How to add Transition Types in Yakindu SCT Editor

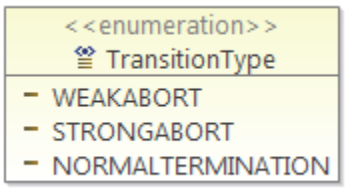
There are different transition types in KIELER SyncCharts, which are "weak abortion", "strong abortion", or "normal termination". The "weak abortion" transition is represented as a simple black arrow. The "strong abortion" transition is represented as a black arrow with a little red circle at the start point. The "normal termination" transition has a little green triangle at the start point.

To add the transition types in Yakindu SCT Editor, we need the following steps:

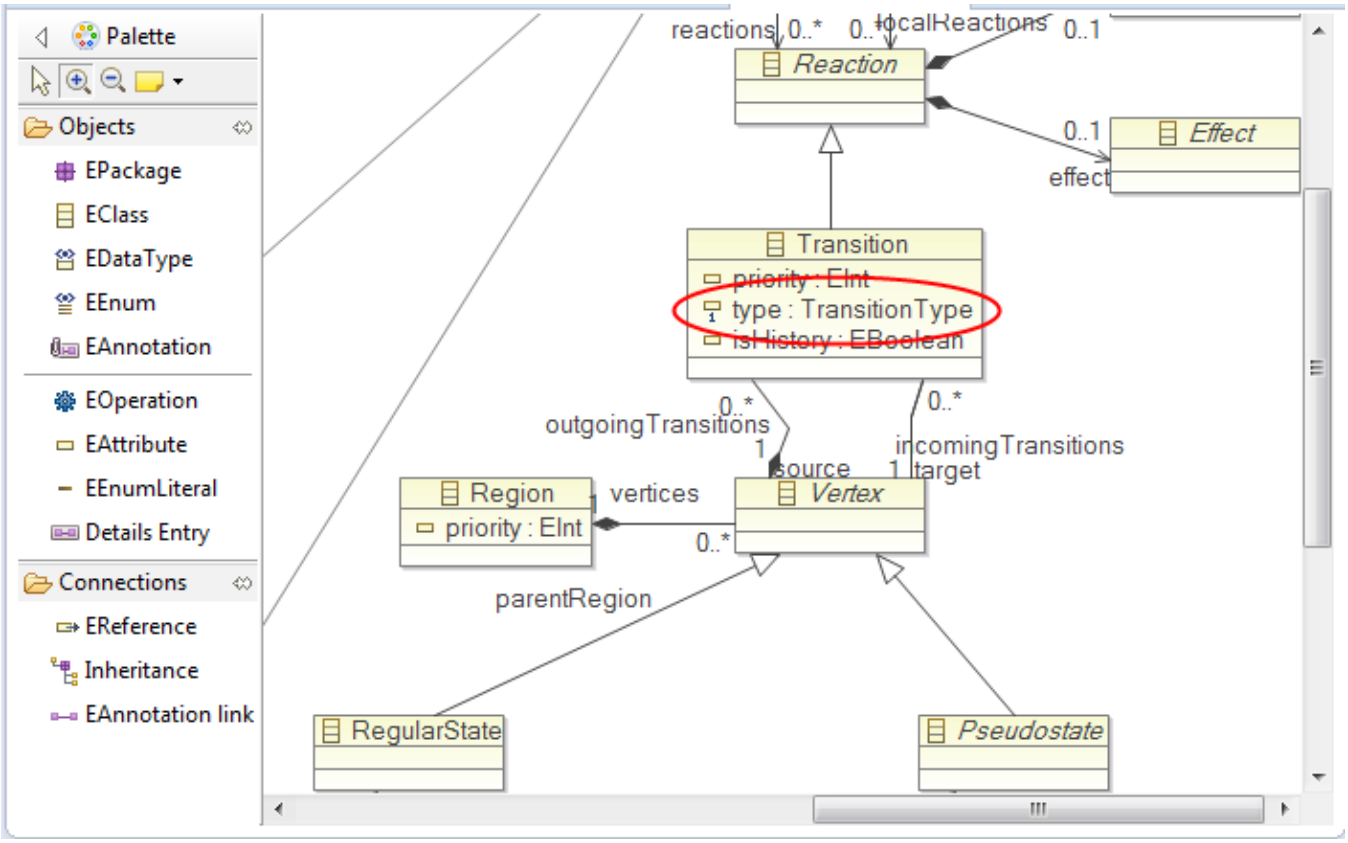
- Modify the Metamodel in org.yakindu.sct.model.sgraph/model/emf/sgraph.genmodel
- Modify the State Figure in org.yakindu.sct.ui.editor/src/org/yakindu/sct/ui/editor/editor/figures/TransitionFigure.java
- Modify the Transition Propertysheets in org.yakindu.sct.ui.editor/src/org/yakindu/sct/ui/editor/propertysheets/TransitionPropertySection.java
- Modify the Transition EditPart in org.yakindu.sct.ui.editor/src/org/yakindu/sct/ui/editor/editparts/TransitionEditPart.java

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

Create an enumeration type "TransitionType".



Add the attribute "type" to the Transition 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/TransitionFigure.java

Modify the constructor

```
public TransitionFigure(IMapMode mapMode, TransitionType transitionType,
        boolean isHistory) {
    this.transitionType = transitionType;
    this.isHistory = isHistory;
    this.mapMode = mapMode;
    setLineWidth(mapMode.DPtoLP(LINE_WIDTH));
    // draw the arrow target decoration
    setTargetDecoration(createTargetDecoration());
    // draw the arrow source decoration.
    switch (transitionType) {
    case NORMALTERMINATION:
        // If the transition is a normal termination transition, set a
        // green triangle as the source of the arrow
        setSourceDecoration(createNormalTerminationDecoration());
        break;
    case STRONGABORT:
        // If the transition is a strong abortion transition, set a red
        // circle as the source of the arrow.
        setSourceDecoration(createStrongAbortDecoration());
        break;
    default:
        // If the transition is a weak abortion transition, the arrow has no
        // source decoration
        setSourceDecoration(null);
        break;
    }
}
```

Add the methods createNormalTerminationDecoration() and createStrongAbortDecoration() to draw the different source decorators.

```
private static final int TERMINATION_SIZE = 2;
private static final double TERMINATION_X_SCALE = 4.0;
private static final double TERMINATION_Y_SCALE = 2.5;

/**
 * Create the normal termination decoration.
 *
 * @return The decoration.
 */
private RotatableDecoration createNormalTerminationDecoration() {
    PolygonDecoration triangleDecoration = new PolygonDecoration();
    triangleDecoration.setLineWidth(getMapMode().DPtoLP(LINE_WIDTH));
    triangleDecoration.setForegroundColor(ColorConstants.black);
    triangleDecoration.setBackgroundColor(ColorConstants.green);
    PointList triangleDecorationPoints = new PointList();
    triangleDecorationPoints.addPoint(0, TERMINATION_SIZE);
    triangleDecorationPoints.addPoint(-TERMINATION_SIZE, 0);
    triangleDecorationPoints.addPoint(0, -TERMINATION_SIZE);
    triangleDecoration.setTemplate(triangleDecorationPoints);
    triangleDecoration.setScale(TERMINATION_X_SCALE, TERMINATION_Y_SCALE);
    return triangleDecoration;
}
```

```java
private static final int STRONG_ABORT_SIZE = 2;
private static final double STRONG_ABORT_SCALE = 2.0;

/**
 * Create the strong abortion decoration.
 *
 * @return The decoration.
 */
private RotatableDecoration createStrongAbortDecoration() {
    PolygonDecoration circleDecoration = new CircleDecoration();
    circleDecoration.setLineWidth(getMapMode().DPtoLP(LINE_WIDTH));
    circleDecoration.setForegroundColor(ColorConstants.black);
    circleDecoration.setBackgroundColor(ColorConstants.red);
    PointList circleDecorationPoints = new PointList();
    circleDecorationPoints.addPoint(STRONG_ABORT_SIZE, STRONG_ABORT_SIZE);
    circleDecorationPoints.addPoint(-STRONG_ABORT_SIZE, -STRONG_ABORT_SIZE);
    circleDecoration.setTemplate(circleDecorationPoints);
    circleDecoration.setScale(STRONG_ABORT_SCALE, STRONG_ABORT_SCALE);
    return circleDecoration;
}
```

And then create a method refreshSourceDecoration() to refresh the source decorator, if the transition type is changed.

```java
/**
 * Refresh the figure source decoration
 */
public void refreshSourceDecoration() {
    switch (transitionType) {
    case NORMALTERMINATION:
        setSourceDecoration(createNormalTerminationDecoration());
        break;
    case STRONGABORT:
        setSourceDecoration(createStrongAbortDecoration());
        break;
    default:
        setSourceDecoration(null);
        break;
    }
}
```

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

Add a transitionType ComboBox. The user can select between the different options: Weak Abort, Strong Abort, and Normal Termination.

```java
/**
 * Create the selection Combo. It allows to select the transition type
 * (WEAKABORT, STRONGABORT or NORMALTERMINATION)
 *
 * @param parent
 *            the parent Composite
 */
private void createTransitionTypeControl(Composite parent) {
    Label kindLabel = getToolkit().createLabel(parent, "Transition Type: ");
    GridDataFactory.fillDefaults().applyTo(kindLabel);
    transitionTypeKindViewer = new ComboViewer(parent, SWT.READ_ONLY
            | SWT.SINGLE);
    transitionTypeKindViewer.setContentProvider(new ArrayContentProvider());
    transitionTypeKindViewer.setLabelProvider(new LabelProvider());
    transitionTypeKindViewer.setInput(TransitionType.values());
    GridDataFactory.fillDefaults().grab(true, false)
            .applyTo(transitionTypeKindViewer.getControl());
}
```

```java
/**
 * This method enables to select the transition type
 *
 * @param context
 */
private void bindTransitionTypeKindControl(EMFDataBindingContext context) {
    IEMFValueProperty property = EMFEditProperties.value(
            TransactionUtil.getEditingDomain(eObject),
            SGraphPackage.Literals.TRANSITION__TYPE);
    context.bindValue(
            ViewerProperties.singleSelection()
                    .observe(transitionTypeKindViewer), property
                    .observe(eObject));
}
```

```java
@Override
public void bindModel(EMFDataBindingContext context) {
    IEMFValueProperty modelProperty = EMFEditProperties.value(
            TransactionUtil.getEditingDomain(eObject),
            SGraphPackage.Literals.SPECIFICATION_ELEMENT__SPECIFICATION);
    ISWTObservableValue uiProperty = WidgetProperties.text(SWT.FocusOut)
            .observe(textControl);
    context.bindValue(uiProperty, modelProperty.observe(eObject));
    bindTransitionTypeKindControl(context);
    bindIsHistoryKindControl(context);
}

@Override
protected void createRightColumnControls(Composite rightColumn) {
    createTransitionTypeControl(rightColumn);
    createIsHistoryControl(rightColumn);
}
```

Modify the Transition EditPart in org.yakindu.sct.ui.editor/src/org/yakindu/sct/ui/editor/editparts/TransitionEditPart.java

Add the following code lines in the method handleNotificationEvent() to update the source decorator when the user changes the Transition Type.

```java
@Override
protected void handleNotificationEvent(Notification notification) {
    if (NotationPackage.eINSTANCE.getFontStyle().getEAllAttributes()
            .contains(notification.getFeature())) {
        refreshFont();
    } else {
        super.handleNotificationEvent(notification);
    }
    // Update the transition source decoration when transition type is modified
    if (SGraphPackage.eINSTANCE.getTransition_Type().equals(
            notification.getFeature())) {
        // set the transition type
        ((TransitionFigure) getFigure())
                .setTransitionType(resolveSemanticElement().getType());
        // refresh the figure
        ((TransitionFigure) getFigure()).refreshSourceDecoration();
    }
```