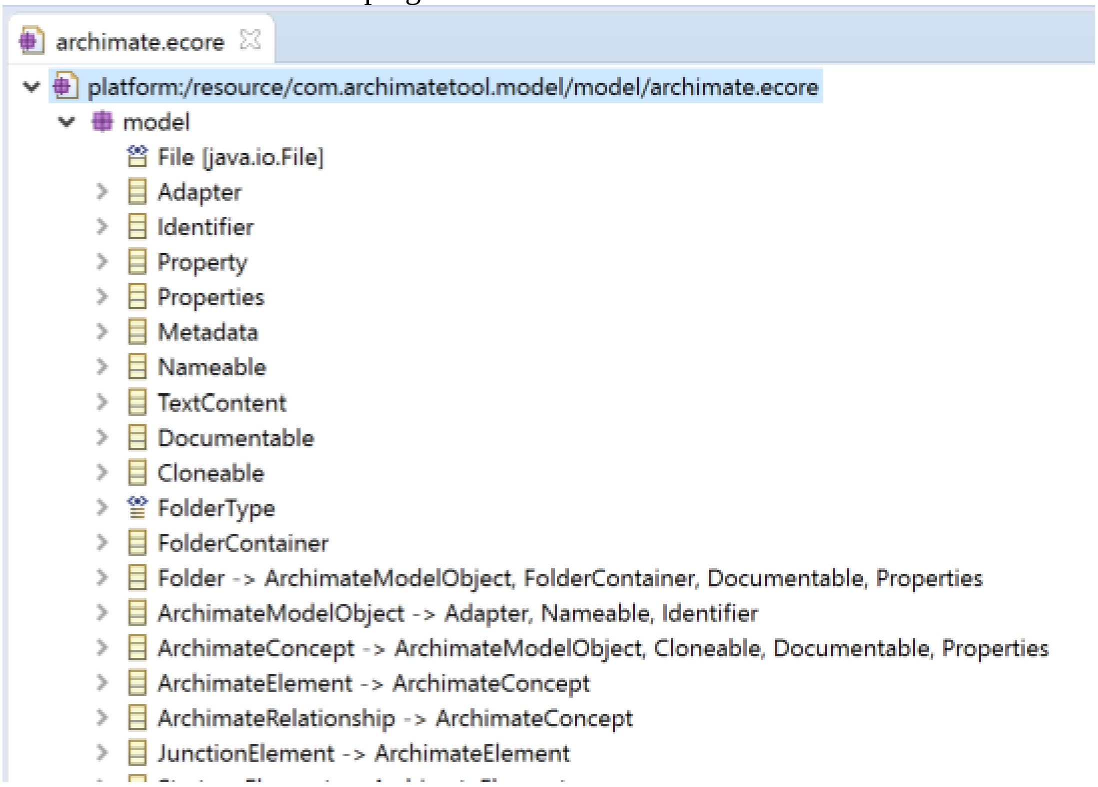
Extending the Model with a new Element

Introduction

This tutorial provides an example of adding a new element concept to the ArchiMate model. This example element will be called "Channel" it will belong to the "Business" ArchiMate layer and will ineffect be a specialisation of a “Business Interface”. Ensure you have the latest code and open Eclipse with all the Archi projects available.

Create the element in the model

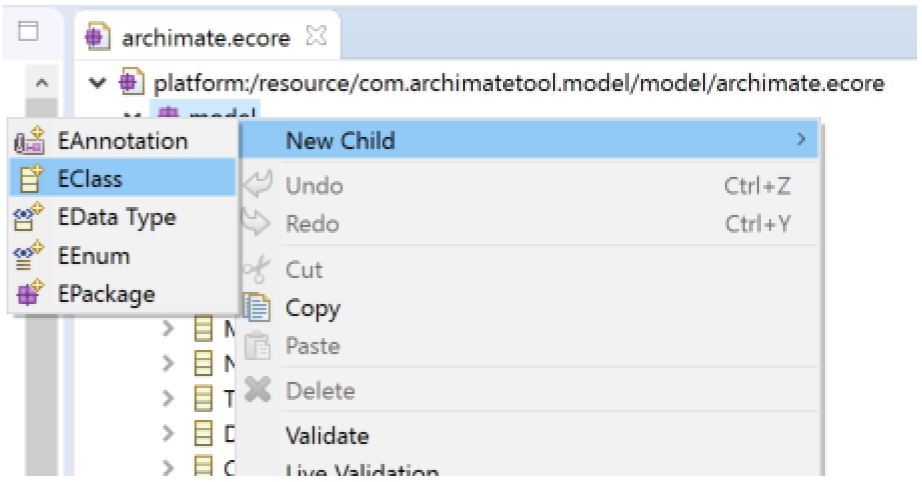
1. In Eclipse, open the archimate.ecore file in the "model" folder of the "com.archimatetool.model" plug-in:



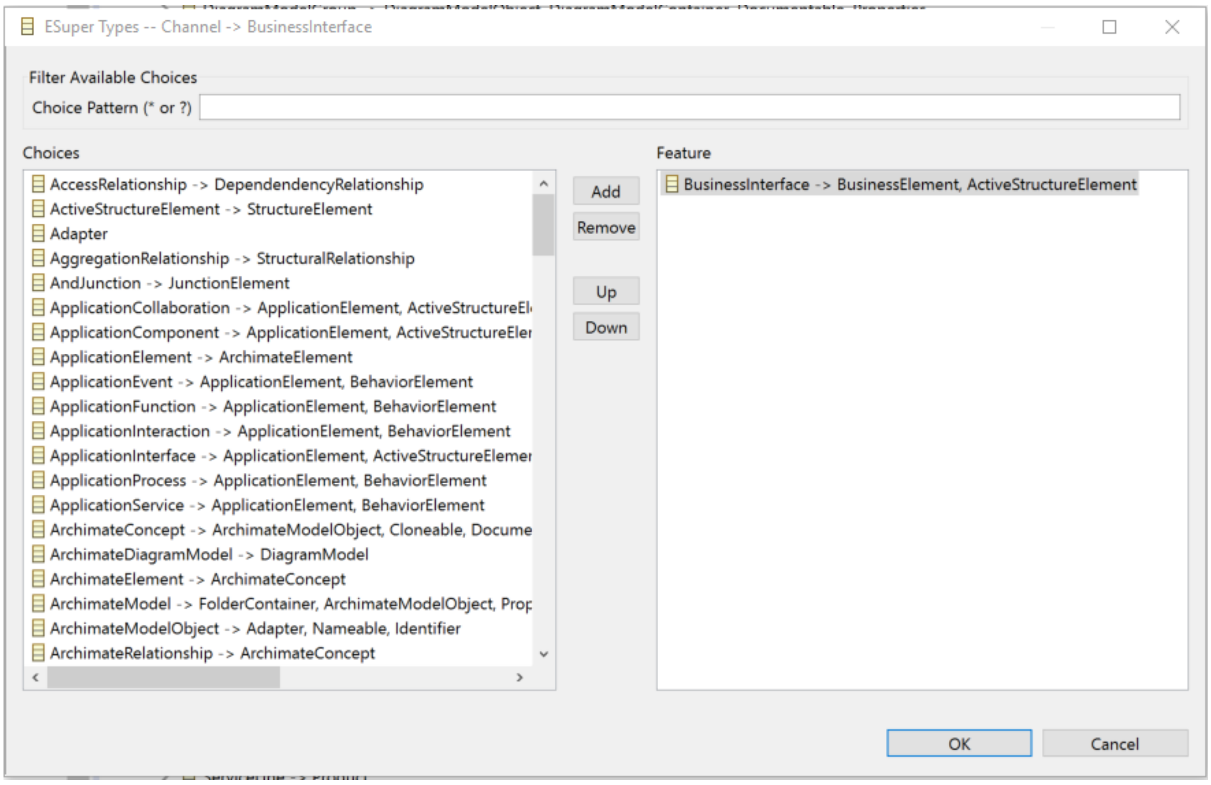
```
archimate.ecore
platform:/resource/com.archimatetool.model/model/archimate.ecore
  model
    File [java.io.File]
    Adapter
    Identifier
    Property
    Properties
    Metadata
    Nameable
    TextContent
    Documentable
    Cloneable
    FolderType
    FolderContainer
    Folder -> ArchimateModelObject, FolderContainer, Documentable, Properties
    ArchimateModelObject -> Adapter, Nameable, Identifier
    ArchimateConcept -> ArchimateModelObject, Cloneable, Documentable, Properties
    ArchimateElement -> ArchimateConcept
    ArchimateRelationship -> ArchimateConcept
    JunctionElement -> ArchimateElement
```

2. In the tree of the Ecore editor, select the "model" node, right click, and select "New Child" and then "EClass". This will create a new blank EClass node at the bottom of

the tree:



3. Select this new blank node and double-click it to open the Properties window.
4. In the Properties window add the name, "Channel".
5. As the element is a specialisation of the "Business Interface" it is classed as a "BusinessElement" and an "ActiveStructureElement", it therefore needs to inherit from these.
   There are two ways of doing this either give it ESuper Types of both "BusinessElement" and "ActiveStructureElement" or give it an ESuper Type of "BusinessInterface". In the Properties window, select the "ESuper Types" row and click on the "..." button at the right of the row. A dialog box will appear. In the left column, select your ESuper Type(s) and add it/them to the right column. Click OK:



Choices:

AccessRelationship -> DependendencyRelationship
ActiveStructureElement -> StructureElement
Adapter
AggregationRelationship -> StructuralRelationship
AndJunction -> JunctionElement
ApplicationCollaboration -> ApplicationElement, ActiveStructureEl
ApplicationComponent -> ApplicationElement, ActiveStructureEler
ApplicationElement -> ArchimateElement
ApplicationEvent -> ApplicationElement, BehaviorElement
ApplicationFunction -> ApplicationElement, BehaviorElement
ApplicationInteraction -> ApplicationElement, BehaviorElement
ApplicationInterface -> ApplicationElement, ActiveStructureElemer
ApplicationProcess -> ApplicationElement, BehaviorElement
ApplicationService -> ApplicationElement, BehaviorElement
ArchimateConcept -> ArchimateModelObject, Cloneable, Docume
ArchimateDiagramModel -> DiagramModel
ArchimateElement -> ArchimateConcept
ArchimateModel -> FolderContainer, ArchimateModelObject, Prop
ArchimateModelObject -> Adapter, Nameable, Identifier
ArchimateRelationship -> ArchimateConcept

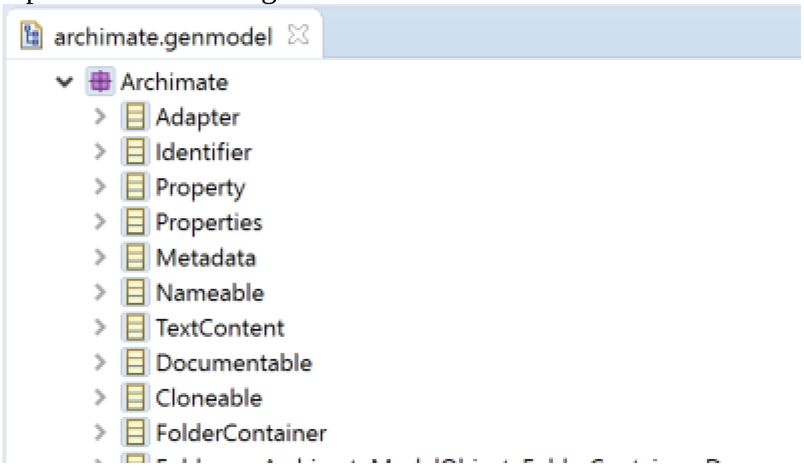
Feature:

BusinessInterface -> BusinessElement, ActiveStructureElement

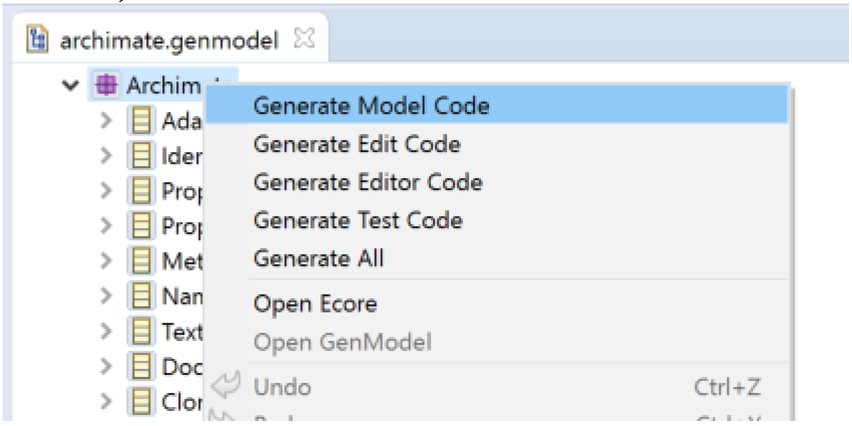
6. NOTE: if you want the element to appear in a specific folder then this is controlled by specific classes in the inheritance hierarchy as detailed in the following table

Folder	Element to inherit from
Strategy	StrategyElement
Business	BusinessElement
Application	ApplicationElement
Technology & Physical	TechnologyElement or PhysicalElement
Motivation	MotivationElement
Implementation & Migration	ImplementationMigrationElement
Other	CompositeElement

7. ApplicationElement, BusinessElement,
8. Make sure you save the archimate.ecore file.
9. Open the archimate.genmodel file:



10. From the main "Generator" or right click context menu in Eclipse, select "Generate Model Code". This will generate some files in the "src" folder - "IChannel.java" (the Java Interface) and "Channel.java" (the implementation Java class). Other files will be modified, too.



Add the element to the relationships rules file

As the ArchiMate language defines a strict set of rules for relationships between elements, we need to declare what relationships are allowed between our new element and the others. This is defined in the "relationships.xml" file found in the "model" folder. This is an XML file that is designed to be reasonably human readable at the expense of verbosity. An element is declared by its name in a "source" tag and allowable target elements are declared as "target" elements. Allowed relationships are set in the "relations" attribute. These are key letters like "o" and "c" and "f". These are defined in the "relationships-keys.xml" file. For example, "a" represents the "AccessRelationship".

1. In Eclipse, open the "relationships.xml" file found in the "model" folder.
2. For this example, copy and paste the XML section for the element that most closely resembles your new element e.g. "BusinessInterface" section and rename it to the name of your new element, e.g. "Channel". You will need to edit it to suit your rules. It should look like this: (Note: now also has an attribute for derived relationships derived="ftv")

```
<source element="Channel">
        <target element="Channel" relations="cfgostv" />
        <target concept="ApplicationCollaboration" relations="fotv" />
        <target concept="ApplicationComponent" relations="fotv" />
        <target concept="ApplicationEvent" relations="fotv" />
        <target concept="ApplicationFunction" relations="fotv" />
        <target concept="ApplicationInteraction" relations="fotv" />
        <target concept="ApplicationInterface" relations="fotv" />
        <target concept="ApplicationProcess" relations="fotv" />
        <target concept="ApplicationService" relations="fotv" />
        <target concept="Artifact" relations="fotv" />
        <target concept="Assessment" relations="no" />
        <target concept="BusinessActor" relations="fotv" />
        <target concept="BusinessCollaboration" relations="fotv" />
        <target concept="BusinessEvent" relations="fotv" />
        <target concept="BusinessFunction" relations="fotv" />
        <target concept="BusinessInteraction" relations="fotv" />
        <target concept="BusinessInterface" relations="cfgostv" />
        <target concept="BusinessObject" relations="ao" />
        <target concept="BusinessProcess" relations="fotv" />
        <target concept="BusinessRole" relations="fotv" />
        <target concept="BusinessService" relations="fiotv" />
        <target concept="Capability" relations="or" />
        <target concept="CommunicationNetwork" relations="fot" />
        <target concept="Constraint" relations="nor" />
        <target concept="Contract" relations="ao" />
        <target concept="CourseOfAction" relations="or" />
        <target concept="DataObject" relations="ao" />
        <target concept="Deliverable" relations="o" />
        <target concept="Device" relations="fotv" />
        <target concept="DistributionNetwork" relations="fot" />
        <target concept="Driver" relations="no" />
        <target concept="Equipment" relations="fotv" />
        <target concept="Facility" relations="fotv" />
        <target concept="Gap" relations="o" />
        <target concept="Goal" relations="nor" />
        <target concept="Grouping" relations="acfginorstv" />
        <target concept="ImplementationEvent" relations="o" />
        <target concept="Location" relations="fot" />
        <target concept="Material" relations="aov" />
        <target concept="Meaning" relations="no" />
        <target concept="Node" relations="fotv" />
        <target concept="Outcome" relations="nor" />
        <target concept="Path" relations="fot" />
        <target concept="Plateau" relations="o" />
        <target concept="Principle" relations="nor" />
        <target concept="Product" relations="fot" />
        <target concept="Relationship" relations="o" />
        <target concept="Representation" relations="ao" />
        <target concept="Requirement" relations="nor" />
        <target concept="Resource" relations="or" />
        <target concept="Stakeholder" relations="no" />
```

```
    <target concept="SystemSoftware" relations="fotv" />
    <target concept="TechnologyCollaboration" relations="fotv" />
    <target concept="TechnologyEvent" relations="fotv" />
    <target concept="TechnologyFunction" relations="fotv" />
    <target concept="TechnologyInteraction" relations="fotv" />
    <target concept="TechnologyInterface" relations="fotv" />
    <target concept="TechnologyProcess" relations="fotv" />
    <target concept="TechnologyService" relations="fotv" />
    <target concept="Value" relations="no" />
    <target concept="WorkPackage" relations="o" />

    <target concept="AndJunction" relations="fiort" />
    <target concept="OrJunction" relations="fiort" />          ...
</source>
```

3. You will need to add "target" XML elements to the other "source" elements in the XML file so that relationship rules can be declared from these elements to the new element. The format will be:

```
<target element="Channel" relations="cfgostv"  derived="cfgtv" />
```

Add the element to the ArchiMateModelUtils.java file

Now that the model element has been created, and the relationship rules defined we need to add it to the list of elements that will appear in the UI. The first task we need to do is add it to the "ArchiMateModelUtils.java" file.

1. In Eclipse, open the "ArchiMateModelUtils.java" file, found in the "com.archimatetool.model.util" package.
2. Edit the getXxxxClasses() method (where Xxxx represents the category of element you are adding : e.g. Business) by inserting the following line amongst the others:

```
IArchimatePackage.eINSTANCE.getChannel(),
```

Define the User Interface for the element

Now that we have added the element to the model plug-in, we need to turn our attention to the "com.archimatetool.editor" plug-in and provide some code to create the UI for the new element. Decision

Add an icon

1. Provide a 16x16 icon for the element. Save it as "business-element.png" and copy it to the "img/archimate" folder in the "com.archimatetool.editor" plug-in.
2. Declare the reference to the icon in the "IArchiImages.java" file in the "com.archimatetool.editor.ui" package:

```
String ICON_NAME_OF_ELEMENT = ARCHIMATE_IMGPATH + "name-of-element.png";
```

In our case

```
String ICON_CHANNEL = ARCHIMATE_IMGPATH + "channel.png";
```

Add a UI Provider

1. Create a UI Provider class for the element in the "com.archimatetool.editor.ui.factory.elements" package. Copy the java class that most closely resembles your new element e.g."BusinessInterfaceUIProvider.java" class and rename it to "NewElementNameUIProvider.java" e.g. "ChannelUIProvider.java".
2. Edit this class to incorporate your classes from before:

```
public class ChannelUIProvider extends AbstractInterfaceUIProvider {

    public EClass providerFor() {
        return IArchimatePackage.eINSTANCE.getChannel();
    }

    @Override
    public EditPart createEditPart() {
        return new ArchimateElementEditPart(InterfaceFigure.class);
    }

    @Override
    public String getDefaultName() {
        return Messages.ChannelUIProvider_0;
    }

    @Override
    public Image getImage() {
        return getImageWithUserFillColor(IArchiImages.ICON_CHANNEL);
    }

    @Override
    public ImageDescriptor getImageDescriptor() {
        return getImageDescriptorWithUserFillColor(IArchiImages.ICON_CHANNEL);
    }

    @Override
    public Color getDefaultColor() {
        return ColorFactory.get(255, 255, 181);
    }
}
```

3. Add a constant to the Messages.java file for your new element:

```
Public static String NewElementNameUIProvider_0;
```

In our case this is

```
Public static String ChannelUIProvider_0;
```

4. Add a value to be assigned to the new constant to the Messages.properties file:

```
NewElementNameUIProvider_0=New Element Name
```

In our example this would be

```
ChannelUIProvider_0=Channel
```

5. Finally, register this UI Provider. Edit the "plugin.xml" in com.archimatetool.editor by adding the following element to the `<extension name="Archimate Elements" point="com.archimatetool.editor.objectUIProvider">` element:

```
<uiProvider
    class="com.archimatetool.editor.ui.factory.elements.NewElementNameUIProvider"
    id="com.archimatetool.editor.NewElementNameUIProvider">
</uiProvider>
```

In our example this would be

```
<uiProvider
    class="com.archimatetool.editor.ui.factory.elements.ChannelUIProvider"
    id="com.archimatetool.editor.ChannelUIProvider">
</uiProvider>
```

Add a Special Figure

1. If you want a figure with your icon to be displayed for your element (rather than just going with the figure for the element you have based your element on) you will have to create one in com.archimatetool.editor.diagram.figures.elements.
2. The naming of this is up to you but the convention is MyElementNameFigure.java.
3. If you want a special alternative shape to be available as well these are just another figure file (but where there are two the convention is to call the second MyElementNameFigureDelegate.java so you will need to add this to the package as well.
4. You may need to refer to the org.eclipse.draw2d.geometry.point (for drawing the icon) and org.eclipse.draw2d.geometry.pointlist (for drawing a special shape) documentation to work out what is happening here (if your new figure is based on a rectangle anyway).
5. The example below shows what to do if you are going to allow a rounded rectangle with icon shape and a special alternative shape:
6. First in your MyElementNameFigure.java file copy the element that most closely resembles the one you want and edit the constructor

```
public class MyElementNameFigure extends AbstractTextControlContainerFigure {
  protected IFigureDelegate fFigureDelegate1;
  protected IFigureDelegate fFigureDelegate2;
  public MyElementNameFigure() {
    super(TEXT_FLOW_CONTROL);
    fFigureDelegate1 = new RoundedRectangleFigureDelegate(this, 20 - getTextControlMarginWidth());
    fFigureDelegate2 = new MyElementNameFigureDelegate(this);
  }
```

7. The method that deals with drawing the icon is called DrawIcon and this is where your knowledge of the point will come in useful

```
  protected void drawIcon(Graphics graphics) {
    graphics.pushState();
    graphics.setLineWidth(1);
    graphics.setForegroundColor(isEnabled() ? ColorConstants.black : ColorConstants.gray);
    PointList points = new PointList();
    // Start at top left
    Point pt = getIconOrigin();
    points.addPoint(pt);
    pt.translate(8, 0);
    points.addPoint(pt);
    pt.translate(6, 5);
    points.addPoint(pt);
    pt.translate(-6, 5);
    points.addPoint(pt);
    pt.translate(-8, 0);
    points.addPoint(pt);
    graphics.drawPolygon(points);
    graphics.popState();
  }
```

8. Now add your MyElementNameFigureDelegate.java (again it is probably easiest to copy the one that most closely resembles the one you want to create)
9. There is not really anything to do in the constructor this time most of the work is in the description of the shape in the GetPointList method

```
public class MyElementNameFigureDelegate extends AbstractFigureDelegate {

  public MyElementNameFigureDelegate(IDiagramModelObjectFigure owner) {
    super(owner);
  }

```

10. This is where you hope it's an easy shape to draw

```
  private PointList getPointList() {
    Rectangle bounds = getBounds();

    bounds.width -= 1;
    bounds.height -= 1;

    PointList points = new PointList();

    points.addPoint(bounds.x, bounds.y);
    points.addPoint(bounds.x + (int)(bounds.width * 0.8), points.getPoint(0).y);
    points.addPoint(bounds.x + bounds.width, bounds.y + (bounds.height / 2));
    points.addPoint(points.getPoint(1).x, bounds.y + bounds.height);
    points.addPoint(points.getPoint(0).x, points.getPoint(3).y);
    points.addPoint(bounds.x + (int)(bounds.width * 0.2), points.getPoint(2).y);
    points.addPoint(points.getPoint(0).x, points.getPoint(0).y);

    return points;
  }
```

11. Now finally make sure that your MyElementNameUIProvider.java you created earlier in com.archimate.editor.ui.factory.elements refers to your new figure in its createEditPart method

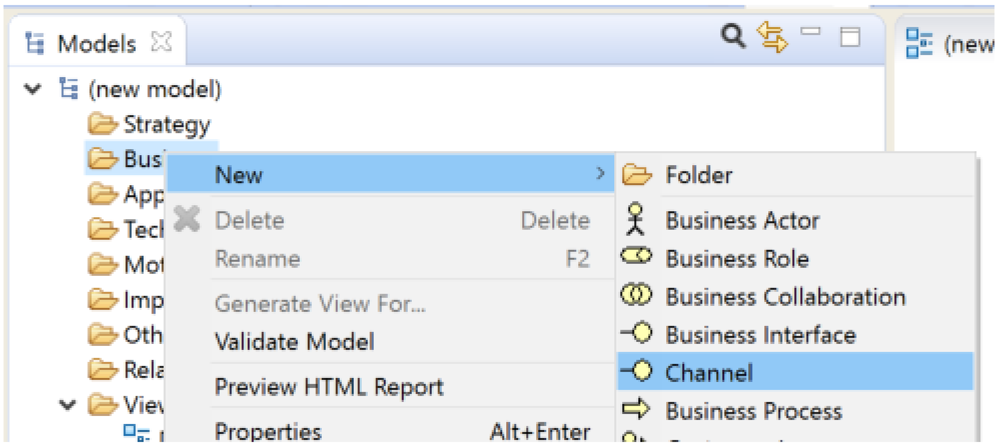
```
  public EditPart createEditPart() {
    return new ArchimateElementEditPart(MyElementNameFigure.class);
  }
```

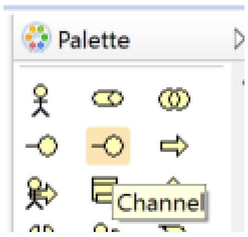
12. And if you have created that alternative special delegate shape make sure the hasAlternativeFigure method returns true

```
  public boolean hasAlternateFigure() {
    return true;
  }
```

Run Archi from Eclipse

Running Archi from Eclipse (see link) will result in your new element appearing in the context menus on the model tree and in the editor palette. The new element will appear in a diagram, and any relationship rules will be evident:





Well it works but you will notice there are no hints appearing (no prizes for guessing why). If you want to add hints then follow the steps below.

Adding Hints for your new element

Where the hints are stored.

The hints are HTML files stored in the hints directory in com.archimatetool.help

To follow the format of the existing hints files these should be called my_element_name.html in our example this would be business_element.html.

To be consistent with the presentation of the other hints it may be best initially to copy the html file of the closest equivalent element included with archi, rename it and edit its contents to contain the information you want to convey about your new element.

If you don't have time to add the content yet you can do this later but I would suggest you create the file and add its editing to your backlog.

How the hints are referenced.

In the com.archimatetool.help directory find the plugin.properties file, this contains the titles of all the hints, just add the elements you intend to add hints for to this list, they are recorded in the format :

```
hint.title.nn = <<element name>>
```

NOTE: I know it’s obvious use the next available number for your element in our example this is :

```
hint.title.94 = Business Element
```

These hint.title.*nn* elements are used in the plugin.xml file as the title attribute of a hint tag/element so now we need to add our new element to this file.
Open plugin.xml and in the <extension point="com.archimatetool.help.hints"> tag add a hint tag for your element, the format is

```
    <hint
        class="com.archimatetool.model.IYourNewElement"
        file="hints/your_new_file.html"
        title="%hint.title.nn">
    </hint>
```

In our example this is

```
    <hint
        class="com.archimatetool.model.IChannel"
        file="hints/business_element.html"
        title="%hint.title.94">
    </hint>
```

You can now run archi and the hints for your new elements should appear.
If you get an error in the hints tab on your new version of archi, check all the files for the elements you added hint tags for are there and that they are spelt correctly.

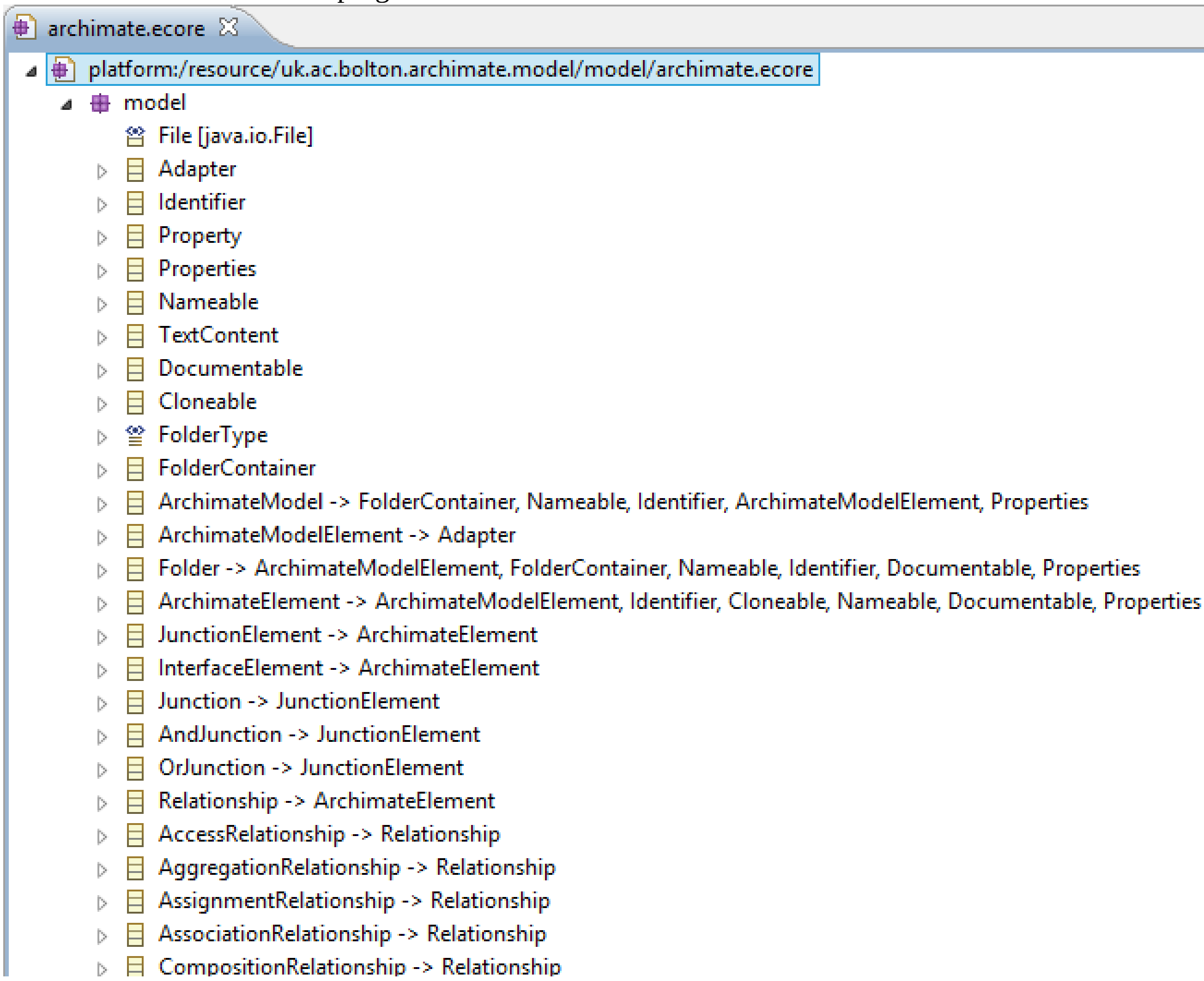
Extending the Model with a new Relationship

Introduction

This tutorial provides an example of adding a new relationship type to the ArchiMate model. This example relationship will be called "Extension". Ensure you have the latest code and open Eclipse with all the Archi projects available.

Create the relationship in the model

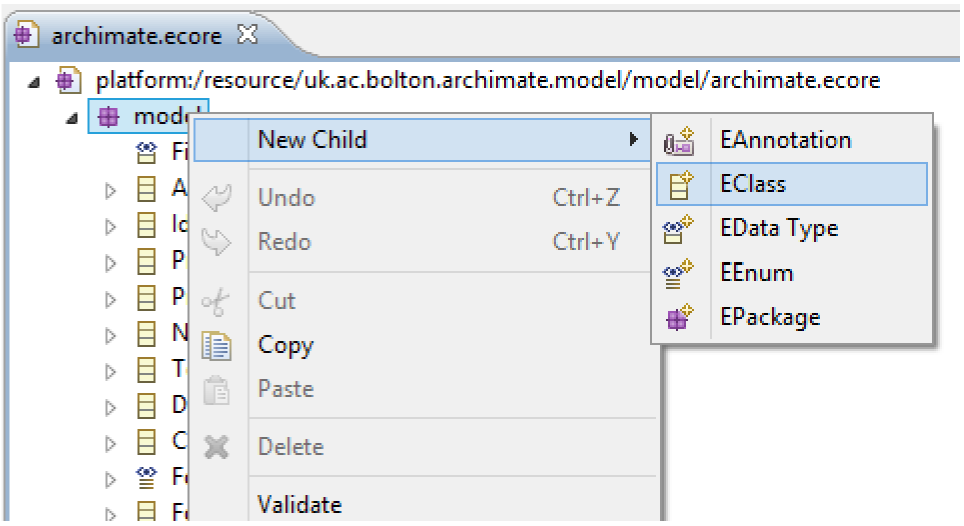
1. In Eclipse, open the archimate.ecore file in the "model" folder of the "com.archimatetool.model" plug-in:



```
archimate.ecore
platform:/resource/uk.ac.bolton.archimate.model/model/archimate.ecore
  model
    File [java.io.File]
    Adapter
    Identifier
    Property
    Properties
    Nameable
    TextContent
    Documentable
    Cloneable
    FolderType
    FolderContainer
    ArchimateModel -> FolderContainer, Nameable, Identifier, ArchimateModelElement, Properties
    ArchimateModelElement -> Adapter
    Folder -> ArchimateModelElement, FolderContainer, Nameable, Identifier, Documentable, Properties
    ArchimateElement -> ArchimateModelElement, Identifier, Cloneable, Nameable, Documentable, Properties
    JunctionElement -> ArchimateElement
    InterfaceElement -> ArchimateElement
    Junction -> JunctionElement
    AndJunction -> JunctionElement
    OrJunction -> JunctionElement
    Relationship -> ArchimateElement
    AccessRelationship -> Relationship
    AggregationRelationship -> Relationship
    AssignmentRelationship -> Relationship
    AssociationRelationship -> Relationship
    CompositionRelationship -> Relationship
```

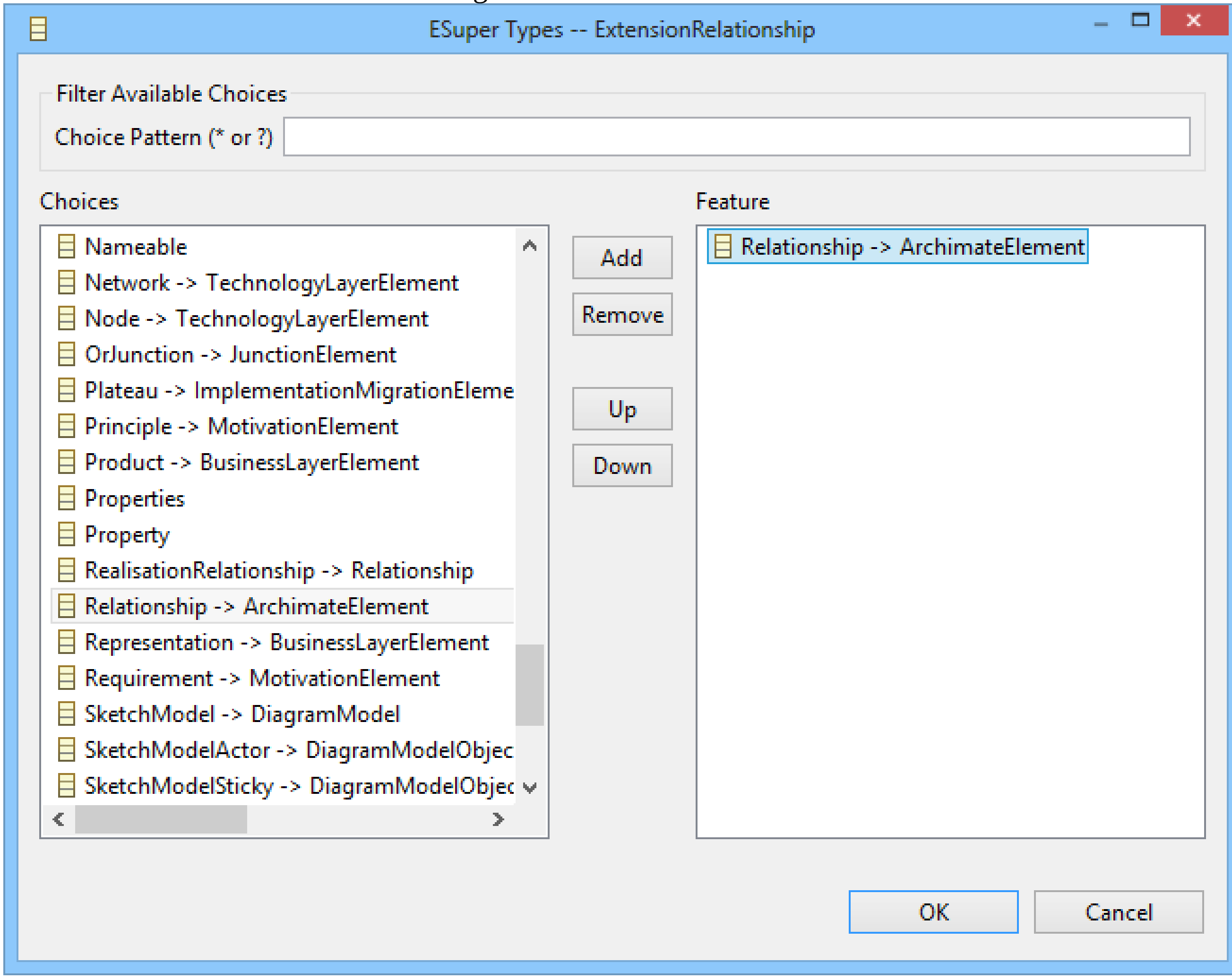
2. In the tree of the Ecore editor, select the "model" node, right click, and select "New Child" and then "EClass". This will create a new blank EClass node at the bottom of

the tree:


archimate.ecore

platform:/resource/uk.ac.bolton.archimate.model/model/archimate.ecore

3. Select this new blank node and double-click it to open the Properties window.
4. In the Properties window add the name, "ExtensionRelationship".
5. The new relationship needs to inherit from the "Relationship" type. In the Properties window, select the "ESuper Types" row and click on the "..." button at the right of the row. A dialog box will appear. In the left column, select "Relationship -> ArchimateElement" and add it to the right column. Click OK:


ESuper Types -- ExtensionRelationship

Filter Available Choices

Choice Pattern (* or ?)

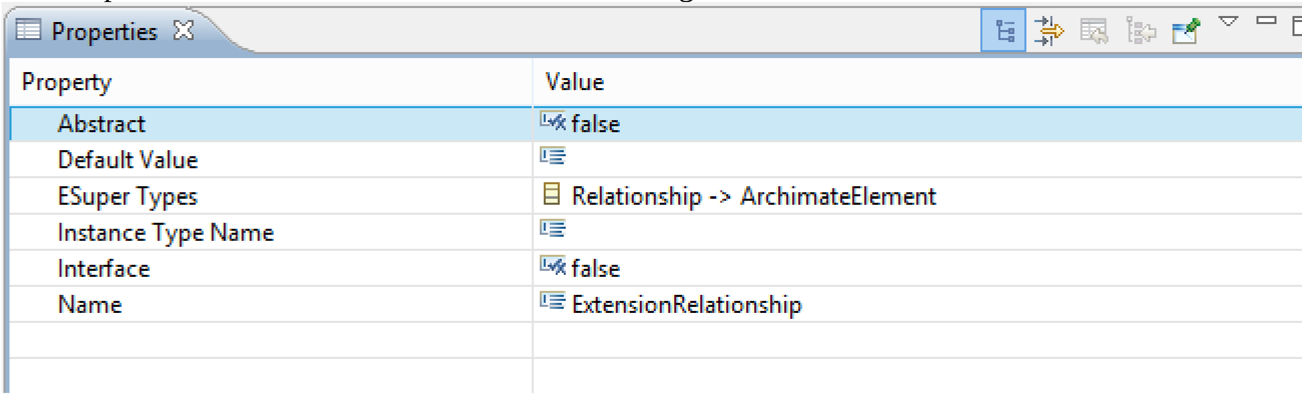
Choices

- Nameable
- Network -> TechnologyLayerElement
- Node -> TechnologyLayerElement
- OrJunction -> JunctionElement
- Plateau -> ImplementationMigrationEleme
- Principle -> MotivationElement
- Product -> BusinessLayerElement
- Properties
- Property
- RealisationRelationship -> Relationship
- Relationship -> ArchimateElement
- Representation -> BusinessLayerElement
- Requirement -> MotivationElement
- SketchModel -> DiagramModel
- SketchModelActor -> DiagramModelObjec
- SketchModelSticky -> DiagramModelObjec

Feature

- Relationship -> ArchimateElement

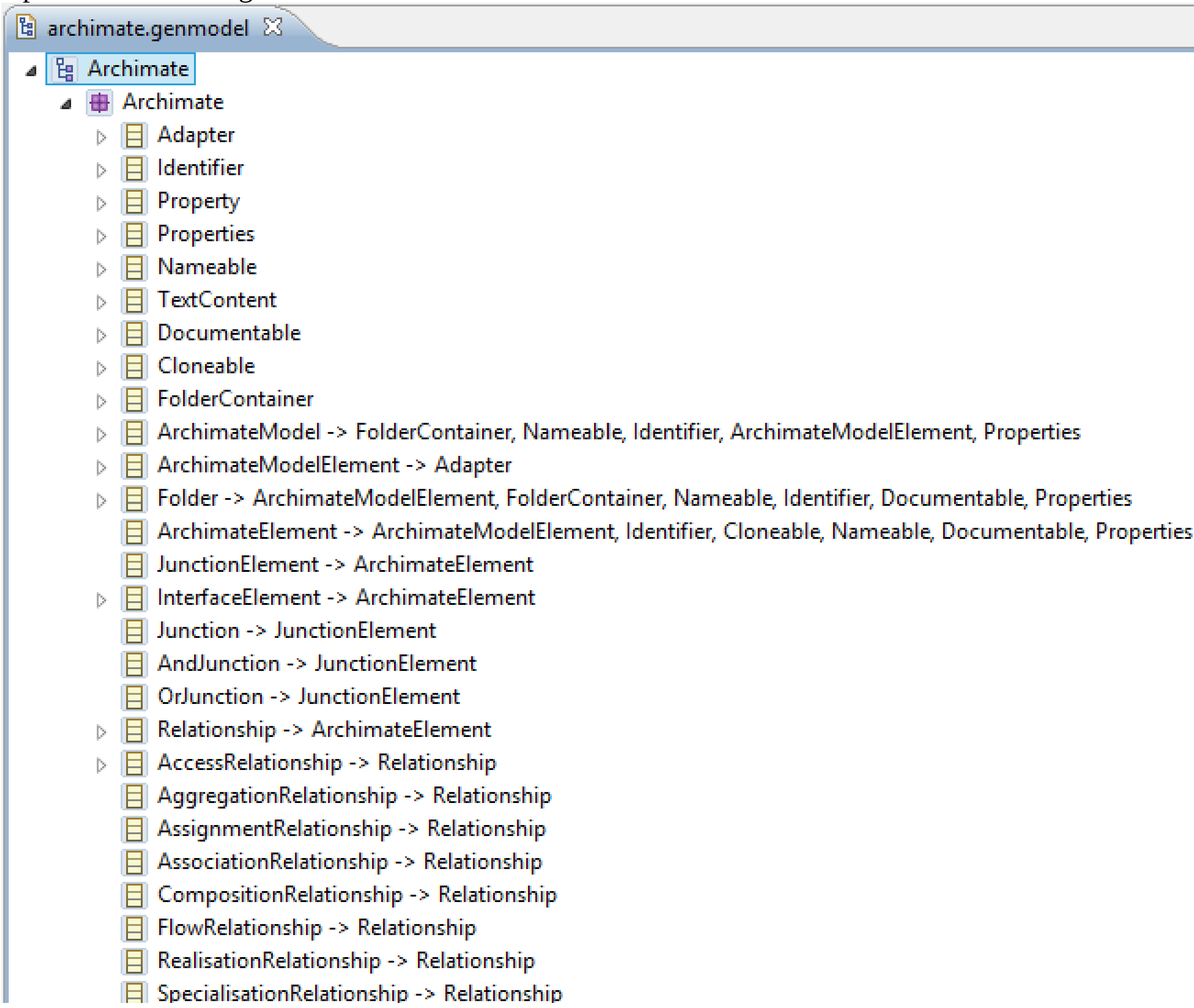
6. The Properties window should look like the following:



Properties

Property	Value
Abstract	false
Default Value	
ESuper Types	Relationship -> ArchimateElement
Instance Type Name	
Interface	false
Name	ExtensionRelationship

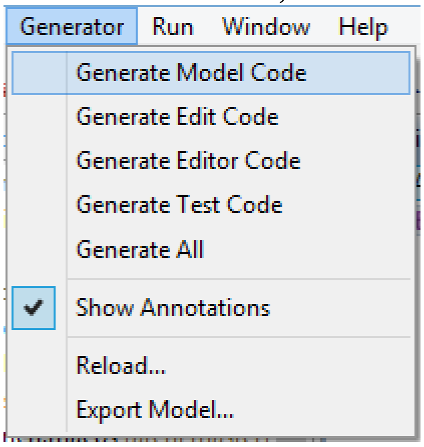
7. Make sure you save the archimate.ecore file.
8. Open the archimate.genmodel file:



```
archimate.genmodel
Archimate
  Archimate
    Adapter
    Identifier
    Property
    Properties
    Nameable
    TextContent
    Documentable
    Cloneable
    FolderContainer
    ArchimateModel -> FolderContainer, Nameable, Identifier, ArchimateModelElement, Properties
    ArchimateModelElement -> Adapter
    Folder -> ArchimateModelElement, FolderContainer, Nameable, Identifier, Documentable, Properties
    ArchimateElement -> ArchimateModelElement, Identifier, Cloneable, Nameable, Documentable, Properties
    JunctionElement -> ArchimateElement
    InterfaceElement -> ArchimateElement
    Junction -> JunctionElement
    AndJunction -> JunctionElement
    OrJunction -> JunctionElement
    Relationship -> ArchimateElement
    AccessRelationship -> Relationship
    AggregationRelationship -> Relationship
    AssignmentRelationship -> Relationship
    AssociationRelationship -> Relationship
    CompositionRelationship -> Relationship
    FlowRelationship -> Relationship
    RealisationRelationship -> Relationship
    SpecialisationRelationship -> Relationship
```

9. From the main "Generator" menu in Eclipse, select "Generate Model Code". This will generate some files in the "src" folder - "IExtensionRelationship.java" (the Java Interface) and "ExtensionRelationship.java" (the implementation Java class). Other

files will be modified, too.



Add the relationship to the relationships rules file

As the ArchiMate language defines a strict set of rules for relationships between elements, we need to declare what relationships are allowed between our new element and the others. This is defined in the "relationships-2.0.xml" file found in the "model" folder. This is an XML file that is designed to be reasonably human readable at the expense of verbosity. An element is declared by its name in a "source" tag and allowable target elements are declared as "target" elements. Allowed relationships are set in the "relations" attribute. These are key letters like "o" and "c" and "f". These are defined in the "relationships-keys.xml" file. For example, "a" represents the "AccessRelationship".

1. In Eclipse, open the "relationships-keys.xml" file found in the "model" folder. Add the following XML declaration:

```
<key char="x" relationship="ExtensionRelationship" />
```

2. In Eclipse, open the "relationships-2.0.xml" file found in the "model" folder.
3. Edit the file adding the letter "x" to the "relations" attribute of each target element where such a relationship is allowed.

Add the relationship to the ArchiMateModelUtils.java file

Now that the model relationship has been created, and the relationship rules defined we need to add it to the list of relationships that will appear in the UI. The first task we need to do is add it to the "ArchiMateModelUtils.java" file.

1. In Eclipse, open the "ArchiMateModelUtils.java" file, found in the "com.archimatetool.model.util" package.
2. Edit the getRelationsClasses() method by inserting the following line amongst the others:

```
IArchimatePackage.eINSTANCE.getExtensionRelationship(),
```

Define the User Interface for the relationship

Now that we have added the relationship to the model plug-in, we need to turn our attention to the "com.archimatetool.editor" plug-in and provide some code to create the UI for the new relationship.

Add an icon

1. Provide a 16x16 icon for the relationship. Save it as "extension-16.png" and copy it to the "img/archimate" folder in the "com.archimatetool.editor" plug-in.
2. Declare the reference to the icon in the "IArchimateImages.java" file in the "com.archimatetool.editor.ui" package:

```
String ICON_EXTENSION_CONNECTION_16 = ARCHIMATE_IMGPATH + "extension-16.png";
```

Add a GEF Figure

1. Create a GEF (Graphical Eclipse Framework) Figure class for the connection in the "com.archimatetool.editor.diagram.figures.connections" package. Copy the "TriggeringConnectionFigure.java" class and rename it to "ExtensionConnectionFigure.java".

Add a GEF Edit Part

1. Create a GEF (Graphical Eclipse Framework) Edit Part class for the connection in the "com.archimatetool.editor.diagram.editparts.connections" package. Copy the "TriggeringConnectionEditPart.java" class and rename it to "ExtensionConnectionEditPart.java".
2. Edit the getFigure() method in this class so that your new figure class is returned:

```
return new ExtensionConnectionFigure(getModel());
```

Add a UI Provider

1. Create a UI Provider class for the connection in the "com.archimatetool.editor.ui.factory.relationships" package. Copy the "TriggeringConnectionUIProvider.java" class and rename it to "ExtensionConnectionUIProvider.java".
2. Edit this class to incorporate your classes from before:

```
public class ExtensionConnectionUIProvider extends AbstractConnectionUIProvider {

  public EClass providerFor() {
    return IArchimatePackage.eINSTANCE.getExtensionRelationship();
  }

  @Override
```

```
    public EditPart createEditPart() {
        return new ExtensionConnectionEditPart();
    }

    @Override
    public String getDefaultName() {
        return "Extension relation";
    }

    @Override
    public String getDefaultShortName() {
        return "Extension";
    }

    @Override
    public Image getImage() {
        return IArchimateImages.ImageFactory.getImage(
            IArchimateImages.ICON_EXTENSION_CONNECTION_16);
    }

    @Override
    public ImageDescriptor getImageDescriptor() {
        return IArchimateImages.ImageFactory.getImageDescriptor(
            IArchimateImages.ICON_EXTENSION_CONNECTION_16);
    }
}
```

3. Finally, register this UI Provider. Edit the "ElementUIFactory.java" class by adding the following line to the ElementUIFactory constructor:

```
registerProvider(new ExtensionConnectionUIProvider());
```

Add a ArchiLabelProvider

In com.archimatetool.editor.ui add the new relationship to the ArchiLabelProvider

```
    public String getRelationshipSentence(IArchimateRelationship relation) {
        if(relation != null) {
            if(relation.getSource() != null && relation.getTarget() != null) {
                String nameSource = ArchiLabelProvider.INSTANCE.getLabel(relation.getSource());
                String nameTarget = ArchiLabelProvider.INSTANCE.getLabel(relation.getTarget());

                switch(relation.eClass().getClassifierID()) {
                    case IArchimatePackage.SPECIALIZATION_RELATIONSHIP:
                        return NLS.bind(Messages.ArchimateLabelProvider_3, nameSource, nameTarget);

                    case IArchimatePackage.COMPOSITION_RELATIONSHIP:
                        return NLS.bind(Messages.ArchimateLabelProvider_4, nameSource, nameTarget);

                    case IArchimatePackage.AGGREGATION_RELATIONSHIP:
                        return NLS.bind(Messages.ArchimateLabelProvider_5, nameSource, nameTarget);

                    case IArchimatePackage.TRIGGERING_RELATIONSHIP:
                        return NLS.bind(Messages.ArchimateLabelProvider_6, nameSource, nameTarget);

                    case IArchimatePackage.FLOW_RELATIONSHIP:
                        return NLS.bind(Messages.ArchimateLabelProvider_7, nameSource, nameTarget);

                    case IArchimatePackage.ACCESS_RELATIONSHIP:
                        return NLS.bind(Messages.ArchimateLabelProvider_8, nameSource, nameTarget);

                    case IArchimatePackage.ASSOCIATION_RELATIONSHIP:
                        return NLS.bind(Messages.ArchimateLabelProvider_9, nameSource, nameTarget);

                    case IArchimatePackage.ASSIGNMENT_RELATIONSHIP:
                        return NLS.bind(Messages.ArchimateLabelProvider_10, nameSource, nameTarget);
                    case IArchimatePackage.REALIZATION_RELATIONSHIP:
                        return NLS.bind(Messages.ArchimateLabelProvider_11, nameSource, nameTarget);

                    case IArchimatePackage.SERVING_RELATIONSHIP:
                        return NLS.bind(Messages.ArchimateLabelProvider_12, nameSource, nameTarget);

                    case IArchimatePackage.INFLUENCE_RELATIONSHIP:
                        return NLS.bind(Messages.ArchimateLabelProvider_13, nameSource, nameTarget);

                    case IArchimatePackage.ENABLES_RELATIONSHIP:
                        return NLS.bind(Messages.ArchimateLabelProvider_36, nameSource, nameTarget);

                    case IArchimatePackage.CONTRIBUTES_RELATIONSHIP:
                        return NLS.bind(Messages.ArchimateLabelProvider_37, nameSource, nameTarget);

                    default:
                        return ""; //$NON-NLS-1$
                }
            }
        }

        return ""; //$NON-NLS-1$
    }
```

Note these labels are sequential so the numbers may look a bit odd.

And

```
public String getRelationshipPhrase(EClass eClass, boolean reverseDirection) {
    if(eClass == null) {
        return ""; //$NON-NLS-1$
    }

    switch(eClass.getClassifierID()) {
        case IArchimatePackage.SPECIALIZATION_RELATIONSHIP:
            return reverseDirection ? Messages.ArchimateLabelProvider_14 : Messages.ArchimateLabelProvider_15;

        case IArchimatePackage.COMPOSITION_RELATIONSHIP:
            return reverseDirection ? Messages.ArchimateLabelProvider_16 : Messages.ArchimateLabelProvider_17;

        case IArchimatePackage.AGGREGATION_RELATIONSHIP:
            return reverseDirection ? Messages.ArchimateLabelProvider_18 : Messages.ArchimateLabelProvider_19;

        case IArchimatePackage.TRIGGERING_RELATIONSHIP:
            return reverseDirection ? Messages.ArchimateLabelProvider_20 : Messages.ArchimateLabelProvider_21;

        case IArchimatePackage.FLOW_RELATIONSHIP:
            return reverseDirection ? Messages.ArchimateLabelProvider_22 : Messages.ArchimateLabelProvider_23;

        case IArchimatePackage.ACCESS_RELATIONSHIP:
            return reverseDirection ? Messages.ArchimateLabelProvider_24 : Messages.ArchimateLabelProvider_25;

        case IArchimatePackage.ASSOCIATION_RELATIONSHIP:
            return reverseDirection ? Messages.ArchimateLabelProvider_26 : Messages.ArchimateLabelProvider_27;

        case IArchimatePackage.ASSIGNMENT_RELATIONSHIP:
            return reverseDirection ? Messages.ArchimateLabelProvider_28 : Messages.ArchimateLabelProvider_29;

        case IArchimatePackage.REALIZATION_RELATIONSHIP:
            return reverseDirection ? Messages.ArchimateLabelProvider_30 : Messages.ArchimateLabelProvider_31;

        case IArchimatePackage.SERVING_RELATIONSHIP:
            return reverseDirection ? Messages.ArchimateLabelProvider_32 : Messages.ArchimateLabelProvider_33;

        case IArchimatePackage.INFLUENCE_RELATIONSHIP:
            return reverseDirection ? Messages.ArchimateLabelProvider_34 : Messages.ArchimateLabelProvider_35;

        default:
            return ""; //$NON-NLS-1$
    }
}
```

6. Finally, register this UI Provider. Edit the "plugin.xml" in com.archimatetool.editor by adding the following element element to the `<extension name="Archimate Relationships" point="com.archimatetool.editor.objectUIProvider">` element:

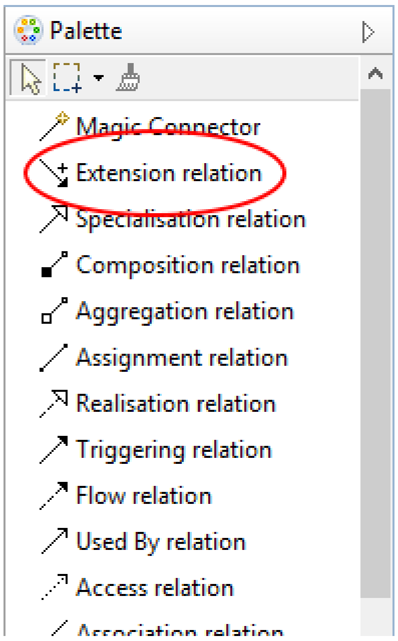
```
<uiProvider
    class="com.archimatetool.editor.ui.factory.elements.NewElementNameUIProvider"
    id="com.archimatetool.editor.NewElementNameUIProvider">
</uiProvider>
```

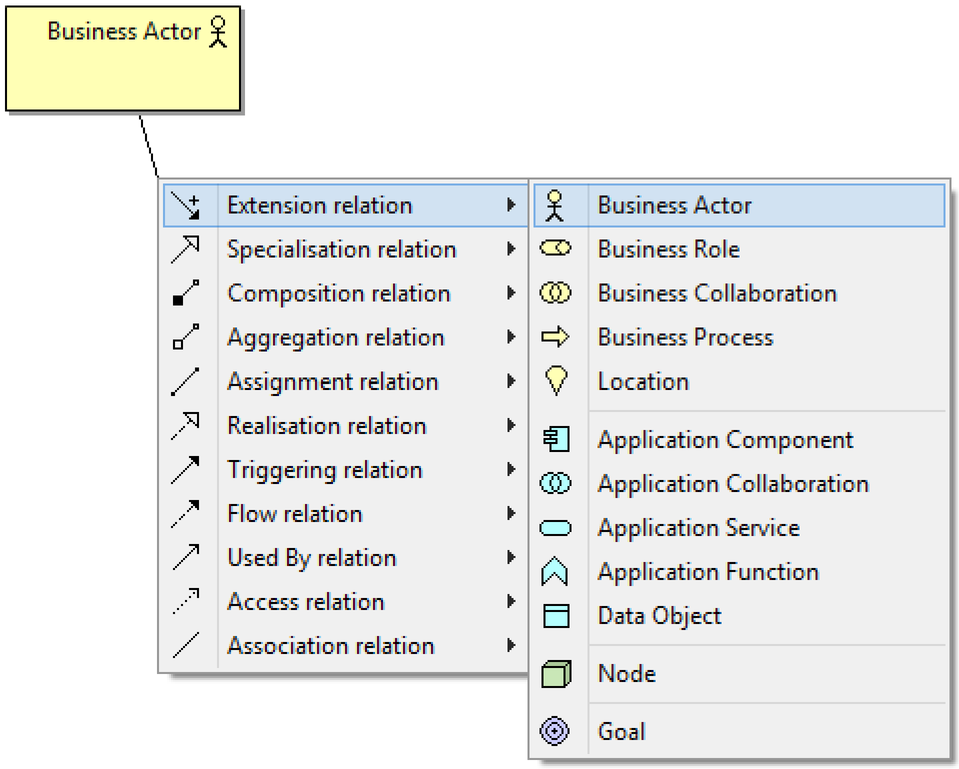
In our example this would be

```
<uiProvider
    class="com.archimatetool.editor.ui.factory.elements.ChannelUIProvider"
    id="com.archimatetool.editor.ChannelUIProvider">
</uiProvider>
```

Run Archi from Eclipse

Running Archi from Eclipse (see link) will result in your new relationship appearing in the context menus on the diagram and in the editor palette. The new relationship will appear in a diagram, and any relationship rules will be evident:





See also

To add a new element to the model, see [Extending the Model with a new Element]()

Add a new viewpoint

Edite the viewpoints.xml in com.archimatetool.model to add the viewpoint and the allowed elements.

```
<viewpoint id="customer_journey_cooperation">
  <name xml:lang="en">Customer Journey View</name>

  <concept>$ApplicationElements$</concept>
  <concept>BusinessActor</concept>
  <concept>BusinessCollaboration</concept>
  <concept>BusinessEvent</concept>
  <concept>BusinessFunction</concept>
  <concept>BusinessInteraction</concept>
  <concept>BusinessInterface</concept>
  <concept>BusinessObject</concept>
  <concept>BusinessProcess</concept>
  <concept>CustomerJourney</concept>
  <concept>BusinessRole</concept>
  <concept>BusinessService</concept>
  <concept>Location</concept>
  <concept>Representation</concept>
</viewpoint>
```