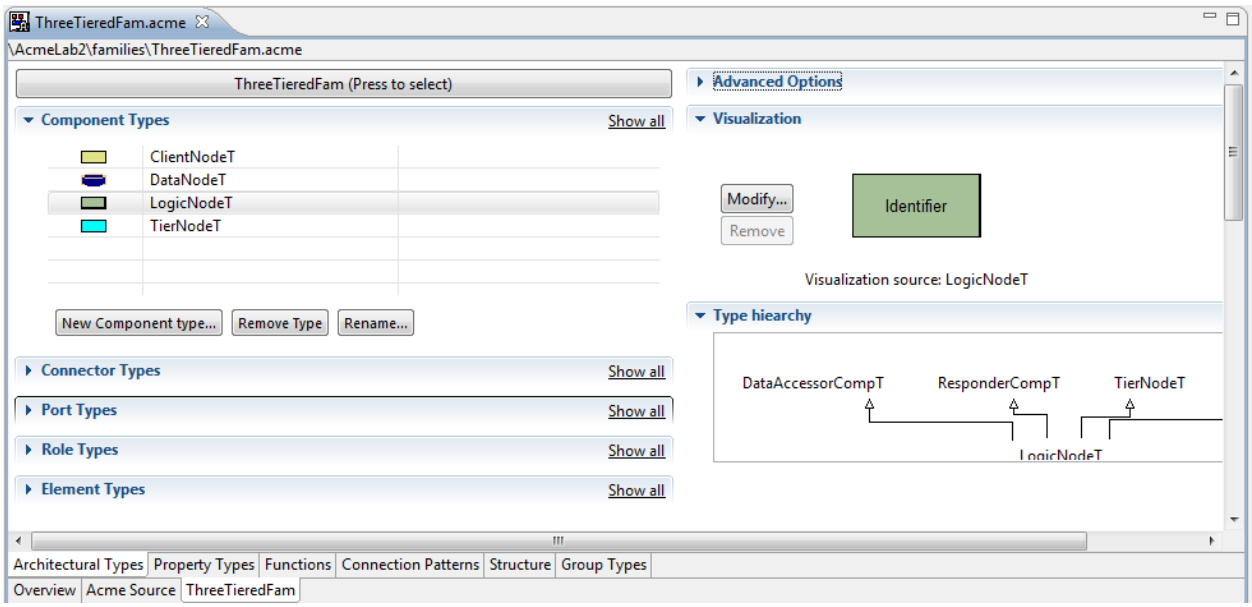


Figure 7: The resulting ThreeTieredFam description after adding LogicNodeT and changing its visualization.

Add a new property type

Since this is a three-tiered family, we want to make it explicit that a node belongs to a tier via a property for each node. This property can be defined using a property type that specifies an enum with three values. The node of each tier can then instantiate a property of this type assigning the appropriate tier value. Let's start by creating a property type (not the property itself yet!).

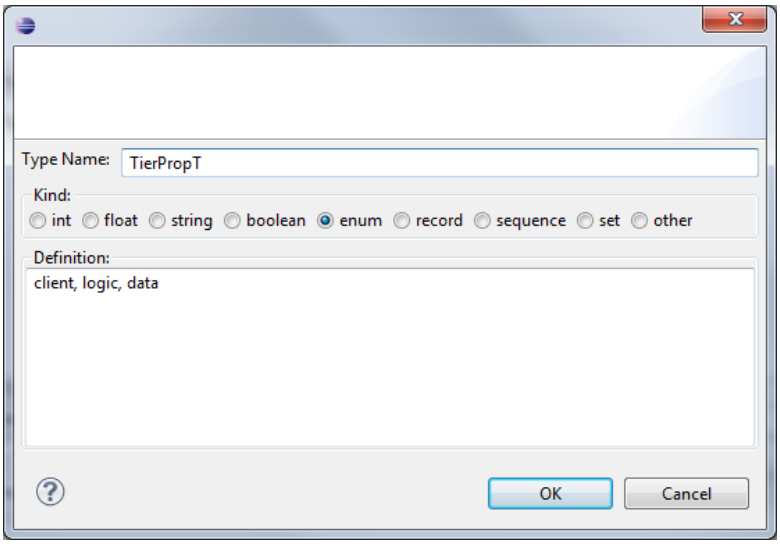


Figure 8: New Property Type dialog box.

17. In the Family editor, select the Property Types editor (at the bottom of the family editor), and open the Property Types section by selecting on the label.
18. Select the New... button in this section. A type entry dialog box appears.
19. Enter `TierPropT` for the Type Name.
20. Select enum as the Kind of property. The Definition field becomes enabled.

21. For the values, enter `client, logic, data`.
22. Figure 8 shows what the dialog box looks like at this point. Click OK.
23. Notice that the newly added property type appears in the Property Types section. To browse the details, you may expand the Property Type Details section.

Editing types with the Properties View

Switch back to the Architectural Types editor. To continue the task of adding a tier property to each type of node, we will now add a property and a rule.

Add a property

We begin by adding a property called tier to each of the three tier node types. To do this, we add a property to the parent node type TierNodeT, since the other types are subtypes. In fact, this is the purpose behind this type.

24. In the Family editor, choose TierNodeT. The details of TierNodeT are shown in the Properties View. We will call this action "focusing on" an element.
25. Select the Properties tab if it's not already selected.
26. Right-click in the empty region and select New Property... or click the  icon on the Properties View toolbar at the right-hand corner of the Properties View title bar.
27. For the Name, enter `tier`.
28. For the Type, choose TierPropT, which we defined in the previous section.
29. Uncheck the Assign Value to Property checkbox. Figure 9 shows what the dialog box looks like at this point.
30. Click OK.

Note: since TierNodeT is a root type, we want to have sub-types and instances of it specify the actual value for the tier property. Therefore, we will not assign a value to the property here. Notice that the property now appears in the Properties tab, and has <<No Value>>.

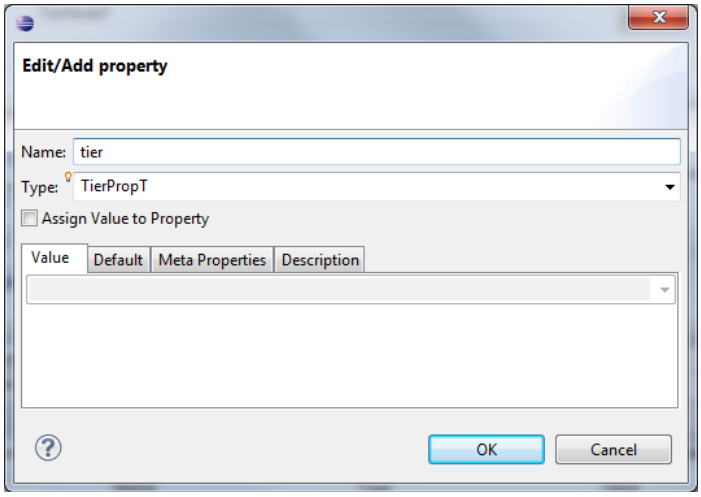


Figure 9: Property editing dialog for the "tier" property.

Add a rule

You may have noticed that the TierNodeT component type already defines two properties called host and allowShareHost. The first property is a string that indicates the host on which an instance of the TierNodeT component type runs. The second property is a Boolean that indicates whether some other nodes can run on the same host. These properties are meant to indicate that a node that doesn't allow sharing a host should not be on the same host as any other node.

We would like AcmeStudio to flag a warning if the condition above is violated. To do this, we will add a rule that captures this constraint. In order to apply to all elements in the system, this rule needs to reside at the family level.

31. In the Family editor, focus on TieredFam by pressing the button above the Component Types section labeled TieredFam (Press to select).
32. Select the Rules tab in the Properties View.
33. Click the  icon on the Properties View toolbar.
34. Name the rule `hostCheck` and ensure that the invariant button is selected.
35. Put the following in the Design Rule textbox:

```
Forall t1 : TierNodeT in self.Components |
  !t1.allowShareHosts -> (Forall t2 : TierNodeT in self.Components |
    t1 != t2 -> t1.host != t2.host)
```

    This is the above rule denoted in the Acme constraint language.
36. Click Parse Rule to make sure there's no error. You must execute this step in order to complete the rule definition. If there is a parse error, you must fix it (or click Cancel) before you can continue.
37. One may write a descriptive label for this rule, which will appear in the Rules list when the rule is satisfied. Let's label this one, "`Tier nodes respect host assignment.`"
38. One may also write a descriptive error label, which will appear in the Rules list when the rule is violated. Let's label this one, "`Two nodes that cannot share a host must not reside on the same host.`"
39. Click OK to complete.
    Note: if OK doesn't respond, you may have to click Parse again.
40. Make sure that you save the family.

Optional challenge problem: See if you can come up with a heuristic that flags a warning if a particular logic node has more than three clients connected to it. Hint: Assume that each client node connects to a logic node on separate ports. Then define a heuristic associated with LogicNodeT that limits the number of ports. To count the number of ports, use the set constructor `select` to select the subset of ports in the component that declares the port type of provideT (e.g., `{select p : Port in SomeSet | declaresType(p, t) }`). A component can refer to itself as `self`. The set of ports defined on a component can be referenced via `self.Ports`. Then apply the `size()` function to that set.

Acme Source editing

Sometimes it's easier to make certain changes by directly editing the Acme source, such as changes that involve significant redundancy where copy and paste becomes handy. This can be accomplished using the Source editor. Note that ordinarily, using the Source editor requires sufficient understanding of the Acme syntax. However, we will walk through a couple of changes without relying on you to create a lot of Acme specs.

Recall that we previously added a tier property to the component supertype TierNodeT. However, we haven't made use of that property in each of the component subtypes. We will now assign the appropriate property value in each subtype – `client` for ClientNodeT, `logic` for LogicNodeT, and `data` for DataNodeT.[4]

41. Notice three tabs at the bottom of the Editor region – Overview, Acme Source and ThreeTieredFam. To switch to the Source editor, click on Acme Source and the editor shows you the source of the family's Acme description.
42. Locate the component type TierNodeT, which can be done in a number of ways. You could scroll through the source until you find it, or you can go to the Outline view, locate the type there, and double click on it. This will bring the node into focus in the source editor.
43. Highlight the line containing the declaration of the property tier and copy it to clipboard (this could be done by pressing Ctrl-C (Option-C on Mac) or Ctrl-Insert or choosing from the menu bar Edit → Copy).
44. Locate ClientNodeT component type and paste the property text inside the component type definition (between the two curly braces).
45. Before the semicolon of the tier-property text, add "`= client`". The entire component type definition should look like:

```
Component Type ClientNodeT extends TierNodeT with {
      Property tier : TierPropT = client;
    }
```

46. Do the same pasting and editing for LogicNodeT (assigning `logic`) and DataNodeT (assigning `data`).
47. After making these changes, the Source editor content must be synchronized with the Family editor content. Clicking on the TieredFam editor tab will initiate the synchronization. Note that if there were major errors from the source change, synchronization will fail, and you will be taken back to the Source editor. Otherwise, the Family editor appears again.

Error indicators: When you change something that causes a parse error or type-check error in the Source editor, red error icons appear at the appropriate error locations. Parse errors must be removed. Certain type-checking and constraint-checking errors in the Family may be ignored, specifically, "does not satisfy constraint" or "does not satisfy type". These will often occur in Family descriptions and can only be satisfied in the instantiated system. For example, if a port type *P* declares a rule that at least one role must be attached, and one port instance *p* of *P* is defined in a component type *C*, the rule

[4] Note that the usual way to do this in AcmeStudio would be to add a property via the Properties tab of the Properties view.

will not be satisfied for *p* until an instantiated system where instances of *C* are defined and roles of connectors attached to the components' ports.

Visual variants

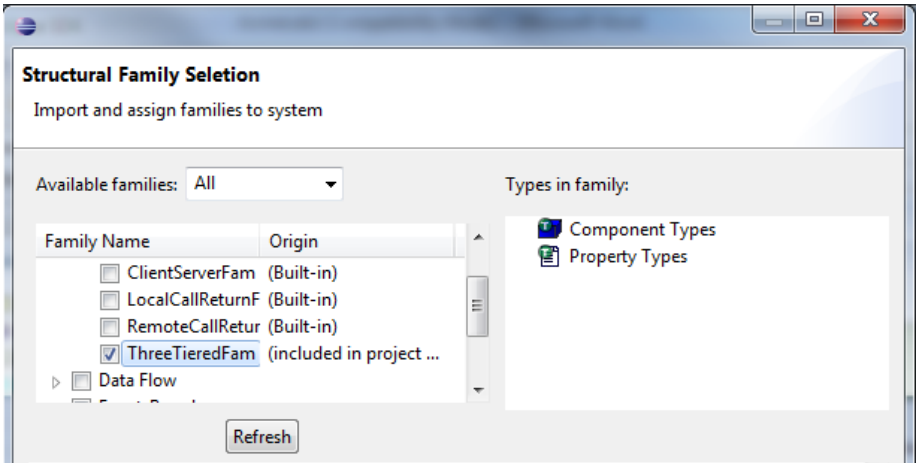
The visualization of an element can be set to vary depending on the values of its properties, through a feature called Visual Variant. Let us create a Visual Variant for the LogicNodeT component type that would change the component outline to a red line if its property indicates that the tier node cannot share hosts.

48. In the Family editor, select Modify… the visualization of the LogicNodeT component.
49. Click on the Variants tab.
50. Click New… to create a new variant.
51. Name the variant `No Sharing`.
52. Make sure the radio button is selected for Trigger is based on a property.
53. Pull the Property dropdown menu and select allowSharedHost, then choose == for the Condition, and enter `false` for the Value.
54. From the Shape tab, section Color/Line Properties, make the outline color red.
55. Click OK to create the new variant and return to the Visualization dialog box.
56. Select the newly added variant to make sure it shows a red outline.
57. Click OK to finish.

Testing the redefined Family

Make sure that your family is saved. To test the family you modified, let us now create a new system based on the family.

58. In the Navigator, right-click anywhere in the current AcmeLab2 project. In the pop-up menu, choose New → Acme System. A progress indicator dialog appears while AcmeStudio searches for and parses the families available to this project.
59. For the System name, write `TestSystem`. Click Next.
60. Make sure that the Container says /AcmeLab2. If not, click Browse and choose AcmeLab2.
61. Click Next.
62. Look a list of Available families. They are grouped by high-level architectural styles. The ThreeTieredFam family resides into the Call Return category. Choose ThreeTieredFam. A structured list of the types defined in this family appears in the Details box.



63. Click Finish, and you have instantiated an empty system of the family ThreeTieredFam. The default system editor is the Diagram editor. Notice the palette of types on the right from which you can click-and-drop an element to create an instance.
64. Compose a simple three-tiered (you may want to try one in Assignment 2). Change a node's allowShareHosts property to false and see if the visualization variant works. Test the constraint you defined that limits the number of clients and see if it works. You can do this by instantiating four client nodes, creating four ports on a logic node, creating four connectors to attach the client nodes to the logic node.

    Don't forget to use the Properties View to check whether rules are satisfied. Recall that if an invariant fails, the icon is placed next to the rule; if a heuristic fails, the icon appears.[5]

Other advanced features to play with

There are many more features of AcmeStudio not covered in this lab that you may wish to experiment with. Try different things out. Some of the more complex features are listed below:

- Adding descriptions to the types
- Defining Port policy and alignment for Component type
- Defining Role policy and alignment for Connector type
- Exporting the diagram to an image or PDF

[5] … indicates that the rule is being evaluated, indicates that the rule has type errors in it, indicates that constraint evaluation has been turned off, indicates that the rule could not evaluated.