How to modify Initial and Final States in Yakindu SCT Editor

Initial and Final States are pseudo states in Yakindu SCT. The Initial State can only have one outgoing transition and no incoming. It is represented with a black filled circle. The Final State can only have one incoming transition and no outgoing. To create an Initial or Final State, we have to select Initial or Final State element in the palette and then draw it on the appropriate region.

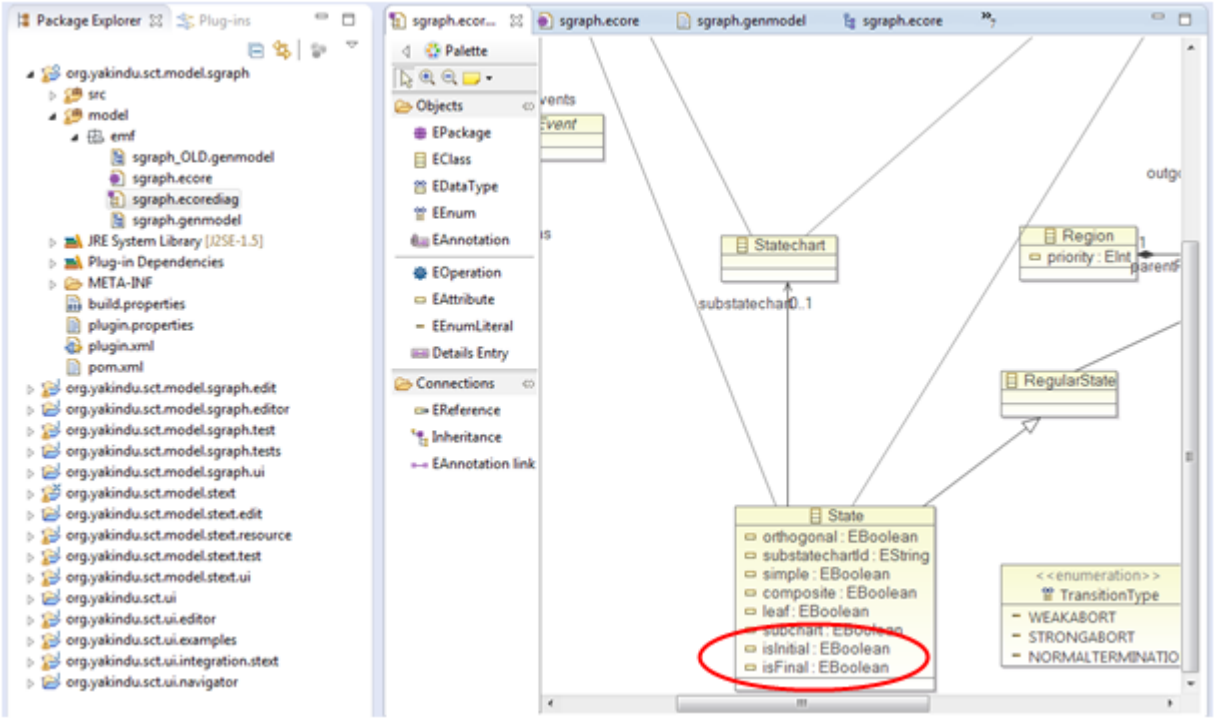
Goal: Initial and Final States are normal or composite states and are represented respectively with a thicker border or a double line border.

To modify the Initial and Final States in Yakindu SCT Editor, we need the following steps:

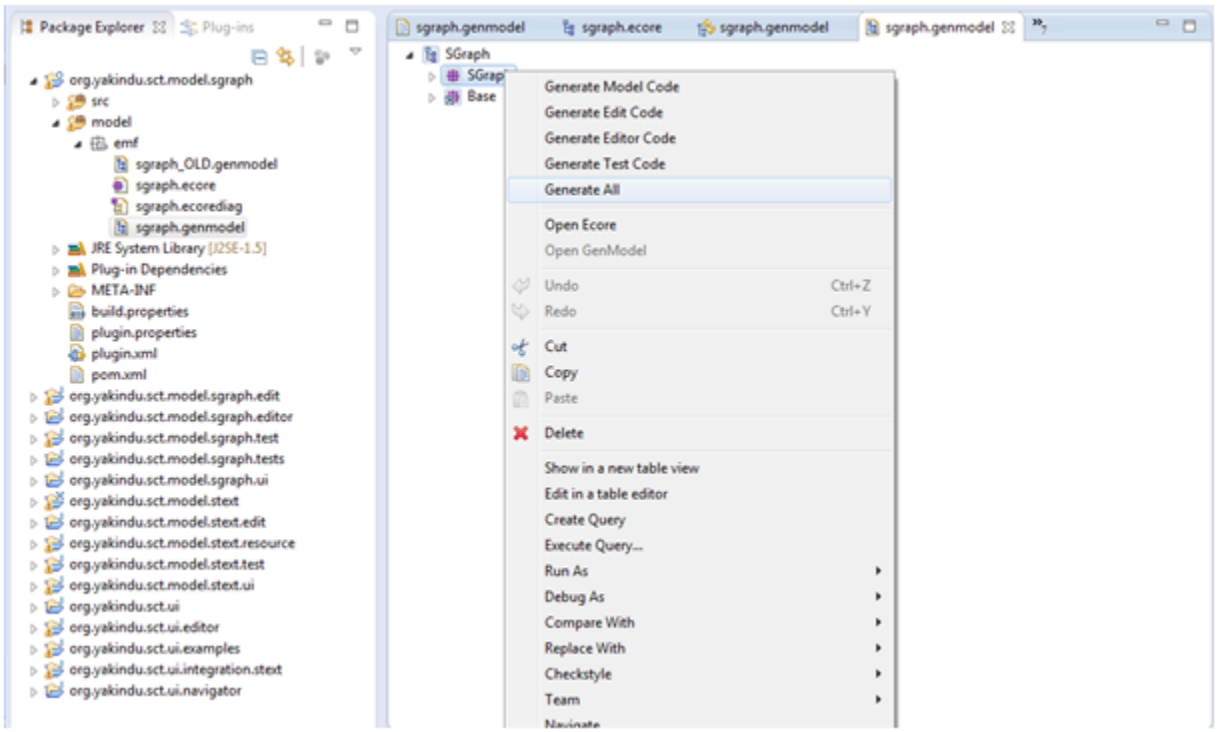
- Modify the Metamodel in org.yakindu.sct.model.sgraph/model/emf/sgraph.genmodel
- Modify the State Figure in org.yakindu.sct.ui.editor/src/org/yakindu/sct/ui/editor/editor/figures/StateFigure.java
- Modify the Propertysheets in org.yakindu.sct.ui.editor/src/org/yakindu/sct/ui/editor/propertysheets/StatePropertySection.java
- Modify the State EditPart in org.yakindu.sct.ui.editor/src/org/yakindu/sct/ui/editor/editparts/StateEditPart.java
- Remove the Initial and Final State Icon from the palette in org.yakindu.sct.ui.editor/plugin.xml
- Modify the SGraphValidator in org.yakindu.sct.model.sgraph/src/org/yakindu/sct/model/sgraph/util/SGraphValidator.java
- Modify the creation wizard in org.yakindu.sct.ui.editor/src/org/yakindu/sct/ui/editor/factories/FactoryUtils.java
- Clean the code

Modify the Metamodel in org.yakindu.sct.model.sgraph/model/emf/sgraph.genmodel

by adding two attributes (isInitial, isFinal) to the State Class.



And then generate the EMF Model Code.



Modify the State Figure in org.yakindu.sct.ui.editor/src/org/yakindu/sct/ui/editor/editor/figures/StateFigure.java

by setting the constants NORMAL_BORDER_WIDTH and THICK_BORDER_WIDTH and adding two variables isDoubleLineBorder and isThickBorder. If the value of isDoubleLineBorder is set to true, the RoundedRectangle is represented with a double line border. If the value of isThickBorder is set to true, the RoundedRectangle is represented with a thicker line border.

```
public class StateFigure extends RoundedRectangle {

    public static final int BLUR_SHADOW_WIDTH = 5;

    /** width of the double rectangle border. */
    public static final int NORMAL_BORDER_WIDTH = 1;
    public static final int THICK_BORDER_WIDTH = 3;

    private WrappingLabel nameFigure;
    private Figure textCompartmentPane;
    private Figure figureCompartmentPane;

    private boolean isDoubleLineBorder = false;
    private boolean isThickBorder = false;

    public StateFigure(IMapMode mapMode, boolean isDoubleLineBorder,
            boolean isThickBorder) {
        this.isDoubleLineBorder = isDoubleLineBorder;
        this.isThickBorder = isThickBorder;
        GridLayout layout = new GridLayout(1, false);
```

Overwrite the outlineShape method.

```
  /**
   * Draw the outline twice. If isDoubleLineBorder is true, draw a second
   * rectangle inside the first one.
   *
   * @param graphics
   *            the graphics object
   */
  protected void outlineShape(final Graphics graphics) {
      // if the isDoubleLineBorder is true, draw a second rectangle.
      if (isDoubleLineBorder) {
          //call the super method to draw the first RoundRectangle
          super.outlineShape(graphics);
          // set the second rectangle. It is calculated according the border
          // line width
          Rectangle rect = new Rectangle();
          rect.x = getBounds().x + 3 * getLineWidth() / 2 + 1;
          rect.y = getBounds().y + 3 * getLineWidth() / 2 + 1;
          rect.width = getBounds().width - 3 * getLineWidth() - 2;
          rect.height = getBounds().height - 3 * getLineWidth() - 2;
          // Draw the second rectangle inside the first one
          graphics.drawRoundRectangle(rect, getCornerDimensions().width
                  - (getLineWidth() + 1) * getLineWidth(),
                  getCornerDimensions().height - (getLineWidth() + 1)
                          * getLineWidth());
      } else {
          this.setLineWidth(getBorderWidth());
          super.outlineShape(graphics);
      }
  }
```

And add the method getBorderWidth

```
/**
 * Return the Rounded Rectangle border width
 *
 * @return the border width
 */
private int getBorderWidth() {
    return (isThickBorder ? THICK_BORDER_WIDTH : NORMAL_BORDER_WIDTH);
}
```

Modify the Propertysheets in org.yakindu.sct.ui.editor/src/org/yakindu/sct/ui/editor/propertysheets/StatePropertySection.java

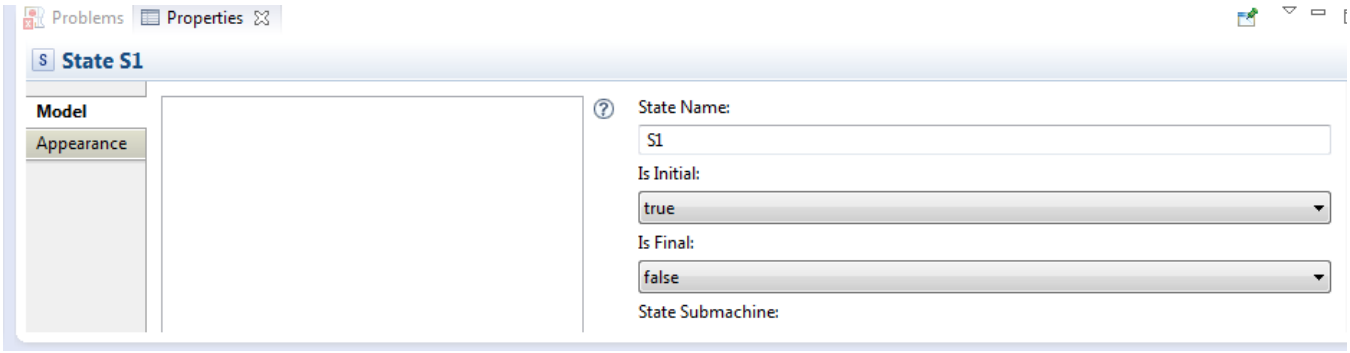
by adding an Is Initial ComboBox and an Is Final ComboBox. The user may select the option true or false.

```java
/**
 * Create the isInitial Combo. It allows to select false or true
 *
 * @param parent
 *            the parent Composite
 */
private void createIsInitialControl(Composite parent) {
    Label kindLabel = getToolkit().createLabel(parent, "Is Initial: ");
    GridDataFactory.fillDefaults().applyTo(kindLabel);
    isInitialKindViewer = new ComboViewer(parent, SWT.READ_ONLY | SWT.SINGLE);
    isInitialKindViewer.setContentProvider(new ArrayContentProvider());
    isInitialKindViewer.setLabelProvider(new LabelProvider());
    isInitialKindViewer.add(false);
    isInitialKindViewer.add(true);
    GridDataFactory.fillDefaults().grab(true, false)
            .applyTo(isInitialKindViewer.getControl());
}

/**
 * Create the isFinal Combo. It allows to select false or true
 *
 * @param parent
 *            the parent Composite
 */
private void createIsFinalControl(Composite parent) {
    Label kindLabel = getToolkit().createLabel(parent, "Is Final: ");
    GridDataFactory.fillDefaults().applyTo(kindLabel);
    isFinalKindViewer = new ComboViewer(parent, SWT.READ_ONLY | SWT.SINGLE);
    isFinalKindViewer.setContentProvider(new ArrayContentProvider());
    isFinalKindViewer.setLabelProvider(new LabelProvider());
    isFinalKindViewer.add(false);
    isFinalKindViewer.add(true);
    GridDataFactory.fillDefaults().grab(true, false)
            .applyTo(isFinalKindViewer.getControl());
}
```

```java
@Override
public void bindModel(EMFDataBindingContext context) {
    bindNameControl(context);
    bindIsInitialKindControl(context);
    bindIsFinalKindControl(context);
    bindSpecificationControl(context);
    orderElementControl.refreshInput();
    updateLabel();
}
```

```java
/**
 * This method enables to select the isInitial attribute
 *
 * @param context
 */
private void bindIsInitialKindControl(EMFDataBindingContext context) {
    IEMFValueProperty property = EMFEditProperties.value(
            TransactionUtil.getEditingDomain(eObject),
            SGraphPackage.Literals.STATE__IS_INITIAL);
    context.bindValue(
            ViewerProperties.singleSelection()
                    .observe(isInitialKindViewer), property
                    .observe(eObject));
}

/**
 * This method enables to select the isFinal attribute
 *
 * @param context
 */
private void bindIsFinalKindControl(EMFDataBindingContext context) {
    IEMFValueProperty property = EMFEditProperties.value(
            TransactionUtil.getEditingDomain(eObject),
            SGraphPackage.Literals.STATE__IS_FINAL);
    context.bindValue(
            ViewerProperties.singleSelection()
                    .observe(isFinalKindViewer), property
                    .observe(eObject));
}
```

After that, the State Properties View will look like this:



Modify the State EditPart in org.yakindu.sct.ui.editor/src/org/yakindu/sct/ui/editor/editparts/StateEditPart.java

by adding the following code lines in the method handleNotificationEvent to update the border width or to add a double line border to the state, if the user changes the Is Initial or Is Final options.

```
@Override
protected void handleNotificationEvent(Notification notification) {

    if (notification.getFeature() == NotationPackage.Literals.DRAWER_STYLE__COLLAPSED) {

        if (isCollapsed()) {
            installEditPolicy(EditPolicy.PRIMARY_DRAG_ROLE,
                    new NonResizableEditPolicyEx());

        } else {

            installEditPolicy(EditPolicy.PRIMARY_DRAG_ROLE,
                    new ResizableEditPolicyEx());
        }
        refreshVisuals();
    }
    // update the state Border
    if (notification.getFeature() == SGraphPackage.Literals.STATE__IS_INITIAL
            || notification.getFeature() == SGraphPackage.Literals.STATE__IS_FINAL) {
        getPrimaryShape().setThickBorder(
                resolveSemanticElement().isIsInitial());
        getPrimaryShape().setDoubleLineBorder(
                resolveSemanticElement().isIsFinal());
        getPrimaryShape().refreshBorder();
        getPrimaryShape().repaint();

    }

    super.handleNotificationEvent(notification);
}
```

Remove the Initial and Final State Icon from the palette in org.yakindu.sct.ui.editor/plugin.xml

by removing them from the list of Elements situated in the file org.yakindu.sct.ui.editor/plugin.xml.

```
            id="org.yakindu.sct.ui.editor.Region"
            kind="tool"
            label="Region"
            large_icon="icons/obj32/Region-32.png"
            path="/tools/"
            small_icon="icons/obj16/Region-16.png">
      </entry>
      <entry
            description="Creates an initial state"
            id="org.yakindu.sct.ui.editor.Entry"
            kind="tool"
            label="Initial State"
            large_icon="icons/obj32/Initial-State-32.png"
            path="/tools/"
            small_icon="icons/obj16/Initial-State-16.png">
      </entry>
       <entry
            description="Creates a shallow history"
            id="org.yakindu.sct.ui.editor.ShallowHistory"
            kind="tool"
            label="Shallow History"
            large_icon="icons/obj32/Shallow-History-32.png"
            path="/tools/"
            small_icon="icons/obj16/Shallow-History-16.png">
      </entry>
```

...

```
            id="org.yakindu.sct.ui.editor.DeepHistory"
            kind="tool"
            label="Deep History"
            large_icon="icons/obj32/Deep-History-32.png"
            path="/tools/"
            small_icon="icons/obj16/Deep-History-16.png">
      </entry>
      <entry
            description="Creates a final state"
            id="org.yakindu.sct.ui.editor.FinalState"
            kind="tool"
            label="Final State"
            large_icon="icons/obj32/Final-State-32.png"
            path="/tools/"
            small_icon="icons/obj16/Final-State-16.png">
      </entry>
      <!--
        <entry
            description="Creates an exit point"
            id="org.yakindu.sct.ui.editor.Exit"
            kind="tool"
            label="Exit Point"
            large_icon="icons/obj32/Exit-Point-32.png"
            path="/tools/"
            small_icon="icons/obj16/Exit-Point-16.png">
```

Modify the SGraphValidator in org.yakindu.sct.model.sgraph/src/org/yakindu/sct/model/sgraph/util/SGraphValidator.java

Modify the SGraphValidator.

```
 * @generated NOT
 */
public boolean validateVertex_IncomingTransitionCount(Vertex vertex,
        DiagnosticChain diagnostics, Map<Object, Object> context) {
    if (vertex instanceof State && !((State) vertex).isIsInitial()) {
        if (vertex.getIncomingTransitions().size() > 0) {
            return true;
        }
        TreeIterator<EObject> eAllContents = vertex.eAllContents();
        while (eAllContents.hasNext()) {
            EObject next = eAllContents.next();
            if (next instanceof State) {
                EList<Transition> incomingTransitions = ((State) next)
                        .getIncomingTransitions();
                for (Transition transition : incomingTransitions) {
                    if (!EcoreUtil.isAncestor(vertex,
                            transition.getSource())) {
                        return true;
                    }
                }
            }
        }
        return error(vertex, diagnostics, ISSUE_NODE_NOT_REACHABLE);
    } else if (vertex.getIncomingTransitions().size() == 0
            && !((State) vertex).isIsInitial()) {
        return error(vertex, diagnostics, ISSUE_NODE_NOT_REACHABLE);
    }

    // modified by wah
    // if (vertex.getIncomingTransitions().size() > 0
    // && vertex instanceof Entry
    // && ((Entry) vertex).getKind().equals(EntryKind.INITIAL)) {
    // return warning(vertex, diagnostics,
    // ISSUE_INITIAL_ENTRY_WITH_IN_TRANS);
    // }
```

Modify the creation wizard in org.yakindu.sct.ui.editor/src/org/yakindu/sct/ui/editor/factories/FactoryUtils.java

The creation wizard creates a new diagram which includes an Initial State and a Normal State. We should modify the creation wizard by modifying the method createStatechartModel (org.yakindu.sct.ui.editor/src/org/yakindu/sct/ui/editor/factories/FactoryUtils.java)

```
// Add to resource
resource.getContents().add(statechart);
resource.getContents().add(diagram);
// Create an initial region
Region region = SGraphFactory.eINSTANCE.createRegion();
region.setName(INITIAL_REGION_NAME);
statechart.getRegions().add(region);
Node regionView = ViewService.createNode(diagram, region,
        SemanticHints.REGION, preferencesHint);
setRegionViewLayoutConstraint(regionView);
// // Create an initial state
State initialState = SGraphFactory.eINSTANCE.createState();
initialState.setName(INITIAL_STATE_NAME);
initialState.setIsInitial(true);
// initialState.setKind(EntryKind.INITIAL);
region.getVertices().add(initialState);
Node initialStateView = ViewService.createNode(
        getRegionCompartmentView(regionView), initialState,
        SemanticHints.STATE, preferencesHint);
setInitialStateViewLayoutConstraint(initialStateView);
// Create the first state
State state = SGraphFactory.eINSTANCE.createState();
state.setIsFinal(true);
state.setName(FINAL_STATE_NAME);
region.getVertices().add(state);
Node stateNode = ViewService.createNode(
        getRegionCompartmentView(regionView), state,
        SemanticHints.STATE, preferencesHint);
setStateViewLayoutConstraint(stateNode);
// Create the transition from Initial State to State
Transition transition = SGraphFactory.eINSTANCE.createTransition();
transition.setSource(initialState);
transition.setTarget(state);
initialState.getOutgoingTransitions().add(transition);
ViewService.createEdge(initialStateView, stateNode, transition,
        SemanticHints.TRANSITION, preferencesHint);
```

Clean the code

Clean the code where the old Initial and Final States are implemented (FinalStateFigure, InitialStateFigure, ...)