```
public abstract class AbstractChangePattern extends
CompoundCommandWithAttributes {
	protected final Graph graph;
	private Map<Bendpoint, Point> bendpointMoveDeltas;
	static boolean ok = false;

	public AbstractChangePattern(Graph graph) {
		Assert.isNotNull(graph);

		this.graph = graph;
		this.bendpointMoveDeltas = new HashMap<Bendpoint, Point>();
		setAttribute(CHANGE_PATTERN_TYPE, getName());
	}

	protected Node addAddActivityCommand(Graph graph, String activityName,
Point location) {
		IGraphElementDescriptor descriptor =
EditorRegistry.getDescriptor(EditorRegistry.BPMN_ACTIVITY);
		Node activity = (Node) descriptor.createModel(graph);
		CreateNodeCommand command = new CreateNodeCommand(graph,
activity, location, activityName);
		add(command);
		return activity;
	}

	protected Node addAddAndGatewayCommand(Graph graph, Point location) {
		IGraphElementDescriptor elementDescriptor =
EditorRegistry.getDescriptor(EditorRegistry.BPMN_AND_GATEWAY);
		Node andSplit = (Node) elementDescriptor.createModel(graph);
		CreateNodeCommand command = new CreateNodeCommand(graph,
andSplit, location);
		add(command);
		return andSplit;
	}

	protected Node addAddConditionCommand(Graph graph, String
conditionName, Point location) {
		IGraphElementDescriptor descriptor =
EditorRegistry.getDescriptor(EditorRegistry.CHANGE_PATTERN_CONDITION);
		Node condition = (Node) descriptor.createModel(graph);
		CreateNodeCommand command = new CreateNodeCommand(graph,
condition, location, conditionName);
		add(command);
		return condition;
	}

	protected CreateEdgeCommand addAddEdgeCommand(Graph graph, Node
source, Node target) {
		IGraphElementDescriptor edgeDescriptor =
EditorRegistry.getDescriptor(EditorRegistry.BPMN_SEQUENCE_FLOW);
		Edge newEdge = (Edge) edgeDescriptor.createModel(graph);
		CreateEdgeCommand createEdgeCommand = new
CreateEdgeCommand(graph, newEdge, source, target, null);
		add(createEdgeCommand);
		return createEdgeCommand;
	}

	protected CreateEdgeCommand addAddEdgeCommand(Graph graph, Node
source, Node target, Point point) {
```

```
            IGraphElementDescriptor edgeDescriptor =
EditorRegistry.getDescriptor(EditorRegistry.BPMN_SEQUENCE_FLOW);
            Edge newEdge = (Edge) edgeDescriptor.createModel(graph);
            newEdge.addBendPoint(point, 0);
            CreateEdgeCommand createEdgeCommand = new
CreateEdgeCommand(graph, newEdge, source, target, null);
            add(createEdgeCommand);
            return createEdgeCommand;
      }

      protected CreateEdgeCommand addAddEdgeDashCommand(Graph graph, Node
source, Node target) {
            IGraphElementDescriptor edgeDescriptor =
EditorRegistry.getDescriptor(EditorRegistry.BPMN_DASH_SEQUENCE_FLOW);
            Edge newEdge = (Edge) edgeDescriptor.createModel(graph);
            CreateEdgeCommand createEdgeCommand = new
CreateEdgeCommand(graph, newEdge, source, target, null);
            add(createEdgeCommand);
            return createEdgeCommand;
      }

      protected CreateEdgeCommand addAddSplitEdgeCommand(Graph graph, Node
source, Node target, Point point) {
            IGraphElementDescriptor edgeDescriptor =
EditorRegistry.getDescriptor(EditorRegistry.BPMN_SEQUENCE_FLOW);
            Edge newEdge = (Edge) edgeDescriptor.createModel(graph);
            newEdge.addBendPoint(point, 0);
            newEdge.addBendPoint(new Point(point.x + 80, point.y), 1);
            CreateEdgeCommand createEdgeCommand = new
CreateEdgeCommand(graph, newEdge, source, target, null);
            add(createEdgeCommand);
            return createEdgeCommand;
      }

      private void addAll(List<AbstractGraphCommand> commands) {
            for (AbstractGraphCommand abstractGraphCommand : commands) {
                  add(abstractGraphCommand);
            }
      }

      protected Node addXorGatewayCommand(Graph graph, Point location) {
            IGraphElementDescriptor elementDescriptor =
EditorRegistry.getDescriptor(EditorRegistry.BPMN_XOR_GATEWAY);
            Node andSplit = (Node) elementDescriptor.createModel(graph);
            CreateNodeCommand command = new CreateNodeCommand(graph,
andSplit, location);
            add(command);
            return andSplit;
      }

      @Override
      public void execute() {
            List<Edge> edges = graph.getEdges();
            for (Edge edge : edges) {
                  List<Bendpoint> bendPoints = edge.getBendPoints();
                  if (bendPoints != null) {
                        for (Bendpoint bendpoint : bendPoints) {
                              if
(bendpointMoveDeltas.containsKey(bendpoint)) {
```

```
                                        Point delta =
bendpointMoveDeltas.get(bendpoint);
                                        Point newLocation =
bendpoint.getLocation().getCopy().translate(delta);
                                        add(new MoveEdgeBendPointCommand(edge,
bendPoints.indexOf(bendpoint), newLocation));
                                  }
                            }
                      }
                }

                super.execute();

                List<IGraphOptimizer> optimizers = getGraphOptimizers();
                boolean optimized = false;
                do {
                      optimized = false;
                      for (IGraphOptimizer optimizer : optimizers) {
                            if (optimized) {
                                  break;
                            }
                            optimized = optimizer.optimize();
                            if (optimized) {
                                  addAll(optimizer.getCommands());
                            }

                      }
                } while (optimized);
          }

          protected Point getActivitySize() {
                return ((ActivityDescriptor)
EditorRegistry.getDescriptor(EditorRegistry.BPMN_ACTIVITY)).getInitialSize().
getCopy();
          }

          protected Point getConditonSize() {
                return ((ConditionDescriptor)
EditorRegistry.getDescriptor(EditorRegistry.CHANGE_PATTERN_CONDITION)).getIni
tialSize().getCopy();
          }

          protected List<Edge> getConnections(Node node1, Node node2) {
                List<Edge> sourceConnections = new ArrayList<Edge>();
                for (Edge edge : node1.getSourceConnections()) {
                      Node target = edge.getTarget();
                      if (target == null) {
                            continue;
                      }
                      if (target.equals(node2)) {
                            sourceConnections.add(edge);
                      }
                }

                for (Edge edge : node2.getSourceConnections()) {
                      Node target = edge.getTarget();
                      if (target == null) {
                            continue;
                      }
```

```
                if (target.equals(node1)) {
                        sourceConnections.add(edge);
                }
        }
        return sourceConnections;
    }

    protected List<IGraphOptimizer> getGraphOptimizers() {
        return Collections.emptyList();
    }

    public abstract String getName();

    private void moveBendPoint(Bendpoint bendpoint, Point delta) {
        if (!bendpointMoveDeltas.containsKey(bendpoint)) {
            bendpointMoveDeltas.put(bendpoint, new Point());
        }

        Point point = bendpointMoveDeltas.get(bendpoint);
        point.translate(delta);
    }

    protected void moveHorizontally(int cutOff, Point delta, List<Edge>
toIgnore) {
        for (Node node : graph.getNodes()) {
            if (node.getLocation().x >= cutOff) {
                add(new MoveNodeCommand(node, delta));
            }
        }
        for (Edge edge : graph.getEdges()) {
            if (toIgnore.contains(edge)) {
                continue;
            }

            if (edge.getBendPoints() != null) {
                for (Bendpoint bendpoint : edge.getBendPoints()) {
                    if (bendpoint.getLocation().x >= cutOff) {
                        moveBendPoint(bendpoint, delta);
                    }
                }
            }
        }
    }

    protected void moveVertically(int cutOff, Point delta) {
        for (Node node : graph.getNodes()) {
            if (node.getLocation().y >= cutOff) {
                add(new MoveNodeCommand(node, delta));
            }
        }
        for (Edge edge : graph.getEdges()) {
            if (edge.getBendPoints() != null) {
                for (Bendpoint bendpoint : edge.getBendPoints()) {
                    if (bendpoint.getLocation().y >= cutOff) {
                        moveBendPoint(bendpoint, delta);
                    }
                }
            }
        }
```

```
	}

	protected void moveVerticallyUp(int cutOff, Point delta) {
		for (Node node : graph.getNodes()) {
			if (node.getLocation().y <= cutOff) {
				add(new MoveNodeCommand(node, delta));
			}
		}
		for (Edge edge : graph.getEdges()) {
			if (edge.getBendPoints() != null) {
				for (Bendpoint bendpoint : edge.getBendPoints()) {
					if (bendpoint.getLocation().y <= cutOff) {
						moveBendPoint(bendpoint, delta);
					}
				}
			}
		}
	}

}
```

```
public abstract class AbstractChangePatternAction extends Action {


	protected final GraphicalGraphViewerWithFlyoutPalette viewer;
	protected final PlainMultiLineButton button;
	private ISelectionChangedListener selectionChangeListener;
	static int count = 0;
	static boolean ok = false;

	public AbstractChangePatternAction(PlainMultiLineButton button,
GraphicalGraphViewerWithFlyoutPalette viewer) {
		Assert.isNotNull(viewer);
		Assert.isNotNull(button);
		this.button = button;
		this.viewer = viewer;

		selectionChangeListener = new ISelectionChangedListener() {
			@Override
			public void selectionChanged(SelectionChangedEvent event)
{
				IStructuredSelection selection =
(IStructuredSelection)
AbstractChangePatternAction.this.viewer.getViewer().getSelection();
				setEnabled(isChangePatternExecutable(selection));
			}
		};

	viewer.getViewer().addSelectionChangedListener(selectionChangeListener
);

		button.addSelectionListener(new SelectionAdapter() {
			@Override
			public void widgetSelected(SelectionEvent e) {
				run();
			}
		});
	}

	protected abstract AbstractChangePattern
createChangePattern(IStructuredSelection selection);

	public void dispose() {

	viewer.getViewer().removeSelectionChangedListener(selectionChangeListe
ner);
	}

	protected Edge extractConnection(IStructuredSelection selection) {
		if (selection.size() != 2) {
			return null;
		}

		Object object = selection.toList().get(0);
		if (!(object instanceof NodeEditPart)) {
			return null;
		}
		Node node1 = ((NodeEditPart) object).getModel();
		Object object2 = selection.toList().get(1);
		if (!(object2 instanceof NodeEditPart)) {
```

```
                return null;
            }
            Node node2 = ((NodeEditPart) object2).getModel();

            Edge connection = getConnection(node1, node2);
            return connection;
        }

        protected Edge extractConnectionXOR(IStructuredSelection selection) {
            if (selection.size() != 2) {
                return null;
            }

            Object object = selection.toList().get(0);
            if (!(object instanceof NodeEditPart)) {
                return null;
            }
            Node node1 = ((NodeEditPart) object).getModel();
            if (!node1.getDescriptor().getName().equals("XOR")) {
                return null;
            }
            Object object2 = selection.toList().get(1);
            if (!(object2 instanceof NodeEditPart)) {
                return null;
            }
            Node node2 = ((NodeEditPart) object2).getModel();
            if (!node2.getDescriptor().getName().equals("XOR")) {
                return null;
            }
            Edge connection = getConnection(node1, node2);
            return connection;
        }

        protected String getActivityName(String initialString) {
            Shell shell =
PlatformUI.getWorkbench().getActiveWorkbenchWindow().getShell();
            InputDialog dialog = new InputDialog(shell, "Activity Name",
"Please enter a name.", initialString, new IInputValidator() {
                @Override
                public String isValid(String newText) {
                    if (newText.trim().isEmpty()) {
                        return "Please enter a valid name";
                    }
                    return null;
                }
            });
            if (!(dialog.open() == Window.OK)) {
                return null;
            }
            return dialog.getValue();
        }

        protected Edge getConnection(Node node1, Node node2) {
            for (Edge edge : node1.getSourceConnections()) {
                Node target = edge.getTarget();
                if (target == null) {
                    continue;
                }
                if (target.equals(node2)) {
```

```
                        return edge;
                    }
                }

                for (Edge edge : node2.getSourceConnections()) {
                    Node target = edge.getTarget();
                    if (target == null) {
                        continue;
                    }
                    if (target.equals(node1)) {
                        return edge;
                    }
                }
                return null;
            }

            @SuppressWarnings("unchecked")
            protected SESEChecker getSESEChecker(IStructuredSelection selection) {
                Set<Node> nodes = new HashSet<Node>();
                List<NodeEditPart> list = selection.toList();
                for (NodeEditPart nodeEditPart : list) {
                    nodes.add(nodeEditPart.getModel());
                }

                SESEChecker seseChecker = new SESEChecker(nodes);
                return seseChecker;
            }

            @SuppressWarnings("rawtypes")
            protected boolean isChangePatternExecutable(IStructuredSelection
selection) {
                HashSet<Node> nodes = new HashSet<Node>();
                List list = selection.toList();
                for (Object object : list) {
                    if (!(object instanceof NodeEditPart)) {
                        return false;
                    }
                    Node node = ((NodeEditPart) object).getModel();
                    nodes.add(node);
                }

                SESEChecker seseChecker = new SESEChecker(nodes);
                return seseChecker.isSESEFragment();
            }

            protected boolean isInXOR(IStructuredSelection selection) {
                if (selection.size() < 2) {
                    return false;
                }
                HashSet<Node> nodes = new HashSet<Node>();
                List list = selection.toList();
                for (Object object : list) {
                    if (!(object instanceof NodeEditPart)) {
                        return false;
                    }
                    Node node = ((NodeEditPart) object).getModel();
                    nodes.add(node);
                }
                if (nodes.size() >= 2) {
```

```
                  SESEChecker seseChecker = new SESEChecker(nodes);
                  if (!seseChecker.isSESEFragment()) {
                        if (nodes.size() == 2) {
                              List<Node> nodes_list = new
ArrayList<Node>(nodes);
                              Node node1 =
nodes_list.get(0).getSourceConnections().size() > 1 ? nodes_list.get(0) :
nodes_list.get(1);
                              Node node2 =
nodes_list.get(0).getSourceConnections().size() > 1 ? nodes_list.get(1) :
nodes_list.get(0);
                              if
(!node1.getDescriptor().getName().equals("XOR") ||
!node2.getDescriptor().getName().equals("XOR")) {
                                    return false;
                              }
                              if (node1.getProperty("Join") == null ||
(Long) node1.getProperty("Join") != node2.getId()) {
                                    return false;
                              }
                        }
                  } else {
                        Node node1 = seseChecker.getFirstNode();
                        Node node2 = seseChecker.getLastNode();
                        if (!node1.getDescriptor().getName().equals("XOR")
|| !node2.getDescriptor().getName().equals("XOR")) {
                              return false;
                        }
                        if (node1.getProperty("Join") == null || (Long)
node1.getProperty("Join") != node2.getId()) {
                              return false;
                        }
                  }
            }
            return true;
      }

      @Override
      public void run() {
            IStructuredSelection selection = (IStructuredSelection)
viewer.getViewer().getSelection();
            AbstractChangePattern changePattern =
createChangePattern(selection);
            if (changePattern != null) {

      viewer.getViewer().getEditDomain().getCommandStack().execute(changePat
tern);
            }
            viewer.getViewer().setSelection(new StructuredSelection());
      }

      @Override
      public void setEnabled(boolean enabled) {
            super.setEnabled(enabled);
            button.setEnabled(enabled);
      }
```

```
public abstract class AbstractGraphOptimizer implements IGraphOptimizer {

	protected final Graph graph;
	private List<AbstractGraphCommand> commands;

	public AbstractGraphOptimizer(Graph graph) {
		Assert.isNotNull(graph);
		this.graph = graph;
		commands = new ArrayList<AbstractGraphCommand>();
	}

	protected void add(AbstractGraphCommand command) {
		commands.add(command);
	}

	protected abstract boolean doOptimize();

	protected Node findCorrespondingAndJoin(Node innerAndSplit) {
		List<Edge> sourceConnections =
innerAndSplit.getSourceConnections();
		for (Edge edge : sourceConnections) {
			return findCorrespondingAndJoin(edge.getTarget(),
innerAndSplit);
		}
		throw new IllegalStateException("Illegal structure");
	}

	private Node findCorrespondingAndJoin(Node current, Node
innerAndSplit) {
		SESEChecker seseChecker = new SESEChecker(innerAndSplit,
current);
		if (seseChecker.isSESEFragment()) {
			return current;
		}

		for (Edge edge : current.getSourceConnections()) {
			return findCorrespondingAndJoin(edge.getTarget(),
innerAndSplit);
		}

		throw new IllegalStateException("Illegal structure");
	}

	/**
	 * @param target1
	 * @param node
	 * @return
	 */
	protected Edge getEdgeToNotComingFromNode(Node target1, Node node) {
		List<Edge> targetConnections = target1.getTargetConnections();
		for (Edge edge : targetConnections) {
			if (!edge.getSource().equals(node)) {
				return edge;
			}
		}

		throw new IllegalArgumentException("Could not find another edge
not starting at node " + node.getNameNullSafe());
	}
```

```
	protected boolean isAndGateway(Node node) {
		return
EditorRegistry.BPMN_AND_GATEWAY.equals(node.getDescriptor().getId());
	}

	protected boolean isJoin(Node node) {
		return node.getTargetConnections().size() >= 2 &&
node.getSourceConnections().size() == 1;
	}

	protected boolean isSplit(Node node) {
		return node.getSourceConnections().size() >= 2;
	}

	protected boolean isXorGateway(Node node) {
		return
EditorRegistry.BPMN_XOR_GATEWAY.equals(node.getDescriptor().getId());
	}

	public List<AbstractGraphCommand> getCommands() {
		return Collections.unmodifiableList(commands);
	}

	@Override
	public boolean optimize() {
		commands.clear();
		boolean optimized = doOptimize();

		for (AbstractGraphCommand abstractGraphCommand : commands) {
			abstractGraphCommand.execute();
		}

		return optimized;
	}

}
```

```
public class AndGraphOptimizer extends AbstractGraphOptimizer {

	public AndGraphOptimizer(Graph graph) {
		super(graph);
	}

	@Override
	protected boolean doOptimize() {
		List<Node> nodes = graph.getNodes();
		for (Node andSplit : nodes) {
			if (!isSplit(andSplit) || !isAndGateway(andSplit)) {
				continue;
			}

			List<Edge> sourceConnections =
andSplit.getSourceConnections();
			if (sourceConnections.size() == 2) {
				return handleAndWithTwoBranches(andSplit);
			}

			for (Edge edge : sourceConnections) {
				Node target = edge.getTarget();
				SESEChecker seseChecker = new SESEChecker(andSplit,
target);
				if (seseChecker.isSESEFragment()) {
					add(new DeleteEdgeCommand(edge));
					return true;
				}
			}
		}

		return false;
	}

	private boolean handleAndWithTwoBranches(Node node) {
		Edge edge1 = node.getSourceConnections().get(0);
		Edge edge2 = node.getSourceConnections().get(1);
		Edge incomingEdge = node.getTargetConnections().get(0);

		Node target1 = edge1.getTarget();
		if (isJoin(target1) && isAndGateway(target1)) {
			remove(node, edge1, edge2, incomingEdge);
			return true;
		}

		Node target2 = edge2.getTarget();
		if (isJoin(target2) && isAndGateway(target2)) {
			remove(node, edge2, edge1, incomingEdge);
			return true;
		}

		return false;
	}

	private void remove(Node node, Edge edge1, Edge edge2, Edge
incomingEdge) {
		Node target1 = edge1.getTarget();
		Node target2 = edge2.getTarget();
```

```
            add(new ReconnectEdgeCommand(incomingEdge,
incomingEdge.getSource(), target2));

            Edge edge = getEdgeToNotComingFromNode(target1, node);
            Edge outgoingEdge = target1.getSourceConnections().get(0);
            add(new ReconnectEdgeCommand(edge, edge.getSource(),
outgoingEdge.getTarget()));

            add(new DeleteEdgeCommand(edge1));
            add(new DeleteEdgeCommand(edge2));
            add(new DeleteEdgeCommand(outgoingEdge));
            add(new DeleteNodeCommand(node));
            add(new DeleteNodeCommand(target1));
      }

}
```

```
public class AndJoinOptimizer extends AbstractGraphOptimizer {

	public AndJoinOptimizer(Graph graph) {
		super(graph);
	}

	@Override
	protected boolean doOptimize() {
		List<Node> nodes = graph.getNodes();
		for (Node outerAndSplit : nodes) {
			if (!isAndGateway(outerAndSplit) ||
!isSplit(outerAndSplit)) {
				continue;
			}

			List<Edge> sourceConnections =
outerAndSplit.getSourceConnections();
			for (Edge edge : sourceConnections) {
				Node innerAndSplit = edge.getTarget();
				if (isAndGateway(innerAndSplit) &&
isSplit(innerAndSplit)) {
					Node innerAndJoin =
findCorrespondingAndJoin(innerAndSplit);
					Node outerAndJoin =
innerAndJoin.getSourceConnections().get(0).getTarget();
					if (isAndGateway(outerAndJoin) &&
isJoin(outerAndJoin)) {
						SESEChecker seseChecker = new
SESEChecker(outerAndSplit, outerAndJoin);
						if (seseChecker.isSESEFragment()) {

	performOptimization(innerAndSplit, innerAndJoin);
							return true;
						}
					}
				}
			}
		}

		return false;
	}

	private void optimizeAndJoin(Node innerAndJoin) {
		Edge andJoinEdge = innerAndJoin.getSourceConnections().get(0);
		Node outerAndJoin = andJoinEdge.getTarget();

		List<Edge> targetConnections =
innerAndJoin.getTargetConnections();
		for (Edge edge : targetConnections) {
			add(new ReconnectEdgeCommand(edge, edge.getSource(),
outerAndJoin));
		}

		add(new DeleteEdgeCommand(andJoinEdge));
		add(new DeleteNodeCommand(innerAndJoin));
	}

	private void optimizeAndSplit(Node innerAndSplit) {
		Edge andSplitEdge = innerAndSplit.getTargetConnections().get(0);
```

```
            Node outerAndSplit = andSplitEdge.getSource();
            List<Edge> sourceConnections =
innerAndSplit.getSourceConnections();
            for (Edge edge : sourceConnections) {
                  add(new ReconnectEdgeCommand(edge, outerAndSplit,
edge.getTarget()));
            }

            add(new DeleteEdgeCommand(andSplitEdge));
            add(new DeleteNodeCommand(innerAndSplit));
      }

      /**
       * @param innerAndSplit
       * @param innerAndJoin
       */
      private void performOptimization(Node innerAndSplit, Node
innerAndJoin) {
            optimizeAndSplit(innerAndSplit);
            optimizeAndJoin(innerAndJoin);
      }
}
```

```
public class ChangePatternDialog extends TitleAreaDialog {

	private ChangePatternDialogComposite composite;
	private final GraphicalGraphViewerWithFlyoutPalette viewer;
	private List<AbstractChangePatternAction> actions;

	public ChangePatternDialog(Shell parentShell,
GraphicalGraphViewerWithFlyoutPalette viewer) {
		super(parentShell);
		setBlockOnOpen(false);
		Assert.isNotNull(viewer);
		this.viewer = viewer;
		actions = new ArrayList<AbstractChangePatternAction>();
	}

	@Override
	public boolean close() {
		for (AbstractChangePatternAction action : actions) {
			action.dispose();
		}
		return super.close();
	}

	@Override
	protected Control createButtonBar(Composite parent) {
		return null;
	}

	@Override
	protected Control createDialogArea(Composite parent) {
		Composite container = (Composite)
super.createDialogArea(parent);
		GridLayout layout = new GridLayout();
		layout.marginWidth = 0;
		layout.marginHeight = 0;
		container.setLayout(layout);
		container.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true,
true));
		container.setBackground(SWTResourceManager.getColor(255, 255,
255));
		setTitle("Change Pattern");
		setMessage("Please select the appropriate change pattern");
		composite = new ChangePatternDialogComposite(container,
SWT.NONE);

		initializeChangePattern();
		return container;
	}

	@Override
	protected int getShellStyle() {
		return ~SWT.CLOSE & SWT.DIALOG_TRIM | SWT.TOOL;
	}

	@Override
	protected void handleShellCloseEvent() {
		// ignore
	}
```

```
	protected void initializeChangePattern() {
		if
(isPatternEnabled(IConfiguration.CONFIGURATION_ALTERNATIVE_INSERT)) {
			actions.add(new
ConfigationAlternativeAction(composite.getConfiguratioAlternativeButton(),
viewer));
		}
		if (isPatternEnabled(IConfiguration.CONFIGURABLE_REGION_DELETE))
{
			actions.add(new
DeleteConfigurableRegionAction(composite.getDeleteConfigurableRegionButton(),
viewer));
		}
		if (isPatternEnabled(IConfiguration.CONFIGURABLE_REGION_INSERT))
{
			actions.add(new
ConfigurableRegionAction(composite.getConfigurableRegionButtonButton(),
viewer));
		}

		if
(isPatternEnabled(IConfiguration.CONFIGURATION_CONSTRAINT_INSERT)) {
			actions.add(new
ConfigurationConstraintAction(composite.getConfigurationConstraintButton(),
viewer));
		}

		if
(isPatternEnabled(IConfiguration.CONFIGURATION_CONSTRAINT_DELETE)) {
			actions.add(new
DeleteConfigurationConstraintAction(composite.getDeleteConfigurationConstrain
tButton(), viewer));
		}

		if
(isPatternEnabled(IConfiguration.CONFIGURATION_ALTERNATIVE_DELETE)) {
			actions.add(new
DeleteConfigurationAlternativeAction(composite.getDeleteConfigurationAlternat
iveButton(), viewer));
		}

		if (isPatternEnabled(IConfiguration.CHANGE_PATTERN_UNDO)) {
			actions.add(new
UndoChangePatternAction(composite.getUndoButton(), viewer));
		}

		if (isPatternEnabled(IConfiguration.CONFIGURATION)) {
			actions.add(new
ConfigurationAction(composite.getConfigurationButton(), viewer));
		}
	}

	private boolean isPatternEnabled(String pattern) {
		return CheetahPlatformConfigurator.getBoolean(pattern);
	}
}
```

```
public class ChangePatternDialogComposite extends Composite {

	private PlainMultiLineButton configurableRegionButton;
	private PlainMultiLineButton deleteConfigurableRegionButton;
	private PlainMultiLineButton configuratioAlternativeButton;
	private PlainMultiLineButton deleteConfigurationAlternativeButton;
	private PlainMultiLineButton configurationConstraintButton;
	private PlainMultiLineButton deleteConfigurationConstraintButton;
	private PlainMultiLineButton configurationButton;
	private PlainMultiLineButton undoButton;

	public ChangePatternDialogComposite(Composite parent, int style) {
		super(parent, style);

		setBackground(SWTResourceManager.getColor(255, 255, 255));
		setLayout(new GridLayout(2, true));
		GridData layoutData = new GridData(SWT.FILL, SWT.FILL, true,
true);
		setLayoutData(layoutData);
		GridData buttonLayoutData = new GridData(SWT.FILL, SWT.FILL,
true, false);

		Image image = null;
		Image disabledImage = null;
		if
(CheetahPlatformConfigurator.getBoolean(IConfiguration.CONFIGURABLE_REGION_IN
SERT)) {
			image =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp1.png");
			disabledImage =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp1Disable.png");
			configurableRegionButton = new PlainMultiLineButton(this,
SWT.HORIZONTAL, "Insert Configurable Region", image, disabledImage);
			configurableRegionButton.setLayoutData(buttonLayoutData);
			configurableRegionButton.setEnabled(false);
		}

		if
(CheetahPlatformConfigurator.getBoolean(IConfiguration.CONFIGURABLE_REGION_DE
LETE)) {
			image =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp2.png");
			disabledImage =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp2Disable.png");
			deleteConfigurableRegionButton = new
PlainMultiLineButton(this, SWT.HORIZONTAL, "Delete Configurable Region",
image,
					disabledImage);

	deleteConfigurableRegionButton.setLayoutData(buttonLayoutData);
			deleteConfigurableRegionButton.setEnabled(false);
		}
```

```
            if
(CheetahPlatformConfigurator.getBoolean(IConfiguration.CONFIGURATION_ALTERNAT
IVE_INSERT)) {
                  image =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp3.png");
                  disabledImage =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp3Disable.png");
                  configuratioAlternativeButton = new
PlainMultiLineButton(this, SWT.HORIZONTAL, "Insert Configuration
Alternative", image,
                              disabledImage);

      configuratioAlternativeButton.setLayoutData(buttonLayoutData);
                  configuratioAlternativeButton.setEnabled(false);
            }

            if
(CheetahPlatformConfigurator.getBoolean(IConfiguration.CONFIGURATION_ALTERNAT
IVE_DELETE)) {
                  image =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp4.png");
                  disabledImage =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp4Disable.png");
                  deleteConfigurationAlternativeButton = new
PlainMultiLineButton(this, SWT.HORIZONTAL, "Delete Configuration
Alternative",
                              image, disabledImage);

      deleteConfigurationAlternativeButton.setLayoutData(buttonLayoutData);
                  deleteConfigurationAlternativeButton.setEnabled(false);
            }

            if
(CheetahPlatformConfigurator.getBoolean(IConfiguration.CONFIGURATION_CONSTRAI
NT_INSERT)) {
                  image =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp8.png");
                  disabledImage =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp8Disable.png");
                  configurationConstraintButton = new
PlainMultiLineButton(this, SWT.HORIZONTAL, "Insert Configuration Constraint",
image,
                              disabledImage);

      configurationConstraintButton.setLayoutData(buttonLayoutData);
                  configurationConstraintButton.setEnabled(false);
            }

            if
(CheetahPlatformConfigurator.getBoolean(IConfiguration.CONFIGURATION_CONSTRAI
NT_DELETE)) {
```

```
                    image =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp9.png");
                    disabledImage =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp9Disable.png");
                    deleteConfigurationConstraintButton = new
PlainMultiLineButton(this, SWT.HORIZONTAL, "Delete Configuration Constraint",
image,
                                  disabledImage);

      deleteConfigurationConstraintButton.setLayoutData(buttonLayoutData);
                    deleteConfigurationConstraintButton.setEnabled(false);
            }

            if
(CheetahPlatformConfigurator.getBoolean(IConfiguration.CHANGE_PATTERN_UNDO))
{
                    image =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/changepattern/undo16.png");
                    disabledImage =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/changepattern/undodisabled16.png");
                    undoButton = new PlainMultiLineButton(this,
SWT.HORIZONTAL, "Undo", image, disabledImage);
                    undoButton.setLayoutData(buttonLayoutData);
                    undoButton.setEnabled(false);
            }

            if
(CheetahPlatformConfigurator.getBoolean(IConfiguration.CONFIGURATION)) {
                    image =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp10.png");
                    disabledImage =
ResourceManager.getPluginImage(Activator.getDefault(),
"img/configurablechangepattern/cp10Disable.png");
                    configurationButton = new PlainMultiLineButton(this,
SWT.HORIZONTAL, "Configuration", image, disabledImage);
                    configurationButton.setLayoutData(buttonLayoutData);
                    configurationButton.setEnabled(false);
            }
      }

      /**
       * Returns the configurableRegionButton.
       *
       * @return the configurableRegionButton
       */
      public PlainMultiLineButton getConfigurableRegionButtonButton() {
            return configurableRegionButton;
      }

      /**
       * Returns the configuratioAlternativeButton.
       *
       * @return the configuratioAlternativeButton
       */
```

```
	public PlainMultiLineButton getConfiguratioAlternativeButton() {
		return configuratioAlternativeButton;
	}

	/**
	 * Returns the configurationButton.
	 *
	 * @return the configurationButton
	 */
	public PlainMultiLineButton getConfigurationButton() {
		return configurationButton;
	}

	/**
	 * Returns the configurationConstraintButton.
	 *
	 * @return the configurationConstraintButton
	 */
	public PlainMultiLineButton getConfigurationConstraintButton() {
		return configurationConstraintButton;
	}

	/**
	 * Returns the deleteConfigurableRegionButton.
	 *
	 * @return the deleteConfigurableRegionButton
	 */
	public PlainMultiLineButton getDeleteConfigurableRegionButton() {
		return deleteConfigurableRegionButton;
	}

	/**
	 * Returns the deleteConfigurationAlternativeButton.
	 *
	 * @return the deleteConfigurationAlternativeButton
	 */
	public PlainMultiLineButton getDeleteConfigurationAlternativeButton()
{
		return deleteConfigurationAlternativeButton;
	}

	/**
	 * Returns the deleteConfigurationConstraintButton.
	 *
	 * @return the deleteConfigurationConstraintButton
	 */
	public PlainMultiLineButton getDeleteConfigurationConstraintButton() {
		return deleteConfigurationConstraintButton;
	}

	public PlainMultiLineButton getUndoButton() {
		return undoButton;
	}

}
```

```
public class ConfigationAlternative extends AbstractChangePattern {
	public static final String CONFIGURATION_ALTERNATIVE_INSERT =
"CONFIGURATION_ALTERNATIVE";

	/**
	 * @param graph
	 * @param selection
	 * @param activityName
	 */

	public ConfigationAlternative(Graph graph, IStructuredSelection
selection, String activityName) {
		super(graph);
		HashSet<Node> nodes = new HashSet<Node>();
		List list = selection.toList();
		for (Object object : list) {
			if ((object instanceof NodeEditPart)) {
				Node node = ((NodeEditPart) object).getModel();
				nodes.add(node);
			}
		}
		Node source = null;
		Node target = null;
		SESEChecker seseChecker = new SESEChecker(nodes);
		if (!seseChecker.isSESEFragment()) {
			if (nodes.size() == 2) {
				List<Node> nodes_list = new ArrayList<Node>(nodes);
				source =
nodes_list.get(0).getSourceConnections().size() > 1 ? nodes_list.get(0) :
nodes_list.get(1);
				target =
nodes_list.get(0).getSourceConnections().size() > 1 ? nodes_list.get(1) :
nodes_list.get(0);
			}
		} else {
			source = seseChecker.getFirstNode();
			target = seseChecker.getLastNode();
		}
		Assert.isNotNull(activityName);
		List<Edge> sourceConnections = getConnections(source, target);
		if (sourceConnections.size() > 0) {
			Edge edge = sourceConnections.get(0);
			Point location =
source.getBounds().getTopRight().getCopy().translate(36,
sourceConnections.size() == 1 ? -90 : 60);
			Node activity = addAddActivityCommand(graph, activityName,
location);
			edge.removeBendPoint(1);
			edge.removeBendPoint(0);
			ReconnectEdgeCommand reconnectEdgeCommand = new
ReconnectEdgeCommand(edge, source, activity);
			add(reconnectEdgeCommand);
			addAddEdgeCommand(graph, activity, target);
		} else {
			Point location =
source.getBounds().getTopRight().getCopy().translate(36, -15);
			Node activity = addAddActivityCommand(graph, activityName,
location);
			addAddEdgeCommand(graph, source, activity);
```

```
                        addAddEdgeCommand(graph, activity, target);
                }
                Point delta = getActivitySize().translate(20, 0);
                delta.y = 0;
                int size = target.getBounds().getLeft().x -
source.getBounds().getRight().x;
                int cutOff = source.getBounds().getRight().x;
                if (size < getActivitySize().x + 36) {
                        moveHorizontally(cutOff, delta, new ArrayList<Edge>());
                }
        }

        @Override
        public String getName() {
                return CONFIGURATION_ALTERNATIVE_INSERT;
        }
}
```

```
public class ConfigationAlternativeAction extends AbstractChangePatternAction
{

	/**
	 * @param configationAlternativeButton
	 * @param viewer
	 */

	public ConfigationAlternativeAction(PlainMultiLineButton
configationAlternativeButton, GraphicalGraphViewerWithFlyoutPalette viewer) {
		super(configationAlternativeButton, viewer);
	}

	@Override
	protected AbstractChangePattern
createChangePattern(IStructuredSelection selection) {
		String name = getActivityName(null);
		if (name == null) {
			return null;
		}
		AbstractChangePattern changePattern = new
ConfigationAlternative(viewer.getGraph(), selection, name);
		return changePattern;
	}

	@Override
	protected boolean isChangePatternExecutable(IStructuredSelection
selection) {
		return (isInXOR(selection));
	}
}
```

```
public class ConfigurableRegion extends AbstractChangePattern {
	public static final String CONFIGURABLE_REGION_INSERT =
"CONFIGURABLE_REGION";

	/**
	 * @param graph
	 * @param selection
	 */

	public ConfigurableRegion(Graph graph, IStructuredSelection selection)
{
		super(graph);
		Object object = selection.toList().get(0);
		Node node1 = ((NodeEditPart) object).getModel();
		object = selection.toList().get(1);
		Node node2 = ((NodeEditPart) object).getModel();
		List<Edge> connections = getConnections(node1, node2);
		Edge edge = connections.get(0);
		Node source = edge.getSource();
		Node target = edge.getTarget();

		Point xorSplitLocation = source
				.getBounds()
				.getTopRight()
				.getCopy()
				.translate(
						20,

	source.getDescriptor().getName().equals("XOR") && (connections.size()
== 2) ? 70 : source.getDescriptor().getName()
							.equals("XOR")
							&&
target.getDescriptor().getName().equals("XOR") && (connections.size() == 1) ?
-70 : -10);
		Node xorSplit = addXorGatewayCommand(graph, xorSplitLocation);

		Point pointLocation1 = xorSplitLocation.getCopy().translate(15,
60);
		Point pointLocation2 = xorSplitLocation.getCopy().translate(15,
-30);

		Point xorJoinLocation = xorSplitLocation.getCopy().translate(80,
0);
		Node xorJoin = addXorGatewayCommand(graph, xorJoinLocation);
		xorJoin.setProperty("Split", xorSplit.getId());
		xorSplit.setProperty("Join", xorJoin.getId());
		addAddSplitEdgeCommand(graph, xorSplit, xorJoin,
pointLocation1);
		addAddSplitEdgeCommand(graph, xorSplit, xorJoin,
pointLocation2);

		Node oldTarget = edge.getTarget();
		int k = edge.getBendPointCount();
		for (int i = k - 1; i >= 0; i--) {
			edge.removeBendPoint(i);
		}
		add(new ReconnectEdgeCommand(edge, edge.getSource(), xorSplit));

		addAddEdgeCommand(graph, xorJoin, oldTarget);
```

```
                Point delta = new Point(120, 0);
                int size = oldTarget.getBounds().getLeft().x -
source.getBounds().getRight().x;
                int cutOff = source.getBounds().getRight().x;
                if (size < delta.x + 36) {
                        moveHorizontally(cutOff, delta, new ArrayList<Edge>());
                }
                if (connections.size() == 2) {
                        cutOff = xorSplitLocation.getCopy().translate(0, 24).y;
                        moveVertically(cutOff, new Point(0, 40));
                } else {
                        cutOff = xorSplitLocation.getCopy().translate(0, -24).y;
                        moveVerticallyUp(cutOff, new Point(0, -40));

                }
        }

        @Override
        public String getName() {
                return CONFIGURABLE_REGION_INSERT;
        }
}
```

```
public class ConfigurableRegionAction extends AbstractChangePatternAction {

	/**
	 * @param configurableRegionButton
	 * @param viewer
	 */

	public ConfigurableRegionAction(PlainMultiLineButton
configurableRegionButton, GraphicalGraphViewerWithFlyoutPalette viewer) {
		super(configurableRegionButton, viewer);
	}

	@Override
	protected AbstractChangePattern
createChangePattern(IStructuredSelection selection) {
		ConfigurableRegion changePattern = new
ConfigurableRegion(viewer.getGraph(), selection);
		return changePattern;
	}

	@Override
	protected boolean isChangePatternExecutable(IStructuredSelection
selection) {
		return extractConnection(selection) != null;
	}
}
```

```
public class Configuration extends AbstractChangePattern {
	public static final String CONFIGURATION_CHANGE_PATTERN =
"CONFIGURATION";
	private List<Node> nodesRemove = new ArrayList<Node>();

	/**
	 * @param graph
	 * @param condicion
	 */

	public Configuration(Graph graph, Node condicion) {
		super(graph);
		if (ok) { // Aqui se borra el modelo de una variante cuando se
preciona el boton "Configuracion" por segunda vez
			List<Node> nodes = graph.getNodes();
			for (Node node : nodes) {
				if (node.getProperty("copy") != null) {
					DeleteNodeCommand command = new
DeleteNodeCommand(node);
					add(command);
					for (Edge edgeC :
node.getAllConnectedEdges()) {
						add(new DeleteEdgeCommand(edgeC));
					}
				}
			}
		} else {// Aqui se construye una variante del modelo desde el
modelo copia. Todos acciones se hacen sobre la copia del modelo
			List<Edge> edges = null;
			Node XORif = null;
			Node XORthen = null;
			Edge edgeIf = null;
			Edge edgeThen = null;
			List<Edge> connections = condicion.getTargetConnections();
			// Se buscan las condiciones
			String[] conditionIf =
condicion.getName().substring(condicion.getName().indexOf("\n") + 1).split("
-> ");
			for (Edge edge : connections) { // Aqui se buscan los
Edges que tienen nombre que coincide con partes de condicion
				Node node = edge.getSource();
				edges = node.getSourceConnections();
				for (Edge edgeC : edges) {
					if (edgeC.getName() != null &&
edgeC.getName().equals(conditionIf[0])) {
						XORif = node; // Puerta logica XOR
"split" de primera Region Configurable
						edgeIf = edgeC; // Edge que tiene
nombre igual a la parte izquierda de Condicion
					}
					if (edgeC.getName() != null &&
edgeC.getName().equals(conditionIf[1])) {
						XORthen = node;// Puerta logica XOR
"split" de segunda Region Configurable
						edgeThen = edgeC;// Edge que tiene
nombre igual a la parte derecha de Condicion
					}
				}
			}
```

```
                    // Ahora tenemos las puertas que participan en
Configuration Constraint, Y tenemos las ramas que cumplen a la condicion
                    edges = XORif.getSourceConnections();
                    List<Edge> removedEdges = new ArrayList<Edge>();
                    for (Edge edge : edges) { // se borran todas las Condition
Constraint conectadas a la puerta XOR split de la primera Region
                                                                        //
Configurable
                          if
(edge.getTarget().getDescriptor().getName().equals("Condition")) {
                                DeleteNodeCommand command = new
DeleteNodeCommand(edge.getTarget());
                                add(command);
                                connections =
edge.getTarget().getAllConnectedEdges();
                                for (Edge edgeC : connections) {
                                      add(new DeleteEdgeCommand(edgeC));
                                      removedEdges.add(edge);
                                }
                          }
                    }
                    for (Edge edge : edges) { // recorre todas las ramas de
primer Region Configurable y se añaden los nodos que no cumplen
                                                                        //
condicion. Tambien se borran estas ramas
                          if (edge != edgeIf &&
!edge.getTarget().getDescriptor().getName().equals("Condition")) {
                                rellenar(edge);
                                add(new DeleteEdgeCommand(edge));
                                removedEdges.add(edge);
                          }
                    }

                    edges = XORthen.getSourceConnections();
                    for (Edge edge : edges) {// recorre todas las ramas de
segundo Region Configurable y se añaden los nodos que no cumplen
                                                                        //
condicion. Tambien se borran estas ramas
                          if (edge != edgeThen &&
!edge.getTarget().getDescriptor().getName().equals("Condition")) {
                                rellenar(edge);
                                add(new DeleteEdgeCommand(edge));
                          }
                    }

                    for (Node node : nodesRemove) {// se borran los nodes de
las ramas que no cumplen la condicion
                          connections = node.getSourceConnections();
                          for (Edge edge : connections) {
                                add(new DeleteEdgeCommand(edge));
                          }
                          add(new DeleteNodeCommand(node));
                    }
              }
              ok = !ok;
        }

        /**
         *
```

```
	 * @param node
	 * @return devuelve un nodo "Join" del nodo "Split" que pasa como
parametro.
	 */
	private Node gateEnd(Node node) {
		Node node0 = node.getSourceConnections().get(0).getTarget();
		Node node1 = node.getSourceConnections().get(1).getTarget();
		while (node0 != null && node0 != node1) {
			if (node0.getSourceConnections().size() <
node0.getTargetConnections().size()) {
				node1 =
node.getSourceConnections().get(1).getTarget();
				while (node1 != null && node0 != node1) {
					node1 = node1.getSourceConnections().size() >
0 ? node1.getSourceConnections().get(0).getTarget() : null;
				}
				if (node1 == null) {
					node0 =
node0.getSourceConnections().get(0).getTarget();
				}
			} else {
				node0 = node0.getSourceConnections().size() > 0 ?
node0.getSourceConnections().get(0).getTarget() : null;
			}
		}
		return node0;
	}

	@Override
	protected List<IGraphOptimizer> getGraphOptimizers() {
		List<IGraphOptimizer> optimizers = new
ArrayList<IGraphOptimizer>();
		optimizers.add(new XorGraphOptimizer(graph));
		// optimizers.add(new LoopGraphOptimizer(graph));
		// optimizers.add(new AndGraphOptimizer(graph));
		return optimizers;
	}

	@Override
	public String getName() {
		return CONFIGURATION_CHANGE_PATTERN;
	}

	/**
	 *
	 * @param edge
	 *            Este metodo recoge todos los Nodes que van en la rama
que pasa como parametro. La idea era recoger todos nodos que no
	 *            estan en la rama que cubre la condicion y añadirlos en
la lista para luego borrarlos.
	 */
	private void rellenar(Edge edge) {
		boolean ok = true;
		Node node = edge.getSource();
		while (ok) {
			if ((node.getSourceConnections().size() > 1 &&
edge.getTarget() == gateEnd(node))) {
				ok = false;
			} else {
```

```
                        nodesRemove.add(edge.getTarget());
                        List<Edge> edges =
edge.getTarget().getSourceConnections();
                        if (edges.size() > 1) {
                              for (Edge edge_aux : edges) {
                                    rellenar(edge_aux);
                              }
                              edge =
edge.getTarget().getSourceConnections().get(0);
                        } else {
                              edge =
edge.getTarget().getSourceConnections().get(0);
                        }
                  }
            }
      }
}
```

```
public class ConfigurationAction extends AbstractChangePatternAction {

	/**
	 * @param configurationButton
	 * @param viewer
	 */

	public ConfigurationAction(PlainMultiLineButton configurationButton,
GraphicalGraphViewerWithFlyoutPalette viewer) {
		super(configurationButton, viewer);
	}

	// este metodo crea un Edge nuevo
	protected Edge addEdge(Graph graph, Node source, Node target, String
name, String editorRegistry) {
		IGraphElementDescriptor edgeDescriptor =
EditorRegistry.getDescriptor(editorRegistry);
		Edge edge = (Edge) edgeDescriptor.createModel(graph);
		edge.setSource(source);
		edge.setTarget(target);
		if (name != null) {
			edge.setName(name);
		}
		return edge;
	}

	@Override
	protected AbstractChangePattern
createChangePattern(IStructuredSelection selection) {
		Graph graph = viewer.getGraph();
		Node node = null;
		if (!ok) { // en este bloque se añade a la variable Graph una
copia del modelo existente.
				// A todos los elementos del modelo-copia se
añade la propiedad "copy". En "copy" se pasa el id del elemento desde que
				// se ha copiado
				// Asi todos nodos copiados tendrán el id del
Nodo desde cual se han copido
				// Esto se hase para distingir la copia del
modelo original
			Object object = selection.toList().get(0);
			node = ((NodeEditPart) object).getModel();
			List<Node> nodes = graph.getNodes();
			List<Edge> edges = graph.getEdges();
			List<Node> nodesConfiguration = new ArrayList<Node>(); //
Aqui se añaden los Nodos copiados
			Node nodeSource = null, nodeTarget = null;
			int sizeEdges = edges.size();
			for (int i = 0; i < sizeEdges; i++) {// El Bucle para
recorrer todos Edges
				nodeSource = edges.get(i).getSource();
				nodeTarget = edges.get(i).getTarget();
				if (nodeSource != null &&
!nodesConfiguration.contains(nodeSource)) { // si no se ha copiado el Nodo.
Lo copiamos
					Node nod = nodeSource;
					// se crea Nodo nuevo
					nodeSource = newNode(graph, nod.getName(),
nod.getLocation().getCopy().translate(0, 400), nod.getDescriptor().getId());
```

```
                                // se añade la propiedad "copy". En la
propiedad "copy" pasa el id del nodo original
                                nodeSource.setProperty("copy", nod.getId());
                                // se añade el nodo al Graph
                                graph.addNode(nodeSource);
                                // se añade el Nodo a la lista de nodos. Esto
se ha hecho para que el mismo Nodo se puede añadir solo una vez.

      nodesConfiguration.add(edges.get(i).getSource());
                        } else {// Si este nodo ya esta copiado. Tenemos
que encontrarlo para poder cread la conexion con otro nodo
                                for (Node nod_x : nodes) { // Recorremos
todos Nodos y lo buscamos
                                        if (nod_x.getProperty("copy") != null
&& nodeSource.getId() == (Long) nod_x.getProperty("copy")) {
                                                nodeSource = nod_x;
                                        }
                                }
                        }
                        if (nodeTarget != null &&
!nodesConfiguration.contains(nodeTarget)) {// si no se ha copiado el Nodo. Lo
copiamos
                                Node nod = nodeTarget;
                                // se crea Nodo nuevo
                                nodeTarget = newNode(graph, nod.getName(),
nod.getLocation().getCopy().translate(0, 400), nod.getDescriptor().getId());
                                // se añade la propiedad "copy". En la
propiedad "copy" pasa el id del nodo original
                                nodeTarget.setProperty("copy", nod.getId());
                                // se añade el nodo al Graph
                                graph.addNode(nodeTarget);
                                // se añade el Nodo a la lista de nodos. Esto
se ha hecho para que el mismo Nodo se puede añadir solo una vez.

      nodesConfiguration.add(edges.get(i).getTarget());
                        } else {// Si este nodo ya esta copiado. Tenemos
que encontrarlo para poder cread la conexion con otro nodo
                                for (Node nod_x : nodes) {// Recorremos todos
Nodos y lo buscamos
                                        if (nod_x.getProperty("copy") != null
&& nodeTarget.getId() == (Long) nod_x.getProperty("copy")) {
                                                nodeTarget = nod_x;
                                        }
                                }
                        }
                        // Se crea la conexxion entre los nodos copiados
                        Edge edge = addEdge(graph, nodeSource, nodeTarget,
edges.get(i).getName(), edges.get(i).getDescriptor().getId());
                        // Se añade Edge a Graph
                        graph.addEdge(edge);
                }

                for (Node nod_x : nodes) {// Aqui se encuentra
Configuracion Constraint copiada al que se aplica el boton "Configuracion"
                        if (nod_x.getProperty("copy") != null &&
node.getId() == (Long) nod_x.getProperty("copy")) {
                                node = nod_x;
                        }
                }
```

```
			}
			ok = !ok;
			// LLama la clase Configuracion y se pasa dos modelos iguales en
variable Graph, y Configuracion Constraint
			Configuration configuration = new Configuration(graph, node);
			return configuration;
		}

		@Override
		protected boolean isChangePatternExecutable(IStructuredSelection
selection) {
			if (selection.size() > 1) {
				return false || ok;
			}
			Object object = selection.toList().get(0);
			if (!(object instanceof NodeEditPart)) {
				return false || ok;
			}
			Node node = ((NodeEditPart) object).getModel();
			if (!node.getDescriptor().getName().equals("Condition")) {
				return false || ok;
			}
			return true || ok;
		}

		// Este metodo crea un Nodo nuevo
		protected Node newNode(Graph graph, String Name, Point location,
String editorRegistry) {
			IGraphElementDescriptor descriptor =
EditorRegistry.getDescriptor(editorRegistry);
			Node node = (Node) descriptor.createModel(graph);
			if (Name != null) {
				node.setName(Name);
			}
			node.setLocation(location);
			return node;
		}

}
```

```
public class ConfigurationConstraint extends AbstractChangePattern {
	public static final String CONFIGURATION_CONSTRAINT_CHANGE_PATTERN =
"CONFIGURATION_CONSTRAINT";

	/**
	 * @param graph
	 * @param sourseNode
	 * @param targetNode
	 * @param condition
	 */

	public ConfigurationConstraint(Graph graph, Node sourseNode, Node
targetNode, String condition) {
		super(graph);
		Node sourseNodeStart = getXORStart(sourseNode);
		Node sourseNodeEnd = getXOREnd(targetNode);
		int a = targetNode.getBounds().getTopLeft().x;
		int b = sourseNode.getBounds().getTopRight().x;
		int c = getConditonSize().x;
		int x = (a - b) / 2 + b - c / 2;
		Point location = new Point(x,
sourseNode.getBounds().getBottomRight().y + 90);
		Node constraint = addAddConditionCommand(graph, condition,
location);
		addAddEdgeDashCommand(graph, sourseNodeStart, constraint);
		addAddEdgeDashCommand(graph, constraint, sourseNode);
		addAddEdgeDashCommand(graph, targetNode, constraint);
		addAddEdgeDashCommand(graph, constraint, sourseNodeEnd);
	}

	@Override
	public String getName() {
		return CONFIGURATION_CONSTRAINT_CHANGE_PATTERN;
	}

	private Node getXOREnd(Node sourseNode) {
		boolean ok = false;
		Node node1 = sourseNode;
		while (!ok && node1 != null) {
			Node node = node1.getSourceConnections().size() > 0 ?
node1.getSourceConnections().get(0).getTarget() : null;
			if (node.getDescriptor().getName().equals("XOR")
					&& (node.getProperty("Split") != null &&
(Long) node.getProperty("Split") == sourseNode.getId())) {
				ok = true;
			}
			node1 = node;
		}
		return node1;
	}

	private Node getXORStart(Node sourseNode) {
		boolean ok = false;
		Node node1 = sourseNode;
		while (!ok && node1 != null) {
			Node node = node1.getTargetConnections().size() > 0 ?
node1.getTargetConnections().get(0).getSource() : null;
			if (node.getDescriptor().getName().equals("XOR") &&
node.getProperty("Join") != null
```

```
                                    && (Long) node.getProperty("Join") ==
sourseNode.getId()) {
                                ok = true;
                        }
                        node1 = node;
                }
                return node1;
        }
}
```

```
public class ConfigurationConstraintAction extends
AbstractChangePatternAction {

	protected boolean ok = false;

	/**
	 * @param configuracionConstraintButton
	 * @param viewer
	 */
	public ConfigurationConstraintAction(PlainMultiLineButton
configuracionConstraintButton, GraphicalGraphViewerWithFlyoutPalette viewer)
{
		super(configuracionConstraintButton, viewer);
	}

	@Override
	protected AbstractChangePattern
createChangePattern(IStructuredSelection selection) {
		Object object = selection.toList().get(0);
		Node node1 = ((NodeEditPart) object).getModel();
		Object object2 = selection.toList().get(1);
		Node node2 = ((NodeEditPart) object2).getModel();

		Shell shell =
PlatformUI.getWorkbench().getActiveWorkbenchWindow().getShell();
		InputDialog dialog = new InputDialog(shell, "Condition", "Please
enter a valid condition.", null, null);
		if (!(dialog.open() == Window.OK)) {
			return null;
		}
		count = count + 1;
		String condition = "Configuration Constraint " + count + "\n" +
dialog.getValue();

		ConfigurationConstraint configurationConstraint = new
ConfigurationConstraint(viewer.getGraph(), node1.getSourceConnections()
				.size() > node2.getSourceConnections().size() ?
node2 : node1, node1.getSourceConnections().size() > node2
				.getSourceConnections().size() ? node1 : node2,
condition);
		return configurationConstraint;
	}

	@Override
	protected boolean isChangePatternExecutable(IStructuredSelection
selection) {
		ok = false;
		if (selection.size() != 2) {
			return false;
		}
		Object object = selection.toList().get(0);
		if (!(object instanceof NodeEditPart)) {
			return false;
		}
		Node node1 = ((NodeEditPart) object).getModel();
		if (!node1.getDescriptor().getName().equals("XOR")) {
			return false;
		}
		Object object2 = selection.toList().get(1);
```

```
            if (!(object2 instanceof NodeEditPart)) {
                  return false;
            }
            Node node2 = ((NodeEditPart) object2).getModel();
            if (!node2.getDescriptor().getName().equals("XOR")) {
                  return false;
            }
            return (((node1.getTargetConnections().size() > 1 &&
node2.getSourceConnections().size() > 1) && isSequence(node1, node2)) ||
((node2
                        .getTargetConnections().size() > 1 &&
node1.getSourceConnections().size() > 1) && isSequence(node2, node1)));
      }

      /**
       *
       * @param node1
       * @param node2
       * @return
       *
       *         este metodo determina si el node 2 sigue despues de nodo 1
       */
      protected boolean isSequence(Node node1, Node node2) {
            List<Edge> sourceConnections = node1.getSourceConnections();
            List<Edge> targetConnections = node2.getTargetConnections();
            for (Edge sourceEdge : sourceConnections) {
                  if
(sourceEdge.getDescriptor().getId().equals("BPMN.DASH_SEQUENCE_FLOW")) {
                        continue;
                  }
                  for (Edge targetEdge : targetConnections) {
                        if
(targetEdge.getDescriptor().getId().equals("BPMN.DASH_SEQUENCE_FLOW")) {
                              continue;
                        }
                        if (sourceEdge.equals(targetEdge)) {
                              ok = true;
                        } else if (!isSequence(sourceEdge.getTarget(),
node2)) {
                              continue;
                        }
                  }
            }
            return ok;
      }
}
```

```
public class DeleteConfigurableRegion extends AbstractChangePattern {
	public static final String DELETE_CONFIGURABLE_REGION =
"DELETE_CONFIGURABLE_REGION";

	/**
	 * @param graph
	 * @param incomingEdge
	 * @param outgoingEdge
	 * @param nodes
	 */

	public DeleteConfigurableRegion(Graph graph, Edge incomingEdge, Edge
outgoingEdge, List<Node> nodes, Node activity) {
		super(graph);
		if (activity != null) {
			add(new ReconnectEdgeCommand(incomingEdge,
incomingEdge.getSource(), activity));
			add(new
ReconnectEdgeCommand(activity.getSourceConnections().get(0), activity,
outgoingEdge.getTarget()));
		} else {
			add(new ReconnectEdgeCommand(incomingEdge,
incomingEdge.getSource(), outgoingEdge.getTarget()));
		}

		List<Edge> removedEdges = new ArrayList<Edge>();

		for (Node node : nodes) {
			List<Edge> sourceConnections =
node.getSourceConnections();
			for (Edge edge : sourceConnections) {
				add(new DeleteEdgeCommand(edge));
				removedEdges.add(edge);
			}

			add(new DeleteNodeCommand(node));
		}

		Point incoming =
incomingEdge.getSource().getBounds().getTopRight().getCopy();
		incoming.y = 0;
		Point outgoing =
outgoingEdge.getTarget().getBounds().getTopLeft().getCopy();
		outgoing.y = 0;

		if (otherNodesFound(incoming, outgoing, nodes)) {
			return;
		}

		// Point delta =
incoming.translate(outgoing.getNegated()).translate(20, 0);
	}

	@Override
	protected List<IGraphOptimizer> getGraphOptimizers() {
		List<IGraphOptimizer> optimizers = new
ArrayList<IGraphOptimizer>();
		// optimizers.add(new XorGraphOptimizer(graph));
		optimizers.add(new LoopGraphOptimizer(graph));
```

```
			optimizers.add(new AndGraphOptimizer(graph));
			return optimizers;
		}

		@Override
		public String getName() {
			return DELETE_CONFIGURABLE_REGION;
		}

		/**
		 * @param incoming
		 * @param outgoing
		 * @param nodes
		 * @return
		 */
		private boolean otherNodesFound(Point incoming, Point outgoing,
List<Node> removedNodes) {
			List<Node> nodes = graph.getNodes();
			for (Node node : nodes) {
				if (removedNodes.contains(node)) {
					continue;
				}

				Rectangle bounds = node.getBounds();
				if (bounds.x > incoming.x && bounds.x < outgoing.x) {
					return true;
				}
			}
			return false;
		}
}
```

```
public class DeleteConfigurableRegionAction extends
AbstractChangePatternAction {

	/**
	 * @param deleteButton
	 * @param viewer
	 */

	public DeleteConfigurableRegionAction(PlainMultiLineButton
deleteButton, GraphicalGraphViewerWithFlyoutPalette viewer) {
		super(deleteButton, viewer);
	}

	@Override
	protected AbstractChangePattern
createChangePattern(IStructuredSelection selection) {
		Shell shell =
PlatformUI.getWorkbench().getActiveWorkbenchWindow().getShell();
		// if (!MessageDialog.openQuestion(shell, "Delete Process
Fragment", "Do you really want to delete the selected process fragment?"))
		// {
		// return null;
		// }
		AbstractChangePattern changePattern = null;
		SESEChecker seseChecker = getSESEChecker(selection);
		Node node1 = seseChecker.getFirstNode();
		Node node2 = seseChecker.getLastNode();
		List<Node> nodes = new ArrayList<Node>(seseChecker.getNodes());
		Object[] possibilities = new Object[nodes.size() - 1];
		possibilities[0] = "Borrar Configurable Region";
		int num = 1;
		for (int i = 0; i < nodes.size(); i++) {
			if (nodes.get(i) != node1 && nodes.get(i) != node2) {
				possibilities[num] = "Dejar solo Actividad " +
nodes.get(i).getName();
				num++;
			}
		}
		Edge incomingEdge = seseChecker.getIncomingEdge();
		Edge outgoingEdge = seseChecker.getOutgoingEdge();
		String s = (String) JOptionPane.showInputDialog(null, "Elige
opcion", "Customized Dialog", JOptionPane.PLAIN_MESSAGE, null,
				possibilities, "Borrar Configurable Region");
		if (s.equals("Configurable Region")) {
			changePattern = new
DeleteConfigurableRegion(viewer.getGraph(), incomingEdge, outgoingEdge,
nodes, null);
		} else {
			for (int i = 0; i < nodes.size(); i++) {
				String name = nodes.get(i).getName();
				if (name != null && name.equals(s.substring(21))) {
					node1 = nodes.get(i);
					nodes.remove(i);
				}
			}
			changePattern = new
DeleteConfigurableRegion(viewer.getGraph(), incomingEdge, outgoingEdge,
nodes, node1);
		}
```

```
			return changePattern;
		}

		@Override
		protected boolean isChangePatternExecutable(IStructuredSelection
selection) {
			if (selection.size() < 2) {
				return false;
			}
			for (int i = selection.toList().size() - 1; i >= 0; i--) {
				Object object = selection.toList().get(i);
				if (!(object instanceof NodeEditPart)) {
					return false;
				}
			}
			SESEChecker seseChecker = getSESEChecker(selection);
			if (!seseChecker.isSESEFragment()) {
				return false;
			}
			Node node1 = seseChecker.getFirstNode();
			Node node2 = seseChecker.getLastNode();
			return (node1.getDescriptor().getName().equals("XOR")) &&
(node2.getDescriptor().getName().equals("XOR"));
		}

}
```

```
public class DeleteConfigurationAlternative extends AbstractChangePattern {
	public static final String DELETE_CONFIGURATION_ALTERNATIVE =
"DELETE_CONFIGURATION_ALTERNATIVE";

	/**
	 * @param graph
	 * @param incomingEdge
	 * @param outgoingEdge
	 * @param node
	 */

	public DeleteConfigurationAlternative(Graph graph, Edge incomingEdge,
Edge outgoingEdge, Node node) {
		super(graph);
		List<Edge> removedEdges = new ArrayList<Edge>();
		if (incomingEdge.getSource().getSourceConnections().size() < 3)
{
			add(new ReconnectEdgeCommand(incomingEdge,
incomingEdge.getSource(), outgoingEdge.getTarget()));
		}
		List<Edge> sourceConnections =
incomingEdge.getSource().getSourceConnections().size() > 2 ?
node.getAllConnectedEdges() : node
				.getSourceConnections();
		for (Edge edge : sourceConnections) {
			add(new DeleteEdgeCommand(edge));
			removedEdges.add(edge);
		}
		add(new DeleteNodeCommand(node));

		Point incoming =
incomingEdge.getSource().getBounds().getTopRight().getCopy();
		incoming.y = 0;
		Point outgoing =
outgoingEdge.getTarget().getBounds().getTopLeft().getCopy();
		outgoing.y = 0;

		// Point delta =
incoming.translate(outgoing.getNegated()).translate(20, 0);
	}

	@Override
	protected List<IGraphOptimizer> getGraphOptimizers() {
		List<IGraphOptimizer> optimizers = new
ArrayList<IGraphOptimizer>();
		return optimizers;
	}

	@Override
	public String getName() {
		return DELETE_CONFIGURATION_ALTERNATIVE;
	}

}
```

```
public class DeleteConfigurationAlternativeAction extends
AbstractChangePatternAction {

	/**
	 * @param deleteButton
	 * @param viewer
	 */

	public DeleteConfigurationAlternativeAction(PlainMultiLineButton
deleteButton, GraphicalGraphViewerWithFlyoutPalette viewer) {
		super(deleteButton, viewer);
	}

	@Override
	protected AbstractChangePattern
createChangePattern(IStructuredSelection selection) {
		Shell shell =
PlatformUI.getWorkbench().getActiveWorkbenchWindow().getShell();
		if (!MessageDialog.openQuestion(shell, "Delete Process
Fragment", "Do you really want to delete the selected process fragment?")) {
			return null;
		}

		SESEChecker seseChecker = getSESEChecker(selection);
		Edge incomingEdge = seseChecker.getIncomingEdge();
		Edge outgoingEdge = seseChecker.getOutgoingEdge();
		Node node = null;
		Object element = selection.getFirstElement();
		if ((element instanceof NodeEditPart)) {
			NodeEditPart nodeEditPart = (NodeEditPart) element;
			node = nodeEditPart.getModel();
		}

		DeleteConfigurationAlternative changePattern = new
DeleteConfigurationAlternative(viewer.getGraph(), incomingEdge, outgoingEdge,
				node);
		return changePattern;
	}

	@Override
	protected boolean isChangePatternExecutable(IStructuredSelection
selection) {
		if (selection.size() != 1) {
			return false;
		}

		Object element = selection.getFirstElement();
		if (!(element instanceof NodeEditPart)) {
			return false;
		}

		NodeEditPart nodeEditPart = (NodeEditPart) element;
		Node node = nodeEditPart.getModel();
		String type = node.getDescriptor().getId();
		return EditorRegistry.BPMN_ACTIVITY.equals(type);
	}

}
```

```
public class DeleteConfigurationConstraint extends AbstractChangePattern {
	public static final String DELETE_CONFIGURATION_CONSTRAINT =
"DELETE_CONFIGURATION_CONSTRAINT";

	/**
	 * @param graph
	 * @param node
	 */
	public DeleteConfigurationConstraint(Graph graph, Node node) {
		super(graph);
		List<Edge> removedEdges = new ArrayList<Edge>();
		DeleteNodeCommand command = new DeleteNodeCommand(node);
		add(command);
		List<Edge> sourceConnections = node.getAllConnectedEdges();
		for (Edge edge : sourceConnections) {
			add(new DeleteEdgeCommand(edge));
			removedEdges.add(edge);
		}
	}

	@Override
	public String getName() {
		return DELETE_CONFIGURATION_CONSTRAINT;
	}

}
```

```
public class DeleteConfigurationConstraintAction extends
AbstractChangePatternAction {

	/**
	 * @param renameActivityButton
	 * @param viewer
	 */
	public DeleteConfigurationConstraintAction(PlainMultiLineButton
renameActivityButton, GraphicalGraphViewerWithFlyoutPalette viewer) {
		super(renameActivityButton, viewer);
	}

	@Override
	protected AbstractChangePattern
createChangePattern(IStructuredSelection selection) {

		NodeEditPart element = (NodeEditPart)
selection.getFirstElement();
		DeleteConfigurationConstraint changePattern = new
DeleteConfigurationConstraint(viewer.getGraph(), element.getModel());
		return changePattern;
	}

	@Override
	protected boolean isChangePatternExecutable(IStructuredSelection
selection) {
		if (selection.size() != 1) {
			return false;
		}

		Object element = selection.getFirstElement();
		if (!(element instanceof NodeEditPart)) {
			return false;
		}

		NodeEditPart nodeEditPart = (NodeEditPart) element;
		Node node = nodeEditPart.getModel();
		String type = node.getDescriptor().getId();
		return EditorRegistry.CHANGE_PATTERN_CONDITION.equals(type);
	}
}
```

```
public class SESEChecker {
	private final Set<Node> nodes;
	private List<Edge> incomingEdges;
	private List<Edge> outgoingEdges;

	public SESEChecker(Node firstNode, Node lastNode) {
		nodes = new HashSet<Node>();
		addNodes(firstNode, lastNode);
		calculate();
	}

	public SESEChecker(Set<Node> nodes) {
		Assert.isNotNull(nodes);
		this.nodes = nodes;
		calculate();
	}

	private void addNodes(Node firstNode, Node lastNode) {
		if (nodes.contains(firstNode)) {
			return;
		}

		nodes.add(firstNode);
		if (firstNode.equals(lastNode)) {
			return;
		}
		List<Edge> sourceConnections = firstNode.getSourceConnections();
		for (Edge edge : sourceConnections) {
			addNodes(edge.getTarget(), lastNode);
		}
	}

	private void calculate() {
		incomingEdges = new ArrayList<Edge>();
		outgoingEdges = new ArrayList<Edge>();
		for (Node node : nodes) {
			for (Edge edge : node.getSourceConnections()) {
				if (!nodes.contains(edge.getTarget())) {
					outgoingEdges.add(edge);
				}
			}

			List<Edge> targetConnections =
node.getTargetConnections();
			for (Edge edge : targetConnections) {
				if (!nodes.contains(edge.getSource())) {
					incomingEdges.add(edge);
				}
			}
		}
	}

	public Node getFirstNode() {
		return getIncomingEdge().getTarget();
	}

	public Edge getIncomingEdge() {
		if (getIncomingEdges().size() != 1) {
```

```
                  throw new IllegalStateException("Not a valid SESE
fragment. See isSESEFragment()");
            }

            return getIncomingEdges().get(0);
      }

      /**
       * Returns the incomingEdges.
       *
       * @return the incomingEdges
       */
      public List<Edge> getIncomingEdges() {
            return Collections.unmodifiableList(incomingEdges);
      }

      public Node getLastNode() {
            return getOutgoingEdge().getSource();
      }

      public Set<Node> getNodes() {
            return Collections.unmodifiableSet(nodes);
      }

      public Edge getOutgoingEdge() {
            if (getOutgoingEdges().size() != 1) {
                  throw new IllegalStateException("Not a valid SESE
fragment. See isSESEFragment()");
            }
            return getOutgoingEdges().get(0);
      }

      /**
       * Returns the outgoingEdges.
       *
       * @return the outgoingEdges
       */
      public List<Edge> getOutgoingEdges() {
            return Collections.unmodifiableList(outgoingEdges);
      }

      public boolean isSESEFragment() {
            return incomingEdges.size() == 1 && outgoingEdges.size() == 1;
      }
}
```

```
public class UndoChangePatternAction extends AbstractChangePatternAction {

	public UndoChangePatternAction(PlainMultiLineButton button,
GraphicalGraphViewerWithFlyoutPalette viewer) {
		super(button, viewer);
		setText("Undo");

	setImageDescriptor(ResourceManager.getPluginImageDescriptor(Activator.
getDefault(), "img/undo.png"));
	}

	@Override
	protected AbstractChangePattern
createChangePattern(IStructuredSelection selection) {
		throw new UnsupportedOperationException("Should not be called");
	}

	@Override
	protected boolean isChangePatternExecutable(IStructuredSelection
selection) {
		CommandStack stack =
viewer.getViewer().getEditDomain().getCommandStack();
		return stack.canUndo();
	}

	@Override
	public void run() {
		CommandStack stack =
viewer.getViewer().getEditDomain().getCommandStack();
		if (!stack.canUndo()) {
			return;
		}

		stack.undo();
		setEnabled(stack.canUndo());
	}
}
```

```
public class XorGraphOptimizer extends AbstractGraphOptimizer {

	public XorGraphOptimizer(Graph graph) {
		super(graph);
	}

	@Override
	protected boolean doOptimize() {
		List<Node> nodes = graph.getNodes();
		for (Node node : nodes) {
			if (!isXorGateway(node) ||
node.getSourceConnections().size() != 1 || node.getTargetConnections().size()
!= 1) {
				// continue;
			} else {
				Edge incomingEdge =
node.getTargetConnections().get(0);
				Edge outgoingEdge =
node.getSourceConnections().get(0);
				add(new ReconnectEdgeCommand(incomingEdge,
incomingEdge.getSource(), outgoingEdge.getTarget()));
				add(new DeleteEdgeCommand(outgoingEdge));
				add(new DeleteNodeCommand(node));
				return true;
			}

		}
		return false;
	}
}
```