```
1: //.setBackground(new Color(169,211,247));
2:
3: package models;
4:
5: import java.awt.BorderLayout;
6: import java.awt.Color;
7: import java.awt.Component;
8: import java.awt.Dimension;
9: import java.awt.Font;
10: import java.awt.Paint;
11: import java.awt.Rectangle;
12: import java.awt.Shape;
13: import java.awt.event.ActionEvent;
14: import java.awt.event.ActionListener;
15: import java.awt.event.MouseEvent;
16: import java.awt.geom.Ellipse2D;
17: import java.awt.geom.Point2D;
18: import java.util.ArrayList;
19: import java.util.Iterator;
20: import java.util.Vector;
21:
22: import javax.swing.BorderFactory;
23: import javax.swing.Box;
24: import javax.swing.BoxLayout;
25: import javax.swing.Icon;
26: import javax.swing.JButton;
27: import javax.swing.JComponent;
28: import javax.swing.JLabel;
29: import javax.swing.JPanel;
30:
31: import main.JeuRelation;
32: import main.RelationComplete;
33:
34: import org.apache.commons.collections15.Predicate;
35: import org.apache.commons.collections15.Transformer;
36:
37: import translation.Messages;
38: import client.SchemaEditingMousePlugin;
39: import dataPack.Acteur;
40: import dataPack.Activite;
41: import dataPack.Content;
42: import dataPack.DataPack;
43: import dataPack.Moyen;
44: import dataPack.TreeListener;
45: import edu.uci.ics.jung.algorithms.layout.Layout;
46: import edu.uci.ics.jung.algorithms.layout.StaticLayout;
47: import edu.uci.ics.jung.graph.DirectedSparseGraph;
48: import edu.uci.ics.jung.graph.Graph;
49: import edu.uci.ics.jung.graph.util.Context;
50: import edu.uci.ics.jung.visualization.GraphZoomScrollPane;
51: import edu.uci.ics.jung.visualization.Layer;
52: import edu.uci.ics.jung.visualization.PluggableRenderContext;
53: import edu.uci.ics.jung.visualization.VisualizationViewer;
54: import edu.uci.ics.jung.visualization.annotations.Annotation;
55: import edu.uci.ics.jung.visualization.annotations.AnnotationManager;
56: import edu.uci.ics.jung.visualization.control.EditingModalGraphMouse;
57: import edu.uci.ics.jung.visualization.decorators.DirectionalEdgeArrowTransformer;
58: import edu.uci.ics.jung.visualization.decorators.EdgeShape.QuadCurve;
59: import edu.uci.ics.jung.visualization.decorators.EllipseVertexShapeTransformer;
60: import edu.uci.ics.jung.visualization.picking.ShapePickSupport;
61: import edu.uci.ics.jung.visualization.renderers.EdgeLabelRenderer;
62: import edu.uci.ics.jung.visualization.renderers.Renderer.VertexLabel.Position;
63: import edu.uci.ics.jung.visualization.renderers.VertexLabelRenderer;
64: import graphicalUserInterface.DialogHandlerFrame;
65: import graphicalUserInterface.IconDatabase;
66: import graphicalUserInterface.PopUpView;
67: import graphicalUserInterface.Program;
68:
69: public class BrickView<V extends BrickVertex, E extends BrickEdge> extends
70: JPanel  {
71:         /**
72:          *
73:          */
74:         private static final long serialVersionUID = -5832896318413480189L;
75:
76:
77:         // TODO positionnement automatique des labels :
78:         //      vv.getRenderer().getVertexLabelRenderer().setPosition(Renderer.Ver
texLabel.Position.AUTO);
79:
80:         protected Brick brick;
81:         protected DirectedSparseGraph<V, E> graph;
82:         protected VisualizationViewer<V, E> vv;
83:         protected Layout<V, E> vertexLocations;
84:         protected GraphZoomScrollPane panel;
85:         protected TriadeEditingMousePlugin<V, E> mousePlugin;
86:         protected EditingModalGraphMouse<V, E> graphMouse;
87:         protected PopUpView popUp;
88:         protected TreeListener treeListener;
89:         protected DataPack datapack;
90:         protected Annotation<String> title;
91:         protected int titleLength;
92:         protected AnnotationManager aM;
93:         protected int vertexShownLevel = 1;
94:         protected PluggableRenderContext<V, E> pr;
95:
96:
97:         boolean rotatedEdge = false;
98:
99:         public BrickView(PopUpView pUV, final DataPack datapack,
100:                         TreeListener treeListener) {
101:                 popUp = pUV;
102:                 this.treeListener = treeListener;
103:                 this.datapack = datapack;
104:                 graph = new DirectedSparseGraph<V, E>();
105:
106:         }
107:
108:         @SuppressWarnings("unchecked")
109:         public BrickView(Brick brick, PopUpView pUV, final DataPack datapack,
110:                         TreeListener treeListener) {
111:                 this(pUV, datapack, treeListener);
112:
113:
114:                 this.brick = brick;
115:
116:
117:
118:                 vertexLocations = new StaticLayout<V, E>(graph);
119:                 vertexLocations.setSize(new Dimension(600, 600));
120:
121:
122:
123:
124:
125:                 if (brick.isNavigationBrick()) {
126:                         mousePlugin = new NavigationMousePlugin<V, E>(treeListener
, brick, (Layout<BrickVertex, BrickEdge>) vertexLocations);
127:                 } else if (Program.isTriades()) {
128:                         mousePlugin = (TriadeEditingMousePlugin<V, E>) new SchemaE
ditingMousePlugin(
129:                                         treeListener, brick, (Layout<BrickVertex,
BrickEdge>) vertexLocations);
130:                 } else {
```

```
131:                        mousePlugin = (TriadeEditingMousePlugin<V, E>) new BrickEd
itingMousePlugin(
132:                                brick, treeListener, (Layout<BrickVertex,
BrickEdge>) vertexLocations);
133:                }
134:
135:                buildComponent();
136:
137:
138:                // vertexLocations.initialize();
139:
140:                if (brick.isNavigationBrick()) {
141:                        //                FRLayout<V,E> kkLayout = new FRLay
out<V, E>(graph);
142:                        //                kkLayout.setSize(new Dimension(600
, 600));
143:                        //                kkLayout.setInitializer(vertexLoca
tions);
144:                        //                kkLayout.setRepulsionMultiplier(1)
;
145:                        //                kkLayout.initialize();
146:                        //
147:                        //                vertexLocations = kkLayout;
148:
149:                        organizeLayout();
150:
151:
152:                }
153:
154:                buildGraph();
155:        }
156:
157:
158:
159:        protected void buildComponent()
160:        {
161:                final BrickView<V, E> access = this;
162:
163:                vv = new VisualizationViewer<V, E>(vertexLocations);
164:
165:                pr = (PluggableRenderContext<V, E>) vv.getRenderContext();
166:                pr.setEdgeArrowTransformer(new DirectionalEdgeArrowTransformer<V,
E>(
167:                                14, 20, 7));
168:
169:                pr.setVertexLabelRenderer(new VertexLabelRenderer() {
170:
171:                        @Override
172:                        public <T> Component getVertexLabelRendererComponent(JComp
onent vv,
173:                                        Object value, Font font, boolean isSelecte
d, T vertex) {
174:                                return access.getVertexLabelRendererComponent(vv,
value, font, isSelected, (V)vertex);
175:                        }
176:                });
177:
178:                pr.setVertexFontTransformer(new Transformer<V, Font>() {
179:                        @Override
180:                        public Font transform(V vertex) {
181:                                return access.getVertexFont(vertex);
182:                        }
183:                });
184:
185:                pr.setVertexIconTransformer(new Transformer<V, Icon>() {
186:                        @Override
187:                        public Icon transform(V vertex) {
188:                                return access.getVertexIcon(vertex);
189:                        }
190:
191:                });
192:
193:                pr.setVertexLabelTransformer(new Transformer<V, String>() {
194:
195:                        @Override
196:                        public String transform(V vertex) {
197:                                return access.getVertexLabel(vertex);
198:                        }
199:                });
200:
201:                vv.setVertexToolTipTransformer(new Transformer<V, String>() {
202:
203:                        @Override
204:                        public String transform(V vertex) {
205:                                return access.getVertexToolTipText(vertex);
206:                        }
207:                });
208:
209:                pr.setEdgeDrawPaintTransformer(new Transformer<E, Paint>() {
210:
211:                        @Override
212:                        public Paint transform(E edge) {
213:                                return access.getEdgeDrawPaint(edge);
214:                        }
215:                });
216:
217:                vv.setEdgeToolTipTransformer(new Transformer<E, String>() {
218:
219:                        @Override
220:                        public String transform(E e) {
221:                                return access.getEdgeToolTipLabel(e);
222:                        }
223:                });
224:
225:                pr.setEdgeIncludePredicate(new Predicate<Context<Graph<V,E>,E>>()
{
226:
227:                        @Override
228:                        public boolean evaluate(
229:                                        Context<Graph<V,E>,E> arg0) {
230:                                return access.getEdgeIncludePredicate(arg0);
231:
232:                        }
233:                });
234:
235:                pr.setVertexIncludePredicate(new Predicate<Context<Graph<V,E>,V>>(
) {
236:
237:                        @Override
238:                        public boolean evaluate(
239:                                        Context<Graph<V,E>,V> arg0) {
240:                                return access.getVertexIncludePredicate(arg0);
241:
242:                        }
243:                });
244:
245:                pr.setEdgeLabelTransformer(new Transformer<E, String>() {
246:                        BrickView<V, E> brickView = access;
247:
248:                        @Override
249:                        public String transform(E e) {
250:                                return brickView.getEdgeLabel(e);
251:                        }
252:                });
```

```
253:                pr.setEdgeFontTransformer(new Transformer<E, Font>() {
254:                        BrickView<V, E> brickView = access;
255:
256:                        @Override
257:                        public Font transform(E e) {
258:                                return brickView.getEdgeFont(e);
259:                        }
260:                });
261:
262:                pr.setEdgeLabelRenderer(new EdgeLabelRenderer() {
263:                        BrickView<V, E> brickView = access;
264:
265:                        @Override
266:                        public void setRotateEdgeLabels(boolean state) {
267:                        }
268:
269:                        @Override
270:                        public boolean isRotateEdgeLabels() {
271:                                return true;
272:                        }
273:
274:                        @Override
275:                        public <T> Component getEdgeLabelRendererComponent(JCompon
ent vv,
276:                                        Object value, Font font, boolean isSelecte
d, T edge) {
277:                                return brickView.getEdgeLabelRendererComponent(vv,
 value, font, isSelected, (E)edge);
278:                        }
279:                });
280:                if (!brick.isNavigationBrick())
281:                        pr.setEdgeArrowTransformer(new DirectionalEdgeArrowTransfo
rmer<V, E>(10, 16, 12));
282:                else
283:                        pr.setEdgeArrowTransformer(new DirectionalEdgeArrowTransfo
rmer<V, E>(0, 0, 0));
284:
285:
286:
287:                vv.setBounds(new Rectangle(-300, -300, 600, 600));
288:                vv.setBackground(Color.white);
289:                vv.setPickSupport(new ShapePickSupport<V, E>(vv));
290:
291:                QuadCurve<V, E> shape = new QuadCurve<V, E>();
292:                shape.setControlOffsetIncrement(45);
293:                pr.setEdgeShapeTransformer(shape);
294:
295:                vv.getRenderer().getVertexLabelRenderer().setPosition(Position.SE)
;
296:
297:                pr.setVertexShapeTransformer(new EllipseVertexShapeTransformer<V>(
new Transformer<V, Integer>() {
298:
299:                        @Override
300:                        public Integer transform(V arg0) {
301:                                return 40;
302:                        }
303:                },
304:                new Transformer<V, Float>() {
305:
306:                        final Float ratio = new Float(1);
307:                        @Override
308:                        public Float transform(V arg0) {
309:                                return ratio;
310:                        }
311:                }));
312:
313:                vv.addMouseListener(mousePlugin);
314:                vv.addMouseMotionListener(mousePlugin);
315:
316:                setLayout(new BorderLayout());
317:
318:                this.add(vv, BorderLayout.CENTER);
319:
320:                if (Program.isTriades() && !brick.isNavigationBrick() && this inst
anceof BrickView)
321:                {
322:                        vertexShownLevel = 1;
323:                        final JButton more = new JButton("Afficher les acteurs sec
ondaires");
324:                        final JButton less = new JButton("Masquer les sommets seco
ndaires");
325:                        less.setEnabled(false);
326:                        more.addActionListener(new ActionListener() {
327:
328:                                @Override
329:                                public void actionPerformed(ActionEvent e) {
330:                                        if (vertexShownLevel == 1)
331:                                        {
332:                                                more.setText("Afficher les acteurs
 restants");
333:                                                less.setText("Masquer les acteurs
secondaires");
334:                                                less.setEnabled(true);
335:                                                vertexShownLevel++;
336:                                                access.repaint();
337:                                        }
338:                                        else if (vertexShownLevel == 2)
339:                                        {
340:                                                more.setEnabled(false);
341:                                                less.setText("Masquer les acteurs
restants");
342:                                                vertexShownLevel++;
343:                                                access.repaint();
344:                                        }
345:                                }
346:                        });
347:                        less.addActionListener(new ActionListener() {
348:
349:                                @Override
350:                                public void actionPerformed(ActionEvent e) {
351:                                        if (vertexShownLevel == 2)
352:                                        {
353:                                                more.setText("Afficher les acteurs
 secondaires");
354:                                                less.setEnabled(false);
355:                                                vertexShownLevel--;
356:                                                access.repaint();
357:                                        }
358:                                        else if (vertexShownLevel == 3)
359:                                        {
360:                                                more.setText("Afficher les acteurs
 restants");
361:                                                more.setEnabled(true);
362:                                                less.setText("Masquer les acteurs
secondaires");
363:                                                vertexShownLevel--;
364:                                                access.repaint();
365:                                        }
366:                                }
367:                        });
368:                        JPanel shownLevelButtons = new JPanel();
369:                        shownLevelButtons.setLayout(new BoxLayout(shownLevelButton
s, BoxLayout.X_AXIS));
```

```
370:                        shownLevelButtons.add(less);
371:                        shownLevelButtons.add(Box.createGlue());
372:                        shownLevelButtons.add(more);
373:                        this.add(shownLevelButtons, BorderLayout.SOUTH);
374:
375:                }
376:                else
377:                {
378:                        vertexShownLevel = 4;
379:                }
380:
381:                mousePlugin.setModelView(this);
382:                graphMouse = new ModelsEditingModalGraphMouse<V, E>(mousePlugin, p
r);
383:
384:                vv.setGraphMouse(graphMouse);
385:                if (!brick.isNavigationBrick())
386:                        vv.getRenderContext().getMultiLayerTransformer().getTransf
ormer(
387:                                        Layer.VIEW).setTranslate(300, 300);
388:                else {
389:
390:                        vv.getRenderContext().getMultiLayerTransformer().getTransf
ormer(
391:                                        Layer.VIEW).setTranslate(150, 150);
392:                }
393:
394:                aM = new AnnotationManager(vv.getRenderContext());
395:                Point2D point = new Point2D.Double(30, 30);
396:                point = vv.getRenderContext().getMultiLayerTransformer()
397:                .inverseTransform(point);
398:                title = new Annotation<String>(
399:                                "<html><TABLE border=1><TR><TD><center><font size=
4> " //$NON-NLS-1$
400:                                + brick.getName() + " <br>(" + brick.getStep() //$
NON-NLS-1$
401:                                + ")</center></font></TD></TR></TABLE></html>", //
$NON-NLS-1$
402:                                Annotation.Layer.LOWER, Color.black, false, point)
;
403:                aM.add(Annotation.Layer.LOWER, title);
404:                titleLength = brick.getName().length() + brick.getStep().length()
+ 2;
405:                titleLength *= 6;
406:
407:                vv.addPreRenderPaintable(aM
408:                                .getAnnotationPaintable(Annotation.Layer.LOWER));
409:                vv.addPostRenderPaintable(aM
410:                                .getAnnotationPaintable(Annotation.Layer.UPPER));
411:
412:                vv.repaint();
413:
414:
415:        }
416:
417:        protected Font getEdgeFont(E e) {
418:                return new Font("Helvetica", Font.PLAIN, 10); //$NON-NLS-1$
419:        }
420:
421:        protected Component getEdgeLabelRendererComponent(JComponent vv,
422:                        Object value, Font font, boolean isSelected, E edge) {
423:
424:                JLabel result = new JLabel();
425:                result.setFont(font);
426:                result.setText(value.toString());
427:
428:                return result;
429:        }
430:
431:        protected String getEdgeLabel(E e) {
432:                String labelText = ""; //$NON-NLS-1$
433:
434:                if(brick.isNavigationBrick()) {
435:                        return ""; //$NON-NLS-1$
436:                } else {
437:                        for(JeuRelation relation : e.relations.getEnsembleRelation
()) {
438:                                labelText += relation.objectifStructurel.toString(
).substring(0, 4) + ". "; //$NON-NLS-1$
439:                        }
440:                }
441:
442:                return labelText;
443:        }
444:
445:        protected String getEdgeToolTipLabel(E e) {
446:                return  e.getStringDescription();
447:        }
448:
449:        protected void buildGraph()
450:        {
451:                for (Iterator<V> iterator = graph.getVertices().iterator(); iterat
or
452:                .hasNext();) {
453:                        vertexLocations.lock(iterator.next(), true);
454:
455:                }
456:
457:
458:
459:                brick.graph = (DirectedSparseGraph<BrickVertex, BrickEdge>) graph;
460:                for (BrickVertex brickVertex : brick.vertices) {
461:                        graph.addVertex((V) brickVertex);
462:                        brickVertex.setSelected(false);
463:                        // if (!brick.isNavigationBrick())
464:                        vertexLocations.setLocation((V) brickVertex, brickVertex
465:                                        .getLocation());
466:                }
467:
468:                Vector<BrickEdge> toRemove = new Vector<BrickEdge>();
469:                for (BrickEdge mEdge : brick.edges) {
470:                        BrickVertex src = mEdge.source;
471:                        BrickVertex dest = mEdge.destination;
472:                        mEdge.setSelected(false);
473:
474:                        try {
475:                                graph.addEdge((E) mEdge, (V) src, (V) dest);
476:
477:                        } catch (IllegalArgumentException e) {
478:                                DialogHandlerFrame
479:                                .showErrorDialog(Messages.getString("BrickView.7")
); //$NON-NLS-1$
480:
481:                                toRemove.add(mEdge);
482:                        }
483:                }
484:                for (BrickEdge edge : toRemove) {
485:                        brick.edges.remove(edge);
486:                }
487:
488:                vertexLocations.initialize();
489:                for (Iterator<V> iterator = graph.getVertices().iterator(); iterat
or
490:                .hasNext();) {
```

```
491:                            vertexLocations.lock(iterator.next(), false);
492:                    }
493:            }
494:
495:            @Override
496:            public void repaint() {
497:                    if (vv != null && title != null) {
498:                            Point2D point = new Point2D.Double(30, 30);
499:                            point = vv.getRenderContext().getMultiLayerTransformer()
500:                            .inverseTransform(point);
501:
502:                            title.setLocation(point);
503:                    }
504:                    validate();
505:                    super.repaint();
506:            }
507:
508:
509:
510:
511:
512:
513:            public TriadeEditingMousePlugin<V, E> getMousePlugin() {
514:                    return mousePlugin;
515:            }
516:
517:            public PopUpView getPopUp() {
518:                    return popUp;
519:            }
520:
521:            public DirectedSparseGraph<V, E> getGraph() {
522:                    return graph;
523:            }
524:
525:            public Component getVertexLabelRendererComponent(JComponent vv,
526:                            Object value, Font font, boolean isSelected, V vertex) {
527:                    JLabel result = new JLabel();
528:                    result.setFont(font);
529:                    result.setText(value.toString());
530:                    result.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
createLineBorder(Color.lightGray), BorderFactory.createEmptyBorder(0, 2, 0, 2)));
531:                    if (vertex.isSelected()) {
532:                            result.setForeground(Color.blue);
533:                    }
534:
535:                    return result;
536:            }
537:
538:            protected Font getVertexFont(V v) {
539:                    if (v.isSelected()) {
540:                            return new Font("Helvetica", Font.ITALIC, 13); //$NON-NLS-
1$
541:                    } else {
542:                            new Font("Helvetica", Font.PLAIN, 12); //$NON-NLS-1$
543:                    }
544:                    return new Font("Helvetica", Font.PLAIN, 10); //$NON-NLS-1$
545:            }
546:
547:            public Paint getEdgeDrawPaint(E mE) {
548:                    if (mE != null) {
549:                            if (mE.isSelected())
550:                                    return Color.CYAN;
551:
552:
553:                            if (Program.isTriades())
554:                            {
555:                                    int state = mE.getCompleteRelation().getState();
556:                                    switch (state) {
557:                                    case RelationComplete.RELATION_OK:
558:                                            return Color.GREEN;
559:                                    case RelationComplete.RELATION_INCOMPLETE:
560:                                            return Color.BLUE;
561:                                    case RelationComplete.RELATION_ECART_OBJECTIF:
562:                                            return Color.RED;
563:                                    case RelationComplete.RELATION_ECART_MOYEN:
564:                                            return Color.getHSBColor(0.0f, 1.0f, 0.41f
);
565:                                    default:
566:                                            return Color.BLACK;
567:                                    }
568:                            }
569:                            else
570:                            {
571:                                    int state = mE.getCompleteRelation().getState();
572:                                    switch (state){
573:                                    case RelationComplete.RELATION_OK:
574:                                            return Color.BLUE;
575:                                    case RelationComplete.RELATION_INCOMPLETE :
576:                                            return Color.RED;
577:                                    default:
578:                                            return Color.black;
579:
580:                                    }
581:                            }
582:                    }
583:
584:                    /*
585:                     * if (mE != null && (mE.source.intValue() < 0 &&
586:                     * mE.source.intValue() > -50) || (mE.destination.intValue() < 0
587:                     * && mE.destination .intValue() > -50)) if (((Integer)
588:                     * e.getUserDatum("selection")).intValue() == 1) return
589:                     * Color.cyan; else return Color.blue;
590:                     */
591:
592:                    if (mE.isSelected())
593:                            return Color.green;
594:                    return Color.black;
595:            }
596:
597:            public boolean getEdgeIncludePredicate(Context<Graph<V, E>, E> arg0)
598:            {
599:                    if (arg0.element instanceof BrickEdge)
600:                    {
601:                            return (!((BrickEdge)arg0.element).getCompleteRelation().i
sNoRelation()) || (((BrickEdge)arg0.element).isSelected());
602:
603:                    }
604:                    return true;
605:            }
606:
607:            public boolean getVertexIncludePredicate(Context<Graph<V, E>, V> arg0) {
608:                    if (!Program.isTriades() || brick.isNavigationBrick())
609:                            return true;
610:                    else if (!(arg0.element.getContent() instanceof Acteur))
611:                            return true;
612:                    else if (arg0.element.getRank() != null && arg0.element.getRank().
getValue() <= vertexShownLevel)
613:                            return true;
614:                    else if (vertexShownLevel >= 3)
615:                            return true;
616:                    else
617:                            return false;
618:
619:
```

```
620:         }
621:
622:         public Icon getVertexIcon(V v) {
623:                 // TODO Doit traiter le cas ou le content est un schÃ©ma.
624:                 // Dans ce cas, afficher le logo de l'activitÃ© du schÃ©ma
625:
626:                 Content content = v.getContent();
627:                 boolean selected = v.isSelected();
628:                 if (content instanceof Activite) {
629:                         return IconDatabase.vectorIconActivities
630:                         .elementAt(((Activite)content).getIconId() + (selected?1:0
));
631:                         // if (selected != null && selected.intValue() == 1)
632:                         // return IconDatabase.iconActivitySelected;
633:                         // return IconDatabase.iconActivity;
634:                 }
635:                 if (content instanceof Moyen)
636:                         return IconDatabase.vectorIconMoyens.elementAt(((Moyen)con
tent).getIdGenerique().intValue()
637:                                         * 2 + (selected?1:0));
638:                 if (content instanceof Acteur) {
639:                         if(v.isSelected())
640:                                 return IconDatabase.vectorIconActorsBig.firstEleme
nt();
641:
642:                         return ((Acteur)content).getIcon();
643:                 }
644:                 if (content instanceof Brick) {
645:                         return IconDatabase.vectorIconActivities.elementAt(((Brick
)content).getActivity().getIconId());
646:                 }
647:
648:                 return IconDatabase.iconRemoveActor0;
649:         }
650:
651:         public Paint getVertexDrawPaint(V v) {
652:                 return Color.black;
653:         }
654:
655:         public String getVertexLabel(V v) {
656:                 String result = v.getContent().toString();
657:                 if (result == null) {
658:                         return Messages.getString("BrickView.11"); //$NON-NLS-1$
659:                 }
660:                 return result;
661:         }
662:
663:         public String getVertexToolTipText(V bV) {
664:                 return bV.getStringDescription();
665:
666:                 //              Integer id = (Integer) v.getUserDatum("id");
667:                 //              if (id != null) {
668:                 //                      if (id.intValue() > -50)
669:                 //                              return "Sommet virtuel permettant
de reprÃ©senter\n des relations sortant de la brique";
670:                 //                      else {
671:                 //                              DataPack dataPack = (DataPack) gra
ph
672:                 //                              .getUserDatum(DataPack.class);
673:                 //                              if (dataPack != null)
674:                 //                                      return "Sommet reprÃ©senta
nt l'activitÃ© centrale de la brique : "
675:                 //                                      + dataPack.getStringById(i
d);
676:                 //                              else
677:                 //                                      return "Sommet reprÃ©senta
nt l'activitÃ© centrale de la brique\n";
678:                 //
679:                 //                      }
680:                 //              }
681:                 //              return "Sommet Ã©trange";
682:         }
683:
684:         public String getEdgeToolTipText(E e) {
685:                 return e.getStringDescription();
686:                 //              Object obj = e.getUserDatum("nom");
687:                 //              if (obj != null && obj.getClass() == String.class)
 {
688:                 //                      return (String) obj;
689:                 //              }
690:                 //              return "ArÃªte";
691:         }
692:
693:         @Override
694:         public String getToolTipText(MouseEvent event) {
695:                 return ((JComponent) event.getSource()).getToolTipText();
696:         }
697:
698:         public TreeListener getTreeListener() {
699:                 return treeListener;
700:         }
701:
702:         public void setTreeListener(TreeListener treeListener) {
703:                 this.treeListener = treeListener;
704:                 mousePlugin.setTreeListener(treeListener);
705:         }
706:
707:
708:         protected void organizeLayout()
709:         {
710:                 double oX = 350;
711:                 double oY = 300;
712:                 if (!brick.isNavigationBrick())
713:                 {
714:                         System.err.println("BrickView.organizeLayout() must be cal
l only on a navigationBrick.");
715:                         return;
716:                 }
717:
718:                 ArrayList<BrickVertex> primaryVertices = new ArrayList<BrickVertex
>();
719:                 ArrayList<BrickVertex> secondaryVertices = new ArrayList<BrickVert
ex>();
720:                 ArrayList<BrickVertex> remainingVertices = new ArrayList<BrickVert
ex>();
721:                 for (BrickVertex bV : brick.getVertices())
722:                 {
723:                         switch (bV.getRank())
724:                         {
725:                         case primary :
726:                                 primaryVertices.add(bV);
727:                                 break;
728:                         case secondary :
729:                                 secondaryVertices.add(bV);
730:                                 break;
731:                         case remaining :
732:                                 remainingVertices.add(bV);
733:                                 break;
734:                         default :
735:                                 System.out.println("Vertex with no rank in a navig
ationBrick (BrickView.organizeLayout).");
736:                                 System.out.println("Vertex info : "+bV.getStringDe
scription());
737:                                 remainingVertices.add(bV);
```

```
  738:                        }
  739:                }
  740:
  741:                int radius = 50 * (primaryVertices.size());
  742:                if (radius <= 75)
  743:                        radius = 0;
  744:                else
  745:                {
  746:                        Ellipse2D shape = new Ellipse2D.Double(oX-radius, oY-radiu
s, 2*radius, 2*radius);
  747:                    Annotation.Layer layer = Annotation.Layer.LOWER;
  748:                        Annotation<Shape> annotation =
  749:                        new Annotation<Shape>(shape, layer, Color.GRAY, false, new
 Point2D.Double(oX+radius, oY+radius));
  750:                aM.add(layer, annotation);
  751:                }
  752:                int count = 0;
  753:
  754:                double delta = Math.PI*2;
  755:                if (primaryVertices.size() > 0)
  756:                        delta /= primaryVertices.size();
  757:                for (BrickVertex bV : primaryVertices)
  758:                {
  759:                        double x = oX+radius*Math.cos(0.1+delta * count);
  760:                        double y = oY+radius*Math.sin(0.1+delta * count);
  761:                        bV.setLocation(new Point2D.Double(x, y));
  762:                        count ++;
  763:
  764:                }
  765:                count = 0;
  766:                radius += 100;
  767:                delta = Math.PI*2;
  768:                {
  769:                Ellipse2D shape = new Ellipse2D.Double(oX-radius, oY-radius, 2*rad
ius, 2*radius);
  770:             Annotation.Layer layer = Annotation.Layer.LOWER;
  771:                Annotation<Shape> annotation =
  772:                        new Annotation<Shape>(shape, layer, Color.GRAY, false, new
 Point2D.Double(oX+radius, oY+radius));
  773:                aM.add(layer, annotation);
  774:                }
  775:                if (secondaryVertices.size() > 0)
  776:                delta /= secondaryVertices.size();
  777:                for (BrickVertex bV : secondaryVertices)
  778:                {
  779:                        double x = oX + radius*Math.cos(-0.1+delta * count);
  780:                        double y = oY + radius*Math.sin(-0.1+delta * count);
  781:                        bV.setLocation(new Point2D.Double(x, y));
  782:                        count ++;
  783:
  784:                }
  785:
  786:                count = 0;
  787:                radius += 100;
  788:                delta = Math.PI*2;
  789:
  790:                {
  791:                        Ellipse2D shape = new Ellipse2D.Double(oX-radius, oY-radiu
s, 2*radius, 2*radius);
  792:                    Annotation.Layer layer = Annotation.Layer.LOWER;
  793:                        Annotation<Shape> annotation =
  794:                                new Annotation<Shape>(shape, layer, Color.GRAY, fa
lse, new Point2D.Double(oX+radius, oY+radius));
  795:                        aM.add(layer, annotation);
  796:                        }
  797:
  798:                if (remainingVertices.size() > 0)
  799:                        delta /= remainingVertices.size();
  800:
  801:                for (BrickVertex bV : remainingVertices)
  802:                {
  803:                        double x = oX + radius*Math.cos(0.1+delta * count);
  804:                        double y = oY + radius*Math.sin(0.1+delta * count);
  805:                        bV.setLocation(new Point2D.Double(x, y));
  806:                        count ++;
  807:
  808:                }
  809:        }
  810:
  811:        //      @Override
  812:        //      public boolean evaluate(Object arg0) {
  813:        //              return true;
  814:        //      }
  815: }
```

```
    1: package models;
    2:
    3: import java.util.HashSet;
    4: import java.util.Set;
    5:
    6: import translation.Messages;
    7:
    8: import dataPack.Content;
    9:
   10: public class NavigationEdge extends BrickEdge {
   11:
   12:         private static final long serialVersionUID = 9007768621574379344L;
   13:         protected Set<Content> sharedContent;
   14:
   15:         public NavigationEdge(BrickVertex source, BrickVertex destination,
   16:                         Set<Content> actorIntersection) {
   17:                 super(source, destination);
   18:                 sharedContent = new HashSet<Content>(actorIntersection);
   19:
   20:         }
   21:
   22:         @Override
   23:         public String getStringDescription() {
   24:                 int size = sharedContent.size();
   25:                 if (size == 1) {
   26:                         return Messages.getString("NavigationEdge.0") + sharedCont
ent.iterator().next().toString(); //$NON-NLS-1$
   27:                 } else {
   28:                         String result = "<html>" + size + Messages.getString("Navi
gationEdge.2"); //$NON-NLS-1$ //$NON-NLS-2$
   29:                         for (Content id : sharedContent) {
   30:                                 result += "<br/>- " + id.toString(); //$NON-NLS-1$
   31:                         }
   32:                         return result + "</html>"; //$NON-NLS-1$
   33:                 }
   34:         }
   35:
   36:         public Set<Content> getSharedContent() {
   37:                 return sharedContent;
   38:         }
   39:
   40:
   41: }
```

```
 1: package models;
 2:
 3: import java.io.Serializable;
 4:
 5: public class BrickType implements Serializable {
 6:
 7:         private static final long serialVersionUID = -59990001611121786L;
 8:
 9:         protected String name;
10:         protected Integer brickTypeId;
11:
12:         public Integer getBrickTypeId() {
13:                 return brickTypeId;
14:         }
15:
16:         public void setBrickTypeId(Integer brickTypeId) {
17:                 this.brickTypeId = brickTypeId;
18:         }
19:
20:         public BrickType(String _name, Integer id) {
21:                 name = _name;
22:                 brickTypeId = id;
23:         }
24:
25:         public boolean equals(BrickType other) {
26:                 return other.name == name;
27:         }
28:
29:         public String toString() {
30:                 return name;
31:         }
32:
33: }
```

```
    1: package models;
    2:
    3: import java.awt.event.MouseEvent;
    4: import java.awt.geom.Point2D;
    5: import java.util.Iterator;
    6:
    7: import edu.uci.ics.jung.algorithms.layout.Layout;
    8: import edu.uci.ics.jung.visualization.VisualizationViewer;
    9: import edu.uci.ics.jung.visualization.control.PickingGraphMousePlugin;
   10: import edu.uci.ics.jung.visualization.picking.PickedState;
   11:
   12: public class MyPickingEditingMousePlugin<V extends BrickVertex, E extends BrickEdg
e>
   13: extends PickingGraphMousePlugin<V, E> {
   14:
   15:         public MyPickingEditingMousePlugin(int selectionModifiers,
   16:                         int addToSelectionModifiers) {
   17:                 super(selectionModifiers, addToSelectionModifiers);
   18:         }
   19:
   20:         @SuppressWarnings("unchecked")
   21:         @Override
   22:         public void mouseDragged(MouseEvent e) {
   23:                 if (locked == false) {
   24:                         VisualizationViewer<V, E> vv = (VisualizationViewer<V, E>)
 e
   25:                         .getSource();
   26:                         if (vertex != null) {
   27:
   28:
   29:
   30:                                 Layout<V, E> layout = vv
   31:                                 .getGraphLayout();
   32:                                 PickedState<V> ps = vv.getPickedVertexState();
   33:
   34:                                 Point2D newLocation = vv.getRenderContext().getMul
tiLayerTransformer().inverseTransform(e.getPoint());
   35:                                 layout.setLocation(vertex, newLocation);
   36:
   37:                                 vertex.setLocation(newLocation);

   38:
   39:
   40:
   41:
   42:                         } else {
   43:                                 Point2D out = e.getPoint();
   44:                                 if (e.getModifiers() == this.addToSelectionModifie
rs
   45:                                                 || e.getModifiers() == modifiers)
{
   46:                                         rect.setFrameFromDiagonal(down, out);
   47:                                 }
   48:                         }
   49:                         if (vertex != null)
   50:                                 e.consume();
   51:                 }
   52:         }
   53:
   54: }
```

```java
package models;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Shape;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.geom.AffineTransform;
import java.awt.geom.Point2D;
import java.awt.geom.QuadCurve2D;

import translation.Messages;

import dataPack.Content;
import dataPack.TreeListener;
import edu.uci.ics.jung.algorithms.layout.Layout;
import edu.uci.ics.jung.visualization.VisualizationServer.Paintable;
import edu.uci.ics.jung.visualization.VisualizationViewer;
import edu.uci.ics.jung.visualization.control.EditingGraphMousePlugin;
import edu.uci.ics.jung.visualization.util.ArrowFactory;

public abstract class TriadeEditingMousePlugin<V extends BrickVertex, E extends BrickEdge>
                extends
 EditingGraphMousePlugin<V, E>
                implements MouseListener, MouseMotionListener, GraphListener {

        protected Layout<V, E> vertexLocations;
        protected V startVertex;
        protected Point2D down;
        protected BrickView<V, E> modelView;

        protected boolean drawArrow;

        final protected QuadCurve2D rawEdge = new QuadCurve2D.Float(0,0,50,35,100,0);
        protected Shape edgeShape;
        protected Shape rawArrowShape;
        protected Shape arrowShape;
        protected Paintable edgePaintable;
        protected Paintable arrowPaintable;
        protected boolean edgeIsDirected;
        protected V mobile = null;

        protected V selectedVertex;
        protected E selectedEdge;
        protected Content activeContent;

        protected TreeListener treeListener;

        abstract public Brick getEditedAbstractSchema();

        abstract public void removeSelectedVertex();

        public TriadeEditingMousePlugin(Layout<V, E> vertexLocation) {
                this(MouseEvent.BUTTON1_MASK);

                if(vertexLocation == null) {
                        throw new IllegalArgumentException("vertexLocation must be not null"); //$NON-NLS-1$
                }

                setVertexLocations(vertexLocation);
        }

        /**
         * create instance and prepare shapes for visual effects
         *
         * @param modifiers
         */
        public TriadeEditingMousePlugin(int modifiers) {
                super(modifiers, null, null);
                rawArrowShape = ArrowFactory.getNotchedArrow(20, 16, 12);
                edgePaintable = new EdgePaintable();
                arrowPaintable = new ArrowPaintable();
                activeContent = null;
                selectedVertex = null;
//              edgeIsDirected = true;

                drawArrow = true;
        }

        public void selectEdge(E newSelection) {
                if (selectedVertex != null) {
                        selectedVertex.setSelected(false);
                        selectedVertex = null;
                }
                if (selectedEdge != null) {
                        selectedEdge.setSelected(false);
                        selectedEdge = null;
                }
                if (newSelection != null) {
                        selectedEdge = newSelection;
                        selectedEdge.setSelected(true);
                }

                getModelView().repaint();
        }

        public void selectVertex(V newSelection) {
                if (selectedVertex != null) {
                        selectedVertex.setSelected(false);
                        selectedVertex = null;
                }
                if (selectedEdge != null) {
                        selectedEdge.setSelected(false);
                        selectedEdge = null;
                }
                if (newSelection != null) {
                        selectedVertex = newSelection;
                        selectedVertex.setSelected(true);
                }

                getModelView().repaint();

                notifyTree(newSelection != null ? newSelection.getContent() : null);
        }

        abstract public void selectVertex(Content content);
        abstract public void notifyTree(Content content);

        /**
         * sets the vertex locations. Needed to place new vertices
         *
         * @param vertexLocations
         */
        public void setVertexLocations(
Layout<V, E> vertexLocations) {
                this.vertexLocations = vertexLocations;
        }

```

```
131:         public void setContent(Content newContent) {
132:                 activeContent = newContent;
133:         }
134:
135:         /**
136:          * overrided to be more flexible, and pass events with key combinations. T
he
137:          * default responds to both ButtonOne and ButtonOne+Shift
138:          */
139:         @Override
140:         public boolean checkModifiers(MouseEvent e) {
141:                 return (e.getModifiers() == modifiers);
142:         }
143:
144:         /**
145:          * If the mouse is pressed in an empty area, create a new vertex there. If
146:          * the mouse is pressed on an existing vertex, prepare to create an edge
147:          * from that vertex to another
148:          */
149:
150:         /**
151:          * If startVertex is non-null, and the mouse is released over an existing
152:          * vertex, create an directed edge from startVertex to the vertex under th
e
153:          * mouse pointer with a transition.
154:          */
155:
156:         /**
157:          * If startVertex is non-null, stretch an edge shape between startVertex a
nd
158:          * the mouse pointer to simulate edge creation
159:          */
160:         @SuppressWarnings("unchecked")
161:         @Override
162:         public void mouseDragged(MouseEvent e) {
163:                 if (checkModifiers(e)) {
164:                         VisualizationViewer<BrickVertex, BrickEdge> vv = (Visualiz
ationViewer<BrickVertex, BrickEdge>) e
165:                                 .getSource();
166:
167:                         if (startVertex != null) {
168:                                 BrickVertex endVertex = vv.getPickSupport().getVer
tex(vv.getGraphLayout(), e.getPoint().getX(), e.getPoint().getY());
169:
170:                                 Point2D endPoint = e.getPoint();
171:                                 if(endVertex != null)
172:                                         endPoint = vv.getRenderContext().getMultiL
ayerTransformer().transform(endVertex.getLocation());
173:
174:
175:                                 transformEdgeShape(down, endPoint);
176:                                 transformArrowShape(down, endPoint);
177:
178:                                 if (down.distance(endPoint) < 20)
179:                                         drawArrow = false;
180:                                 else
181:                                         drawArrow = true;
182:
183:                         }
184:
185:                         vv.repaint();
186:                 }
187:         }
188:
189:         /**
190:          * code lifted from PluggableRenderer to move an edge shape into an
191:          * arbitrary position
192:          */
193:         protected void transformEdgeShape(Point2D down, Point2D out) {
194:                 float x1 = (float) down.getX();
195:                 float y1 = (float) down.getY();
196:                 float x2 = (float) out.getX();
197:                 float y2 = (float) out.getY();
198:
199:                 AffineTransform xform = AffineTransform.getTranslateInstance(x1, y
1);
200:
201:                 float dx = x2 - x1;
202:                 float dy = y2 - y1;
203:                 float thetaRadians = (float) Math.atan2(dy, dx);
204:                 xform.rotate(thetaRadians);
205:                 float dist = (float) Math.sqrt(dx * dx + dy * dy);
206:                 xform.scale(dist / rawEdge.getBounds().getWidth(), 1.0);
207:                 edgeShape = xform.createTransformedShape(rawEdge);
208:         }
209:
210:         protected void transformArrowShape(Point2D down, Point2D out) {
211:                 float x1 = (float) down.getX();
212:                 float y1 = (float) down.getY();
213:                 float x2 = (float) out.getX();
214:                 float y2 = (float) out.getY();
215:
216:                 AffineTransform xform = AffineTransform.getTranslateInstance(x2, y
2);
217:
218:                 float dx = x2 - x1;
219:                 float dy = y2 - y1;
220:                 float thetaRadians = (float) Math.atan2(dy, dx);
221:                 xform.rotate(thetaRadians);
222:                 arrowShape = xform.createTransformedShape(rawArrowShape);
223:         }
224:
225:         /**
226:          * Used for the edge creation visual effect during mouse drag
227:          */
228:         class EdgePaintable implements Paintable {
229:
230:                 public void paint(Graphics g) {
231:                         if (edgeShape != null && drawArrow) {
232:                                 Color oldColor = g.getColor();
233:                                 g.setColor(Color.BLACK);
234:                                 ((Graphics2D) g).draw(edgeShape);
235:                                 g.setColor(oldColor);
236:                         }
237:                 }
238:
239:                 public boolean useTransform() {
240:                         return false;
241:                 }
242:         }
243:
244:         /**
245:          * Used for the directed edge creation visual effect during mouse drag
246:          */
247:         class ArrowPaintable implements Paintable {
248:
249:                 public void paint(Graphics g) {
250:                         if (arrowShape != null && drawArrow) {
251:                                 Color oldColor = g.getColor();
252:                                 g.setColor(Color.BLACK);
253:                                 ((Graphics2D) g).fill(arrowShape);
254:                                 g.setColor(oldColor);
255:                         }
256:                 }
```

```
257:
258:                    public boolean useTransform() {
259:                            return false;
260:                    }
261:            }
262:
263:            @Override
264:            public void mouseClicked(MouseEvent e) {
265:
266:            }
267:
268:            @Override
269:            public void mouseEntered(MouseEvent e) {
270:            }
271:
272:            @Override
273:            public void mouseExited(MouseEvent e) {
274:            }
275:
276:            @Override
277:            public void mouseMoved(MouseEvent e) {
278:            }
279:
280:            public BrickView<V, E> getModelView() {
281:                    return modelView;
282:            }
283:
284:            public void setModelView(BrickView<V, E> modelView) {
285:                    this.modelView = modelView;
286:            }
287:
288:            public void setTreeListener(TreeListener treeListener) {
289:                    this.treeListener = treeListener;
290:            }
291: }
```

```java
package models;

import java.awt.geom.Point2D;
import java.io.Serializable;
import java.util.HashMap;
import java.util.Vector;

import client.stringTranslator.StringTranslator;
import dataPack.ActeurBase;
import dataPack.Activite;
import dataPack.Content;
import dataPack.DataPack;
import dataPack.JeuActeur;
import edu.uci.ics.jung.graph.DirectedGraph;

public class Brick implements Serializable,  Content {

	/**
	 *
	 */
	private static final long serialVersionUID = -3672273059633190927L;



	protected Vector<BrickVertex> vertices;
	protected String name;
	protected String step;
	protected Vector<BrickEdge> edges;

	protected Vector<BrickVertex> genericActors;

	protected DataPack datapack;

	protected transient DirectedGraph<BrickVertex, BrickEdge> graph;

	protected Activite activity;

	protected boolean navigationBrick;

	protected Brick() {
		vertices = null;

		graph = null;
	}

	private Brick(String _step, String _nom, DataPack _datapack) {
		step = _step;
		vertices = new Vector<BrickVertex>(5);
		edges = new Vector<BrickEdge>(10);
		name = _nom;
		datapack = _datapack;
		graph = null;
		genericActors = new Vector<BrickVertex>();
		navigationBrick = false;
		activity = null;
	}

	public Brick(String _step, String _nom, DataPack _datapack,
			Activite _activity) {
		this(_step, _nom, _datapack);
		if (_activity != null) {
			activity = _activity;
			BrickVertex activityVertex = addBrickVertex(activity);
			activityVertex.setLocation(new Point2D.Double(0, 0));
			navigationBrick = false;
		} else {
			this.navigationBrick = true;
		}
	}

	public Brick(Brick brick) {
		step = brick.step;
		name = brick.name;
		datapack = brick.datapack;
		graph = brick.graph;
		activity = brick.activity;
		navigationBrick = brick.navigationBrick;
		genericActors = new Vector<BrickVertex>(brick.genericActors);

		HashMap<BrickVertex, BrickVertex> oldNew = new HashMap<BrickVertex, BrickVertex>();

		vertices = new Vector<BrickVertex>(brick.vertices.size());
		for(BrickVertex brickVertex : brick.vertices) {
			BrickVertex value = new BrickVertex(brickVertex);
			oldNew.put(brickVertex, value);
			vertices.add(value);
		}

		edges = new Vector<BrickEdge>(brick.edges.size());
		for(BrickEdge brickEdge: brick.edges) {
			edges.add(new BrickEdge(oldNew.get(brickEdge.getSource()), oldNew.get(brickEdge.getDestination()) ,brickEdge.getCompleteRelation()));
		}
	}

	public BrickVertex addBrickVertex(Content newContent) {
		BrickVertex temp = new BrickVertex(datapack);
		temp.setContent(newContent);
		vertices.add(temp);

		if (newContent instanceof ActeurBase
				&& ((ActeurBase) newContent).isBase()) {
			genericActors.add(temp);
		}

		if (graph != null)
		{
			graph.addVertex(temp);
		}
		return temp;
	}

	public BrickEdge addEdge(BrickEdge newEdge) {
		edges.add(newEdge);
		return newEdge;
	}

	public void removeActor(Content oldActor) {

		for (int i = 0; i < vertices.size();) {

			if (vertices.elementAt(i).getContent().equals(oldActor)) {
				if (oldActor instanceof ActeurBase      && ((ActeurBase) oldActor).isBase())
				{
					genericActors.remove(vertices.elementAt(i));
				}
				removeBrickVertex(vertices.elementAt(i));

			} else
				i++;
		}
```

```
131:
132:
133:        }
134:
135:        public void removeBrickVertex(BrickVertex brickVertex) {
136:                if (vertices.contains(brickVertex)) {
137:                        for (int i = 0; i < edges.size();) {
138:                                if (edges.elementAt(i).source.equals(brickVertex)
139:                                                || edges.elementAt(i).destination
140:                                                .equals(brickVertex))
141:                                        removeModelEdge(edges.elementAt(i));
142:                                else
143:                                        i++;
144:                        }
145:                        vertices.remove(brickVertex);
146:
147:
148:                        if (brickVertex.getContent() instanceof ActeurBase)
149:                        {
150:                                if (((ActeurBase)brickVertex.getContent()).isBase(
))
151:                                {
152:                                        while (genericActors.contains(brickVertex)
)
153:                                                genericActors.remove(brickVertex);
154:                                }
155:                        }
156:
157:                        if (graph != null) {
158:                                graph.removeVertex(brickVertex);
159:                        }
160:                } else {
161:                        System.out.println("removeBrickVertex failed"); //$NON-NLS
-1$
162:                }
163:        }
164:
165:
166:        public void removeModelEdge(BrickEdge modelEdge) {
167:                if (edges.contains(modelEdge)) {
168:                        edges.remove(modelEdge);
169:                        if (graph != null)
170:                        {
171:                                graph.removeEdge(modelEdge);
172:                        }
173:                }
174:        }
175:
176:        public void switchActivity(Activite oldActivity, Activite newActivity)
177:        {
178:                BrickVertex oldVertex = getBrickVertexByContent(oldActivity);
179:                if (oldVertex == null)
180:                {
181:                        System.err.println("Brick.switchActivity : no vertex corre
spond to the oldActivity, switch aborted"); //$NON-NLS-1$
182:                }
183:
184:                BrickVertex newVertex = addBrickVertex(newActivity);
185:                for (BrickEdge bE : edges)
186:                {
187:                        if (bE.getSource().equals(oldVertex))
188:                        {
189:                                bE.setSource(newVertex);
190:                                if (graph != null)
191:                                {
192:                                        graph.removeEdge(bE);
193:                                        graph.addEdge(bE, newVertex, bE.getDestina
tion());
194:                                }
195:                        }
196:
197:                        if (bE.getDestination().equals(oldVertex))
198:                        {
199:                                bE.setDestination(newVertex);
200:                                if (graph != null)
201:                                {
202:                                        graph.removeEdge(bE);
203:                                        graph.addEdge(bE, bE.getSource(), newVerte
x);
204:                                }
205:                        }
206:                }
207:                newVertex.setLocation(oldVertex.getLocation());
208:                removeBrickVertex(oldVertex);
209:                activity = newActivity;
210:        }
211:
212:
213:        public BrickVertex getBrickVertexByContent(Content lookedContent) {
214:                for (BrickVertex vertex : vertices)
215:                        if (vertex.getContent().equals(lookedContent))
216:                                return vertex;
217:
218:                System.out
219:                .println("Aucun sommet trouvÃ© correspondant Ã  l'id fourni (Brick
.getBrickVertexByContent avec le contentId:" //$NON-NLS-1$
220:                                + lookedContent + ")"); //$NON-NLS-1$
221:
222:                System.out.println(vertices);
223:
224:                return null;
225:
226:        }
227:
228:        public Vector<BrickVertex> getVertices() {
229:                return vertices;
230:        }
231:
232:        public DataPack getDatapack() {
233:                return datapack;
234:        }
235:
236:
237:        public Vector<BrickEdge> getEdges() {
238:                return edges;
239:        }
240:
241:
242:
243:        public Activite getActivity() {
244:                return activity;
245:        }
246:
247:        public String getName() {
248:                return name;
249:        }
250:
251:        public String getStep() {
252:                return step;
253:        }
254:
255:        public boolean isGeneric() {
256:                if (genericActors == null) {
257:                        genericActors = new Vector<BrickVertex>();
```

```
  258:
  259:                         //rÃ©cupÃ©re la liste des acteur gÃ©nÃ©rique dans la brick
  260:                         for (BrickVertex vertex : vertices) {
  261:                                 Content vertexContent = vertex.getContent();
  262:
  263:                                 if (vertexContent instanceof ActeurBase
  264:                                                 && ((ActeurBase) vertexContent).is
Base()) {
  265:                                         genericActors.add(vertex);
  266:                                 }
  267:                         }
  268:                 }
  269:
  270:                 return genericActors.size() > 0;
  271:         }
  272:
  273:         private void replaceBaseByCorps(BrickVertex vertex, String nomCorps) {
  274:                 if (((ActeurBase)vertex.getContent()).isBase())
  275:                 {
  276:                         Content newContent = ((ActeurBase) vertex.getContent())
  277:                         .getJeuActeur().getRealActor(
  278:                                         (ActeurBase) vertex.getContent(), nomCorps
);
  279:                         vertex.setContent(newContent);
  280:                 }
  281:         }
  282:
  283:         public Vector<Brick> createNoGenericBrick() {
  284:                 if (isGeneric() == false) {
  285:                         throw new RuntimeException("Demande de gÃ©nÃ©rer les briqu
es non gÃ©nÃ©rique d'une brique deja non gÃ©nÃ©rique"); //$NON-NLS-1$
  286:                 }
  287:
  288:                 Vector<Brick> noGenericBricks = new Vector<Brick>();
  289:                 JeuActeur genericJeuActeur = ((ActeurBase) genericActors.firstElem
ent()
  290:                                 .getContent()).getJeuActeur();
  291:
  292:                 for(String nomCorps : genericJeuActeur.getListeCorps()) {
  293:                         //CrÃ©er une brique pour chaques corps du jeu d'acteur ide
ntique a celle la (constructeur par recopi)
  294:                         Brick newBrick = new Brick(this);
  295:                         newBrick.name += " " + nomCorps; //$NON-NLS-1$
  296:                         newBrick.genericActors.clear();
  297:
  298:                         for (BrickVertex genericVertex : genericActors) {
  299:                                 // rÃ©cupÃ©re l'id du corps de la base
  300:                                 BrickVertex vertexToBeChanged = newBrick
  301:                                 .getBrickVertexByContent(genericVertex.getContent(
));
  302:
  303:                                 //remplacer les acteur generique par les acteur no
n generique de la base
  304:                                 newBrick.replaceBaseByCorps(vertexToBeChanged, nom
Corps);
  305:                         }
  306:
  307:                         noGenericBricks.add(newBrick);
  308:                 }
  309:
  310:                 return noGenericBricks;
  311:         }
  312:
  313:
  314:         public boolean isNavigationBrick() {
  315:                 return navigationBrick;
  316:         }
  317:
  318:         public void setNavigationBrick(boolean navigationBrick) {
  319:                 this.navigationBrick = navigationBrick;
  320:         }
  321:
  322:         public String getNoTranslatedString() {
  323:                 return name + " (" + step + ")";
  324:         }
  325:
  326:         @Override
  327:         public String toString() {
  328:                 return StringTranslator.getTranslatedString(this, StringTranslator
.StringType.brickType);
  329:         }
  330:
  331:         public void setStep(String newName) {
  332:                 step = newName;
  333:         }
  334:
  335:         public void setName(String newName) {
  336:                 name = newName;
  337:
  338:         }
  339:
  340:         public void resetGraph() {
  341:                 graph = null;
  342:
  343:         }
  344: }
```

```
    1: package models;
    2:
    3: import java.awt.geom.Point2D;
    4: import java.io.Serializable;
    5:
    6:
    7: import edu.uci.ics.jung.graph.Vertex;
    8:
    9: public class ModelVertex implements Serializable{
   10:
   11:         /**
   12:          *
   13:          */
   14:         private static final long serialVersionUID = -643197059838492478L;
   15:
   16:         /**
   17:          * contentId contient un entier indiquant l'acteur ou le type de brique dÃ
©signÃ© par ce sommet
   18:          * nÃ©gatif pour un type de brique (de -10 Ã  -49) positif pour un acteur
   19:          * Pour connaitre l'acteur correspondant (ou la brique correspondante) se
rÃ©fÃ©rer au datapack
   20:          */
   21:         protected Integer contentId;
   22:         protected Point2D location;
   23:         protected Integer vertexId;
   24:
   25:         protected transient Vertex associatedVertex;
   26:
   27:
   28:         public ModelVertex(Integer id, Integer _contentId)
   29:         {
   30:                 vertexId = id;
   31:                 contentId = _contentId;
   32:                 location = null;
   33:         }
   34:
   35:         public Point2D getLocation() {
   36:                 return location;
   37:         }
   38:
   39:         public void setLocation(Point2D location) {
   40:                 this.location = location;
   41:         }
   42:
   43:         public Vertex getAssociatedVertex() {
   44:                 return associatedVertex;
   45:         }
   46:
   47:         public void setAssociatedVertex(Vertex associatedVertex) {
   48:                 this.associatedVertex = associatedVertex;
   49:         }
   50:
   51:         public Integer getContentId() {
   52:                 return contentId;
   53:         }
   54:
   55:         public Integer getVertexId() {
   56:                 return vertexId;
   57:         }
   58:
   59:
   60: }
```

```
    1: package models;
    2:
    3: import java.awt.geom.Point2D;
    4: import java.io.Serializable;
    5:
    6: import dataPack.Content;
    7:
    8: public class SchemaVertex implements Serializable {
    9:
   10:         /**
   11:          *
   12:          */
   13:         private static final long serialVersionUID = -643197059838492478L;
   14:
   15:         /**
   16:          * contentId contient un entier indiquant l'acteur ou le type de brique
   17:          * dÃ©signÃ© par ce sommet nÃ©gatif pour un type de brique (de -10 Ã  -49)
   18:          * positif pour un acteur Pour connaitre l'acteur
   19:          * corassociatedContentrespondant (ou la brique correspondante) se rÃ©fÃ©r
er
   20:          * au datapack
   21:          */
   22:         protected Content content;
   23:         private Point2D location;
   24:         protected Integer vertexId;
   25:         protected boolean isSelected;
   26:
   27:         public SchemaVertex(Integer id, Content content) {
   28:                 vertexId = id;
   29:                 location = null;
   30:                 this.content = content;
   31:                 isSelected = false;
   32:         }
   33:
   34:         public Point2D getLocation() {
   35:                 return location;
   36:         }
   37:
   38:         public void setLocation(Point2D location) {
   39:                 this.location = location;
   40:         }
   41:
   42:         public Content getContent() {
   43:                 return content;
   44:         }
   45:
   46:         public Integer getVertexId() {
   47:                 return vertexId;
   48:         }
   49:
   50:         public boolean isSelected() {
   51:                 return isSelected;
   52:         }
   53:
   54:         public void setSelected(boolean isSelected) {
   55:                 this.isSelected = isSelected;
   56:         }
   57:
   58:         @Override
   59:         public String toString() {
   60:                 return "" + vertexId + " content : " + content;
   61:         }
   62: }
```

```
    1: package models;
    2:
    3: import java.io.Serializable;
    4:
    5: import translation.Messages;
    6:
    7: import main.RelationComplete;
    8: import client.export.ExportVertexData;
    9:
   10: public class BrickEdge implements Serializable {
   11:         private static final long serialVersionUID = -5466333956926991387L;
   12:
   13:         protected BrickVertex source;
   14:         protected BrickVertex destination;
   15:         protected RelationComplete relations;
   16:
   17:         protected boolean selected;
   18:
   19:         public BrickEdge(BrickVertex _source, BrickVertex _destination) {
   20:                 source = _source;
   21:                 destination = _destination;
   22:                 relations = new RelationComplete();
   23:                 selected = false;
   24:         }
   25:
   26:         public BrickEdge(BrickVertex source2, BrickVertex destination2,
   27:                         RelationComplete relationComplete) {
   28:                 source = source2;
   29:                 destination = destination2;
   30:                 relations = new RelationComplete(relationComplete);
   31:         }
   32:
   33:         public BrickEdge(BrickEdge brickEdge) {
   34:                 this(brickEdge.source, brickEdge.destination, brickEdge.relations)  $ //$NON-NLS-2$
;
   35:         }
   36:
   37:         public BrickEdge(BrickEdge baseEdge, boolean doNotClone) {
   38:                 source = baseEdge.source;
   39:                 destination = baseEdge.destination;
   40:                 if (doNotClone)
   41:                         relations = baseEdge.relations;
   42:                 else
   43:                         relations = new RelationComplete(baseEdge.relations);
   44:         }
   45:
   46:         public BrickEdge(ExportVertexData source2, ExportVertexData destination2,
   47:                         RelationComplete completeRelation, boolean doNotClone) {
   48:                 source = source2;
   49:                 destination = destination2;
   50:
   51:                 if (doNotClone)
   52:                         relations = completeRelation;
   53:                 else
   54:                         relations = new RelationComplete(completeRelation);
   55:         }
   56:
   57:         public BrickVertex getSource() {
   58:                 return source;
   59:         }
   60:
   61:
   62:         public void setSource(BrickVertex newVertex) {
   63:                 source = newVertex;
   64:
   65:         }
   66:
   67:         public BrickVertex getDestination() {
   68:                 return destination;
   69:         }
   70:
   71:         public void setDestination(BrickVertex newVertex)
   72:         {
   73:                 destination = newVertex;
   74:         }
   75:
   76:         public RelationComplete getCompleteRelation() {
   77:                 return relations;
   78:         }
   79:
   80:         public boolean equals(BrickEdge other) {
   81:                 return (source.equals(other.source) && destination
   82:                                 .equals(other.destination));
   83:         }
   84:
   85:         public String getStringDescription() {
   86:                 return relations.getStringDescription();
   87:         }
   88:
   89:         public boolean isSelected() {
   90:                 return selected;
   91:         }
   92:
   93:         public void setSelected(boolean selected) {
   94:                 this.selected = selected;
   95:         }
   96:
   97:         @Override
   98:         public String toString() {
   99:                 return "src = " + source + ", dst = " + destination;  //$NON-NLS-1
  100:         }
  101:
  102: }
```

```java
package models;

import java.awt.geom.Point2D;
import java.io.Serializable;


import translation.Messages;
import dataPack.Acteur;
import dataPack.Activite;
import dataPack.Content;
import dataPack.DataPack;
import dataPack.Moyen;

public class BrickVertex implements Serializable {

	/**
	 *
	 */
	private static final long serialVersionUID = 241153997966787582 8L;
	protected Content content;
	protected Point2D location;
	protected boolean selected;
	protected VerticeRank rank;

	public enum VerticeRank implements Comparable<VerticeRank>
	{
		primary(1), secondary(2), remaining(3), undefined(4);

		private int r;

		private VerticeRank(int r)
		{
			this.r = r;
		}

		public int getValue()
		{
			return r;
		}

	};

	protected BrickVertex(DataPack _dataPack) {
		content = null;
		selected = false;
		rank = VerticeRank.remaining;
	}

	protected BrickVertex(BrickVertex brickVertex) {
		content = brickVertex.content;
		location = brickVertex.location;
		rank = brickVertex.rank;
	}

	protected void setActor(Content newContent) {
		content = newContent;
	}

	protected void setLocation(Point2D newLocation) {
		location = newLocation;
	}

	public Point2D getLocation() {
		return location;
	}

	public void setRank(VerticeRank r)
	{
		rank = r;
	}

	public VerticeRank getRank()
	{
		return rank;
	}



	public String getStringDescription() {

		if (content instanceof Moyen)
			return Messages.getString("BrickVertex.0") + content; //$N
ON-NLS-1$
		else if (content instanceof Acteur)
			return Messages.getString("BrickVertex.1") + content; //$N
ON-NLS-1$
		else if (content instanceof Activite)
			return Messages.getString("BrickVertex.2") + content; //$N
ON-NLS-1$
		else if (content instanceof Brick)
			return "Brick : " + content;
		else
			return Messages.getString("BrickVertex.3"); //$NON-NLS-1$
	}

	public Content getContent() {
		return content;
	}

	@Override
	public boolean equals(Object other) {
		if (other instanceof BrickVertex) {
			return ((BrickVertex) other).content.equals(content);
		}
		return false;
	}

	@Override
	public String toString() {
		return "content : " + content; //$NON-NLS-1$
	}

	public void setContent(Content newContent) {
		content = newContent;
	}

	public boolean isSelected() {
		return selected;
	}

	public void setSelected(boolean selected) {
		this.selected = selected;
	}
}
```

```
    1: package models;
    2:
    3: import java.awt.geom.Point2D;
    4: import java.util.HashMap;
    5: import java.util.Map;
    6: import java.util.Vector;
    7:
    8: import models.BrickVertex.VerticeRank;
    9: import translation.Messages;
   10: import dataPack.Content;
   11: import dataPack.DataPack;
   12:
   13: public class SchemaGenerator {
   14:
   15:         protected DataPack datapack;
   16:         protected Map<Content, VerticeRank> actors;
   17:
   18:         public SchemaGenerator(DataPack _datapack, Map<Content, VerticeRank> _acto
rs) {
   19:                 datapack = _datapack;
   20:                 actors = _actors;
   21:         }
   22:
   23:         public Vector<Brick> generateBricks() {
   24:                 Vector<Brick> result = new Vector<Brick>();
   25:                 HashMap<String, Brick> mainBricks = new HashMap<String, Brick>();
   26:                 for (String step : datapack.getSteps()) {
   27:                         Brick stepBrick = new Brick(step, Messages.getString("Sche
maGenerator.0") //$NON-NLS-1$
   28:                                         + step, datapack, null);
   29:                         mainBricks.put(step, stepBrick);
   30:                         result.add(stepBrick);
   31:                 }
   32:
   33:                 Vector<Brick> usedBrick = new Vector<Brick>();
   34:                 Vector<Vector<Content>> brickActors = new Vector<Vector<Content>>(
);
   35:
   36:                 for (Brick brick : datapack.getBrickList()) {
   37:                         if (!brick.isGeneric()) {
   38:                                 Vector<Content> brickSet = new Vector<Content>();
   39:                                 VerticeRank maxRank = VerticeRank.undefined;
   40:                                 for (BrickVertex vertex : brick.getVertices()) {
   41:                                         if (actors.containsKey(vertex.getContent()
)) {
   42:                                                 brickSet.add(vertex.getContent());
   43:                                                 if (vertex.getRank() != null && ma
xRank.getValue() > vertex.getRank().getValue())
   44:                                                 {
   45:                                                         maxRank = vertex.getRank()
;
   46:                                                 }
   47:                                         }
   48:                                 }
   49:
   50:                                 if (brickSet.size() > 0) {
   51:                                         usedBrick.add(brick);
   52:                                         brickActors.add(brickSet);
   53:                                         String step = brick.getStep();
   54:                                         BrickVertex newVertex = mainBricks.get(ste
p)
   55:                                         .addBrickVertex(brick);
   56:                                         newVertex.setLocation(new Point2D.Double(u
sedBrick.size() * 5,
   57:                                                         usedBrick.size() * 5));
   58:                                         newVertex.setRank(maxRank);
   59:                                 }
   60:                         }
   61:                 }
   62:
   63: //              for (int i = 0; i < usedBrick.size(); i++) {
   64: //                      for (int j = i + 1; j < usedBrick.size(); j++) {
   65: //                              if (usedBrick.elementAt(i).getStep().equals(
   66: //                                              usedBrick.elementAt(j).getStep()))
 {
   67: //                                  Brick stepBrick = mainBricks.get(usedBrick
.elementAt(i)
   68: //                                                  .getStep());
   69: //                                  Set<Content> intersection = new HashSet<Co
ntent>();
   70: //                                  intersection.addAll(brickActors.elementAt(
i));
   71: //                                  intersection.retainAll(brickActors.element
At(j));
   72: //
   73: //                                  if (intersection.size() > 0) {
   74: //
   75: //
   76: //                                          BrickVertex source = stepBrick
   77: //                                          .getBrickVertexByContent(usedBrick
.elementAt(i));
   78: //                                          BrickVertex dest = stepBrick
   79: //                                          .getBrickVertexByContent(usedBrick
.elementAt(j));
   80: //
   81: //                                          stepBrick.addEdge(new NavigationEd
ge(source, dest,
   82: //                                                          intersection));
   83: //                                  }
   84: //                              }
   85: //                      }
   86: //              }
   87:
   88:                 for (Brick brick : usedBrick) {
   89:
   90:                         result.add(brick);
   91:                 }
   92:
   93:                 return result;
   94:         }
   95: }
```

```
1: package models;
2:
3: public interface AbstractSchema {
4:
5:         public void removeElement(Object Element);
6:
7:         public void removeModelEdge(BrickEdge mEdge);
8: }
```

```java
package models;

import java.awt.event.MouseEvent;
import java.awt.geom.Point2D;

import translation.Messages;

import client.export.ExportsContextualMenu;
import dataPack.Content;
import dataPack.TreeListener;
import edu.uci.ics.jung.algorithms.layout.GraphElementAccessor;
import edu.uci.ics.jung.algorithms.layout.Layout;
import edu.uci.ics.jung.visualization.VisualizationViewer;
import graphicalUserInterface.DialogHandlerFrame;
import graphicalUserInterface.Program;

public class NavigationMousePlugin<V extends BrickVertex, E extends BrickEdge>
extends TriadeEditingMousePlugin<V, E> {

	protected Brick editedBrick;

	public NavigationMousePlugin(TreeListener _treeListener,
			Brick _editedSchema, Layout<BrickVertex, BrickEdge> vertexPosition) {
		super((Layout<V, E>) vertexPosition);
		editedBrick = _editedSchema;

		treeListener = _treeListener;
		if (treeListener != null)
			treeListener.setGrapheListener(this);
	}

	@Override
	public void mousePressed(MouseEvent e) {

	}


	@Override
	@SuppressWarnings("unchecked")
	public void mouseReleased(MouseEvent e) {
		if (e.getButton() == MouseEvent.BUTTON3) {
			final VisualizationViewer<V, E> vv = (VisualizationViewer<V, E>) e
			.getSource();
			final Point2D p = e.getPoint();
			GraphElementAccessor<V, E> pickSupport = vv
			.getPickSupport();
			if (pickSupport != null) {

				if (pickSupport.getVertex(vv.getGraphLayout(), p.getX(), p.getY()) != null)
				{
					final Brick brick = (Brick) pickSupport.getVertex(
							vv.getGraphLayout(), p.getX(), p.getY()).getContent();
					if (brick != null) {
						ExportsContextualMenu menu = new ExportsContextualMenu(
								brick);
						menu.show(e.getComponent(), e.getX(), e.getY());
					}
				}
				else
				{

					ExportsContextualMenu menu = new ExportsContextualMenu(editedBrick);
					menu.show(e.getComponent(), e.getX(), e.getY());
				}

			}
		}

		if (checkModifiers(e)) {

			final VisualizationViewer<BrickVertex, BrickEdge> vv = (VisualizationViewer<BrickVertex, BrickEdge>) e
			.getSource();
			final Point2D p = e.getPoint();
			GraphElementAccessor<BrickVertex, BrickEdge> pickSupport = vv
			.getPickSupport();
			if (pickSupport != null) {
				final BrickVertex vertex = pickSupport.getVertex(
						vv.getGraphLayout(), p.getX(), p.getY());
				if (vertex != null) {
					if (e.getClickCount() > 1
							&& e.getButton() == MouseEvent.BUTTON1) {
						Content content = vertex.getContent();
						if (content instanceof Brick) {

							Program.myMainFrame.addTab(content);
						}

					}

				}
			}
			vv.repaint();
		}
	}

	@Override
	public Brick getEditedAbstractSchema() {
		return editedBrick;
	}

	@Override
	public void removeSelectedVertex() {
		DialogHandlerFrame.showErrorDialog(Messages.getString("NavigationMousePlugin.0")); //$NON-NLS-1$
	}

	@Override
	public void selectVertex(Content content) {
		selectVertex((V)editedBrick.getBrickVertexByContent(content));
	}

	@Override
	public void notifyTree(Content content) {
		treeListener.setSelectedContent(content);
	}

}
```

```
    1: package models;
    2:
    3: import java.awt.event.MouseEvent;
    4: import java.awt.geom.Point2D;
    5: import java.util.Iterator;
    6:
    7: import javax.swing.JOptionPane;
    8:
    9: import client.export.ExportsContextualMenu;
   10:
   11: import translation.Messages;
   12:
   13: import dataPack.Activite;
   14: import dataPack.Content;
   15: import dataPack.TreeListener;
   16: import edu.uci.ics.jung.algorithms.layout.GraphElementAccessor;
   17: import edu.uci.ics.jung.algorithms.layout.Layout;
   18: import edu.uci.ics.jung.graph.Graph;
   19: import edu.uci.ics.jung.visualization.VisualizationViewer;
   20: import graphicalUserInterface.DialogHandlerFrame;
   21: import graphicalUserInterface.RankContextualMenu;
   22: import graphicalUserInterface.RelationChooserPopUp;
   23:
   24: public class BrickEditingMousePlugin
   25: extends TriadeEditingMousePlugin<BrickVertex, BrickEdge> {
   26:
   27:         protected final Brick editedBrick;
   28:
   29:         public BrickEditingMousePlugin(Brick _editedBrick,
   30:                         TreeListener _treeListener, Layout<BrickVertex, BrickEdge>
 vertexPosition) {
   31:                 super(vertexPosition);
   32:                 editedBrick = _editedBrick;
   33:                 treeListener = _treeListener;
   34:                 treeListener.setGrapheListener(this);
   35:         }
   36:
   37:         @Override
   38:         public void removeSelectedVertex() {
   39:                 if (selectedVertex != null) {
   40:                         if (selectedVertex.getContent() instanceof Activite) {
   41:                                 DialogHandlerFrame
   42:                                 .showErrorDialog(Messages.getString("BrickEditingM
ousePlugin.0")); //$NON-NLS-1$
   43:                                 return;
   44:                         }
   45:
   46:                         if (DialogHandlerFrame
   47:                                         .showYesNoDialog(Messages.getString("Brick
EditingMousePlugin.1")) == JOptionPane.YES_OPTION) { //$NON-NLS-1$
   48:                                 treeListener.showActor(selectedVertex.getContent()
);
   49:                                 editedBrick.removeBrickVertex(selectedVertex);
   50:                                 selectedVertex = null;
   51:                         }
   52:
   53:                 } else if (selectedEdge != null) {
   54:                         if (DialogHandlerFrame
   55:                                         .showYesNoDialog(Messages.getString("Brick
EditingMousePlugin.2")) == JOptionPane.YES_OPTION) { //$NON-NLS-1$
   56:                                 editedBrick.removeModelEdge(selectedEdge);
   57:                                 selectedEdge = null;
   58:                         }
   59:                 }
   60:         }
   61:
   62:         @SuppressWarnings("unchecked")
   63:         @Override
   64:         public void mousePressed(MouseEvent e) {
   65:                 if (checkModifiers(e)) {
   66:                         final VisualizationViewer<BrickVertex, BrickEdge> vv = (Vi
sualizationViewer<BrickVertex, BrickEdge>) e
   67:                         .getSource();
   68:                         final Point2D transformedPoint = vv.getRenderContext()
   69:                         .getMultiLayerTransformer()
   70:                         .inverseTransform(e.getPoint());
   71:                         final Point2D p = e.getPoint();
   72:                         GraphElementAccessor<BrickVertex, BrickEdge> pickSupport =
 vv
   73:                         .getPickSupport();
   74:                         if (pickSupport != null) {
   75:                                 final BrickVertex vertex = pickSupport.getVertex(v
ertexLocations, p.getX(), p.getY());
   76:
   77:                                 BrickEdge edge = pickSupport.getEdge(vertexLocatio
ns, p.getX(), p.getY());
   78:
   79:                                 if (vertex == null && edge != null)
   80:                                 {
   81:                                         if (((RelationChooserPopUp) modelView.getP
opUp()).showRelationChooserView(edge, this,Messages.getString("BrickEditingMousePlugin.3"
)) != -1) //$NON-NLS-1$
   82:                                                 selectEdge(edge);
   83:
   84:                                 }
   85:                                 else
   86:                                 {
   87:                                         selectEdge(null);
   88:                                 }
   89:
   90:                                 if (vertex != null)
   91:                                 { // get ready to make an edge
   92:                                         if (selectedVertex != vertex)
   93:                                                 selectVertex(vertex);
   94:                                         startVertex = vertex;
   95:                                         down = vv.getRenderContext().getMultiLayer
Transformer().transform(vertex.getLocation());
   96:                                         transformEdgeShape(down, down);
   97:                                         vv.addPostRenderPaintable(edgePaintable);
   98:                                         edgeIsDirected = true;
   99:                                         transformArrowShape(down, e.getPoint());
  100:                                         vv.addPostRenderPaintable(arrowPaintable);
  101:                                         drawArrow = false;
  102:                                 } else if (activeContent != null) {
  103:                                         // On ajoute un sommet au graph en le lian
t Ã  une nouvelle
  104:                                         // brick.
  105:                                         Graph<BrickVertex, BrickEdge> graph = vv.g
etGraphLayout().getGraph();
  106:                                         BrickVertex bV = editedBrick.addBrickVerte
x(activeContent);
  107:                                         bV.setLocation(p);
  108:                                         vertexLocations.setLocation(bV, transforme
dPoint);
  109:                                         bV.setLocation(transformedPoint);
  110:
  111:                                         Content oldActiveContent = activeContent;
  112:                                         activeContent = null;
  113:                                         treeListener.hideActor(oldActiveContent);
  114:                                         selectVertex(bV);
  115:
  116:
  117:                                         for (Iterator<BrickVertex> iterator = grap
```

```
h.getVertices()
                                                        .iterator(); iterator
                                                        .hasNext();) {
                                                vertexLocations.lock(iterator.next
(), true);
                                        }
                                        graph.addVertex(bV);
                                        for (Iterator<BrickVertex> iterator = grap
h.getVertices()
                                                        .iterator(); iterator
                                                        .hasNext();) {
                                                vertexLocations.lock(iterator.next
(), false);
                                        }
                                }
                                else
                                {
                                        selectVertex((BrickVertex)null);
                                }
                        }
                        vv.repaint();
                }
        }

        /**
         * If startVertex is non-null, and the mouse is released over an existing
         * vertex, create an undirected edge from startVertex to the vertex under
         * the mouse pointer. If shift was also pressed, create a directed edge
         * instead.
         */
        @SuppressWarnings("unchecked")
        @Override
        public void mouseReleased(MouseEvent e) {
                if (checkModifiers(e)) {
                        final VisualizationViewer<BrickVertex, BrickEdge> vv = (Vi
sualizationViewer<BrickVertex, BrickEdge>) e
                        .getSource();
                        final Point2D p = e.getPoint();
                        GraphElementAccessor<BrickVertex, BrickEdge> pickSupport =
 vv
                        .getPickSupport();
                        if (pickSupport != null) {
                                final BrickVertex vertex = pickSupport.getVertex(
                                                vertexLocations, p.getX(), p.getY(
));


                                if (vertex != null && startVertex != null) {
                                        if (vertex != startVertex) {

                                                if (vv.getGraphLayout().getGraph()
.isSuccessor(
                                                                startVertex, verte
x)) {

                                                // l'arrete existe deja, o
n l'Ã©dite
                                                BrickEdge mE = vv.getGraph
Layout().getGraph().findEdge(startVertex, vertex);
                                                if (((RelationChooserPopUp
) modelView
                                                                .getPopUp(
))
                                                                .showRelat
ionChooserView(mE, this,
Messages.getString("BrickEditingMousePlugin.4")) != -1) //$NON-NLS-1$
                                                                selectEdge(mE);


                                                } else {
                                                // Ajout d'une nouvelle ar
rÃªte

                                                Graph graph = vv.getGraphL
ayout().getGraph();
                                                BrickVertex source = start
Vertex;
                                                BrickVertex destination =
vertex;
                                                BrickEdge mEdge = editedBr
ick
                                                .addEdge(new BrickEdge(sou
rce, destination));


                                                startVertex = null;
                                                down = null;


                                                boolean ok = false;
                                                if (((RelationChooserPopUp
) modelView.getPopUp())
                                                                .showRelat
ionChooserView(mEdge, this,
Messages.getString("BrickEditingMousePlugin.5")) != -1) //$NON-NLS-1$
                                                                ok = true;


                                                if (ok) {

                                                        graph.addEdge(mEdg
e, source, destination);
                                                        selectEdge(mEdge);
                                                }

                                                startVertex = null;
                                                down = null;
                                                edgeIsDirected = false;
                                                vv.removePostRenderPaintab
le(edgePaintable);
                                                vv.removePostRenderPaintab
le(arrowPaintable);
                                                return;

                                                }

                                        } else {

                                                if (activeContent != null) {
                                                // l'utilisateur veut remp
lacer le contenu d'un
                                                // sommet
                                                if (vertex.getContent() in
stanceof Activite)
                                                {
                                                        DialogHandlerFrame
.showErrorDialog(Messages.getString("BrickEditingMousePlugin.6")); //$NON-NLS-1$

                                                }
                                                else if (JOptionPane
                                                                .showConfi
```

```
rmDialog(null,
  222:
Messages.getString("BrickEditingMousePlugin.7")) == JOptionPane.YES_OPTION) { //$NON-NLS-
1$
  223:                                                                    treeListener.showA
ctor(vertex
  224:                                                                                    .g
etContent());
  225:                                                                    vertex.setContent(
activeContent);
  226:                                                                    Content oldContent
 = activeContent;
  227:                                                                    activeContent = nu
ll;
  228:                                                                    treeListener.hideA
ctor(oldContent);
  229:                                                            }
  230:                                                    }
  231:
  232:                                            }
  233:                                    }
  234:                                    vv.repaint();
  235:                            }
  236:
  237:                            startVertex = null;
  238:                            down = null;
  239:                            edgeIsDirected = false;
  240:                            vv.removePostRenderPaintable(edgePaintable);
  241:                            vv.removePostRenderPaintable(arrowPaintable);
  242:                    }
  243:
  244:                    if (e.getButton() == MouseEvent.BUTTON3) {
  245:                            final VisualizationViewer<BrickVertex, BrickEdge> vv = (Vi
sualizationViewer<BrickVertex, BrickEdge>) e
  246:                            .getSource();
  247:                            final Point2D p = e.getPoint();
  248:                            GraphElementAccessor<BrickVertex, BrickEdge> pickSupport =
 vv
  249:                            .getPickSupport();
  250:                            if (pickSupport != null) {
  251:                                    final BrickVertex vertex = pickSupport.getVertex(
  252:                                                    vertexLocations, p.getX(), p.getY(
));
  253:                                    if (vertex != null)
  254:                                    {
  255:                                            if (! (vertex.getContent() instanceof Acti
vite))
  256:                                            {
  257:                                                    RankContextualMenu menu = new Rank
ContextualMenu(vertex, editedBrick);
  258:                                                    menu.show(e.getComponent(), e.getX
(), e.getY());
  259:                                            }
  260:                                    }
  261:                            }
  262:
  263:                    }
  264:
  265:            }
  266:
  267:            @Override
  268:            public Brick getEditedAbstractSchema() {
  269:                    return editedBrick;
  270:            }
  271:
  272:            @Override
  273:            public void selectVertex(Content content) {
  274:                    selectVertex(editedBrick.getBrickVertexByContent(content));
  275:            }
  276:
  277:            @Override
  278:            public void notifyTree(Content content) {
  279:
  280:            }
  281: }
```

```
   1: package models;
   2:
   3: import java.io.Serializable;
   4: import java.util.Iterator;
   5: import java.util.Vector;
   6:
   7: import dataPack.*;
   8: import edu.uci.ics.jung.graph.Edge;
   9: import edu.uci.ics.jung.graph.Graph;
  10: import edu.uci.ics.jung.graph.Vertex;
  11:
  12: public class Model implements Serializable, AbstractSchema{
  13:
  14:         /**
  15:          *
  16:          */
  17:         private static final long serialVersionUID = 4211763759080896241L;
  18:
  19:         protected final DataPack dataPack;
  20:         protected Vector<ModelEdge> edges;
  21:         protected String name;
  22:         protected Vector<ModelVertex> vertices;
  23:         protected transient Graph graph;
  24:
  25:         public Model(DataPack _dataPack, String _nom)
  26:         {
  27:                 dataPack = _dataPack;
  28:                 edges = new Vector<ModelEdge>();
  29:                 vertices = new Vector<ModelVertex>();
  30:                 name = _nom;
  31:         }
  32:
  33:         public ModelVertex addModelVertex(Integer actorId,Vertex vertex)
  34:         {
  35:                 ModelVertex result = new ModelVertex(new Integer(getIdMax().intVal
ue()+1),actorId);
  36:                 result.setAssociatedVertex(vertex);
  37:                 vertices.add(result);
  38:                 return result;
  39:         }
  40:
  41:         public ModelEdge addModelEdge(Integer source, Integer destination, Edge as
sociatedEdge)
  42:         {
  43:                 ModelEdge result = new ModelEdge(source,destination);
  44:                 edges.add(result);
  45:                 result.setAssociatedEdge(associatedEdge);
  46:                 return result;
  47:         }
  48:
  49:         public synchronized void removeModelEdge(ModelEdge modelEdge)
  50:         {
  51:                 if (edges.contains(modelEdge))
  52:                 {
  53:                         edges.remove(modelEdge);
  54:                         Graph graph = (Graph) modelEdge.getAssociatedEdge().getGra
ph();
  55:                         if (graph != null)
  56:                                 graph.removeEdge(modelEdge.getAssociatedEdge());
  57:                 }
  58:         }
  59:
  60:         public synchronized void removeModelVertex(ModelVertex modelVertex)
  61:         {
  62:                 if (vertices.contains(modelVertex))
  63:                 {
  64:                         System.out.println("removeModelVertex");
  65:
  66:                         vertices.remove(modelVertex);
  67:
  68: //                      Enumeration<Edge> iterator = modelVertex.getAssociatedVert
ex().getInEdges().iterator();
  69: //                      while (modelVertex.getAssociatedVertex().getInEdges().size
() < 0)
  70: //                      {
  71: //
  72: //                              //Edge edge = iterator.;
  73: //                              ModelEdge modelEdge = (ModelEdge)edge.getUserDatum
(ModelEdge.class);
  74: //                              removeModelEdge(modelEdge);
  75: //                      }
  76: //
  77:                         if (modelVertex.getAssociatedVertex() != null)
  78:                         {
  79:                         for(Object edgeObj:modelVertex.getAssociatedVertex().getIn
Edges())
  80:                         {
  81:                                 Edge edge = (Edge)edgeObj;
  82:                                 ModelEdge modelEdge = (ModelEdge)edge.getUserDatum
(ModelEdge.class);
  83:                                 removeModelEdge(modelEdge);
  84:                         }
  85:                         for(Object edgeObj:modelVertex.getAssociatedVertex().getOu
tEdges())
  86:                         {
  87:                                 Edge edge = (Edge)edgeObj;
  88:                                 ModelEdge modelEdge = (ModelEdge)edge.getUserDatum
(ModelEdge.class);
  89:                                 removeModelEdge(modelEdge);
  90:                         }
  91:                         graph.removeVertex(modelVertex.associatedVertex);
  92:                         }
  93:                         else
  94:                         {
  95:                                 for (ModelEdge mE: edges)
  96:                                 {
  97:                                         if (mE.source.equals(modelVertex.vertexId)
 || mE.destination.equals(modelVertex.vertexId))
  98:                                                                 removeModelEdge(mE);
  99:                                 }
 100:
 101:                         }
 102:                 }
 103:         }
 104:
 105:         public void removeActor(Integer content)
 106:         {
 107:                 //Iterator<ModelVertex> iterator = vertices.iterator();
 108:
 109:                 for (int i = 0; i< vertices.size();)
 110:                 {
 111:                         if (vertices.elementAt(i).contentId.intValue() == content.
intValue())
 112:                         {
 113:                                 removeModelVertex(vertices.elementAt(i));
 114:                         }
 115:                         else
 116:                         {
 117:                                 i++;
 118:                         }
 119:                 }
 120:         }
 121:
 122:
```

```
  123:         protected Integer getIdMax()
  124:         {
  125:                 Integer result = new Integer(-1);
  126:                 for (ModelVertex vertex:vertices)
  127:                 {
  128:                         if (vertex.vertexId > result)
  129:                                 result = vertex.vertexId;
  130:
  131:                 }
  132:
  133:                 return result;
  134:         }
  135:
  136:         public ModelVertex getModelVertex(Integer id)
  137:         {
  138:
  139:                 if (id.intValue() < vertices.size() && vertices.elementAt(id.intVa
lue()).vertexId == id)
  140:                         return vertices.elementAt(id.intValue());
  141:                 else
  142:                         for (ModelVertex vertex:vertices)
  143:                                 if (vertex.vertexId == id)
  144:                                         return vertex;
  145:
  146:                 System.err.println("Aucun sommet trouvÃ© correspondant Ã  l'id fou
rni (Model.getModelVertex avec l'id:"+id+")");
  147:                 return null;
  148:         }
  149:
  150:         public DataPack getDataPack() {
  151:                 return dataPack;
  152:         }
  153:
  154:         public Vector<ModelVertex> getVertices() {
  155:                 return vertices;
  156:         }
  157:
  158:         public String toString()
  159:         {
  160:                 return name;
  161:         }
  162:
  163:         @Override
  164:         public void removeElement(Object element) {
  165:                 if (element.getClass() == ModelVertex.class)
  166:                         removeModelVertex((ModelVertex)element);
  167:                 else
  168:                         System.err.println("Demande de suppression d'un objet qui
n'est pas un ModelVertex dans un Model");
  169:
  170:         }
  171:
  172:         public Vector<ModelEdge> getEdges() {
  173:                 return edges;
  174:         }
  175:
  176:         public ModelVertex getVertexByContent(Integer contentId)
  177:         {
  178:                 for (ModelVertex vertex:vertices)
  179:                 {
  180:                         if (vertex.contentId.equals(contentId))
  181:                                 return vertex;
  182:                 }
  183:                 return null;
  184:         }
  185:
  186:         public String getName() {
  187:
  188:                 return name;
  189:         }
  190: }
```

```
    1: package models;
    2:
    3: import java.awt.event.InputEvent;
    4:
    5: import edu.uci.ics.jung.visualization.RenderContext;
    6: import edu.uci.ics.jung.visualization.control.CrossoverScalingControl;
    7: import edu.uci.ics.jung.visualization.control.EditingModalGraphMouse;
    8: import edu.uci.ics.jung.visualization.control.ScalingGraphMousePlugin;
    9: import edu.uci.ics.jung.visualization.control.TranslatingGraphMousePlugin;
   10:
   11: public class ModelsEditingModalGraphMouse<V extends BrickVertex, E extends BrickEd
ge>
   12:                 extends EditingModalGraphMouse<V, E> {
   13:
   14:         protected TriadeEditingMousePlugin<V, E> mousePlugin;
   15:
   16:         protected ModelsEditingModalGraphMouse(
   17:                         TriadeEditingMousePlugin<V, E> _mousePlugin,
   18:                         RenderContext<V, E> renderContext) {
   19:                 super(renderContext, null, null);
   20:                 in = 1.1f;
   21:                 out = 1 / 1.1f;
   22:                 mousePlugin = _mousePlugin;
   23:                 loadPlugins();
   24:
   25:         }
   26:
   27:         @Override
   28:         protected void loadPlugins() {
   29:                 pickingPlugin = new MyPickingEditingMousePlugin<BrickVertex, Brick
Edge>(
   30:                                 InputEvent.BUTTON1_MASK
   31:                                 | InputEvent.SHIFT_MASK, 0);
   32:                 translatingPlugin = new TranslatingGraphMousePlugin(
   33:                                 InputEvent.BUTTON1_MASK | InputEvent.CTRL_MASK);
   34:                 scalingPlugin = new ScalingGraphMousePlugin(
   35:                                 new CrossoverScalingControl(), 0, in, out);
   36:                 editingPlugin = mousePlugin;
   37:
   38:                 setMode(Mode.EDITING);
   39:         }
   40:
   41:         @Override
   42:         protected void setEditingMode() {
   43:                 add(pickingPlugin);
   44:                 remove(animatedPickingPlugin);
   45:                 add(translatingPlugin);
   46:                 remove(rotatingPlugin);
   47:                 remove(shearingPlugin);
   48:                 add(editingPlugin);
   49:         }
   50:
   51: }
```

```
    1: package models;
    2:
    3: import java.awt.BorderLayout;
    4: import java.awt.Color;
    5: import java.awt.Component;
    6: import java.awt.Dimension;
    7: import java.awt.Font;
    8: import java.awt.Paint;
    9: import java.awt.Rectangle;
   10: import java.awt.event.MouseEvent;
   11: import java.util.Iterator;
   12:
   13: import javax.swing.BorderFactory;
   14: import javax.swing.Icon;
   15: import javax.swing.JComponent;
   16: import javax.swing.JLabel;
   17: import javax.swing.JPanel;
   18:
   19: import org.apache.commons.collections15.Predicate;
   20: import org.apache.commons.collections15.Transformer;
   21:
   22:
   23: import client.export.ExportingMousePlugin;
   24: import dataPack.DataPack;
   25: import dataPack.TreeListener;
   26: import edu.uci.ics.jung.algorithms.layout.Layout;
   27: import edu.uci.ics.jung.algorithms.layout.StaticLayout;
   28: import edu.uci.ics.jung.graph.Graph;
   29: import edu.uci.ics.jung.graph.SparseGraph;
   30: import edu.uci.ics.jung.graph.util.Context;
   31: import edu.uci.ics.jung.visualization.PluggableRenderContext;
   32: import edu.uci.ics.jung.visualization.RenderContext;
   33: import edu.uci.ics.jung.visualization.VisualizationViewer;
   34: import edu.uci.ics.jung.visualization.control.EditingModalGraphMouse;
   35: import edu.uci.ics.jung.visualization.decorators.DirectionalEdgeArrowTransformer;
   36: import edu.uci.ics.jung.visualization.renderers.VertexLabelRenderer;
   37: import graphicalUserInterface.IconDatabase;
   38: import graphicalUserInterface.PopUpView;
   39: import graphicalUserInterface.Program;
   40:
   41: public class SchemaView extends JPanel {
   42:
   43:         /**
   44:          *
   45:          */
   46:         private static final long serialVersionUID = 4352715056596473408L;
   47:
   48:         private static final int NOMODE = -1; // mode non initialiser, tout est
   49:
// affichÃ©
   50:         private static final int STARMODE = 0; // affiche seulement les acteur
   51:
// voisin d'un acteur dÃ©fini par
   52:
// startTargetId
   53:         private static final int CONFLICTMODE = 1; // affiche seulement les lieu o
u
   54:
        // il y a conflit (+
   55:
        // possiblement l'Ã©toile de
   56:
        // cette acteur
   57:         private static final int ACTIONTIMEMODE = 2; // affiche seulement les
   58:
                 // element actif a un
   59:
                  // certain temps d'action
   60:         private static final int FREEMODE = 3; // l'utilisateur est libre de faire
   61:
// ce qu'il veut.
   62:
   63:         private int mode;
   64:
   65:         private Integer starTargetId; // id de l'element centre de l'etoile
   66:         private Integer actionTime; // temps
   67:
   68:         protected SparseGraph<SchemaVertex, BrickEdge> graph;
   69:         protected VisualizationViewer<SchemaVertex, BrickEdge> vv;
   70:         protected edu.uci.ics.jung.algorithms.layout.StaticLayout<SchemaVertex, Br
ickEdge> vertexLocations;
   71:
   72:         protected TriadeEditingMousePlugin mousePlugin;
   73:         protected final EditingModalGraphMouse<SchemaVertex, BrickEdge> graphMouse
;
   74:         protected PopUpView popUp;
   75:
   76:         protected Schema shema;
   77:
   78:         // protected ModelExport modelExport; // modification a effectuer sur les
   79:         // element visible du shema
   80:
   81:         @SuppressWarnings("unchecked")
   82:         private SchemaView(TriadeEditingMousePlugin _mousePlugin, PopUpView pUV,
   83:                         final DataPack datapack) {
   84:                 mode = NOMODE;
   85:
   86:                 graph = new SparseGraph<SchemaVertex, BrickEdge>();
   87:                 popUp = pUV;
   88:                 vertexLocations = new StaticLayout<SchemaVertex, BrickEdge>(graph)
;
   89:
   90:                 vv = new VisualizationViewer<SchemaVertex, BrickEdge>(vertexLocati
ons);
   91:
   92:                 PluggableRenderContext<SchemaVertex, BrickEdge> pr = (PluggableRen
derContext<SchemaVertex, BrickEdge>) vv
   93:                                 .getRenderer();
   94:                 // pr.setVertexLabelCentering(true);
   95:                 pr.setEdgeArrowTransformer(new DirectionalEdgeArrowTransformer(14,
 20,
   96:                                 7));
   97:                 pr.setVertexLabelRenderer(new VertexLabelRenderer() {
   98:
   99:                         boolean rotatedEdge = false;
  100:
  101:                         public <T> Component getVertexLabelRendererComponent(
  102:                                         JComponent vv,
  103:                                         Object value, Font font, boolean isSelecte
d,
  104:  T vertex) {
  105:
  106:                                 JLabel result = new JLabel();
  107:                                 result.setFont(font);
  108:                                 result.setText((String) value);
  109:                                 result.setBorder(BorderFactory.createEmptyBorder(5
5, 0, 0, 0));
  110:
  111:                                 if (vertex instanceof SchemaVertex
  112:                                                 && ((SchemaVertex) vertex).isSelec
ted()) {
  113:
  114:                                         result.setForeground(Color.blue);
  115:                                 }
```

```
  116:
  117:                                    return result;
  118:                            }
  119:
  120:
  121:                    });
  122:                    pr.setVertexFontTransformer(new Transformer<SchemaVertex, Font>()
{
  123:
  124:                            @Override
  125:                            public Font transform(SchemaVertex v) {
  126:
  127:                                    if (v.isSelected()) {
  128:                                            return new Font("Helvetica", Font.
ITALIC, 13);
  129:                                    } else
  130:                                            new Font("Helvetica", Font.PLAIN,
12);
  131:
  132:                            }
  133:
  134:                    });
  135:
  136:                    pr.setEdgeDrawPaintTransformer(new Transformer<BrickEdge, Paint>()
 {
  137:
  138:                            @Override
  139:                            public Paint transform(BrickEdge mE) {
  140:                                    if (mE != null
  141:                                            && (mE.source.intValue() < 0 && mE
.source.intValue() > -50)
  142:                                            || (mE.destination.intValue() < 0
&& mE.destination
  143:                                                            .intValue() > -50)
)
  144:                                    if (mE.isSelected())
  145:                                            return Color.cyan;
  146:                                    else
  147:                                            return Color.blue;
  148:
  149:                                    if (mE.isSelected())
  150:                                            return Color.green;
  151:                                    return Color.black;
  152:                            }
  153:                    });
  154:                    // pr.setEdgeFillPaintTransformer(new Transformer<BrickEdge, Paint
>() {
  155:                    //
  156:                    // @Override
  157:                    // public Paint transform(BrickEdge mE) {
  158:                    // return pr.
  159:                    // }
  160:                    // Voir si il faut mettre quelque chose ici
  161:                    // });
  162:
  163:                    pr.setVertexIconTransformer(new Transformer<SchemaVertex, Icon>()
{
  164:
  165:
  166:                            @Override
  167:                            public Icon transform(SchemaVertex e) {
  168:
  169:                                    Integer id = e.getVertexId(); // <=  Attention ! i
l faut utiliser contentId
  170:                                    boolean selected = e.isSelected();
  171:                                    if (id != null) {
  172:                                            int i = id.intValue();
  173:                                            if (i <= -50) {
  174:                                                    return IconDatabase.vectorIconActi
vities
  175:                                                                    .elementAt(datapac
k.getActivities()
  176:                                                                    .g
etActivite(id).getIconId()
  177:                                                                    +
(selected?1:0));
  178:                                                    // if (selected != null && selecte
d.intValue() == 1)
  179:                                                    // return IconDatabase.iconActivit
ySelected;
  180:                                                    // return IconDatabase.iconActivit
y;
  181:                                            }
  182:
  183:                                            if ((i <= -10 && i > -50) || i == -1 || i
== -2) {
  184:                                                    if (selected)
  185:                                                            return IconDatabase.iconAr
rowOutSelected;
  186:                                                    return IconDatabase.iconArrowOut;
  187:                                            }
  188:
  189:                                    }
  190:
  191:                                    if (selected )
  192:                                            return IconDatabase.iconHumanActorSelected
;
  193:                                    return IconDatabase.iconHumanActor;
  194:                            }
  195:                    });
  196:
  197:                    pr.setVertexDrawPaintTransformer(new Transformer<SchemaVertex, Pai
nt>() {
  198:                            @Override
  199:                            public Paint transform(SchemaVertex v) {
  200:                                    return Color.black;
  201:                            }
  202:
  203:                            });
  204:
  205:
  206: pr
  207:                            .setVertexFillPaintTransformer(new Transformer<Sch
emaVertex, Paint>() {
  208:
  209:                                    @Override
  210:                                    public Paint transform(SchemaVertex v) {
  211:                                            Integer id = v.getVertexId();
  212:                                    if (id != null) {
  213:                                            int i = id.intValue();
  214:                                            if (i <= -50)
  215:                                                    return Color.yellow;
  216:
  217:                                            if (i <= -10 && i > -50)
  218:                                                    return Color.blue;
  219:
  220:                                            if (i == -1 || i == -2)
  221:                                                    return Color.cyan;
  222:                                    }
  223:
  224:                                            if (v.isSelected())
  225:                                    return Color.white;
  226:
  227:                                    return Color.red;
```

```
228:                         }
229:
230:                 });
231:
232:                 pr.setVertexIncludePredicate(new Predicate<Context<Graph<SchemaVer
tex, BrickEdge>, SchemaVertex>>() {
233:
234:                         @Override
235:                         public boolean evaluate(
236:                                         Context<Graph<SchemaVertex, BrickEdge>, Sc
hemaVertex> arg0) {
237:                                 if (arg0 == null)
238:                                         return false;
239:
240:                                 if (arg0.getClass() == DirectedSparseEdge.class) {
241:                                         DirectedSparseEdge edge = (DirectedSparseE
dge) arg0;
242:
243:                                         Integer idEdge = (Integer) edge.getUserDat
um("id");
244:
245:                                         if (idEdge == null)
246:                                                 return false;
247:                                 return false;
248:                         }
249:
250:                 });
251:
252:                 this.graphLayout = new StaticLayout(graph);
253:                 graphLayout.initialize(new Dimension(600, 600), vertexLocations);
254:
255:
256:                 vv.setBounds(new Rectangle(-300, -300, 600, 600));
257:                 vv.setBackground(Color.white);
258:                 vv.setPickSupport(new ShapePickSupport(50));
259:                 pr.setVertexShapeFunction(new EllipseVertexShapeFunction(
260:                                 new ConstantVertexSizeFunction(60),
261:                                 new ConstantVertexAspectRatioFunction(1.0f)));
262:                 pr.setEdgeShapeFunction(new EdgeShape.QuadCurve());
263:                 pr.setVertexStringer(new VertexStringer() {
264:
265:                         public String getLabel(ArchetypeVertex v) {
266:
267:                                 String result = (String) v.getUserDatum("nom");
268:                                 if (result == null) {
269:                                         return "Sommet non nommÃ©";
270:                                 }
271:                                 return result;
272:                         }
273:                 });
274:
275:                 vv.setToolTipFunction(new ToolTipFunction() {
276:
277:                         public String getToolTipText(Vertex v) {
278:
279:                                 BrickVertex bV = (BrickVertex) v
280:                                                 .getUserDatum(BrickVertex.class);
281:                                 if (bV != null)
282:                                         return bV.getStringDescription();
283:
284:                                 Integer id = (Integer) v.getUserDatum("id");
285:                                 if (id != null) {
286:                                         if (id.intValue() > -50)
287:                                                 return "Sommet virtuel permettant
de reprÃ©senter\n des relations sortant de la brique";
288:                                         else {
289:                                                 DataPack dataPack = (DataPack) gra
ph
290:                                                                 .getUserDatum(Data
Pack.class);
291:                                                 if (dataPack != null)
292:                                                         return "Sommet reprÃ©senta
nt l'activitÃ© centrale de la brique : "
293:                                                                 + dataPack
.getStringById(id);
294:                                                 else
295:                                                         return "Sommet reprÃ©senta
nt l'activitÃ© centrale de la brique\n";
296:
297:                                         }
298:                                 }
299:                                 return "Sommet Ã©trange";
300:                         }
301:
302:                         public String getToolTipText(Edge e) {
303:                                 Object obj = e.getUserDatum("nom");
304:                                 if (obj != null && obj.getClass() == String.class)
 {
305:                                         return (String) obj;
306:                                 }
307:                                 return "ArÃªte";
308:                         }
309:
310:                         @Override
311:                         public String getToolTipText(MouseEvent event) {
312:                                 return ((JComponent) event.getSource()).getToolTip
Text();
313:                         }
314:                 });
315:
316:                 vv.addMouseListener(_mousePlugin);
317:                 vv.addMouseMotionListener(_mousePlugin);
318:                 setLayout(new BorderLayout());
319:
320:                 this.add(vv, BorderLayout.CENTER);
321:                 mousePlugin = _mousePlugin;
322:                 // mousePlugin.setModelView(this);
323:                 graphMouse = new ModelsEditingModalGraphMouse(mousePlugin);
324:
325:                 // the EditingGraphMouse will pass mouse event coordinates to the
326:                 // vertexLocations function to set the locations of the vertices a
s
327:                 // they are created
328:                 graphMouse.setVertexLocations(vertexLocations);
329:                 vv.setGraphMouse(graphMouse);
330:                 // graphMouse.add(new EditingPopupGraphMousePlugin(vertexLocations
));
331:
332:         }
333:
334:         // public SchemaView(Schema schema, TreeListener _treeListener, PopUpView
335:         // pUV) {
336:         // this(new ExportingMousePlugin(_treeListener, schema), pUV,
337:         // Program.myMainFrame.getDataPack());
338:         //
339:         // this.shema = schema;
340:         //
341:         // // ajoute tout les element du temps d'action au shema.
342:         // // il sont tous visible et peuvent Ã©tre manipuler par la suite.
343:         //
344:         // Iterator<SchemaVertex> vertexIterator = shema.getVertices().iterator();
345:         //
346:         // while (vertexIterator.hasNext()) {
347:         // SchemaVertex curent = vertexIterator.next();
```

```
348:         //
349:         // Vertex curentVertex = new SimpleSparseVertex();
350:         // vertexLocations.setLocation(curentVertex, curent.getLocation());
351:         // curentVertex.addUserDatum("id", curent.getVertexId(),
352:         // new UserDataContainer.CopyAction.Clone());
353:         // curentVertex.addUserDatum("contentId", curent.getContent(),
354:         // new UserDataContainer.CopyAction.Clone());
355:         // curentVertex.addUserDatum("selection", new Integer(0),
356:         // new UserDataContainer.CopyAction.Clone());
357:         // curentVertex.addUserDatum(SchemaVertex.class, curent,
358:         // new UserDataContainer.CopyAction.Shared());
359:         //
360:         // String nom = null;
361:         // if (curent.getContent() <= -100)
362:         // // activite
363:         // nom = new String(schema.dataPack.getActivite(
364:         // curent.getContent()).toString());
365:         // else if (curent.getContent() >= 0)
366:         // // acteur
367:         // nom = new String(schema.dataPack.getActorsModel().getActorName(
368:         // curent.getContent()));
369:         // else if (curent.getContent() <= -50)
370:         // // moyen
371:         // nom = new String(shema.dataPack.getMoyen(curent.getContent())
372:         // .toString());
373:         // else
374:         // nom = new String("Contenue de l'indice bizarre");
375:         //
376:         // curentVertex.addUserDatum("nom", nom,
377:         // new UserDataContainer.CopyAction.Clone());
378:         //
379:         // curent.setAssociatedVertex(curentVertex);
380:         //
381:         // graph.addVertex(curentVertex);
382:         // }
383:         //
384:         // Iterator<BrickEdge> edgeIterator = shema.getEdges().iterator();
385:         //
386:         // while (edgeIterator.hasNext()) {
387:         // BrickEdge mEdge = edgeIterator.next();
388:         //
389:         // Vertex src = schema.getSchemaVertex(mEdge.source)
390:         // .getAssociatedVertex();
391:         // Vertex dst = schema.getSchemaVertex(mEdge.destination)
392:         // .getAssociatedVertex();
393:         //
394:         // DirectedSparseEdge edge = new DirectedSparseEdge(src, dst);
395:         //
396:         // mEdge.setAssociatedEdge(edge);
397:         //
398:         // edge.addUserDatum(BrickEdge.class, mEdge,
399:         // new UserDataContainer.CopyAction.Shared());
400:         // edge.addUserDatum("selection", new Integer(0),
401:         // new UserDataContainer.CopyAction.Clone());
402:         //
403:         // graph.addEdge(edge);
404:         // }
405:         //
406:         // mode = NOMODE;
407:         // }
408:
409:         public void setStarMode(Integer idActor, Integer actionTime) {
410:
411:                 starTargetId = idActor;
412:
413:                 mode = STARMODE;
414:         }
415:
416:         public TriadeEditingMousePlugin getMousePlugin() {
417:                 return mousePlugin;
418:         }
419:
420:         public PopUpView getPopUp() {
421:                 return popUp;
422:         }
423:
424: }
```

```
1: package models;
2:
3: import dataPack.Content;
4:
5: public interface GraphListener {
6:
7:         public void setContent(Content newContent);
8:
9: }
```

```
 1: package models;
 2:
 3: import java.awt.geom.Point2D;
 4: import java.util.Vector;
 5:
 6: import main.RelationComplete;
 7: import dataPack.ActeurSelectionne;
 8: import dataPack.Content;
 9: import dataPack.DataPack;
10: import dataPack.SavableObject;
11: import edu.uci.ics.jung.graph.SparseGraph;
12:
13: public class Schema implements AbstractSchema, SavableObject {
14:
15:         /**
16:          *
17:          */
18:         private static final long serialVersionUID = 56L;
19:         protected final DataPack dataPack;
20:         protected Vector<BrickEdge> edges;
21:         protected String nom;
22:         protected Vector<BrickVertex> vertices;
23:         protected Brick associatedBrick;
24:         protected transient SparseGraph<BrickVertex, BrickEdge> graph;
25:         protected transient String path;
26:         protected Integer sessionId;
27:
28:         /**
29:          * Indique si le schÃ©ma reprÃ©sente une Ã©tape globale (vrai) ou si il
30:          * reprÃ©sente une brique simple (faux)
31:          */
32:         protected boolean mainSchema;
33:
34:         public Schema(DataPack _dataPack, String _nom, boolean isMainSchema,
35:                         Brick associatedBrick, Integer sessionId) {
36:                 dataPack = _dataPack;
37:                 edges = new Vector<BrickEdge>();
38:                 vertices = new Vector<BrickVertex>();
39:                 nom = _nom;
40:                 graph = null;
41:                 path = null;
42:                 mainSchema = isMainSchema;
43:                 this.associatedBrick = associatedBrick;
44:                 this.sessionId = sessionId;
45:         }
46:
47:         public Schema(Schema schema) {
48:                 dataPack = schema.dataPack;
49:                 edges = new Vector<BrickEdge>(schema.edges);
50:                 vertices = new Vector<BrickVertex>(schema.vertices);
51:                 nom = new String(schema.nom);
52:                 graph = schema.graph;
53:                 path = null;
54:                 mainSchema = schema.mainSchema;
55:                 associatedBrick = schema.associatedBrick;
56:                 this.sessionId = schema.sessionId;
57:         }
58:
59:         public BrickVertex addSchemaVertex(Content content) {
60:                 BrickVertex result = new BrickVertex(new Integer(
61:                                 getIdMax()
62:                                 .intValue() + 1), content);
63:                 result.setLocation(new Point2D.Double());
64:                 vertices.add(result);
65:                 return result;
66:         }
67:
68:         public BrickEdge addModelEdge(BrickVertex source, BrickVertex destination,
69:                         RelationComplete relationComplete) {
70:                 BrickEdge result = new BrickEdge(source, destination, relationComp
lete);
71:                 edges.add(result);
72:                 return result;
73:         }
74:
75:         public void addModelEdge(BrickEdge newEdge) {
76:                 newEdge.setAssociatedEdge(null);
77:                 edges.add(newEdge);
78:
79:         }
80:
81:         protected Integer getIdMax() {
82:                 Integer result = new Integer(-1);
83:                 for (BrickVertex vertex : vertices) {
84:                         if (vertex.vertexId > result)
85:                                 result = vertex.vertexId;
86:
87:                 }
88:
89:                 return result;
90:         }
91:
92:         public BrickVertex getSchemaVertex(Integer id) {
93:                 for (BrickVertex vertex : vertices) {
94:                         if (vertex.getVertexId().equals(id)) {
95:                                 return vertex;
96:                         }
97:                 }
98:                 System.out
99:                                 .println("Aucun sommet trouvÃ© correspondant Ã  l'
id fourni (Schema.getSchemaVertex avec l'id:"
100:                                                 + id + ")");
101:                 System.out.println(vertices);
102:                 return null;
103:         }
104:
105:         protected BrickVertex getSchemaVertexByContent(Integer id) {
106:
107:                 // if (vertices.elementAt(id.intValue()).vertexId.equals(id))
108:                 // return vertices.elementAt(id.intValue());
109:                 // else
110:                 for (BrickVertex vertex : vertices)
111:                         if (vertex.associatedContent.equals(id))
112:                                 return vertex;
113:
114:                 System.err
115:                                 .println("Aucun sommet trouvÃ© correspondant au co
ntent fourni (Schema.getSchemaVertexByContent avec l'id:"
116:                                                 + id + ")");
117:                 System.out.println(vertices);
118:                 return null;
119:         }
120:
121:         public BrickVertex getVertexByActor(ActeurSelectionne selectedActor) {
122:                 for (BrickVertex vertex : vertices) {
123:                         if (selectedActor.getIdActeur().equals(vertex.getContent()
))
124:                                 return vertex;
125:                 }
126:                 return null;
127:         }
128:
129:         public Vector<BrickVertex> getVertices() {
130:                 return vertices;
```

```
131:         }                                                                  198: }
132:
133:         public void setVertices(Vector<BrickVertex> vertices) {
134:                 this.vertices = vertices;
135:         }
136:
137:         public Vector<BrickEdge> getEdges() {
138:                 return edges;
139:         }
140:
141:         public DataPack getDataPack() {
142:                 return dataPack;
143:         }
144:
145:         @Override
146:         public void removeElement(Object Element) {
147:
148:         }
149:
150:         @Override
151:         public void removeModelEdge(BrickEdge edge) {
152:
153:         }
154:
155:         public String getFilePath() {
156:                 return path;
157:         }
158:
159:         @Override
160:         public void setFilePath(String _path) {
161:                 path = _path;
162:
163:         }
164:
165:         public boolean isMainSchema() {
166:                 return mainSchema;
167:         }
168:
169:         public void setMainSchema(boolean mainSchema) {
170:                 this.mainSchema = mainSchema;
171:         }
172:
173:         public Brick getAssociatedBrick() {
174:                 return associatedBrick;
175:         }
176:
177:         public void setAssociatedBrick(Brick associatedBrick) {
178:                 this.associatedBrick = associatedBrick;
179:         }
180:
181:         public Integer getSessionId() {
182:                 return sessionId;
183:         }
184:
185:         public void setSessionId(Integer sessionId) {
186:                 this.sessionId = sessionId;
187:         }
188:
189:         public String getName() {
190:                 return nom;
191:         }
192:
193:         @Override
194:         public String toString() {
195:                 return nom;
196:         }
197:
```

```
    1: package models;
    2:
    3: import java.awt.event.MouseEvent;
    4: import java.awt.geom.Point2D;
    5: import java.util.Iterator;
    6:
    7: import javax.swing.JOptionPane;
    8:
    9: import dataPack.TreeListener;
   10: import edu.uci.ics.jung.graph.Edge;
   11: import edu.uci.ics.jung.graph.Graph;
   12: import edu.uci.ics.jung.graph.Vertex;
   13: import edu.uci.ics.jung.graph.impl.DirectedSparseEdge;
   14: import edu.uci.ics.jung.graph.impl.SimpleSparseVertex;
   15: import edu.uci.ics.jung.utils.UserDataContainer;
   16: import edu.uci.ics.jung.visualization.Layout;
   17: import edu.uci.ics.jung.visualization.PickSupport;
   18: import edu.uci.ics.jung.visualization.VisualizationViewer;
   19: import graphicalUserInterface.DialogHandlerFrame;
   20:
   21: public class ModelEditingMousePlugin extends TriadeEditingMousePlugin {
   22:
   23:
   24:         protected TreeListener treeListener;
   25:         protected Model editedModel;
   26:
   27:         public ModelEditingMousePlugin(TreeListener _treeListener, Model _editedMo
del)
   28:         {
   29:                 super();
   30:                 editedModel = _editedModel;
   31:                 treeListener = _treeListener;
   32:                 treeListener.setGrapheListener(this);
   33:
   34:         }
   35:
   36:         @Override
   37:         public void removeSelectedVertex()
   38:         {
   39:                 if (selectedVertex != null)
   40:                 {
   41:                         ModelVertex modelVertex = (ModelVertex)selectedVertex.getU
serDatum(ModelVertex.class);
   42:                         if (modelVertex != null)
   43:                         {
   44:
   45:
   46:                                 if (DialogHandlerFrame.showYesNoDialog("Voulez vou
s vraiment supprimer le sommet sÃ©lectionnÃ©?") == JOptionPane.YES_OPTION)
   47:                                 {
   48:                                         editedModel.removeModelVertex(modelVertex)
;
   49:                                         selectedVertex = null;
   50:                                         treeListener.showActor(modelVertex.content
Id);
   51:                                 }
   52:                         }
   53:                 }
   54:                 else if (selectedEdge != null)
   55:                 {
   56:                         ModelEdge modelEdge = (ModelEdge)selectedEdge.getUserDatum
(ModelEdge.class);
   57:                         if (DialogHandlerFrame.showYesNoDialog("Voulez vous vraime
nt supprimer l'arÃªte sÃ©lectionnÃ©e?") == JOptionPane.YES_OPTION)
   58:                         {
   59:                                 editedModel.removeModelEdge(modelEdge);
   60:                                 selectedEdge = null;
   61:                         }
   62:                 }
   63:         }
   64:
   65:
   66:         public void mousePressed(MouseEvent e) {
   67:                 if(checkModifiers(e)) {
   68:                         final VisualizationViewer vv =
   69:                                 (VisualizationViewer)e.getSource();
   70:                         final Point2D p = vv.inverseViewTransform(e.getPoint());
   71:                         PickSupport pickSupport = vv.getPickSupport();
   72:                         if(pickSupport != null) {
   73:                                 final Vertex vertex = pickSupport.getVertex(p.getX
(), p.getY());
   74:                                 if(vertex != null) { // get ready to make an edge
   75:                                         startVertex = vertex;
   76:                                         down = e.getPoint();
   77:                                         transformEdgeShape(down, down);
   78:                                         vv.addPostRenderPaintable(edgePaintable);
   79:                                         edgeIsDirected = true;
   80:                                         transformArrowShape(down, e.getPoint());
   81:                                         vv.addPostRenderPaintable(arrowPaintable);
   82:
   83:                                 } else { // make a new vertex
   84:                                         if (activeContent != null)
   85:                                         {
   86:                                                 Graph graph = vv.getGraphLayout().
getGraph();
   87:                                                 Vertex newVertex = new SimpleSpars
eVertex();
   88:                                                 vertexLocations.setLocation(newVer
tex, vv.inverseTransform(e.getPoint()));
   89:
   90:                                                 ModelVertex mV = editedModel.addMo
delVertex(activeContent, newVertex);
   91:                                                 mV.setLocation(vv.inverseTransform
(e.getPoint()));
   92:                                                 newVertex.addUserDatum(ModelVertex
.class,mV , new UserDataContainer.CopyAction.Shared());
   93:                                                 newVertex.addUserDatum("nom",edite
dModel.dataPack.getStringById(activeContent),new UserDataContainer.CopyAction.Clone());
   94:                                                 newVertex.addUserDatum("selection"
,new Integer(0),new UserDataContainer.CopyAction.Clone());
   95:                                                 newVertex.addUserDatum("id",mV.con
tentId,new UserDataContainer.CopyAction.Clone());
   96:                                                 Integer oldContent = activeContent
;
   97:                                                 activeContent = null;
   98:                                                 treeListener.hideActor(oldContent)
;
   99:
  100:                                                 Layout layout = vv.getGraphLayout(
);
  101:                                                 for(Iterator<?> iterator=graph.get
Vertices().iterator(); iterator.hasNext(); ) {
  102:                                                         layout.lockVertex((Vertex)
iterator.next());
  103:                                                 }
  104:                                                 graph.addVertex(newVertex);
  105:                                                 vv.getModel().restart();
  106:                                                 for(Iterator<?> iterator=graph.get
Vertices().iterator(); iterator.hasNext(); ) {
  107:                                                         layout.unlockVertex((Verte
x)iterator.next());
  108:                                                 }
  109:                                                 vv.repaint();
  110:                                         }
```

```
                                }
                        }
                }
        }

        /**
         * If startVertex is non-null, and the mouse is released over an
         * existing vertex, create an undirected edge from startVertex to
         * the vertex under the mouse pointer. If shift was also pressed,
         * create a directed edge instead.
         */
        public void mouseReleased(MouseEvent e) {
                if(checkModifiers(e)) {
                        final VisualizationViewer vv =
                                (VisualizationViewer)e.getSource();
                        final Point2D p = vv.inverseViewTransform(e.getPoint());
                        PickSupport pickSupport = vv.getPickSupport();
                        if(pickSupport != null) {
                                final Vertex vertex = pickSupport.getVertex(p.getX
(), p.getY());
                                if(vertex != null && startVertex != null) {
                                        if (vertex != startVertex)
                                        {
                                                if (startVertex.getSuccessors().co
ntains(vertex))
                                                {

                                                        Edge edge = startVertex.fi
ndEdge(vertex);
                                                        ModelEdge mE = (ModelEdge)
edge.getUserDatum(ModelEdge.class);
                                                         if (modelView.getPopUp().
showRelationChooserView(mE,this,"Edition d’une relation") != -1)
                                                                selectEdge(edge);


                                                }
                                                else
                                                {
                                                Graph graph = vv.getGraphLayout().
getGraph();
                                                Edge edge = new DirectedSparseEdge
(startVertex,vertex);
                                                ModelVertex source = (ModelVertex)
startVertex.getUserDatum(ModelVertex.class);
                                                ModelVertex destination = (ModelVe
rtex)vertex.getUserDatum(ModelVertex.class);
                                                Integer sourceID;
                                                Integer destinationID;

                                                //Normalement dans un model, ni so
urce, ni destination ne devrait etre null toutefois, les test sont conservÃ© en cas de be
soin
                                                if (source == null)
                                                        sourceID = (Integer)startV
ertex.getUserDatum("id");
                                                else
                                                        sourceID = source.vertexId
;

                                                if (destination == null)
                                                        destinationID = (Integer)s
tartVertex.getUserDatum("id");
                                                else
                                                        destinationID = destinatio
n.vertexId;

                                                if (destinationID == null || sourc
eID == null)
                                                {
                                                        System.err.println(destina
tionID+" : "+destination+"  --  "+sourceID+" : "+source);
                                                        System.err.println("Erreur
 lors de l’ajout d’une arrÃªte, un des sommets est mal construit");

                                                        startVertex = null;
                                                        down = null;
                                                        edgeIsDirected = false;
                                                        vv.removePostRenderPaintab
le(edgePaintable);
                                                        vv.removePostRenderPaintab
le(arrowPaintable);
                                                        return;
                                                }

                                                ModelEdge mEdge = editedModel.addM
odelEdge(sourceID, destinationID, edge);
                                                if (modelView.getPopUp().showRelat
ionChooserView(mEdge,this,"CrÃ©ation d’une relation") != -1)
                                                {
                                                        edge.addUserDatum(ModelEdg
e.class, mEdge, new UserDataContainer.CopyAction.Shared());
                                                        edge.addUserDatum("selecti
on", new Integer(1), new UserDataContainer.CopyAction.Clone());
                                                        graph.addEdge(edge);
                                                        selectEdge(edge);
                                                }
                                                else
                                                {
                                                        editedModel.removeModelEdg
e(mEdge);
                                                }

                                                }
                                        }
                                        else
                                        {
                                                if (vertex != selectedVertex)
                                                {
                                                        selectVertex(vertex);

                                                }
                                        }



                                }
                                startVertex = null;
                                down = null;
                                edgeIsDirected = false;
                                vv.removePostRenderPaintable(edgePaintable);
                                vv.removePostRenderPaintable(arrowPaintable);
                                vv.repaint();
                        }
                }
        }

        @Override
        public AbstractSchema getEditedAbstractSchema() {
                return editedModel;
        }
```

```
220: }
```

```
 1: package automate;
 2:
 3: import edu.uci.ics.jung.graph.impl.*;
 4: import edu.uci.ics.jung.graph.*;
 5: import java.util.*;
 6: import automateGraph.*;
 7:
 8:
 9: public class Automaton extends AbstractSparseGraph {
10:
11:         protected Hashtable<DirectedSparseEdge,Transition> transitions;
12:         protected Hashtable<Vertex,Action> actions;
13:
14:         public Automaton()
15:         {
16:                 super();
17:                 transitions = new Hashtable<DirectedSparseEdge,Transition>();
18:                 actions = new Hashtable<Vertex,Action>();
19:         }
20:
21:         public Transition getTransition(Vertex a, Vertex b)
22:         {
23:                 return transitions.get(new DirectedSparseEdge(a,b));
24:         }
25:
26:         public void editAction(Vertex vertex)
27:         {
28:                 Action action = actions.get(vertex);
29:                 if (action == null)
30:                 {
31:                         action = new EmptyAction();
32:                         actions.put(vertex, action);
33:
34:                 }
35:                 ActionEditingFrame.editAction(action);
36:         }
37:
38:         public void editTransition(DirectedSparseEdge edge)
39:         {
40:                 Transition transition = transitions.get(edge);
41:                 if (transition == null)
42:                 {
43:                         transition = new EpsilonTransition(0);
44:                         transitions.put(edge, transition);
45:
46:                 }
47:                 //TransitionEditingFrame.editTransition(transition);
48:
49:         }
50: }
```

```
    1: package automate;
    2:
    3: import java.awt.Color;
    4: import java.awt.Graphics;
    5: import java.awt.Graphics2D;
    6: import java.awt.Shape;
    7: import java.awt.event.MouseEvent;
    8: import java.awt.event.MouseListener;
    9: import java.awt.event.MouseMotionListener;
   10: import java.awt.geom.AffineTransform;
   11: import java.awt.geom.CubicCurve2D;
   12: import java.awt.geom.Point2D;
   13: import java.util.Iterator;
   14:
   15: import edu.uci.ics.jung.graph.Graph;
   16: import edu.uci.ics.jung.graph.Vertex;
   17: import edu.uci.ics.jung.graph.impl.DirectedSparseEdge;
   18: import edu.uci.ics.jung.graph.impl.SparseVertex;
   19: import edu.uci.ics.jung.graph.impl.UndirectedSparseEdge;
   20: import edu.uci.ics.jung.visualization.ArrowFactory;
   21: import edu.uci.ics.jung.visualization.Layout;
   22: import edu.uci.ics.jung.visualization.PickSupport;
   23: import edu.uci.ics.jung.visualization.PickedState;
   24: import edu.uci.ics.jung.visualization.SettableVertexLocationFunction;
   25: import edu.uci.ics.jung.visualization.VisualizationViewer;
   26: import edu.uci.ics.jung.visualization.VisualizationViewer.Paintable;
   27: import edu.uci.ics.jung.visualization.control.AbstractGraphMousePlugin;
   28: import edu.uci.ics.jung.visualization.control.EditingGraphMousePlugin;
   29: //import edu.uci.ics.jung.visualization.control.EditingGraphMousePlugin.ArrowPaint
able;
   30: //import edu.uci.ics.jung.visualization.control.EditingGraphMousePlugin.EdgePainta
ble;
   31:
   32: public class EditingAutomatonMousePlugin extends EditingGraphMousePlugin implement
s
   33:     MouseListener, MouseMotionListener {
   34:
   35:     SettableVertexLocationFunction vertexLocations;
   36:     Vertex startVertex;
   37:     Point2D down;
   38:
   39:     CubicCurve2D rawEdge = new CubicCurve2D.Float();
   40:     Shape edgeShape;
   41:     Shape rawArrowShape;
   42:     Shape arrowShape;
   43:     Paintable edgePaintable;
   44:     Paintable arrowPaintable;
   45:     boolean edgeIsDirected;
   46:
   47:     public EditingAutomatonMousePlugin() {
   48:         this(MouseEvent.BUTTON1_MASK);
   49:     }
   50:
   51:     /**
   52:      * create instance and prepare shapes for visual effects
   53:      * @param modifiers
   54:      */
   55:     public EditingAutomatonMousePlugin(int modifiers) {
   56:         super(modifiers);
   57:         rawEdge.setCurve(0.0f, 0.0f, 0.33f, 100, .66f, -50,
   58:                 1.0f, 0.0f);
   59:         rawArrowShape = ArrowFactory.getNotchedArrow(20, 16, 8);
   60:         edgePaintable = new EdgePaintable();
   61:         arrowPaintable = new ArrowPaintable();
   62:     }
   63:
   64:     /**
   65:      * sets the vertex locations. Needed to place new vertices
   66:      * @param vertexLocations
   67:      */
   68:     public void setVertexLocations(SettableVertexLocationFunction vertexLocations)
 {
   69:         this.vertexLocations = vertexLocations;
   70:     }
   71:
   72:     /**
   73:      * overrided to be more flexible, and pass events with
   74:      * key combinations. The default responds to both ButtonOne
   75:      * and ButtonOne+Shift
   76:      */
   77:     public boolean checkModifiers(MouseEvent e) {
   78:         return (e.getModifiers() & modifiers) != 0;
   79:     }
   80:
   81:     /**
   82:      * If the mouse is pressed in an empty area, create a new vertex there.
   83:      * If the mouse is pressed on an existing vertex, prepare to create
   84:      * an edge from that vertex to another
   85:      */
   86:     public void mousePressed(MouseEvent e) {
   87:         if((e.getModifiersEx() &  MouseEvent.BUTTON1_DOWN_MASK) ==  MouseEvent.BUT
TON1_DOWN_MASK)
   88:         {
   89:
   90:                 final VisualizationViewer vv =(VisualizationViewer)e.getSource();
   91:                 Automaton graph = (Automaton)vv.getGraphLayout().getGraph();
   92:                 final Point2D p = vv.inverseViewTransform(e.getPoint());
   93:                 PickSupport pickSupport = vv.getPickSupport();
   94:             PickedState pickedState = vv.getPickedState();
   95:                 if(pickSupport != null)
   96:                 {
   97:                         final Vertex vertex = pickSupport.getVertex(p.getX(), p.ge
tY());
   98:
   99:
  100:                         if(vertex != null)
  101:                         { //Il y a un sommet Ã  l'endroit oÃ¹ l'on a cliquÃ©
  102:                                 if ((e.getModifiersEx() & MouseEvent.CTRL_DOWN_MAS
K) == MouseEvent.CTRL_DOWN_MASK)
  103:                                 {
  104:                                         //CTRL est enfoncÃ©, on Ã©dite l'action du
 sommet
  105:
  106:                                         graph.editAction(vertex);
  107:                                 }
  108:                                 else
  109:                                 {
  110:                                         //CTRL n'est pas appuyÃ©
  111:
  112:                                         startVertex = vertex;
  113:                                         down = e.getPoint();
  114:                                         transformEdgeShape(down, down);
  115:                                         vv.addPostRenderPaintable(edgePaintable);
  116:                                         if((e.getModifiers() & MouseEvent.SHIFT_MA
SK) != 0)
  117:                                         {
  118:                                                 edgeIsDirected = true;
  119:                                                 transformArrowShape(down, e.getPoi
nt());
  120:                                                 vv.addPostRenderPaintable(arrowPai
ntable);
  121:                         }
  122:                                 }
  123:                 } else if ((e.getModifiersEx() & MouseEvent.SHIFT_DOWN_MASK) == Mo
```

```
useEvent.SHIFT_DOWN_MASK)
                {       // S'il n'y a pas de sommet sous la souris et que shift e
st appuyÃ©, on crÃ©Ã© un nouveau sommet avec l'Ã©diteur d'action Ã  lui associer.

                    Vertex newVertex = new SparseVertex();
                    vertexLocations.setLocation(newVertex, vv.inverseTransform(e.g
etPoint()));
                    Layout layout = vv.getGraphLayout();
                    for(Iterator iterator=graph.getVertices().iterator(); iterator
.hasNext(); ) {
                        layout.lockVertex((Vertex)iterator.next());
                    }
                    graph.addVertex(newVertex);
                    vv.getModel().restart();
                    for(Iterator iterator=graph.getVertices().iterator(); iterator
.hasNext(); ) {
                        layout.unlockVertex((Vertex)iterator.next());
                    }
                    vv.repaint();
                    graph.editAction(newVertex);
                }
            }
        }
    }

    /**
     * If startVertex is non-null, and the mouse is released over an
     * existing vertex, create an directed edge from startVertex to
     * the vertex under the mouse pointer with a transition.
     */
    public void mouseReleased(MouseEvent e) {
        if(checkModifiers(e)) {
            final VisualizationViewer vv =
                (VisualizationViewer)e.getSource();
            Automaton graph = (Automaton)vv.getGraphLayout().getGraph();
            final Point2D p = vv.inverseViewTransform(e.getPoint());
            PickSupport pickSupport = vv.getPickSupport();
            if(pickSupport != null) {
                final Vertex vertex = pickSupport.getVertex(p.getX(), p.getY());
                if(vertex != null && startVertex != null) {
                    DirectedSparseEdge arete = new DirectedSparseEdge(startVertex,
 vertex);
                    if (graph.getEdges().contains(arete))
                    {
                        graph.editTransition(arete);
                    }
                    else
                    {
                        graph.addEdge(arete);
                        graph.editTransition(arete);

                    }

                    vv.repaint();
                }
            }
            startVertex = null;
            down = null;
            edgeIsDirected = false;
            vv.removePostRenderPaintable(edgePaintable);
            vv.removePostRenderPaintable(arrowPaintable);
        }
    }

    /**
     * If startVertex is non-null, stretch an edge shape between
     * startVertex and the mouse pointer to simulate edge creation
     */
    public void mouseDragged(MouseEvent e) {
        if(checkModifiers(e)) {
            if(startVertex != null) {
                transformEdgeShape(down, e.getPoint());
                if(edgeIsDirected) {
                    transformArrowShape(down, e.getPoint());
                }
            }
        }
    }

    /**
     * code lifted from PluggableRenderer to move an edge shape into an
     * arbitrary position
     */
    private void transformEdgeShape(Point2D down, Point2D out) {
        float x1 = (float) down.getX();
        float y1 = (float) down.getY();
        float x2 = (float) out.getX();
        float y2 = (float) out.getY();

        AffineTransform xform = AffineTransform.getTranslateInstance(x1, y1);

        float dx = x2-x1;
        float dy = y2-y1;
        float thetaRadians = (float) Math.atan2(dy, dx);
        xform.rotate(thetaRadians);
        float dist = (float) Math.sqrt(dx*dx + dy*dy);
        xform.scale(dist / rawEdge.getBounds().getWidth(), 1.0);
        edgeShape = xform.createTransformedShape(rawEdge);
    }

    private void transformArrowShape(Point2D down, Point2D out) {
        float x1 = (float) down.getX();
        float y1 = (float) down.getY();
        float x2 = (float) out.getX();
        float y2 = (float) out.getY();

        AffineTransform xform = AffineTransform.getTranslateInstance(x2, y2);

        float dx = x2-x1;
        float dy = y2-y1;
        float thetaRadians = (float) Math.atan2(dy, dx);
        xform.rotate(thetaRadians);
        arrowShape = xform.createTransformedShape(rawArrowShape);
    }

    /**
     * Used for the edge creation visual effect during mouse drag
     */
    class EdgePaintable implements Paintable {

        public void paint(Graphics g) {
            if(edgeShape != null) {
                Color oldColor = g.getColor();
                g.setColor(Color.black);
                ((Graphics2D)g).draw(edgeShape);
                g.setColor(oldColor);
            }
        }

        public boolean useTransform() {
            return false;
        }
    }
```

```
252:
253:     /**
254:      * Used for the directed edge creation visual effect during mouse drag
255:      */
256:     class ArrowPaintable implements Paintable {
257:
258:         public void paint(Graphics g) {
259:             if(arrowShape != null) {
260:                 Color oldColor = g.getColor();
261:                 g.setColor(Color.black);
262:                 ((Graphics2D)g).fill(arrowShape);
263:                 g.setColor(oldColor);
264:             }
265:         }
266:
267:         public boolean useTransform() {
268:             return false;
269:         }
270:     }
271:     public void mouseClicked(MouseEvent e) {}
272:     public void mouseEntered(MouseEvent e) {}
273:     public void mouseExited(MouseEvent e) {}
274:     public void mouseMoved(MouseEvent e) {}
275: }
276:
277:
```

```
 1: package automate;
 2:
 3: import java.util.Hashtable;
 4:
 5: public class EpsilonTransition implements Transition {
 6:
 7:         protected int weight;
 8:
 9:         public EpsilonTransition(int w)
10:         {
11:                 weight = w;
12:         }
13:
14:         public boolean evalTransition(Hashtable info) {
15:                 return true;
16:         }
17:
18:         public int getWeight()
19:         {
20:                 return weight;
21:         }
22:
23: }
```

```
 1: package automate;
 2:
 3: import java.util.*;
 4:
 5: public class KeywordTransition extends EpsilonTransition {
 6:
 7:         public static String key = "keyword";
 8:         protected String keyword;
 9:
10:         public KeywordTransition(String _keyword, int w)
11:         {
12:                 super(w);
13:                 keyword = _keyword;
14:         }
15:
16:         public boolean evalTransition(Hashtable info) {
17:                 Set<String> keywordSet = (Set<String>)info.get(key);
18:                 if (keywordSet != null)
19:                 {
20:                         return keywordSet.contains(keyword);
21:                 }
22:                 return false;
23:         }
24:
25: }
```

```
    1: package automate;
    2:
    3: import main.*;
    4: import edu.uci.ics.jung.graph.*;
    5: import java.util.*;
    6:
    7: public class AutomatonExecution {
    8:
    9:         protected Vertex activeState;
   10:         protected Automaton automate;
   11:
   12:
   13:         public Vertex nextState(Hashtable info)
   14:         {
   15:
   16:                 Vertex result = null;
   17:                 if (activeState != null)
   18:                 {
   19:                 int weight = -1;
   20:                 Set<Vertex> successeurs = activeState.getOutEdges();
   21:                 for (Vertex successeur:successeurs)
   22:                 {
   23:                         if(automate.getTransition(activeState,successeur).evalTran
sition(info))
   24:                         {
   25:                                 if (automate.getTransition(activeState,successeur)
.getWeight() > weight)
   26:                                 {
   27:                                         result = successeur;
   28:                                         weight = automate.getTransition(activeStat
e,successeur).getWeight();
   29:                                 }
   30:                         }
   31:                 }
   32:                 }
   33:                 else
   34:                 {
   35:                         System.out.println("Cette execution Ã  dÃ©jÃ  Ã©tÃ© achÃ©v
Ã©e.");
   36:                 }
   37:                 activeState = result;
   38:                 return result;
   39:
   40:         }
   41:
   42:         public Vertex getActiveState()
   43:         {
   44:                 return activeState;
   45:         }
   46: }
```

```
 1: package automate;
 2:
 3: import javax.swing.*;
 4:
 5: public class EmptyAction extends Action {
 6:
 7:         public JComponent toComponent()
 8:         {
 9:                 return new JTextField("Action vide");
10:         }
11:
12: }
```

```
1: package automate;
2:
3: import javax.swing.*;
4:
5: public abstract class Action {
6:
7:
8:         abstract public JComponent toComponent();
9: }
```

```
1: package automate;
2:
3: import java.util.*;
4:
5: public interface Transition {
6:
7:                 public boolean evalTransition(Hashtable info);
8:                 public int getWeight();
9: }
```

```java
package automate;

import java.util.Hashtable;

public class BooleanOpTransition extends EpsilonTransition {

	public static enum Operator{AND, OR, XOR, NAND}
	protected Transition[] elements;
	protected Operator type;
	protected int weight;


	public BooleanOpTransition(Operator _type, Transition[] _elements, int w)
	{
		super(w);
		elements = _elements;
		type = _type;
	}

	public boolean evalTransition(Hashtable info) {
		boolean result = false;
		switch (type)
		{
		case AND:
			result = true;
			for (Transition trans:elements)
			{
				result = result && trans.evalTransition(info);
			}
			break;
		case NAND:
			result = true;
			for (Transition trans:elements)
			{
				result = result && trans.evalTransition(info);
			}
			result = !result;
			break;
		case OR:
			result = false;
			for (Transition trans:elements)
			{
				result = result || trans.evalTransition(info);
			}
			break;
		case XOR:
			result = false;
			for (Transition trans:elements)
			{
				if (result && trans.evalTransition(info))
					return false;
				result = result || trans.evalTransition(info);
			}
		}

		return result;
	}

	public int getWeight()
	{
		return weight;
	}


	public static BooleanOpTransition or(Transition a, Transition b, int w)
	{
		Transition[] temp = new Transition[2];
		temp[0] = a;
		temp[1] = b;
		return new BooleanOpTransition(Operator.OR,temp,w);
	}

	public static BooleanOpTransition and(Transition a, Transition b, int w)
	{
		Transition[] temp = new Transition[2];
		temp[0] = a;
		temp[1] = b;
		return new BooleanOpTransition(Operator.AND,temp,w);
	}

	public static BooleanOpTransition nand(Transition a, Transition b, int w)
	{
		Transition[] temp = new Transition[2];
		temp[0] = a;
		temp[1] = b;
		return new BooleanOpTransition(Operator.NAND,temp,w);
	}

	public static BooleanOpTransition xor(Transition a, Transition b, int w)
	{
		Transition[] temp = new Transition[2];
		temp[0] = a;
		temp[1] = b;
		return new BooleanOpTransition(Operator.XOR,temp,w);
	}

}
```

```
    1: package dataPack;
    2:
    3: import graphicalUserInterface.IconDatabase;
    4: import graphicalUserInterface.Program;
    5:
    6: import java.io.Serializable;
    7:
    8: import javax.swing.Icon;
    9: import javax.swing.JLabel;
   10:
   11: import client.stringTranslator.StringTranslator;
   12:
   13: public class Groupe extends MyDefaultMutableTreeNode implements Serializable {
   14:
   15:         private static final long serialVersionUID = -560872084918416287L;
   16:
   17:         protected Integer idGroupe;
   18:         protected String nom;
   19:
   20:         public Groupe(String n_nom) {
   21:                 nom = n_nom;
   22:
   23:                 DataPack dataPack = Program.myMainFrame.getDataPack();
   24:
   25:                 if (dataPack != null) {
   26:                         idGroupe = Program.myMainFrame.getDataPack().getNewGroupEI
d();
   27:                 } else {
   28:                         idGroupe = new Integer(-1);
   29:                 }
   30:         }
   31:
   32:         public Integer getIdGroupe() {
   33:                 return idGroupe;
   34:         }
   35:
   36:         public String getNom() {
   37:                 return nom;
   38:         }
   39:
   40:         public void setNom(String nom) {
   41:                 this.nom = nom;
   42:         }
   43:
   44:         @Override
   45:         public JLabel getJComponent(boolean selected, boolean expanded, boolean le
af,
   46:                         int row, boolean hasFocus) {
   47:                 if (getChildCount() == 0) {
   48:                         return TreeActorsCellRendere.createDefaultLabel(toString()
,
   49:                                         IconDatabase.iconFileEmpty, selected);
   50:                 } else {
   51:                         if (expanded) {
   52:                                 return TreeActorsCellRendere.createDefaultLabel(to
String(),
   53:                                                 IconDatabase.iconFileOpen, selecte
d);
   54:                         } else {
   55:                                 return TreeActorsCellRendere.createDefaultLabel(to
String(),
   56:                                                 IconDatabase.iconFileClose, select
ed);
   57:                         }
   58:                 }
   59:         }
   60:
   61:         @Override
   62:         public Icon getIcon() {
   63:                 return null;
   64:         }
   65:
   66:         @Override
   67:         public MyTreeNodeType getType() {
   68:                 return MyTreeNodeType.GroupeType;
   69:         }
   70:
   71:         @Override
   72:         public void changeStringValue(String newString) {
   73:                 setNom(newString);
   74:         }
   75:
   76:         @Override
   77:         public Integer getId() {
   78:                 return idGroupe;
   79:         }
   80:
   81:         public String getNoTranslatedString() {
   82:                 return nom;
   83:         }
   84:
   85:         @Override
   86:         public String toString() {
   87:                 return StringTranslator.getTranslatedString(this, StringTranslator
.StringType.groupType);
   88:         }
   89: }
```

```java
package dataPack;

import graphicalUserInterface.DialogHandlerFrame;
import graphicalUserInterface.Program;

import java.io.File;

import javax.swing.JOptionPane;

import translation.Messages;

public class AutoSaveCreator implements Runnable {
	public static String extentionAutoSave = "~"; //$NON-NLS-1$
	private final static String defaultAutoSamveName = "." + File.separatorChar + "defaultAutoSave.dtp"; //$NON-NLS-1$
	private final static int saveTimer = 300; // seconde

	private DataPack dataPack;
	private final Thread thread;
	private boolean enable;

	public AutoSaveCreator(DataPack dataPack) {
		this.dataPack = dataPack;

		enable = true;
		thread = new Thread(this);
		thread.start();
	}

	public AutoSaveCreator() {
		this(null);
	}

	@Override
	public void run() {
		boolean crypted = true;

		while (enable) {
			try {
				Thread.currentThread();
				Thread.sleep(saveTimer * 1000);

				if (getDataPack() == null || (enable == false)) {
					continue;
				}

				String filePath = getDataPack().getFilePath();

				if (filePath != null) {
					if(filePath.endsWith(extentionAutoSave)) {
						filePath = getPathWithoutAutoSaveExtention(filePath);
					}

					Program.saveObject(getDataPack(), filePath + extentionAutoSave, crypted);
				} else {
					Program.saveObject(getDataPack(), defaultAutoSamveName
							+ extentionAutoSave, crypted);
				}
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}

	static public String getPathWithoutAutoSaveExtention(String filePath) {
		if(filePath.endsWith(extentionAutoSave)) {
		return filePath.substring(0, filePath.length()
				- extentionAutoSave.length());
		} else {
			return filePath;
		}
	}

	static public String getAutosaveFilePath(String filePath) {
		if(filePath.endsWith(extentionAutoSave)) {
			return filePath;
		} else {
			return filePath + extentionAutoSave;
		}
	}

	static public void dataPackSaved(SavableObject dataPack) {
		File autoSaveFilePath = new File(getAutosaveFilePath(dataPack.getFilePath()));

		if (autoSaveFilePath.exists() && autoSaveFilePath.isFile()) {
			autoSaveFilePath.delete();
		}
	}

	public static boolean isAutoSaveFile(String filePath) {
		return filePath.endsWith(extentionAutoSave);
	}

	public static boolean hasAutoSaveFile(String filePath) {
		File autoSaveFile = new File(getAutosaveFilePath(filePath));
		return autoSaveFile.exists() && autoSaveFile.isFile();
	}

	public static String askLoadAutoSaveFile(String filePath) {
		if (hasAutoSaveFile(filePath)) {
			int loadAutosave = DialogHandlerFrame
			.showYesNoDialog(Messages.getString("AutoSaveCreator.2")); //$NON-NLS-1$

			if (loadAutosave == JOptionPane.YES_OPTION) {
				File autoSaveFile = new File(getAutosaveFilePath(filePath));

				File file = new File(filePath);
				if(file.exists()) {
					file.delete();
				}
				autoSaveFile.renameTo(file);
			}
		}

		return filePath;
	}

	public void stopAutoSave() {
		setEnable(false);
	}

	synchronized private void setEnable(boolean value) {
		enable = value;
	}

	public synchronized DataPack getDataPack() {
		return dataPack;
	}
```

```
127:
128:         public synchronized void setDataPack(DataPack dataPack) {
129:                 this.dataPack = dataPack;
130:         }
131:
132:
133: }
```

```
    1: package dataPack;
    2:
    3: import java.util.Enumeration;
    4:
    5: import javax.swing.JTree;
    6: import javax.swing.SwingUtilities;
    7: import javax.swing.event.TreeModelEvent;
    8: import javax.swing.event.TreeModelListener;
    9: import javax.swing.tree.DefaultMutableTreeNode;
   10: import javax.swing.tree.TreePath;
   11:
   12: public class MyMutableJTree extends JTree implements TreeModelListener {
   13:         /**
   14:          *
   15:          */
   16:         private static final long serialVersionUID = -7868105658865493729L;
   17:
   18:         @Override
   19:         public void treeNodesChanged(TreeModelEvent e) {
   20:                 reloadModel();
   21:         }
   22:
   23:         @Override
   24:         public void treeNodesInserted(TreeModelEvent e) {
   25:                 reloadModel();
   26:         }
   27:
   28:         @Override
   29:         public void treeNodesRemoved(TreeModelEvent e) {
   30:                 reloadModel();
   31:         }
   32:
   33:         @Override
   34:         public void treeStructureChanged(TreeModelEvent e) {
   35:                 reloadModel();
   36:         }
   37:
   38:         protected void reloadModel() {
   39:                 Enumeration<TreePath> expendedPaths = getExpandedDescendants(new T
reePath(((DefaultMutableTreeNode)getModel().getRoot()).getPath()));
   40:                 final int[] selectedNodes = getSelectionRows();
   41:
   42:                 if(expendedPaths != null) {
   43:                         while (expendedPaths.hasMoreElements()) {
   44:                                 final TreePath path = expendedPaths.nextElement();
   45:                                 SwingUtilities.invokeLater(new Runnable() {
   46:                                         @Override
   47:                                         public void run() {
   48:                                                 expandPath(path);
   49:                                         }
   50:                                 });
   51:                                 makeVisible(path);
   52:                         }
   53:                 }
   54:
   55:                 if (selectedNodes != null) {
   56:                         SwingUtilities.invokeLater(new Runnable() {
   57:
   58:                                 @Override
   59:                                 public void run() {
   60:                                         setSelectionRows(selectedNodes);
   61:                                 }
   62:                         });
   63:                 }
   64:
   65:                 validate();
   66:         }
   67: }
```

```
    1: package dataPack;
    2:
    3: import graphicalUserInterface.DialogHandlerFrame;
    4: import graphicalUserInterface.Program;
    5:
    6: import java.io.Externalizable;
    7: import java.io.IOException;
    8: import java.io.ObjectInput;
    9: import java.io.ObjectOutput;
   10: import java.util.Enumeration;
   11: import java.util.Vector;
   12:
   13: import javax.swing.DefaultCellEditor;
   14: import javax.swing.event.CellEditorListener;
   15: import javax.swing.event.ChangeEvent;
   16: import javax.swing.tree.DefaultMutableTreeNode;
   17: import javax.swing.tree.DefaultTreeModel;
   18: import javax.swing.tree.TreePath;
   19: import javax.swing.tree.TreeSelectionModel;
   20:
   21: import models.Brick;
   22: import models.BrickEdge;
   23: import models.BrickVertex;
   24: import models.TriadeEditingMousePlugin;
   25:
   26: /*
   27:  * Affiche la hiÃ©archie des acteurs dans leur entreprise et permet dans cacher ce
rtain ou de les afficher.
   28:  * Ainsi seul les acteurs pouvant Ã©tre manipuler sont visible, ce qui facilite la
 visibilitÃ© et la manipulation.
   29:  * De plus elle ne touche pas au model et les changement sont mis a jours automati
quement en cas de changement sur ce model.
   30:  */
   31:
   32: public class JTreeActors extends MyMutableJTree implements Externalizable, CellEdi
torListener {
   33:         private static final long serialVersionUID = -5070764781286563264L;
   34:
   35:         protected TreeActorsCellRendere treeCellRenderer;
   36:         protected TreeListener treeListener;
   37:
   38:         protected FiltreActeursEntreprise filtreActeurs;
   39:         protected Vector<JeuActeur> jeuxActeur;
   40:         protected DataPack dataPack;
   41:
   42:         protected Vector<Acteur> hidedBaseActors;
   43:         protected boolean isHidedBases;
   44:
   45:         protected Brick brick;
   46:
   47:         protected TriadeEditingMousePlugin<BrickVertex, BrickEdge> triadeEditingMo
usePlugin;
   48:
   49:         protected Groupe otherGroup;
   50:
   51:         public JTreeActors() {
   52:                 //TODO trouver comment ne pas sauvegarder cette arbre
   53:                 treeListener = null;
   54:         }
   55:
   56:         public JTreeActors(DataPack dataPack) {
   57:                 treeListener = new TreeListener(dataPack.getActorsModel(), null);
   58:                 if(!Program.isTriades()) {
   59:                         setToolTipText("Clic droit pour renomer");
   60:                 }
   61:                 addMouseListener(treeListener);
   62:                 build(dataPack);
   63:                 initJTree();
   64:         }
   65:
   66:         public JTreeActors(DataPack dataPack, Brick brick) {
   67:                 this(dataPack);
   68:
   69:                 this.brick = brick;
   70:
   71:                 for (JeuActeur jeuActeur : Program.myMainFrame.getDataPack()
   72:                                 .getJeuxActeur()) {
   73:                         ajouterJeuActeur(jeuActeur);
   74:                 }
   75:
   76:                 // les acteurs present dans la brique doivent Ã©tre cacher
   77:                 // les moyen sont affichÃ©s pour les ajouter a la brique
   78:
   79:                 Vector<Content> actorsSet = new Vector<Content>();
   80:
   81:                 // BrickType brickType =
   82:                 // brick.getDatapack().getBrickTypeById(brick.getType().getBrickTy
peId());
   83:
   84:                 // rÃ©cupÃ©re tout les acteurs existant dans la brique
   85:                 for (BrickVertex brickVertex : brick.getVertices()) {
   86:                         actorsSet.add(brickVertex.getContent());
   87:                 }
   88:
   89:                 // cache les acteurs deja present
   90:                 for (Content actor : actorsSet) {
   91:                         if(actor instanceof MyDefaultMutableTreeNode)
   92:                                 hideActor(actor);
   93:                 }
   94:
   95:                 // femre les dossier des Jeux d'acteurs
   96:                 for (JeuActeur jeuActeur : jeuxActeur) {
   97:                         collapsePath(new TreePath(jeuActeur.getGroupeJeuActeur().g
etPath()));
   98:                 }
   99:         }
  100:
  101:         private void build(DataPack dataPack) {
  102:                 filtreActeurs = new FiltreActeursEntreprise(dataPack.getActorsMode
l());
  103:                 jeuxActeur = new Vector<JeuActeur>();
  104:                 this.dataPack = dataPack;
  105:
  106:                 hidedBaseActors = new Vector<Acteur>();
  107:                 isHidedBases = false;
  108:
  109:                 setModel(filtreActeurs);
  110:
  111:                 getModel().addTreeModelListener(this);
  112:                 filtreActeurs.addTreeModelListener(this);
  113:         }
  114:
  115:         protected void initJTree() {
  116:                 treeListener.setTree(this);
  117:
  118:                 filtreActeurs.setTreeListener(treeListener);
  119:
  120:                 getSelectionModel().setSelectionMode(
  121:                                 TreeSelectionModel.SINGLE_TREE_SELECTION);
  122:
  123:                 setRootVisible(false);
  124:
  125:                 expandPath(new TreePath((((DefaultMutableTreeNode)getModel()
  126:
```

```
  127:                                .getRoot())
  128:                                .getPath())));
  129:
  130:                treeCellRenderer = new TreeActorsCellRendere();
  131:                setCellRenderer(treeCellRenderer);
  132:                TreeCellEditor cellEditor = new TreeCellEditor();
  133:                setCellEditor(cellEditor);
  134:                cellEditor.addCellEditorListener(this);
  135:
  136:                dataPack.initDataPackActorsView(filtreActeurs);
  137:
  138:                if (Program.isTriadesLoading()) {
  139:                        for (JeuActeur jeuActeur : dataPack.getJeuxActeur()) {
  140:                                hideBases(jeuActeur);
  141:                        }
  142:
  143:                        filtreActeurs.hideNode(dataPack.getActorsModel().getGroupe
Moyens());
  144:                }
  145:
  146:                expendAllNode();
  147:
  148:                treeListener.setSelectedNode(dataPack.getActorsModel()
  149:                                .getGroupeActeurs());
  150:                // setDragEnabled(true);
  151:                // setDropMode(DropMode.ON_OR_INSERT);
  152:                // setTransferHandler(new TreeTransferHandler());
  153:        }
  154:
  155:        protected void expendAllNode(DefaultMutableTreeNode node) {
  156:                if (node == null)
  157:                        return;
  158:
  159:                expendNode(node);
  160:
  161:                int nbChild = node.getChildCount();
  162:                for (int i = 0; i < nbChild; i++) {
  163:                        DefaultMutableTreeNode child = (DefaultMutableTreeNode) no
de
  164:                        .getChildAt(i);
  165:
  166:                        expendAllNode(child);
  167:                }
  168:        }
  169:
  170:        protected void expendNode(DefaultMutableTreeNode node) {
  171:                // tree.expandPath(new TreePath(node.getPath()));
  172:                makeVisible(new TreePath(node.getPath()));
  173:        }
  174:
  175:        protected void collapseNode(DefaultMutableTreeNode node) {
  176:                collapsePath(new TreePath(node.getPath()));
  177:        }
  178:
  179:        public void expendAllNode() {
  180:                expendAllNode(getRoot());
  181:        }
  182:
  183:        protected MyDefaultMutableTreeNode getSelectedNode() {
  184:                TreePath path = getSelectionPath();
  185:                if (path == null) {
  186:                        return null;
  187:                }
  188:
  189:                MyDefaultMutableTreeNode selectedNode = (MyDefaultMutableTreeNode)
 path
  190:                                .getLastPathComponent();
  191:
  192:                return selectedNode;
  193:        }
  194:
  195:        public void hideActor(Content content) {
  196:                hideActor((MyDefaultMutableTreeNode)content);
  197:        }
  198:
  199:        public void hideActor(MyDefaultMutableTreeNode actor) {
  200:                filtreActeurs.hideNode(actor);
  201:
  202:                // si l'"acteur est une base ou un corps il faut cacher les corps
ou les
  203:                // bases
  204:                if (MyDefaultMutableTreeNode.isCorps(actor)) {
  205:                        hideBases(((ActeurBase) actor).jeuActeur);
  206:                        hidedBaseActors.add((ActeurBase) actor);
  207:                } else if (MyDefaultMutableTreeNode.isBase(actor)) {
  208:                        hideAllBases(((ActeurBase) actor).jeuActeur);
  209:                        hideCorps(((ActeurBase) actor).jeuActeur);
  210:                        hidedBaseActors.add((ActeurBase) actor);
  211:                }
  212:        }
  213:
  214:        public void hideAll() {
  215:                hideAll(dataPack.getActorsModel().getRoot());
  216:        }
  217:
  218:        public void hideAll(DefaultMutableTreeNode father) {
  219:                if(father.isLeaf() == false) {
  220:                        for(Enumeration<?> enumeration = father.children() ; enume
ration.hasMoreElements() ;)
  221:                                hideAll((DefaultMutableTreeNode)enumeration.nextEl
ement());
  222:                }
  223:
  224:                try{
  225:                        Acteur acteur = (Acteur)father;
  226:                        hideActor(acteur);
  227:                } catch (Exception e) {
  228:                        e.printStackTrace();
  229:                }
  230:                //                      filtreActeurs.hideNode(node);
  231:                //                      hidedBaseActors.add(new Acteur(null, null,
 null)); // hack pour corriger le fait que ce tableau n'est pas rempli lors d'un hideall
  232:                //                      isHidedBases = true;
  233:        }
  234:
  235:        public void showActor(Content content) {
  236:                MyDefaultMutableTreeNode actor = (MyDefaultMutableTreeNode)content
;
  237:
  238:                hidedBaseActors.remove(actor);
  239:                if (hidedBaseActors.size() == 0) {
  240:                        showAllBases();
  241:                        showAllCorps();
  242:                }
  243:
  244:                filtreActeurs.shwoNode(actor);
  245:        }
  246:
  247:        public void simpleHideAll() {
  248:                filtreActeurs.hideNodes(dataPack.getAllActors());
  249:        }
  250:
  251:        public MyDefaultMutableTreeNode simpleShowActor(Content content) {
  252:                MyDefaultMutableTreeNode actor = (MyDefaultMutableTreeNode)content
```

```
;
  253:                  filtreActeurs.shwoNode(actor);
  254:                  return actor;
  255:          }
  256:
  257:          public void hideCorps(JeuActeur jeuActeur) {
  258:                  filtreActeurs.hideNodes(jeuActeur.getGroupesCorps());
  259:          }
  260:
  261:          public void hideBases(JeuActeur jeuActeur) {
  262:                  filtreActeurs.hideNodes(jeuActeur.getListeBase());
  263:          }
  264:
  265:          public void hideAllBases(JeuActeur jeuActeurReference) {
  266:                  if (isHidedBases == false) {
  267:
  268:                          for (JeuActeur jeuActeur : jeuxActeur) {
  269:                                  if (jeuActeur != jeuActeurReference) {
  270:                                          hideBases(jeuActeur);
  271:                                  }
  272:                          }
  273:                  }
  274:          }
  275:
  276:          public void hideAllCorps(JeuActeur jeuActeurReference) {
  277:                  for (JeuActeur jeuActeur : jeuxActeur) {
  278:                          if (jeuActeur != jeuActeurReference) {
  279:                                  filtreActeurs.hideNodes(jeuActeur.getGroupesCorps(
));
  280:                          }
  281:                  }
  282:          }
  283:
  284:          public void showAllBases() {
  285:                  isHidedBases = false;
  286:                  for(JeuActeur jeuActeur : jeuxActeur) {
  287:                          filtreActeurs.shwoNodes(jeuActeur.getListeBase());
  288:                  }
  289:          }
  290:
  291:          public void showAllCorps() {
  292:                  for (JeuActeur jeuActeur : jeuxActeur) {
  293:                          filtreActeurs.shwoNodes((jeuActeur.getGroupesCorps()));
  294:
  295:                          //pour chaque groupe corps ajoute tout les acteurs
  296:                          for (String corps : jeuActeur.getListeCorps()) {
  297:                                  filtreActeurs.shwoNodes(jeuActeur.getCorps(corps))
;
  298:                          }
  299:                  }
  300:          }
  301:
  302:          public TreeListener getListener() {
  303:                  return treeListener;
  304:          }
  305:
  306:          public void ajouterJeuActeur(JeuActeur jeuActeur) {
  307:                  jeuxActeur.add(jeuActeur);
  308:          }
  309:
  310:          public DefaultMutableTreeNode getRoot() {
  311:                  return (DefaultMutableTreeNode) getModel().getRoot();
  312:          }
  313:
  314:          @Override
  315:          public DefaultTreeModel getModel() {
  316:                  return (DefaultTreeModel)super.getModel();
  317:          }
  318:
  319:          public void readExternal(ObjectInput in) throws IOException,
  320:          ClassNotFoundException {
  321:          }
  322:
  323:          public void writeExternal(ObjectOutput out) throws IOException {
  324:          }
  325:
  326:          public void startEditing() {
  327:                  startEditingAtPath(getSelectionPath());
  328:          }
  329:
  330:          @Override
  331:          public void editingStopped(ChangeEvent e) {
  332:                  String newValue = ((DefaultCellEditor)e.getSource()).getCellEditor
Value().toString();
  333:                  if(getSelectedNode() instanceof ActeurBase) {
  334:                          DialogHandlerFrame.showErrorDialog("Impossible de renomer
cette acteur");
  335:                          setEditable(false);
  336:                          return;
  337:                  }
  338:                  getSelectedNode().changeStringValue(newValue);
  339:                  setEditable(false);
  340:          }
  341:
  342:          @Override
  343:          public void editingCanceled(ChangeEvent e) {
  344:                  setEditable(false);
  345:          }
  346:
  347:          public void setTriadeEditingMousePlugin(TriadeEditingMousePlugin<BrickVert
ex, BrickEdge> triadeEditingMousePlugin) {
  348:                  this.triadeEditingMousePlugin = triadeEditingMousePlugin;
  349:          }
  350:
  351:          public void setVertexSelected(Content content) {
  352:                  if(triadeEditingMousePlugin != null) {
  353:                          triadeEditingMousePlugin.selectVertex(content);
  354:                  }
  355:          }
  356:
  357:          public void addContent(Content content) {
  358:                  if(otherGroup == null) {
  359:                          otherGroup = new Groupe("Autres"); //$NON-NLS-1$
  360:                          getModel().insertNodeInto(otherGroup, getRoot(), getRoot()
.getChildCount());
  361:                  }
  362:                  getModel().insertNodeInto(new DefaultMutableTreeNode(content), oth
erGroup, otherGroup.getChildCount());
  363:          }
  364: }
```

```
    1: package dataPack;
    2:
    3: import graphicalUserInterface.Program;
    4:
    5: import java.io.Serializable;
    6: import java.util.Hashtable;
    7: import java.util.Vector;
    8:
    9: import translation.Messages;
   10:
   11: public class MoyenEntreprise implements Serializable {
   12:
   13:         private static final long serialVersionUID = -1013542159489711501L;
   14:
   15:         final static Integer DEBUTINDICEID = -50;
   16:
   17:         protected Hashtable<Integer, Moyen> moyens;
   18:
   19:         public MoyenEntreprise() {
   20:                 moyens = new Hashtable<Integer, Moyen>(0);
   21:         }
   22:
   23:         public Moyen ajouterMoyen(Integer idGenerique, String nom) {
   24:                 Integer id = new Integer(MoyenEntreprise.DEBUTINDICEID - moyens.si
ze());
   25:                 Moyen newMoyen = new Moyen(nom, id, idGenerique);
   26:                 moyens.put(id, newMoyen);
   27:
   28:                 Program.myMainFrame.getDataPack().getActorsModel().ajouterMoyen(
   29:                                 newMoyen);
   30:
   31:                 return newMoyen;
   32:         }
   33:
   34:         public void supprimerMoyen(Integer idMoyen) {
   35:
   36:                 Program.myMainFrame.getDataPack().getActorsModel()
   37:                                 .removeNodeFromParent(getMoyen(idMoyen));
   38:
   39:                 moyens.remove(idMoyen);
   40:         }
   41:
   42:         public Vector<Moyen> getMoyenVector() {
   43:                 return new Vector<Moyen>(moyens.values());
   44:         }
   45:
   46:         public Moyen getMoyen(Integer idMoyen) {
   47:                 return moyens.get(idMoyen);
   48:         }
   49:
   50:         public int getNbMoyen() {
   51:                 return moyens.size();
   52:         }
   53:
   54:         public String getName(Integer id) {
   55:                 Moyen moyen = moyens.get(id);
   56:                 if (moyen == null) {
   57:                         System.err.println("!!! getName sur un Moyen inÃ©xistant !
!!"); //$NON-NLS-1$
   58:                         return Messages.getString("MoyenEntreprise.1"); //$NON-NLS
-1$
   59:                 }
   60:
   61:                 return moyens.get(id).getNom();
   62:         }
   63: }
```

```java
package dataPack;

import java.awt.datatransfer.DataFlavor;

import javax.swing.JComponent;
import javax.swing.TransferHandler;

import translation.Messages;

public class TreeTransferHandler extends TransferHandler {
	private static final long serialVersionUID = 8808467211310175768L;

	public TreeTransferHandler() {
		super(null);
	}

	@Override
	public boolean canImport(TransferHandler.TransferSupport support) {
		System.out.println("Youpi 1"); //$NON-NLS-1$
		return true;
	}

	@Override
	public boolean canImport(JComponent comp, DataFlavor[] transferFlavors) {
		System.out.println("Youpi 2"); //$NON-NLS-1$
		return true;
	}
}
```

```
 1: package dataPack;
 2:
 3: import java.io.Serializable;
 4:
 5: public interface SavableObject extends Serializable {
 6:
 7:         public String getFilePath();
 8:
 9:         public void setFilePath(String _filePath);
10: }
```

```java
package dataPack;

import java.util.Collection;
import java.util.Enumeration;
import java.util.Vector;

import javax.swing.event.EventListenerList;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreeNode;
import javax.swing.tree.TreePath;

import translation.Messages;

public class FiltreActeursEntreprise extends DefaultTreeModel {

	/**
	 *
	 */
	private static final long serialVersionUID = 6095674535112875250L;
	private final ModelActeursEntreprise model;
	private final Vector<Object> hidedNodes;

	private transient TreeListener treeListener;

	public FiltreActeursEntreprise(ModelActeursEntreprise model) {
		super(model.getRoot());
		if(model.getRoot() == null) {
			System.out.println("\n\nRoot null !!!\n"); //$NON-NLS-1$
		}
		this.model = model;
//		model.addTreeModelListener(this);

		hidedNodes = new Vector<Object>();
		listenerList = new EventListenerList();
	}

	public TreeModelEvent convertTreeModelEvent(TreeModelEvent e) {

		Object source = e.getSource();
		TreePath path = new TreePath(e.getPath());
		Object[] children = e.getChildren();
		int[] childIndices = null;
		if (children != null) {
			childIndices = new int[children.length];
			for (int i = 0; i < childIndices.length; i++) {
				DefaultMutableTreeNode father = (DefaultMutableTreeNode) path
						.getLastPathComponent();
				childIndices[i] = e.getChildIndices()[i];

				Enumeration<DefaultMutableTreeNode> fatherChildren = father
						.children();
				int childIndiceModel = e.getChildIndices()[i];
				while (fatherChildren.hasMoreElements() && childIndiceModel > 0) {
					if (hidedNodes.contains(fatherChildren.nextElement())) {
						childIndices[i]--;
					}

					childIndiceModel--;
				}
			}
		}

		TreeModelEvent event = new TreeModelEvent(source, path, childIndices,
				children);
		return event;
	}

	public void treeNodesChanged(TreeModelEvent e) {
		TreeModelEvent event = convertTreeModelEvent(e);

		if (event.getSource() != null) {
			fireTreeNodesChanged(this, event.getPath(),
					event
					.getChildIndices(), event.getChildren());
		}
	}

	@Override
	public void addTreeModelListener(TreeModelListener listener) {
		model.addTreeModelListener(listener);
		super.addTreeModelListener(listener);
	}

	public boolean isHidedNode(DefaultMutableTreeNode node) {
		return hidedNodes.contains(node);
	}

	public boolean hideNode(DefaultMutableTreeNode node) {
		if (node == getRoot() || hidedNodes.contains(node)) {
			return false;
		}

		hidedNodes.add(node);

		if (isLeaf(node) == false) {
			Enumeration<TreeNode> childrenEnum = node.children();

			while (childrenEnum.hasMoreElements()) {
				hideNode((DefaultMutableTreeNode) childrenEnum.nextElement());
			}
		}

		DefaultMutableTreeNode father = (DefaultMutableTreeNode) node
				.getParent();
		if (father != null &&  getChildCount(father) == 0) {
			if (hideNode(father) == false) {
				reload();
			}
		} else {
			reload();
		}

		return true;
	}

	public void hideNodes(Collection<? extends DefaultMutableTreeNode> nodes) {
		if (nodes != null) {
			for (DefaultMutableTreeNode node : nodes) {
				hideNode(node);
			}
		}
	}

	public boolean shwoNode(DefaultMutableTreeNode node) {
```

```
128:                 if (node == getRoot()
129:                                 || hidedNodes.contains(node) == false)
130:                         return false;
131:
132:                 hidedNodes.remove(node);
133:
134:                 if (shwoNode((DefaultMutableTreeNode) node.getParent()) == false)
{
135:                         reload();
136:                 }
137:
138:                 return true;
139:         }
140:
141:         public void shwoNodes(Collection<? extends DefaultMutableTreeNode> nodes)
{
142:                 if (nodes != null) {
143:                         for (DefaultMutableTreeNode node : nodes) {
144:                                 shwoNode(node);
145:                         }
146:                 }
147:         }
148:
149:         @Override
150:         public Object getChild(Object parent, int index) {
151:                 int modelChildCount = model.getChildCount(parent);
152:                 int childIndex = 0;
153:                 for (int i = 0; i < modelChildCount; i++) {
154:                         Object child = model.getChild(parent, i);
155:                         if (hidedNodes.contains(child) == false) {
156:                                 if (index == childIndex) {
157:                                         return child;
158:                                 } else {
159:                                         childIndex++;
160:                                 }
161:                         }
162:                 }
163:                 return null;
164:         }
165:
166:         @Override
167:         public int getChildCount(Object parent) {
168:                 int modelChildCount = model.getChildCount(parent);
169:                 int childCount = 0;
170:                 for (int i = 0; i < modelChildCount; i++) {
171:                         Object child = model.getChild(parent, i);
172:                         if (hidedNodes.contains(child) == false) {
173:                                 childCount++;
174:                         } else {
175: //                              System.out.println("Node " + child.toString() + "
cachÃ©");
176:                         }
177:                 }
178:                 //System.out.println("childcount de " + parent.toString() + " : "
+ childCount + "(hided : " + hidedNodes + ")");
179:                 return childCount;
180:         }
181:
182:         @Override
183:         public boolean isLeaf(Object node) {
184:                 return getChildCount(node) == 0;
185:         }
186:
187:         @Override
188:         public int getIndexOfChild(Object parent, Object child) {
189:                 int modelChildCount = model.getChildCount(parent);
190:                 int childIndex = 0;
191:                 for (int i = 0; i < modelChildCount; i++) {
192:                         Object modelChild = model.getChild(parent, i);
193:
194:                         if (hidedNodes.contains(modelChild) == false) {
195:                                 if (modelChild == child) {
196:                                         return childIndex;
197:                                 }
198:                                 childIndex++;
199:                         }
200:                 }
201:                 return -1;
202:         }
203:
204:         public void setTreeListener(TreeListener treeListener) {
205:                 this.treeListener = treeListener;
206:         }
207:
208:         public TreeListener getTreeListener() {
209:                 return treeListener;
210:         }
211: }
```

```
    1: package dataPack;
    2:
    3: import graphicalUserInterface.DialogHandlerFrame;
    4: import graphicalUserInterface.MainFrameDatapackCreator;
    5: import graphicalUserInterface.Program;
    6:
    7: import java.awt.event.MouseEvent;
    8: import java.awt.event.MouseListener;
    9: import java.io.Externalizable;
   10: import java.io.IOException;
   11: import java.io.ObjectInput;
   12: import java.io.ObjectOutput;
   13: import java.util.Set;
   14: import java.util.TreeSet;
   15: import java.util.Vector;
   16:
   17: import javax.swing.JOptionPane;
   18: import javax.swing.JTree;
   19: import javax.swing.SwingUtilities;
   20: import javax.swing.tree.DefaultMutableTreeNode;
   21: import javax.swing.tree.TreePath;
   22:
   23: import models.Brick;
   24: import models.GraphListener;
   25: import translation.Messages;
   26: import client.NavigationTree;
   27:
   28: public class TreeListener implements Externalizable, MouseListener {
   29:         static private final long serialVersionUID = 123L;
   30:
   31:         protected ModelActeursEntreprise modelActeur;
   32:         protected GraphListener graphe;
   33:         protected JTreeActors tree;
   34:
   35:         private NavigationTree navigationTree;
   36:
   37:         public TreeListener() {
   38:
   39:         }
   40:
   41:         public TreeListener(ModelActeursEntreprise modelActeur, JTreeActors tree)
{
   42:                 this.modelActeur = modelActeur;
   43:                 this.tree = tree;
   44:                 graphe = null;
   45:         }
   46:
   47:         public TreeListener(NavigationTree navigationTree) {
   48:                 this.navigationTree = navigationTree;
   49:         }
   50:
   51:         @Override
   52:         public void mousePressed(MouseEvent e) {
   53:                 // TreePath courant = acteurs.getTree().getPathForLocation(e.getX(
),
   54:                 // e.getY());
   55:
   56:                 // rÃ©cupÃ©re le path du noeud sÃ©lÃ©ctionÃ©
   57:                 TreePath path = ((JTree) e.getSource()).getPathForLocation(e.getX(
), e
   58:                                 .getY());
   59:
   60:                 if (path == null) {
   61:                         // rien n'est sÃ©lÃ©ctionÃ©
   62:                         return;
   63:                 }
   64:
   65:                 // rÃ©cupÃ©re le noeud
   66:                 MyDefaultMutableTreeNode selectedNode = (MyDefaultMutableTreeNode)
 path
   67:                 .getLastPathComponent();
   68:
   69:                 if(e.getButton() != MouseEvent.BUTTON1) {
   70:                         if(!Program.isTriades()) {
   71:                                 tree.setEditable(true);
   72:                                 tree.startEditingAtPath(new TreePath(selectedNode.
getPath()));
   73:                                 return;
   74:                         }
   75:                 }
   76:
   77:                 if(graphe != null) {
   78:                         if(!MyDefaultMutableTreeNode.isGroupe(selectedNode)) {
   79:                                 System.out.println("Set content (TreeListener) : "
 + selectedNode.toString()); //$NON-NLS-1$
   80:                                 graphe.setContent(selectedNode);
   81:                         } else {
   82:                                 graphe.setContent(null);
   83:                         }
   84:                 }
   85:         }
   86:
   87:         public void setGrapheListener(GraphListener n_graphe) {
   88:                 graphe = n_graphe;
   89:         }
   90:
   91:         public void ajouterJeuAceteur(JeuActeur jeuActeur) {
   92:                 tree.ajouterJeuActeur(jeuActeur);
   93:         }
   94:
   95:         public void hideAllBases(JeuActeur jeuActeur) {
   96:                 tree.hideAllBases(jeuActeur);
   97:                 tree.validate();
   98:         }
   99:
  100:         public void hideAllCorps(JeuActeur jeuActeur) {
  101:                 tree.hideAllCorps(jeuActeur);
  102:                 tree.validate();
  103:         }
  104:
  105:         public void ajouterJeuxActeur(Vector<JeuActeur> jeuxActeur) {
  106:                 for (JeuActeur jeuActeur : jeuxActeur) {
  107:                         ajouterJeuAceteur(jeuActeur);
  108:                 }
  109:         }
  110:
  111:         public Acteur ajouterActeur(Integer statutId, String nom) {
  112:
  113:                 if (checkOpenTab() != JOptionPane.YES_OPTION)
  114:                         return null;
  115:
  116:                 MyDefaultMutableTreeNode selectedNode = tree.getSelectedNode();
  117:
  118:                 if (selectedNode == null
  119:                                 || selectedNode.isNodeAncestor(modelActeur.getGrou
peActeurs()) == false) {
  120:                         DialogHandlerFrame
  121:                         .showErrorDialog(Messages.getString("TreeListener.1")); //
$NON-NLS-1$
  122:                         return null;
  123:                 }
  124:
  125:                 Groupe fatherGroupe = null;
  126:
```

```
  127:                 if (MyDefaultMutableTreeNode.isActor(selectedNode)) {
  128:                         fatherGroupe = (Groupe) selectedNode.getParent();
  129:                 } else if (selectedNode == modelActeur.getGroupeActeurs()) {
  130:                         DialogHandlerFrame
  131:                         .showErrorDialog(Messages.getString("TreeListener.2")); //
$NON-NLS-1$
  132:                         return null;
  133:                 } else if (MyDefaultMutableTreeNode.isGroupe(selectedNode)) {
  134:                         fatherGroupe = (Groupe) selectedNode;
  135:                 } else {
  136:                         DialogHandlerFrame
  137:                         .showErrorDialog(Messages.getString("TreeListener.3")); //
$NON-NLS-1$
  138:                         return null;
  139:                 }
  140:
  141:                 Acteur newActor = new Acteur(nom, statutId, fatherGroupe.getIdGrou
pe());
  142:                 modelActeur.ajouterActeur(newActor, fatherGroupe);
  143:
  144:                 tree.expendNode(newActor);
  145:                 setSelectedNode(newActor);
  146:                 return newActor;
  147:         }
  148:
  149:         private int checkOpenTab() {
  150:                 // afin que les arbres soit bien mis a jour il faut que tout les o
nglet
  151:                 // soit fermer pour ne pas crÃ©e d'incoÃ©rence
  152:
  153:                 if (Program.myMainFrame.getTabLength().intValue() > 1)
  154:                 {
  155:                         DialogHandlerFrame.showErrorDialog("Tous les onglets vont
Ãªtre fermÃ© lors de cette action.");
  156:                         if (!Program.isTriades())
  157:                                 ((MainFrameDatapackCreator)Program.myMainFrame).cl
eanTabAndGraphs();
  158:                 }
  159:                 return JOptionPane.YES_OPTION;
  160:         }
  161:
  162:         public Content deleteActor() {
  163:                 // supprime l'acteur/brique selectionÃ©
  164:
  165:                 MyDefaultMutableTreeNode selectedNode = tree.getSelectedNode();
  166:
  167:                 if (selectedNode == null)
  168:                         return null;
  169:
  170:                 if (checkOpenTab() != JOptionPane.YES_OPTION)
  171:                         return null;
  172:
  173:                 if (selectedNode.isNodeAncestor(modelActeur.getGroupeActeurs()) ==
 true
  174:                                 && selectedNode != modelActeur.getGroupeActeurs())
 {
  175:                         if (MyDefaultMutableTreeNode.isActor(selectedNode)) {
  176:                                 Acteur actor = (Acteur) selectedNode;
  177:
  178:                                 if (DialogHandlerFrame
  179:                                                 .showYesNoDialog(Messages.getStrin
g("TreeListener.5")) != JOptionPane.YES_OPTION) //$NON-NLS-1$
  180:                                         return null;
  181:
  182:                                 DefaultMutableTreeNode newSelectedNode = (DefaultM
utableTreeNode) actor
  183:                                 .getParent();
  184:
  185:                                 // l'acteur selectionÃ© est supprimÃ©
  186:                                 modelActeur.removeNodeFromParent(actor);
  187:
  188:                                 setSelectedNode(newSelectedNode);
  189:
  190:                                 return actor;
  191:                         } else if (MyDefaultMutableTreeNode.isGroupe(selectedNode)
) {
  192:                                 Groupe groupe = (Groupe) selectedNode;
  193:                                 int confirmation = 0;
  194:
  195:                                 // si le groupe contient des acteurs ou des groupe
s on demande
  196:                                 // confirmation pour les supprimer
  197:                                 if (selectedNode.getChildCount() > 0) {
  198:                                         confirmation = DialogHandlerFrame
  199:                                         .showYesNoDialog(Messages.getString("TreeL
istener.6")); //$NON-NLS-1$
  200:                                 } else {
  201:                                         confirmation = DialogHandlerFrame
  202:                                         .showYesNoDialog(Messages.getString("TreeL
istener.7")); //$NON-NLS-1$
  203:                                 }
  204:
  205:                                 if (confirmation != JOptionPane.YES_OPTION)
  206:                                         return null;
  207:
  208:                                 DefaultMutableTreeNode newSelectedNode = (DefaultM
utableTreeNode) groupe
  209:                                 .getParent();
  210:
  211:                                 modelActeur.removeNodeFromParent(groupe);
  212:
  213:                                 setSelectedNode(newSelectedNode);
  214:
  215:                                 return groupe;
  216:                         }
  217:                 } else {
  218:                         DialogHandlerFrame
  219:                         .showErrorDialog(Messages.getString("TreeListener.8")); //
$NON-NLS-1$
  220:                 }
  221:
  222:                 return null;
  223:         }
  224:
  225:         public Groupe ajouterGroupe(String nom) {
  226:
  227:                 if (checkOpenTab() != JOptionPane.YES_OPTION)
  228:                         return null;
  229:
  230:                 MyDefaultMutableTreeNode selectedNode = tree.getSelectedNode();
  231:
  232:                 MyDefaultMutableTreeNode fatherGroupe = null;
  233:
  234:                 if (selectedNode == null) {
  235:                         int choix = DialogHandlerFrame
  236:                         .showYesNoDialog(Messages.getString("TreeListener.9")); //
$NON-NLS-1$
  237:                         if (choix == JOptionPane.YES_OPTION) {
  238:                                 fatherGroupe = modelActeur.getGroupeActeurs();
  239:                         } else {
  240:                                 return null;
  241:                         }
  242:                 } else if (selectedNode.isNodeAncestor(modelActeur.getGroupeActeur
s()) == false) {
```

```
243:                         DialogHandlerFrame
244:                         .showErrorDialog(Messages.getString("TreeListener.10")); //
/$NON-NLS-1$
245:                         return null;
246:
247:                 } else {
248:                         if (MyDefaultMutableTreeNode.isActor(selectedNode)) {
249:                                 fatherGroupe = (MyDefaultMutableTreeNode) selected
Node
250:                                 .getParent();
251:                         } else {
252:                                 fatherGroupe = selectedNode;
253:                         }
254:                 }
255:
256:                 Groupe newGroupe = new Groupe(nom);
257:                 modelActeur.ajouterGroupe(newGroupe, fatherGroupe);
258:
259:                 tree.expendNode(newGroupe);
260:                 setSelectedNode(newGroupe);
261:
262:                 return newGroupe;
263:         }
264:
265:         public void mouseReleased(MouseEvent e) {
266:
267:         }
268:
269:         public void hideActor(Content oldActiveContent) {
270:                 tree.hideActor(oldActiveContent);
271:         }
272:
273:         public void hideActors(Set<Content> actorsSet) {
274:                 for (Content actor : actorsSet) {
275:                         hideActor(actor);
276:                 }
277:         }
278:
279:         public void showActor(Content content) {
280:                 tree.showActor(content);
281:         }
282:
283:         public void showActors(Set<Content> actorsSet) {
284:                 for (Content actorId : actorsSet) {
285:                         showActor(actorId);
286:                 }
287:         }
288:
289:         public void setSelectedNode(DefaultMutableTreeNode selectedNode) {
290:                 if (Program.isTriades()) {
291:                         tree.setSelectionPath(new TreePath(selectedNode.getPath())
);
292:                         tree.validate();
293:                 }
294:         }
295:
296:         public void setTree(JTreeActors tree) {
297:                 this.tree = tree;
298:         }
299:
300:         public void setSelectedContent(Content content) {
301:                 if(content == null) {
302:                         if(tree !=null)
303:                                 tree.setSelectionPath(null);
304:                         else
305:                                 System.out.println("Tree null !!"); //$NON-NLS-1$
306:
307:                         return;
308:                 }
309:
310:                 if(content instanceof MyDefaultMutableTreeNode) {
311:                         MyDefaultMutableTreeNode node = (MyDefaultMutableTreeNode)
content;
312:                         tree.setSelectionPath(new TreePath(node.getPath()));
313:                 } else if (content instanceof Brick) {
314:                         navigationTree.setSelectedContent(content);
315:                 }
316:         }
317:
318:         public JTreeActors getTree() {
319:                 return tree;
320:         }
321:
322:         public Set<Acteur> getSelectedActors() {
323:                 MyDefaultMutableTreeNode selectedNode = tree.getSelectedNode();
324:
325:                 if (selectedNode == null)
326:                         return null;
327:
328:                 return getAllActorVisible(selectedNode);
329:         }
330:
331:         protected Set<Acteur> getAllActorVisible(MyDefaultMutableTreeNode node) {
332:                 /*
333:                  * retourne tout les acteur visible contenue dans ce groupe et de
tout
334:                  * ses sous groupe
335:                  */
336:                 TreeSet<Acteur> actors = new TreeSet<Acteur>();
337:
338:                 if (MyDefaultMutableTreeNode.isActor(node) || MyDefaultMutableTree
Node.isBase(node) || MyDefaultMutableTreeNode.isCorps(node)) {
339:                         actors.add((Acteur) node);
340:                         return actors;
341:                 } else if (MyDefaultMutableTreeNode.isGroupe(node)) {
342:                         // if
343:                         // (DialogHandlerFrame.showYesNoDialog("Tout les acteurs d
u groupe seront ajouter a la liste.\n voulez vous continuer?")
344:                         // != JOptionPane.YES_OPTION)
345:                         // return actors;
346:
347:                         // rÃ©cupÃ©re et retourne tout les acteurs du groupe
348:                         int nbChildren = tree.getModel().getChildCount(node);
349:                         for (int i = 0; i < nbChildren; i++) {
350:                                 MyDefaultMutableTreeNode child = (MyDefaultMutable
TreeNode) tree
351:                                 .getModel().getChild(node, i);
352:                                 actors.addAll(getAllActorVisible(child));
353:                         }
354:                         return actors;
355:                 } else
356:                         return null;
357:         }
358:
359:         public Acteur getSelectedActor() {
360:                 MyDefaultMutableTreeNode selectedNode = tree.getSelectedNode();
361:
362:                 if (selectedNode == null)
363:                         return null;
364:
365:                 if (MyDefaultMutableTreeNode.isActor(selectedNode)) {
366:                         return (Acteur) selectedNode;
367:                 } else
368:                         return null;
```

```
  369:         }
  370:
  371:         public void readExternal(ObjectInput in) throws IOException,
  372:         ClassNotFoundException {
  373:                 SwingUtilities.invokeLater(new Runnable() {
  374:
  375:                         @Override
  376:                         public void run() {
  377:                                 Program.myMainFrame.getDataPack().getActorsModel()
  378:                                 .cleanTreeListener();
  379:                         }
  380:                 });
  381:         }
  382:
  383:         public void writeExternal(ObjectOutput out) throws IOException {
  384:                 throw new RuntimeException("D'ou vient la presence de ce TreeListe
ner ?"); //$NON-NLS-1$
  385:         }
  386:
  387:         @Override
  388:         public void mouseClicked(MouseEvent e) {
  389:         }
  390:
  391:         @Override
  392:         public void mouseEntered(MouseEvent e) {
  393:         }
  394:
  395:         @Override
  396:         public void mouseExited(MouseEvent e) {
  397:         }
  398: }
```

```
 1: package dataPack;
 2:
 3: import models.BrickVertex.VerticeRank;
 4:
 5: public class ActeurSelectionne {
 6:
 7:         final static int NOACTIVITE = -1;
 8:
 9:         protected Acteur acteur; // indexÃ© a leur idActeur
10:
11:         protected VerticeRank rank;
12:         // protected Integer idActivite;
13:
14:         public ActeurSelectionne(Acteur acteur) {
15:                 this.acteur = acteur;
16:
17:                 rank = VerticeRank.primary;
18:
19:                 // idActivite = new Integer(NOACTIVITE);
20:         }
21:
22:         public ActeurSelectionne(Acteur acteur, VerticeRank rank) {
23:                 this.acteur = acteur;
24:                 this.rank = rank;
25:                 // idActivite = activite;
26:         }
27:
28: public void setRank(VerticeRank newRank)
29:                 {
30:         rank = newRank;
31:                 }
32:
33:         public VerticeRank getRank()
34:         {
35:                 return rank;
36:         }
37:
38:
39:         public Acteur getActeur() {
40:                 return acteur;
41:         }
42:
43:
44:         public String toString() {
45:                 String r;
46:                 if (rank==VerticeRank.primary)
47:                         r = " (acteur principal)";
48:                 else if (rank == VerticeRank.secondary)
49:                         r = " (acteur secondaire)";
50:                 else if (rank == VerticeRank.remaining)
51:                         r = " (acteur restant)";
52:                 else
53:                         r = "";
54:                 return acteur.toString()+ r;
55:         }
56:
57:
58: }
```

```java
package dataPack;

import java.util.HashMap;

import javax.swing.event.TreeModelListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.MutableTreeNode;

import translation.Messages;

/*
 * ReprÃ©sente le model des acteurs, moyens, briques et autres.
 *
 */

public class ModelActeursEntreprise extends DefaultTreeModel {

	private static final long serialVersionUID = -6046384331733497154L;

	private final Groupe groupeActeurs;
	private final Groupe groupeMoyens;

	private DataPack dataPack;

	HashMap<Integer, Acteur> allActeurs;

	public ModelActeursEntreprise(DataPack dataPack) {
		super(new Groupe("ActeursEntreprise"), true); //$NON-NLS-1$

		groupeActeurs = new GroupeRoot(Messages.getString("ModelActeursEnt
reprise.1")); //$NON-NLS-1$
		groupeMoyens = new GroupeRoot(Messages.getString("ModelActeursEntr
eprise.2")); //$NON-NLS-1$

		insertNodeInto(groupeActeurs, getRoot(), getRoot().getChildCount()
);
		insertNodeInto(groupeMoyens, getRoot(), getRoot().getChildCount())
;

		this.dataPack = dataPack;

		allActeurs = new HashMap<Integer, Acteur>();
	}

	public void ajouterActeur(Acteur acteur, MutableTreeNode father) {
		// possibilitÃ© d'ajouter les acteurpar ordre alphabÃ©tique
		insertNodeInto(acteur, father, 0);

		allActeurs.put(acteur.getIdActeur(), acteur);
	}

	public void ajouterGroupe(Groupe groupe, MutableTreeNode father) {
		insertNodeInto(groupe, father, father.getChildCount());
	}

	public void ajouterMoyen(Moyen moyen) {
		insertNodeInto(moyen, groupeMoyens, groupeMoyens.getChildCount());
	}

	public void ajouterBase(ActeurBase base, JeuActeur jeuActeur) {
		DefaultMutableTreeNode groupeBase = jeuActeur.getGroupeBase();
		insertNodeInto(base, groupeBase, 0);

		allActeurs.put(base.getIdActeur(), base);
	}

	public void ajouterCorps(ActeurBase corps, JeuActeur jeuActeur) {
		DefaultMutableTreeNode groupeCorps = jeuActeur.getGroupeCorps(corp
s.getCorpsMetier());
		insertNodeInto(corps,
				groupeCorps, groupeCorps
					.getChildCount());

		allActeurs.put(corps.getIdActeur(), corps);
	}

	public void supprimerAnyActeur(Acteur acteur) {
		removeNodeFromParent(acteur);
	}

	public void supprimerGroupe(Groupe groupe) {
		removeNodeFromParent(groupe);
	}

	public void supprimerMoyen(Moyen moyen) {
		removeNodeFromParent(moyen);
	}

	@Override
	public DefaultMutableTreeNode getRoot() {
		return (DefaultMutableTreeNode)super.getRoot();
	}

	public Groupe getGroupeMoyens() {
		return groupeMoyens;
	}

	public Groupe getGroupeActeurs() {
		return groupeActeurs;
	}

	@Override
	public DefaultMutableTreeNode getChild(Object parent, int index) {
		return (DefaultMutableTreeNode) super.getChild(parent, index);
	}

	public void setDataPack(DataPack dataPack) {
		this.dataPack = dataPack;
	}

	public DataPack getDataPack() {
		return dataPack;
	}

	public String getActorName(Integer idActeur) {
		return allActeurs.get(idActeur).toString();
	}

	public void cleanTreeListener() {
		//TODO a virer
	for(TreeModelListener listener : getListeners(TreeModelListener.class)) {
		removeTreeModelListener(listener);
	}
	}

	public boolean isRacine(DefaultMutableTreeNode node) {
		return node == groupeActeurs || node == groupeMoyens;
	}

	public boolean isBase(Integer actorId) {
		Acteur acteur = allActeurs.get(actorId);
		if (acteur instanceof ActeurBase) {
			return ((ActeurBase)acteur).isBase();
```

```
  130:                 }
  131:
  132:                 return false;
  133:         }
  134:
  135:         public JeuActeur getJeuActeurOfGenericActor(Integer actorGenericId) {
  136:                 Acteur actor = allActeurs.get(actorGenericId);
  137:                 if (actor instanceof ActeurBase) {
  138:                         ActeurBase genericActor = (ActeurBase)actor;
  139:                         if (genericActor.isBase()) {
  140:                                 return genericActor.getJeuActeur();
  141:                         }
  142:                 }
  143:
  144:                 throw new IllegalArgumentException("Cet acteur n’est pas une base
!!"); //$NON-NLS-1$
  145:         }
  146:
  147:         @Override
  148:         public void reload() {
  149:                 super.reload();
  150:         }
  151: }
  152:
```

```
    1: package dataPack;
    2:
    3: import java.awt.Component;
    4: import java.awt.Dimension;
    5:
    6: import javax.swing.DefaultCellEditor;
    7: import javax.swing.JTextField;
    8: import javax.swing.JTree;
    9:
   10: import translation.Messages;
   11:
   12: public class TreeCellEditor extends DefaultCellEditor {
   13:         private static final long serialVersionUID = -6234212637865246632L;
   14:
   15:         public TreeCellEditor() {
   16:                 super(new JTextField());
   17:         }
   18:
   19:         @Override
   20:         public Component getTreeCellEditorComponent(JTree tree, Object value,
   21:                         boolean isSelected, boolean expanded, boolean leaf, int ro
w) {
   22:                 if (value instanceof MyDefaultMutableTreeNode) {
   23:                         JTextField result = (JTextField) super.getTreeCellEditorCo
mponent(
   24:                                         tree, value, isSelected, expanded, leaf, r
ow);
   25:
   26:                         if (value instanceof Acteur) {
   27:                                 result.setText(((Acteur) value).poste);
   28:                         } else if (value instanceof Moyen) {
   29:                                 result.setText(((Moyen) value).nom);
   30:                         } else {
   31:                                 result.setText(value.toString());
   32:                         }
   33:
   34:                         result.setPreferredSize(new Dimension(200, 25));
   35:                         result.validate();
   36:
   37:                         return result;
   38:                 } else {
   39:                         throw new IllegalStateException(
   40:                                         "Un Ã©lÃ©ment de l’arbre n’est pas un MyDe
faultMutableTreeNode"); //$NON-NLS-1$
   41:                 }
   42:         }
   43: }
```

```java
package dataPack;

import graphicalUserInterface.IconDatabase;
import graphicalUserInterface.Program;

import java.io.Serializable;

import javax.swing.Icon;
import javax.swing.JLabel;

import main.StatutObjet;
import translation.Messages;
import client.stringTranslator.StringTranslator;

public class Moyen extends MyDefaultMutableTreeNode implements Serializable,
		Content {

	/**
	 *
	 */
	private static final long serialVersionUID = 3631023820177794653L;

	protected String nom;
	protected Integer idMoyen;
	protected Integer idGenerique;

	public Moyen(String nom, Integer id, Integer generique) {
		this.nom = nom;
		idMoyen = id;
		idGenerique = generique;
	}

	@Override
	public Integer getId() {
		return idMoyen;
	}

	public Integer getIdGenerique() {
		return idGenerique;
	}

	public String getNom() {
		return nom;
	}

	public void setNom(String nouveauNom) {
		nom = nouveauNom;
	}

	public String getNomGenerique() {
		if (idGenerique == null || idGenerique.intValue() < 0
				|| idGenerique.intValue() > StatutObjet.values().length) {
			return Messages.getString("Moyen.0") + idGenerique + Messages.getString("Moyen.1"); //$NON-NLS-1$ //$NON-NLS-2$
		} else
			return StatutObjet.values()[idGenerique.intValue()].toString();
	}

	@Override
	public JLabel getJComponent(boolean selected, boolean expanded, boolean leaf,
			int row, boolean hasFocus) {
		return TreeActorsCellRendere.createDefaultLabel(toString(), getIcon(), selected);
	}

	@Override
	public Icon getIcon() {
		return IconDatabase.vectorIconMoyensMin.get(idGenerique.intValue()
				% IconDatabase.vectorIconMoyensMin.size());
	}

	@Override
	public MyTreeNodeType getType() {
		return MyTreeNodeType.MoyenType;
	}

	@Override
	public void changeStringValue(String newString) {
		if (newString.compareTo(this.toString()) != 0) {
			setNom(newString);
		}
	}

	@Override
	public boolean equals(Object object) {
		if (object instanceof Moyen)
			return ((Moyen) object).getId().equals(idMoyen);
		return false;
	}

	public String getNoTranslatedString() {
		return nom;
	}

	@Override
	public String toString() {
		if (Program.isTriades())
			return StringTranslator.getTranslatedString(this, StringTranslator.StringType.moyenType);
		else
			return StringTranslator.getTranslatedString(this, StringTranslator.StringType.moyenType) + " (" + getNomGenerique() + ")"; //$NON-NLS-1$ //$NON-NLS-2$
	}
}
```

```
    1: package dataPack;
    2:
    3: import graphicalUserInterface.DataPackView;
    4:
    5: import java.awt.GridLayout;
    6: import java.awt.event.ActionEvent;
    7: import java.awt.event.ActionListener;
    8:
    9: import javax.swing.Box;
   10: import javax.swing.BoxLayout;
   11: import javax.swing.JButton;
   12: import javax.swing.JFrame;
   13: import javax.swing.JLabel;
   14: import javax.swing.JPanel;
   15: import javax.swing.JTextField;
   16:
   17: public class CompanyInfoView extends JFrame {
   18:         private static final long serialVersionUID = -4610853408447255737L;
   19:
   20:         public CompanyInfoView(final CompanyInfo info, final DataPackView datapack
View) {
   21:                 super();
   22:
   23:                 setTitle("Informations sur l’entreprise");
   24:                 setSize(400, 300);
   25:                 setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
   26:
   27:                 JLabel nameLabel = new JLabel("Nom");
   28:                 JLabel phoneLabel = new JLabel("TÃ©lÃ©phone");
   29:                 JLabel emailLabel = new JLabel("Mail");
   30:                 JLabel managerLabel = new JLabel("Responsable");
   31:
   32:                 final JTextField nameField = new JTextField(info.getName());
   33:                 final JTextField phoneField = new JTextField(info.getPhone());
   34:                 final JTextField emailField = new JTextField(info.getEmail());
   35:                 final JTextField managerField = new JTextField(info.getManager());
   36:
   37:                 final CompanyInfoView access = this;
   38:
   39:                 JButton okButton = new JButton("Valider");
   40:                 JButton cancelButton = new JButton("Annuler");
   41:
   42:                 okButton.addActionListener(new ActionListener() {
   43:                         @Override
   44:                         public void actionPerformed(ActionEvent e) {
   45:                                 info.setEmail(emailField.getText());
   46:                                 info.setManager(managerField.getText());
   47:                                 info.setPhone(phoneField.getText());
   48:                                 info.setName(nameField.getText());
   49:                                 datapackView.updateView();
   50:                                 access.setVisible(false);
   51:                         }
   52:                 });
   53:
   54:                 cancelButton.addActionListener(new ActionListener() {
   55:                         @Override
   56:                         public void actionPerformed(ActionEvent e) {
   57:                                 access.setVisible(false);
   58:                         }
   59:                 });
   60:
   61:                 JPanel infoPanel = new JPanel(new GridLayout(0, 2));
   62:                 JPanel buttonsPanel = new JPanel();
   63:
   64:                 infoPanel.add(nameLabel);
   65:                 infoPanel.add(nameField);
   66:                 infoPanel.add(phoneLabel);
   67:                 infoPanel.add(phoneField);
   68:                 infoPanel.add(emailLabel);
   69:                 infoPanel.add(emailField);
   70:                 infoPanel.add(managerLabel);
   71:                 infoPanel.add(managerField);
   72:
   73:                 buttonsPanel.add(okButton);
   74:                 buttonsPanel.add(cancelButton);
   75:
   76:                 JPanel panel = new JPanel();
   77:                 panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
   78:
   79:                 panel.add(infoPanel);
   80:                 panel.add(Box.createVerticalGlue());
   81:                 panel.add(buttonsPanel);
   82:
   83:                 add(panel);
   84:
   85:                 validate();
   86:                 setVisible(true);
   87:         }
   88:
   89: }
```

```
 1: package dataPack;
 2:
 3: public class GroupeJeuActeur extends Groupe {
 4:         private static final long serialVersionUID = 7272453862281473980L;
 5:
 6:         protected JeuActeur jeuActeur;
 7:
 8:         GroupeJeuActeur(String nNom, JeuActeur jeuActeur) {
 9:                 super(nNom);
10:                 this.jeuActeur = jeuActeur;
11:         }
12:
13:         @Override
14:         public void changeStringValue(String newString) {
15:                 jeuActeur.modifierCorps(nom, newString);
16:         }
17: }
```

```
    1: package dataPack;
    2:
    3: import java.awt.Color;
    4: import java.awt.Component;
    5:
    6: import javax.swing.Icon;
    7: import javax.swing.JLabel;
    8: import javax.swing.JTree;
    9: import javax.swing.SwingConstants;
   10: import javax.swing.tree.DefaultTreeCellRenderer;
   11:
   12: import translation.Messages;
   13:
   14: public class TreeActorsCellRendere extends DefaultTreeCellRenderer {
   15:
   16:         /**
   17:          *
   18:          */
   19:         private static final long serialVersionUID = -1296032005781927796L;
   20:
   21:         @Override
   22:         public Component getTreeCellRendererComponent(JTree tree, Object value,
   23:                         boolean selected, boolean expanded, boolean leaf, int row,
   24:                         boolean hasFocus) {
   25:
   26:                 if(value instanceof MyDefaultMutableTreeNode) {
   27:                         MyDefaultMutableTreeNode myNode = (MyDefaultMutableTreeNod
e) value;
   28:                         return myNode
   29:                                         .getJComponent(selected, expanded, leaf, r
ow, hasFocus);
   30:                 } else {
   31:                         throw new IllegalStateException("Un Ã©lÃ©ment de l'arbre n
'est pas un MyDefaultMutableTreeNode"); //$NON-NLS-1$
   32:                 }
   33:         }
   34:
   35:         static public JLabel createDefaultLabel(String labelText, Icon icon, boole
an selected) {
   36:                 JLabel label = new JLabel(labelText, icon, SwingConstants.LEFT);
   37:                 label.setSize(label.getSize().width, label.getSize().height + 100)
;
   38:                 label.setBackground(selected ? new Color(208, 227, 252) : Color.WH
ITE);
   39:                 label.setOpaque(true);
   40:                 return label;
   41:         }
   42: }
```

```
1: package dataPack;
2:
3: public class ActeursShema {
4:
5: }
```

```java
package dataPack;

import graphicalUserInterface.Program;

import java.io.Serializable;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;
import java.util.Vector;
import java.util.Map.Entry;

import javax.swing.tree.DefaultMutableTreeNode;

import translation.Messages;

public class JeuActeur implements Serializable {

	/**
	 *
	 */
	private static final long serialVersionUID = 5603838022343558666L;
	protected Vector<String> corpsMetier;
	protected Vector<ActeurBase> bases;
	protected Map<String, Vector<ActeurBase>> acteurs;
	protected String nom;

	protected Groupe groupeJeuActeur;
	protected Map<String, GroupeJeuActeur> groupesCorps;
	ModelActeursEntreprise modelActeur;

	public JeuActeur(String _nom) {
		nom = _nom;
		corpsMetier = new Vector<String>();
		bases = new Vector<ActeurBase>();
		acteurs = new TreeMap<String, Vector<ActeurBase>>();

		groupeJeuActeur = new GroupeRoot(_nom);
		groupesCorps = new HashMap<String, GroupeJeuActeur>();
		modelActeur = Program.myMainFrame.getDataPack().getActorsModel();
		modelActeur.ajouterGroupe(groupeJeuActeur, modelActeur.getRoot());
	}

	public void ajouterBase(String nom, Integer idStatut) {
		ActeurBase nouvelleBase = new ActeurBase(nom, idStatut, this);

		for (Entry<String, Vector<ActeurBase>> corps : acteurs.entrySet()) {

			ActeurBase nouveauCorp = new ActeurBase(corps.getKey(),
					nouvelleBase);
			corps.getValue().add(nouveauCorp);

			modelActeur.ajouterCorps(nouveauCorp, this);
		}

		bases.add(nouvelleBase);

		modelActeur.ajouterBase(nouvelleBase, this);
	}

	public void supprimerBase(ActeurBase oldBase) {
		int index = bases.indexOf(oldBase);
		if (index != -1) {
			for (Entry<String, Vector<ActeurBase>> corps : acteurs.entrySet()) {
				Vector<ActeurBase> vecteurActeur = corps.getValue();
				if (vecteurActeur.elementAt(index).getBase().equals(oldBase)) {
					vecteurActeur.remove(index);

					modelActeur.removeNodeFromParent(vecteurActeur
							.elementAt(index));
				} else {
					for (ActeurBase acteur : vecteurActeur) {
						if (acteur.getBase().equals(oldBase)) {
							vecteurActeur.remove(acteur);
							modelActeur.removeNodeFromParent(acteur);
							break;
						}
					}
				}

			}
			bases.remove(index);

			modelActeur.removeNodeFromParent(oldBase);
		} else {
			throw new IllegalArgumentException();
		}
	}

	public void ajouterCorps(String nom) {
		corpsMetier.add(nom);
		Vector<ActeurBase> nouveauCorps = new Vector<ActeurBase>();

		GroupeJeuActeur newGroupeCorps = new GroupeJeuActeur(nom, this);
		groupesCorps.put(nom, newGroupeCorps);
		modelActeur.ajouterGroupe(newGroupeCorps, groupeJeuActeur);

		for (ActeurBase acteurBase : bases) {
			ActeurBase newCorps = new ActeurBase(nom, acteurBase);
			nouveauCorps.add(newCorps);

			modelActeur.ajouterCorps(newCorps, this);
		}
		acteurs.put(nom, nouveauCorps);
	}

	public boolean modifierCorps(String oldCorps, String newCorps) {
		Vector<ActeurBase> acteursCorps = acteurs.get(oldCorps);
		if (acteursCorps == null)
			return false;

		for (ActeurBase acteur : acteursCorps) {
			acteur.setCorpsMetier(newCorps);

			modelActeur.nodeChanged(acteur);
		}

		int index = corpsMetier.indexOf(oldCorps);
		corpsMetier.remove(index);
		corpsMetier.add(index, newCorps);

		acteurs.remove(oldCorps);
		acteurs.put(newCorps, acteursCorps);

		GroupeJeuActeur groupeCorps = getGroupeCorps(oldCorps);
		groupesCorps.remove(groupeCorps);
		groupeCorps.setNom(newCorps);
```

```
127:                groupesCorps.put(newCorps, groupeCorps);
128:
129:                return true;
130:        }
131:
132:        public void supprimerCorps(String nom) {
133:                if (acteurs.get(nom) == null) {
134:                        throw new IllegalArgumentException();
135:                }
136:                acteurs.remove(nom);
137:
138:                DefaultMutableTreeNode groupeCorps = groupesCorps.get(nom);
139:                modelActeur.removeNodeFromParent(groupeCorps);
140:                groupesCorps.remove(nom);
141:        }
142:
143:        public void computeName() {
144:                for (Entry<String, Vector<ActeurBase>> entry : acteurs.entrySet())
 {
145:                        for (ActeurBase aB : entry.getValue()) {
146:                                if (aB.getPoste() == null || !aB.manuallyRenamed)
{
147:                                        aB.setPoste(aB.base.toString() + " " + aB.
corpsMetier); //$NON-NLS-1$
148:                                        aB.manuallyRenamed = false;
149:                                }
150:                        }
151:                }
152:        }
153:
154:
155:        public ActeurBase getRealActor(ActeurBase base, String nomCorps) {
156:                Vector<ActeurBase> selectedCorps = acteurs.get(nomCorps);
157:                for (ActeurBase acteurBase : selectedCorps) {
158:                        if (acteurBase.getBase().equals(base)) {
159:                                return acteurBase;
160:                        }
161:                }
162:
163:                return null;
164:
165:        }
166:
167:        @Override
168:        public String toString() {
169:                return nom;
170:        }
171:
172:        public Vector<ActeurBase> getCorps(String name) {
173:                return acteurs.get(name);
174:        }
175:
176:        public GroupeJeuActeur getGroupeCorps(String name) {
177:                return groupesCorps.get(name);
178:        }
179:
180:        public Collection<GroupeJeuActeur> getGroupesCorps() {
181:                return groupesCorps.values();
182:        }
183:
184:        public Groupe getGroupeBase() {
185:                return groupeJeuActeur;
186:        }
187:
188:        public Groupe getGroupeJeuActeur() {
189:                return groupeJeuActeur;
190:        }
191:
192:        public Vector<String> getListeCorps() {
193:                return corpsMetier;
194:        }
195:
196:        public Vector<ActeurBase> getListeBase() {
197:                return bases;
198:        }
199:
200:        public void setNom(String nouveauNom) {
201:                nom = nouveauNom;
202:        }
203:
204:        public Integer getCorpsId(String nomCorps, Integer baseId) {
205:                Vector<ActeurBase> corps = getCorps(nomCorps);
206:
207:                if (corps == null) {
208:                        throw new IllegalArgumentException(nomCorps + " n'est pas
un corps de ce jeu d'acteur"); //$NON-NLS-1$
209:                }
210:
211:                for (ActeurBase actor : corps) {
212:                        if (actor.getBase().getIdActeur().equals(baseId)) {
213:                                return actor.getIdActeur();
214:                        }
215:                }
216:
217:                throw new IllegalArgumentException("Impossible de trouver le corps
 de cette base : nomCorps = " + nomCorps + ", baseId = " + baseId); //$NON-NLS-1$ //$NON-
NLS-2$
218:        }
219:
220:        public int getBaseIndexByName(String baseName) {
221:                for (int i = 0; i < bases.size(); i++) {
222:
223:                        if (bases.elementAt(i).getPoste().equals(baseName))
224:                                return i;
225:                }
226:                return -1;
227:        }
228: }
```

```
 1: package dataPack;
 2:
 3: import java.io.Serializable;
 4:
 5:
 6: public class CompanyInfo implements Serializable {
 7:         private static final long serialVersionUID = 6874655645781201215L;
 8:
 9:         private String name;
10:         private String email;
11:         private String phone;
12:         private String manager;
13:
14:         public String getName() {
15:                 return name;
16:         }
17:
18:         public String getPhone() {
19:                 return phone;
20:         }
21:
22:         public String getEmail() {
23:                 return email;
24:         }
25:
26:         public String getManager() {
27:                 return manager;
28:         }
29:
30:         public void setName(String name) {
31:                 this.name = name;
32:         }
33:
34:         public void setEmail(String email) {
35:                 this.email = email;
36:         }
37:
38:         public void setPhone(String phone) {
39:                 this.phone = phone;
40:         }
41:
42:         public void setManager(String manager) {
43:                 this.manager = manager;
44:         }
45: }
```

```
    1: package dataPack;
    2:
    3: import java.io.Serializable;
    4:
    5: import client.stringTranslator.StringTranslator;
    6:
    7: public class Activite implements Serializable, Content {
    8:
    9:         private static final long serialVersionUID = 5295995249408229062L;
   10:
   11:
   12:         protected String nom;
   13:
   14:         protected int iconId;
   15:
   16:         @SuppressWarnings("unused")
   17:         private Activite() {
   18:                 this(null, -1);
   19:         }
   20:
   21:         public Activite(String nom, int iconId) {
   22:                 this.nom = nom;
   23:                 this.iconId = iconId;
   24:         }
   25:
   26:         public int getIconId() {
   27:                 return iconId;
   28:         }
   29:
   30:         public void setNom(String nom) {
   31:                 this.nom = nom;
   32:         }
   33:
   34:         @Override
   35:         public boolean equals(Object object) {
   36:                 if (object instanceof Activite)
   37:                         return ((Activite) object).nom.equals(nom);
   38:
   39:                 return false;
   40:         }
   41:
   42:         public String getNoTranslatedString() {
   43:                 return nom;
   44:         }
   45:
   46:         @Override
   47:         public String toString() {
   48:                 return StringTranslator.getTranslatedString(this, StringTranslator
.StringType.activityType);
   49:         }
   50: }
```

```
 1: package dataPack;
 2:
 3: import java.util.Vector;
 4:
 5: import models.BrickVertex.VerticeRank;
 6:
 7: public class ListeActeurSelectionne {
 8:
 9:         protected Vector<ActeurSelectionne> acteurs;
10:
11:         public ListeActeurSelectionne() {
12:                 acteurs = new Vector<ActeurSelectionne>(5);
13:         }
14:
15:         public void ajouterActeurSelectionne(ActeurSelectionne acteur) {
16:                 if (acteur == null)
17:                         return;
18:                 else
19:                         acteurs.add(acteur);
20:         }
21:
22:
23:         public void ajouterActeurSelectionne(Acteur newActor, VerticeRank rank) {
24:                 ActeurSelectionne nouveau = new ActeurSelectionne(newActor, rank);
25:                 acteurs.add(nouveau);
26:         }
27:
28:         public void supprimerActeur(Integer idActeur) {
29:                 acteurs.remove(idActeur);
30:         }
31:
32:         public Vector<ActeurSelectionne> getActorsSelection() {
33:                 return acteurs;
34:         }
35:
36: }
```

```
    1: package dataPack;
    2:
    3: import graphicalUserInterface.Program;
    4:
    5: import javax.swing.Icon;
    6: import javax.swing.JComponent;
    7: import javax.swing.JLabel;
    8: import javax.swing.tree.DefaultMutableTreeNode;
    9:
   10: public abstract class MyDefaultMutableTreeNode extends DefaultMutableTreeNode impl
ements Content {
   11:
   12:         /**
   13:          *
   14:          */
   15:         private static final long serialVersionUID = -5695300567627123042L;
   16:
   17:         public enum MyTreeNodeType {
   18:                 ActorType, GroupeType, ActiviteType, MoyenType, BaseType, CorpsTyp
e, BrickType
   19:         };
   20:
   21:         public abstract JLabel getJComponent(boolean selected, boolean expanded,
   22:                         boolean leaf, int row, boolean hasFocus);
   23:
   24:         public abstract Icon getIcon();
   25:         public abstract Integer getId();
   26:         public abstract MyTreeNodeType getType();
   27:
   28:         public abstract void changeStringValue(String newString);
   29:
   30:         public JComponent getJcomComponent() {
   31:                 return getJComponent(false, false, false, 0, false);
   32:         }
   33:
   34:         static public boolean isGroupe(MyDefaultMutableTreeNode myNode) {
   35:                 return myNode.getType() == MyTreeNodeType.GroupeType;
   36:         }
   37:
   38:         static public boolean isActor(MyDefaultMutableTreeNode myNode) {
   39:                 return myNode.getType() == MyTreeNodeType.ActorType;
   40:         }
   41:
   42:         static public boolean isActivite(MyDefaultMutableTreeNode myNode) {
   43:                 return myNode.getType() == MyTreeNodeType.ActiviteType;
   44:         }
   45:
   46:         static public boolean isMoyen(MyDefaultMutableTreeNode myNode) {
   47:                 return myNode.getType() == MyTreeNodeType.MoyenType;
   48:         }
   49:
   50:         static public boolean isBrick(MyDefaultMutableTreeNode myNode) {
   51:                 return myNode.getType() == MyTreeNodeType.BrickType;
   52:         }
   53:
   54:         static public boolean isBase(MyDefaultMutableTreeNode myNode) {
   55:                 return myNode.getType() == MyTreeNodeType.BaseType;
   56:         }
   57:
   58:         static public boolean isCorps(MyDefaultMutableTreeNode myNode) {
   59:                 return myNode.getType() == MyTreeNodeType.CorpsType;
   60:         }
   61:
   62:         static public ActeurBase isBaseCorps(MyDefaultMutableTreeNode myNode) {
   63:                 if (isBase(myNode) || isCorps(myNode)) {
   64:                         return (ActeurBase) myNode;
   65:                 } else {
   66:                         return null;
   67:                 }
   68:         }
   69:
   70:         static public ActeurBase isBaseCorps(Integer idActor) {
   71:                 return isBaseCorps(Program.myMainFrame.getDataPack().getActor(idAc
tor));
   72:         }
   73: }
```

```java
package dataPack;

import translation.Messages;
import client.stringTranslator.StringTranslator;
import client.stringTranslator.StringTranslator.StringType;


public class ActeurBase extends Acteur {
	private static final long serialVersionUID = 1L;

	private final static int BASE_GROUPE_ID = 20;
	private final static int ACTEUR_MILIEU_GROUPE_ID = 30;

	protected JeuActeur jeuActeur;
	protected String corpsMetier;
	protected ActeurBase base;
	protected boolean manuallyRenamed;

	private ActeurBase(String n_poste, Integer statut, Integer groupe,
			JeuActeur jeuActeur, String corpsMetier, ActeurBase base)
{
		super(n_poste, statut, groupe);

		this.base = base;
		this.corpsMetier = corpsMetier;
		this.jeuActeur = jeuActeur;
		manuallyRenamed = false;
	}

	/**
	 * Base
	 *
	 * @param poste
	 * @param statut
	 * @param jeuActeur
	 */
	public ActeurBase(String poste, Integer statut, JeuActeur jeuActeur) {
		this(poste, statut, BASE_GROUPE_ID, jeuActeur, null, null);
	}

	/**
	 * Corps
	 *
	 * @param corpsMetier
	 *            : corps de l'acteur
	 * @param base
	 *            : Base de l'acteur
	 */
	public ActeurBase(String corpsMetier, ActeurBase base) {
		this(null, base.getIdStatut(), ACTEUR_MILIEU_GROUPE_ID, base.jeuAc
teur,
				corpsMetier, base);
	}

	public ActeurBase getBase() {
		return base;
	}

	public boolean equals(ActeurBase acteurBase) {
		return super.equals(acteurBase) && jeuActeur == acteurBase.jeuActe
ur
				&& ((corpsMetier == null &&  acteurBase.corpsMetie
r == null) || (
				   corpsMetier != null && corpsMetier.equals(acteu
rBase.corpsMetier)))
				&& base == acteurBase.base;
	}

	public boolean isBase() {
		return base == null;
	}

	@Override
	public MyTreeNodeType getType() {
		if (isBase()) {
			return MyTreeNodeType.BaseType;
		} else {
			return MyTreeNodeType.CorpsType;
		}
	}

	public JeuActeur getJeuActeur() {
		return jeuActeur;
	}

	public String getCorpsMetier() {
		return corpsMetier;
	}

	public void setCorpsMetier(String corpsMetier) {
		this.corpsMetier = corpsMetier;
	}

	@Override
	public void setPoste(String newPoste) {
		manuallyRenamed = true;
		super.setPoste(newPoste);
	}

	@Override
	public void changeStringValue(String newString) {
		setPoste(newString);
	}

	@Override
	public Integer getIdStatut() {
		if (isBase())
			return idStatut;
		else
			return base.getIdStatut();
		// TODO VÃ©rifier que le bug a bien Ã©tÃ© corrigÃ© (relation entre
acteur
		// gÃ©nÃ©rique)
	}

	@Override
	public String getNoTranslatedString() {
		if (poste != null)
			return poste;

		if (base == null) {
			return Messages.getString("ActeurBase.0"); //$NON-NLS-1$
		} else {
			return base + " " + corpsMetier; //$NON-NLS-1$
		}
	}

	@Override
	public String toString() {
		return StringTranslator.getTranslatedString(this, StringType.actor
Type);
	}
}
```

```
  1: package dataPack;
  2:
  3: import graphicalUserInterface.DialogHandlerFrame;
  4: import graphicalUserInterface.Program;
  5:
  6: import java.util.ArrayList;
  7: import java.util.Collection;
  8: import java.util.HashMap;
  9: import java.util.Vector;
 10:
 11: import javax.swing.JOptionPane;
 12:
 13: import main.ActionTime;
 14: import main.ActionTimeListe;
 15: import main.JeuRelation;
 16: import main.RelationComplete;
 17: import models.Brick;
 18: import models.BrickEdge;
 19: import models.BrickVertex;
 20: import relations.EnsembleRelation;
 21: import relations.Mean;
 22: import relations.RelationPossibility;
 23: import translation.Messages;
 24: import client.Session;
 25: import client.export.ExportedDatapackModule;
 26: import client.stringTranslator.StringTranslator;
 27:
 28: public class DataPack implements SavableObject {
 29:
 30:         /**
 31:          *
 32:          */
 33:         private static final long serialVersionUID = 5641347596624328060L;
 34:
 35:         protected Vector<Brick> brickList;
 36:         // protected ActeursEntreprise actors;
 37:         protected ActionTimeListe actionTimeListe;
 38:         protected Vector<String> steps;
 39:
 40:         protected ActivitesEntreprise activities;
 41:         protected MoyenEntreprise moyens;
 42:         protected String name;
 43:         protected String filePath;
 44:         protected Vector<JeuActeur> jeuxActeur;
 45:         protected Integer newActorId;
 46:         protected Integer newGroupeId;
 47:         protected ModelActeursEntreprise actorsModel;
 48:         protected EnsembleRelation relationSet;
 49:         protected CompanyInfo companyInfo;
 50:         protected transient FiltreActeursEntreprise dataPackActorsView;
 51:
 52:         protected transient Session currentSession;
 53:
 54:         protected ExportedDatapackModule exportModule;
 55:
 56:         protected HashMap<Integer, MyDefaultMutableTreeNode> allActors;
 57:         protected StringTranslator stringTranslator;
 58:
 59:         public DataPack(String _name) {
 60:                 newActorId = new Integer(0);
 61:                 newGroupeId = new Integer(0);
 62:                 actorsModel = new ModelActeursEntreprise(this);
 63:                 brickList = new Vector<Brick>();
 64:                 activities = new ActivitesEntreprise();
 65:                 moyens = new MoyenEntreprise();
 66:                 actionTimeListe = new ActionTimeListe();
 67:                 jeuxActeur = new Vector<JeuActeur>();
 68:                 steps = new Vector<String>();
 69:                 newActorId = new Integer(0);
 70:                 name = _name;
 71:                 filePath = null;
 72:                 allActors = new HashMap<Integer, MyDefaultMutableTreeNode>();
 73:                 dataPackActorsView = null;
 74:                 relationSet = new EnsembleRelation();
 75:                 exportModule = null;
 76:                 companyInfo = new CompanyInfo();
 77:                 stringTranslator = new StringTranslator();
 78:         }
 79:
 80:         public void init() {
 81:                 actorsModel.setDataPack(this);
 82:         }
 83:
 84:         public void addAnActor(Acteur newNode) {
 85:                 allActors.put(newNode.getIdActeur(), newNode);
 86:         }
 87:
 88:
 89:
 90:         public Activite addActivity(String nom, int iconId) {
 91:                 return activities.ajouterActivite(nom, iconId);
 92:         }
 93:
 94:         public boolean renameActivity(String oldName, String newName)
 95:         {
 96:                 return activities.renameActivity(oldName,newName);
 97:         }
 98:
 99:         public Moyen addMoyen(String nom, Integer idGenerique) {
100:                 if (idGenerique != null) {
101:                         Moyen newMoyen = moyens.ajouterMoyen(idGenerique, nom);
102:                         allActors.put(newMoyen.getId(), newMoyen);
103:                         return newMoyen;
104:                 }
105:                 return null;
106:         }
107:
108:         public void addActionTime(String name) {
109:                 ActionTime newActionTime = actionTimeListe.addActionTime(name);
110:                 for (Brick brick : brickList) {
111:                         for (BrickEdge edge : brick.getEdges()) {
112:                                 edge.getCompleteRelation().addJeuRelation(newActio
nTime);
113:                         }
114:                 }
115:         }
116:
117:         public boolean removeActionTime(ActionTime actionTime) {
118:                 int index = actionTimeListe.values().indexOf(actionTime);
119:                 boolean ok = true;
120:                 for (Brick brick : getBrickList()) {
121:                         for (BrickEdge edge : brick.getEdges()) {
122:                                 RelationComplete relation = edge.getCompleteRelati
on();
123:                                 if (!relation.getJeuRelation(index).isEmpty(
124:                                                 RelationPossibility.RELATIONSTRUCT
URELLE))
125:                                 {
126:                                         ok = false;
127:                                         break;
128:                                 }
129:                         }
130:                 }
131:
```

```
  132:                if (!ok && DialogHandlerFrame.showYesNoCancelDialog(Messages.getSt
ring("DataPack.0")) != JOptionPane.YES_OPTION) //$NON-NLS-1$
  133:                        return false;
  134:
  135:
  136:                for (Brick brick : getBrickList()) {
  137:                        for (BrickEdge edge : brick.getEdges()) {
  138:                                RelationComplete relation = edge.getCompleteRelati
on();
  139:                                relation.removeJeuRelation(index);
  140:                        }
  141:                }
  142:
  143:                actionTimeListe.removeActionTime(actionTime);
  144:
  145:
  146:
  147:                return true;
  148:        }
  149:
  150:        public boolean renameActionTime(String oldName, String newName)
  151:        {
  152:                if (actionTimeListe.getActionTimeByName(newName) != null)
  153:                        return false;
  154:
  155:                actionTimeListe.getActionTimeByName(oldName).setName(newName);
  156:                return true;
  157:        }
  158:
  159:        public void addStep(String name) {
  160:                steps.add(name);
  161:        }
  162:
  163:        public boolean removeStep(String name) {
  164:                boolean ok = true;
  165:                for (Brick brick : brickList) {
  166:                        if (brick.getStep().equals(name))
  167:                                ok = false;
  168:                }
  169:
  170:                if (ok)
  171:                        steps.remove(name);
  172:                else
  173:                        JOptionPane
  174:                        .showMessageDialog(
  175:                                        null,
  176:                                        Messages.getString("DataPack.1"), //$NON-N
LS-1$
  177:                                        Messages.getString("DataPack.2"), //$NON-N
LS-1$
  178:                                        JOptionPane.WARNING_MESSAGE);
  179:
  180:                return ok;
  181:        }
  182:
  183:        public boolean renameStep(String oldName, String newName)
  184:        {
  185:                if (steps.contains(newName))
  186:                        return false;
  187:                int index = steps.indexOf(oldName);
  188:                steps.remove(oldName);
  189:                steps.insertElementAt(newName, index);
  190:
  191:                for (Brick brick : brickList)
  192:                        if (brick.getStep().equals(oldName))
  193:                                brick.setStep(newName);
  194:
  195:
  196:                return true;
  197:        }
  198:
  199:        public boolean addActor(Integer idGenerique, String nom) {
  200:                Acteur newActor = dataPackActorsView.getTreeListener().ajouterActe
ur(
  201:                                idGenerique, nom);
  202:                if (newActor != null) {
  203:                        allActors.put(newActor.idActeur, newActor);
  204:                        return true;
  205:                }
  206:                return false;
  207:        }
  208:
  209:        public boolean addGroup(String nom) {
  210:                return dataPackActorsView.getTreeListener().ajouterGroupe(nom) !=
null;
  211:        }
  212:
  213:
  214:
  215:        public synchronized void removeActor() {
  216:
  217:                Content oldContent = dataPackActorsView.getTreeListener().deleteAc
tor();
  218:
  219:                if (oldContent != null) {
  220:                        System.out.println("remove " + oldContent); //$NON-NLS-1$
  221:
  222:                        // si c'est un acteur on le supprime des type brique et de
s model
  223:                        cleanActor(oldContent);
  224:
  225:                        //                      dataPackActorsView.getTreeListener
().tree.updateJTree();
  226:                }
  227:        }
  228:
  229:        protected synchronized void cleanActor(Content oldActor) {
  230:
  231:                for (Brick brick : brickList)
  232:                        brick.removeActor(oldActor);
  233:        }
  234:
  235:
  236:
  237:
  238:
  239:        public MyDefaultMutableTreeNode getActor(Integer id) {
  240:                MyDefaultMutableTreeNode node = allActors.get(id);
  241:                if (node == null) {
  242:                        System.out.println(allActors.keySet() + "\n On cherchait :
" + id); //$NON-NLS-1$
  243:                        throw new NullPointerException();
  244:                }
  245:                return allActors.get(id);
  246:        }
  247:
  248:        public Vector<Brick> getBrickList() {
  249:                return brickList;
  250:        }
  251:
  252:        public Vector<Moyen> getMoyenListe() {
  253:                return moyens.getMoyenVector();
  254:        }
  255:
```

```
256:         public Moyen getMoyen(Integer id) {
257:                 return moyens.getMoyen(id);
258:         }
259:
260:         public MoyenEntreprise getMoyens() {
261:                 return moyens;
262:         }
263:
264:         public Vector<ActionTime> getActionTimeList() {
265:                 return actionTimeListe.values();
266:         }
267:
268:         public ActivitesEntreprise getActivities() {
269:                 return activities;
270:         }
271:
272:
273:         public ModelActeursEntreprise getActorsModel() {
274:                 return actorsModel;
275:         }
276:
277:         public Integer getNewActorId() {
278:                 Integer newId = newActorId;
279:                 newActorId = new Integer(newActorId.intValue() + 1);
280:                 return newId;
281:         }
282:
283:         public Integer getNewGroupeId() {
284:                 Integer newId = newGroupeId;
285:                 newGroupeId = new Integer(newGroupeId.intValue() + 1);
286:                 return newId;
287:         }
288:
289:
290:
291:         @Override
292:         public String toString() {
293:                 return name;
294:         }
295:
296:         @Override
297:         synchronized public String getFilePath() {
298:                 System.out.println("getFilePath (" + filePath + ")" + Thread.curre
ntThread().getStackTrace());
299:                 return filePath;
300:         }
301:
302:         @Override
303:         synchronized public void setFilePath(String _filePath) {
304:                 System.out.println("setFilePath (" + _filePath + ")" + Thread.curr
entThread().getStackTrace());
305:                 filePath = _filePath;
306:         }
307:
308:         public Vector<String> getSteps() {
309:                 return steps;
310:         }
311:
312:         public Vector<JeuActeur> getJeuxActeur() {
313:                 return jeuxActeur;
314:         }
315:
316:         public JeuActeur getJeuActeurByName(String name) {
317:                 for (JeuActeur ja : jeuxActeur) {
318:                         if (name.equals(ja.toString())) {
319:                                 return ja;
320:                         }
321:                 }
322:                 return null;
323:         }
324:
325:         public void initDataPackActorsView(FiltreActeursEntreprise view) {
326:                 if (dataPackActorsView == null) {
327:                         dataPackActorsView = view;
328:                 }
329:         }
330:
331:         public void addJeuActeur(JeuActeur jA) {
332:                 // ajoute toutes lesbases
333:                 for (Acteur newActor : jA.bases) {
334:                         allActors.put(newActor.idActeur, newActor);
335:                 }
336:
337:                 // ajoute tout les corps
338:                 for (String corps : jA.corpsMetier) {
339:                         for (Acteur actor : jA.getCorps(corps)) {
340:                                 allActors.put(actor.idActeur, actor);
341:                         }
342:                 }
343:
344:                 jeuxActeur.add(jA);
345:         }
346:
347:         public void removeJeuActeur(JeuActeur deletedActorSet) {
348:                 for (String nomCorps : deletedActorSet.getListeCorps()) {
349:                         for (ActeurBase actor : deletedActorSet.getCorps(nomCorps)
) {
350:                                 cleanActor(actor);
351:                         }
352:                 }
353:                 for (ActeurBase actor : deletedActorSet.getListeBase()) {
354:                         cleanActor(actor);
355:                 }
356:
357:                 jeuxActeur.remove(deletedActorSet);
358:         }
359:
360:         public void updateAllActors() {
361:                 for (JeuActeur ja : jeuxActeur) {
362:                         addJeuActeur(ja);
363:                 }
364:         }
365:
366:         public boolean removeActivity(Activite deletedActivity) {
367:                 boolean ok = true;
368:                 for (Brick brick : brickList) {
369:                         if (brick.getActivity().equals(deletedActivity))
370:                                 ok = false;
371:                 }
372:                 if (ok)
373:                         activities.supprimerActivite(deletedActivity);
374:                 else
375:                         JOptionPane
376:                         .showMessageDialog(
377:                                         null,
378:                                         Messages.getString("DataPack.5"), //$NON-N
LS-1$
379:                                         Messages.getString("DataPack.6"), //$NON-N
LS-1$
380:                                         JOptionPane.WARNING_MESSAGE);
381:                 return ok;
382:         }
383:
384:         public boolean removeMoyen(Moyen deletedMoyen) {
```

```
385:                boolean ok = true;
386:                for (Brick brick : brickList) {
387:                        for (BrickVertex vertex : brick.getVertices()) {
388:                                if (vertex.getContent().equals(deletedMoyen.getId(
))) {
389:                                        ok = false;
390:                                        break;
391:                                }
392:                        }
393:                }
394:
395:                if (ok)
396:                        moyens.supprimerMoyen(deletedMoyen.getId());
397:                else
398:                        JOptionPane
399:                        .showMessageDialog(
400:                                        null,
401:                                        Messages.getString("DataPack.7"), //$NON-N
LS-1$
402:                                        Messages.getString("DataPack.8"), //$NON-N
LS-1$
403:                                        JOptionPane.WARNING_MESSAGE);
404:                return ok;
405:        }
406:
407:        public void exportDataPack() {
408:
409:        /*      if (!checkRelationForExport())
410:                {
411:                        DialogHandlerFrame.showErrorDialog("Ce datapack ne peut pa
s Ãªtre exportÃ©. Veuillez corriger les erreurs signalÃ©es avant de relancer la procÃ©dur
e d'exportation.");
412:                        return;
413:                }*/
414:
415:                String oldFilePath = filePath;
416:                filePath = null;
417:
418:                exportModule = new ExportedDatapackModule(this);
419:
420:                Vector<Brick> addedBrick = new Vector<Brick>();
421:
422:                Vector<Brick> brickListTemps = new Vector<Brick>(brickList);
423:
424:                for (Brick brick : brickListTemps) {
425:                        if (brick.isGeneric()) {
426:                                Collection<Brick> exportedBrick = brick.createNoGe
nericBrick();
427:                                addedBrick.addAll(exportedBrick);
428:                                brickList.addAll(exportedBrick);
429:                        }
430:                }
431:                try {
432:                        Program.save(this, Messages.getString("DataPack.9"), Messa
ges.getString("DataPack.10"), true); //$NON-NLS-1$ //$NON-NLS-2$
433:                } catch (Exception e) {
434:                        DialogHandlerFrame
435:                        .showErrorDialog(Messages.getString("DataPack.11") //$NON-
NLS-1$
436:                                        + e.toString());
437:                        e.printStackTrace();
438:                }
439:                exportModule = null;
440:                filePath = oldFilePath;
441:                brickList.removeAll(addedBrick);
442:        }
443:
444:        public boolean checkRelationForExport()
445:        {
446:                boolean ok = true;
447:                for (Brick b : brickList)
448:                {
449:                        for (BrickEdge bE : b.getEdges())
450:                        {
451:                                RelationPossibility possibility = relationSet.getR
elationPossibility(bE.getSource().getContent(), bE.getDestination().getContent(), true);
452:                                int i = 0;
453:                                for (JeuRelation jR : bE.getCompleteRelation().get
EnsembleRelation())
454:                                {
455:                                        ArrayList<Mean> means = possibility.getMap
().get(jR.objectifStructurel);
456:
457:                                        if (jR.objectifStructurel.equals(RelationP
ossibility.UNDEFINED_GOAL))
458:                                        {
459:                                                DialogHandlerFrame.showErrorDialog
("La relation de la brique "+b.getName()+" entre les acteurs "+bE.getSource()+" et "+bE.g
etDestination()+"\nau temps d'action "+actionTimeListe.values().elementAt(i)+"\nn'est pas
 dÃ©finie, veuillez la complÃ©ter.");
460:                                                ok = false;
461:                                        }
462:
463:                                        if (means == null || !means.contains(jR.mo
yenStructurel))
464:                                        {
465:                                                DialogHandlerFrame.showErrorDialog
("La relation de la brique "+b.getName()+" entre les acteurs "+bE.getSource()+" et "+bE.g
etDestination()+"\nau temps d'action "+actionTimeListe.values().elementAt(i)+"\nn'est plu
s disponible dans le datapack,\n veuillez la corriger. \nCette relation va Ãªtre dÃ©finie
 Ã  \""+RelationPossibility.UNDEFINED_STRING+".");
466:                                                ok = false;
467:                                                jR.moyenStructurel = RelationPossi
bility.UNDEFINED_MEAN;
468:                                                jR.objectifStructurel = RelationPo
ssibility.UNDEFINED_GOAL;
469:                                        }
470:                                        i++;
471:                                }
472:                        }
473:                }
474:                return ok;
475:        }
476:
477:        public EnsembleRelation getRelations() {
478:                return relationSet;
479:
480:        }
481:
482:        public void setRelations(EnsembleRelation ensembleRelation) {
483:                System.out.println(Messages.getString("DataPack.12")); //$NON-NLS-
1$
484:                relationSet = ensembleRelation;
485:
486:        }
487:
488:        public ExportedDatapackModule getExportModule() {
489:                return exportModule;
490:        }
491:
492:        public CompanyInfo getCompanyInfo() {
493:                if(companyInfo == null) {
494:                        companyInfo = new CompanyInfo(); //TODO a virer dans la fo
nction de mise a jour de version du datapack
```

```
  495:                 }
  496:                 return companyInfo;
  497:         }
  498:
  499:         public Collection<MyDefaultMutableTreeNode> getAllActors() {
  500:                 return allActors.values();
  501:         }
  502:
  503:         public Session getCurrentSession() {
  504:                 return currentSession;
  505:         }
  506:
  507:         public void setCurrentSession(Session currentSession) {
  508:                 this.currentSession = currentSession;
  509:         }
  510:
  511:         public StringTranslator getStringTranslator() {
  512:                 return stringTranslator;
  513:         }
  514:
  515:         public void setStringTranslator(StringTranslator stringTranslator) {
  516:                 this.stringTranslator = stringTranslator;
  517:         }
  518:
  519:         public void removeBrick(Brick brick) {
  520:                 brickList.remove(brick);
  521:
  522:         }
  523:
  524:         public void checkDatapackValidity()
  525:         {
  526:                 if (activities.activities == null)
  527:                 {
  528:                         activities.activities = new Vector<Activite>();
  529:                         for (Brick b : brickList)
  530:                         {
  531:                                 if (!activities.activities.contains(b.getActivity(
))))
  532:                                 {
  533:                                         activities.activities.add(b.getActivity())
;
  534:                                 }
  535:                         }
  536:                         DialogHandlerFrame.showErrorDialog("Suite Ã  une Ã©volutio
n du datapack, la liste des activitÃ©s a du Ãªtre reconstruite\nL'ensemble des activitÃ©s
 utilisÃ©es dans au moins une brique ont pu Ãªtre rÃ©cupÃ©rÃ©s\nCelles qui n'Ã©taient pas
 encore utilisÃ©e doivent Ãªtre ajoutÃ©es manuellement au datapack.");
  537:                 }
  538:
  539:                 if (actionTimeListe.values() == null)
  540:                 {
  541:                         actionTimeListe.restoreFromOldVersion();
  542:                 }
  543:
  544:
  545:         }
  546:
  547: }
```

```
1: package dataPack;
2:
3: public interface Content {
4:
5:         public boolean equals(Object other);
6:
7: }
```

```java
package dataPack;

import graphicalUserInterface.IconDatabase;
import graphicalUserInterface.Program;

import java.io.Serializable;

import javax.swing.Icon;
import javax.swing.JLabel;

import main.Statut;
import client.stringTranslator.StringTranslator;

/*
 * dÃ©fini un acteur pour la liste de selection des acteur
 */

public class Acteur extends MyDefaultMutableTreeNode implements Content,
                Serializable, Comparable<Acteur> {

        private static final long serialVersionUID = -7013072226067306012L;

        protected String poste;
        protected Integer idActeur;
        protected Integer idStatut;
        protected Integer idGroupe;

        public Acteur(String n_poste, Integer n_idStatut, Integer groupe) {
                super();
                poste = n_poste;
                idActeur = Program.myMainFrame.getDataPack().getNewActorId();

                if (n_idStatut == null)
                        System.out.println(n_idStatut);
                else
                        idStatut = n_idStatut;

                idGroupe = groupe;

                Program.myMainFrame.getDataPack().addAnActor(this);
        }

        public void setPoste(String poste) {
                this.poste = poste;
        }

        public String getPoste() {
                return poste;
        }

        public void setIdActeur(Integer idActeur) {
                this.idActeur = idActeur;
        }

        public Integer getIdActeur() {
                return idActeur;
        }

        public Statut getStatut() {
                return main.Statut.values()[getIdStatut()];
        }

        public void setIdStatut(Integer idStatut) {
                this.idStatut = idStatut;
        }

        public Integer getIdStatut() {
                return idStatut;
        }

        public void setIdGroupe(Integer groupe) {
                idGroupe = groupe;
        }

        public Integer getIdGroupe() {
                return idGroupe;
        }

        public boolean estDansGroupe(Integer groupe) {
                return (idGroupe == groupe);
        }

        public int compareTo(Acteur acteur) {
                if (acteur.getIdActeur().equals(idActeur))
                        return 0;
                else if (acteur.getIdActeur().intValue() < idActeur.intValue())
                        return 1;
                else {
                        return -1;
                }
        }

        public boolean equals(Acteur acteur) {
                return idStatut.equals(acteur.idStatut);
        }

        @Override
        public JLabel getJComponent(boolean selected, boolean expanded, boolean le
af,
                        int row, boolean hasFocus) {
                return TreeActorsCellRendere.createDefaultLabel(this.toString(),
                                getIcon(), selected);
        }

        @Override
        public Icon getIcon() {
                return IconDatabase.vectorIconActorsMin.get(idGroupe
                                % IconDatabase.vectorIconActorsMin.size());
        }

        @Override
        public MyTreeNodeType getType() {
                return MyTreeNodeType.ActorType;
        }

        @Override
        public void changeStringValue(String newString) {
                if (newString.compareTo(this.toString()) != 0) {
                        setPoste(newString);
                }
        }

        @Override
        public Integer getId() {
                return idActeur;
        }

        @Override
        public boolean equals(Object object) {
                if (object instanceof Acteur) {
                        return ((Acteur) object).idActeur.equals(idActeur);
                }
                return false;
        }
```

```
  134:
  135:         public String getNoTranslatedString() {
  136:                 if (Program.isTriades())
  137:                         return poste;
  138:                 else
  139:                         return poste + "(" + getStatut().toString() + ")"; //$NON-
NLS-1$ //$NON-NLS-2$
  140:         }
  141:
  142:         @Override
  143:         public String toString() {
  144:                 return StringTranslator.getTranslatedString(this, StringTranslator
.StringType.actorType);
  145:         }
  146: }
```

```
 1: package dataPack;
 2:
 3: import java.io.Serializable;
 4: import java.util.Vector;
 5:
 6: public class ActivitesEntreprise implements Serializable {
 7:         private static final long serialVersionUID = 5347414089243015128L;
 8:
 9:         final static Integer DEBUTINDICEID = -100;
10:
11:         protected Vector<Activite> activities;
12:
13:         public ActivitesEntreprise() {
14:                 activities = new Vector<Activite>(0);
15:         }
16:
17:         public Activite ajouterActivite(String nom, int iconId) {
18:
19:                 Activite newActivity = new Activite(nom, iconId);
20:                 activities.add(newActivity);
21:
22:                 return newActivity;
23:         }
24:
25:         public void supprimerActivite(Activite activite) {
26:                 activities.remove(activite);
27:         }
28:
29:         public Vector<Activite> getActivities() {
30:                 return activities;
31:         }
32:
33:         public int getNbActivites() {
34:                 return activities.size();
35:         }
36:
37:         public boolean renameActivity(String oldName, String newName) {
38:                 for (Activite act : activities)
39:                 {
40:                         if (act.toString().equals(newName))
41:                                 return false;
42:                 }
43:
44:                 for (Activite act : activities)
45:                 {
46:                         if (act.toString().equals(oldName))
47:                         {
48:                                 act.setNom(newName);
49:                                 return true;
50:                         }
51:                 }
52:
53:                 return false;
54:         }
55:
56: }
```

```
    1: package dataPack;
    2:
    3: import translation.Messages;
    4: import graphicalUserInterface.DialogHandlerFrame;
    5:
    6: public class GroupeRoot extends Groupe {
    7:
    8:         private static final long serialVersionUID = -6533221997848227528L;
    9:
   10:         GroupeRoot(String nNom) {
   11:                 super(nNom);
   12:         }
   13:
   14:         @Override
   15:         public void changeStringValue(String newString) {
   16:                 DialogHandlerFrame
   17:                                 .showErrorDialog(Messages.getString("GroupeRoot.0"
)); //$NON-NLS-1$
   18:         }
   19: }
```

```
 1: package automateGraph;
 2:
 3: import javax.swing.JFrame;
 4: import automate.*;
 5:
 6:
 7: public class ActionEditingFrame extends JFrame {
 8:
 9:         public static final long serialVersionUID = 54365765;
10:
11:
12:
13:         protected static ActionEditingFrame instance;
14:         protected Action currentAction;
15:
16:
17:         public static void editAction(Action a)
18:         {
19:
20:         }
21:
22:         protected void setAction(Action action)
23:         {
24:                 if (action != currentAction)
25:                 {
26:                         removeAll();
27:                         add(action.toComponent());
28:
29:                 }
30:         }
31:
32:
33:
34: }
```

```
 1: package main;
 2:
 3:
 4: public class ActeurGenerique extends Pole{
 5:
 6:         /**
 7:          *
 8:          */
 9:         private static final long serialVersionUID = -1644787848967220296L;
10:         public Statut statut;
11:         public String nom;
12:         public String role;
13:
14:         public ActeurGenerique(String _nom, Statut _statut)
15:         {
16:                 nom = _nom;
17:                 statut = _statut;
18:         }
19:
20:         public ActeurGenerique()
21:         {
22:                 this("Non-dÃ©fini",Statut.nonDef);
23:         }
24:
25:         public String toString()
26:         {
27:                 return (nom+"  -  "+statut.toString());
28:         }
29:
30:         public int getType()
31:         {
32:                 return Pole.ACTEUR;
33:         }
34:
35: }
36:
```

```
    1: package main;
    2:
    3: import java.io.Serializable;
    4:
    5: import client.stringTranslator.StringTranslator;
    6: import client.stringTranslator.StringTranslator.StringType;
    7:
    8: public class ActionTime implements Serializable {
    9:
   10:         // action1("time 1", 0), action2("time 2", 0), action3("time 2", 0),
   11:         // action4("time 2", 0);
   12:
   13:         /**
   14:          *
   15:          */
   16:         private static final long serialVersionUID = -5435459194851632383L;
   17:         public String nom;
   18:         public Integer id;
   19:
   20:         public ActionTime(String n_nom, Integer n_id) {
   21:                 nom = n_nom;
   22:                 id = n_id;
   23:         }
   24:
   25:         public String getNoTranslatedString() {
   26:                 return nom;
   27:         }
   28:
   29:         @Override
   30:         public String toString() {
   31:                 return StringTranslator.getTranslatedString(this, StringType.actio
nTimesType);
   32:         }
   33:
   34:         public void setName(String newName) {
   35:                 nom = newName;
   36:
   37:         }
   38:
   39: }
```

```
1: package main;
2:
3: public interface Objectif {
4:
5:         public String toString();
6:
7: }
```

```java
package main;

import graphicalUserInterface.Program;

import java.io.Serializable;
import java.util.BitSet;
import java.util.Vector;

import relations.Goal;
import relations.Mean;
import relations.RelationPossibility;
import translation.Messages;

public class RelationComplete implements Serializable {

	/**
	 *
	 */
	private static final long serialVersionUID = 1067545404384529172L;

	public static final int RELATION_OK = 0;
	public static final int RELATION_INCOMPLETE = 1;
	public static final int RELATION_ECART_OBJECTIF = 3;
	public static final int RELATION_ECART_MOYEN = 2;

	protected Vector<JeuRelation> ensembleRelations;

	public RelationComplete() {
		ensembleRelations = new Vector<JeuRelation>(Program.myMainFrame
				.getDataPack().getActionTimeList().size());
		for (int i = 0; i < Program.myMainFrame.getDataPack()
				.getActionTimeList().size(); i++) {
			ensembleRelations.add(new JeuRelation());
		}

	}

	public RelationComplete(boolean noRelationFlag)
	{
		ensembleRelations = new Vector<JeuRelation>(Program.myMainFrame
				.getDataPack().getActionTimeList().size());
		for (int i = 0; i < Program.myMainFrame.getDataPack()
				.getActionTimeList().size(); i++) {
			ensembleRelations.add(new JeuRelation(noRelationFlag));
		}
	}

	@SuppressWarnings("unchecked")
	public RelationComplete(RelationComplete relationComplete) {

		ensembleRelations = (Vector<JeuRelation>) relationComplete
				.getEnsembleRelation().clone();
	}

	public JeuRelation getJeuRelation(int index) {
		return ensembleRelations.elementAt(index);
	}

	public Vector<JeuRelation> getEnsembleRelation() {
		return ensembleRelations;
	}

	public boolean isEmpty(int mode) {
		for (JeuRelation relationSet : ensembleRelations)
			if (!relationSet.isEmpty(mode))
				return false;
		return true;
	}

	public boolean isNoRelation() {

		for (JeuRelation relationSet : ensembleRelations)
			if (!relationSet.isNoRelation())
				return false;
		return true;
	}

	public BitSet getActiveTime() {
		BitSet result = new BitSet(ensembleRelations.size());
		result.clear();
		for (int i = 0; i < ensembleRelations.size(); i++) {
			if (!ensembleRelations.elementAt(i).isEmpty(
					RelationPossibility.RELATIONSTRUCTURELLE))
				result.set(i);
		}
		return result;
	}

	public void addJeuRelation(ActionTime newActionTime) {
		ensembleRelations.add(new JeuRelation());
	}

	public int getState() {
		int result = RELATION_OK;
		for (JeuRelation jR : ensembleRelations) {
			int stateJR = jR.getState();
			if (result < stateJR)
				result = stateJR;
		}
		return result;
	}

	public Vector<JeuRelation> getUndefineRelations() {
		Vector<JeuRelation> result = new Vector<JeuRelation>();

		for(JeuRelation jeuRelation : ensembleRelations) {
			if(jeuRelation.getState() == RELATION_INCOMPLETE) {
				result.add(jeuRelation);
			}
		}

		return result;
	}

	public Vector<JeuRelation> getValideRelations() {
		Vector<JeuRelation> result = new Vector<JeuRelation>();

		for(JeuRelation jeuRelation : ensembleRelations) {
			if(jeuRelation.getState() == RELATION_OK) {
				result.add(jeuRelation);
			}
		}

		return result;
	}

	public Vector<JeuRelation> getFalseObjectifRelations() {
		Vector<JeuRelation> result = new Vector<JeuRelation>();

		for(JeuRelation jeuRelation : ensembleRelations) {
			if(jeuRelation.getState() == RELATION_ECART_OBJECTIF) {
				result.add(jeuRelation);
			}
		}
```

```
  135:
  136:                return result;
  137:        }
  138:
  139:        public Vector<JeuRelation> getFalseMoyenRelation() {
  140:                Vector<JeuRelation> result = new Vector<JeuRelation>();
  141:
  142:                for(JeuRelation jeuRelation : ensembleRelations) {
  143:                        if(jeuRelation.getState() == RELATION_ECART_MOYEN) {
  144:                                result.add(jeuRelation);
  145:                        }
  146:                }
  147:
  148:                return result;
  149:        }
  150:
  151:        private String createStringColor(String color, String text) {
  152:                //TODO si possible a virer dans une class d'outils de manipe html
  153:                if(color != null && color.compareTo("") != 0) { //$NON-NLS-1$
  154:                        return "<font color=\"" + color + "\">" + text + "</font>"
; //$NON-NLS-1$ //$NON-NLS-2$ //$NON-NLS-3$
  155:                } else {
  156:                        return text;
  157:                }
  158:        }
  159:
  160:        protected String getStringStructurelRelation() {
  161:                String result = ""; //$NON-NLS-1$
  162:
  163:                for(JeuRelation relation : ensembleRelations) {
  164:                        result += createRelationString(relation, true, "", "");//
"  Objectif : '" + relation.objectifStructurel + "' - Moyen : '" + relation.moyenStructur
el + "' <br/>"; //$NON-NLS-1$ //$NON-NLS-2$
  165:                }
  166:
  167:                result += "</br>"; //$NON-NLS-1$
  168:
  169:                return result;
  170:        }
  171:
  172:        private String createRelationString(JeuRelation relation, boolean structur
el, String objectifColor, String moyenColor) {
  173:                Goal objectif;
  174:                Mean moyen;
  175:
  176:                if(structurel) {
  177:                        objectif = relation.objectifStructurel;
  178:                        moyen = relation.moyenStructurel;
  179:                } else {
  180:                        objectif = relation.objectifReel;
  181:                        moyen = relation.moyenReel;
  182:                }
  183:
  184:                String actionTimeName = Program.myMainFrame.getDataPack().getActio
nTimeList().elementAt(ensembleRelations.indexOf(relation)).toString();
  185:
  186:                if(structurel) {
  187:                        return (Program.isTriades() ? Messages.getString("Relation
Complete.0") : actionTimeName + " : " )+ createStringColor(objectifColor, objectif.toStri
ng()) + ((relation.objectifStructurel.compareTo(RelationPossibility.UNDEFINED_GOAL) == 0)
 ? "<br/>" : " (" + createStringColor(moyenColor, moyen.toString()) + ")<br/>"); //$NON-N
LS-1$ //$NON-NLS-2$ //$NON-NLS-3$ //$NON-NLS-4$ //$NON-NLS-5$
  188:                } else {
  189:                        return actionTimeName + " : " + createStringColor(objectif
Color, objectif.toString()) + ((relation.objectifReel.compareTo(RelationPossibility.UNDEF
INED_GOAL) == 0) ? "<br/>" : " (" + createStringColor(moyenColor, moyen.toString()) + ")<
br/>"); //$NON-NLS-1$ //$NON-NLS-2$ //$NON-NLS-3$ //$NON-NLS-4$
  190:                }
  191:        }
  192:
  193:        public String getStringDescription() {
  194:                String result = "<html>"; //$NON-NLS-1$
  195:
  196:                int relationCompleteState = getState();
  197:
  198:                if(Program.isTriades() == false) {
  199:                        result += getStringStructurelRelation();
  200:                }else if(relationCompleteState == RELATION_OK) {
  201:                        result += getStringStructurelRelation();
  202:                } else if(relationCompleteState == RELATION_INCOMPLETE) {
  203:                        result += getStringStructurelRelation();
  204:                        result += Messages.getString("RelationComplete.4");
        //$NON-NLS-1$
  205:                } else {
  206:                        //Ecart sur les relations
  207:                        result += Messages.getString("RelationComplete.5"); //$NON
-NLS-1$
  208:
  209:                        Vector<JeuRelation> valideRelations = getValideRelations()
;
  210:                        Vector<JeuRelation> undefineRelations = getUndefineRelatio
ns();
  211:                        Vector<JeuRelation> falseObjectifRelations = getFalseObjec
tifRelations();
  212:                        Vector<JeuRelation> falseMoyenRelations = getFalseMoyenRel
ation();
  213:
  214:                        result += Messages.getString("RelationComplete.6"); //$NON
-NLS-1$
  215:                        for(JeuRelation relation : falseObjectifRelations) {
  216:                                result += createRelationString(relation, false, "r
ed", "red");//  "  Objectif : " + createStringColor("red", relation.objectifReel) + " - M
oyen : " + createStringColor("red", relation.moyenReel) + "<br/>"; //$NON-NLS-1$ //$NON-N
LS-2$
  217:                                result += createRelationString(relation, true, "",
 ""); //"  Objectif : " + relation.objectifStructurel + " - Moyen : " + relation.moyenStr
ucturel + "<br/> <br/>"; //$NON-NLS-1$ //$NON-NLS-2$
  218:                        }
  219:
  220:                        result += Messages.getString("RelationComplete.7"); //$NON
-NLS-1$
  221:                        for(JeuRelation relation : falseMoyenRelations) {
  222:                                result += createRelationString(relation, false, ""
, "red");// "  Objectif : " + createStringColor("red", relation.objectifReel) + " - Moyen
 : " + createStringColor("red", relation.moyenReel) + "<br/>"; //$NON-NLS-1$ //$NON-NLS-2
$
  223:                                result += createRelationString(relation, true, "",
 ""); //"  Objectif : " + relation.objectifStructurel + " - Moyen : " + relation.moyenStr
ucturel + "<br/> <br/>"; //$NON-NLS-1$ //$NON-NLS-2$
  224:                        }
  225:
  226:                        if(undefineRelations.size() > 0)
  227:                                result += createStringColor("blue", undefineRelati
ons.size() + (undefineRelations.size() > 1 ? Messages.getString("RelationComplete.1") : M
essages.getString("RelationComplete.2")) + " : <br/><br/>"); //$NON-NLS-1$ //$NON-NLS-2$
//$NON-NLS-3$ //$NON-NLS-4$
  228:
  229:                        result += Messages.getString("RelationComplete.3"); //$NON
-NLS-1$
  230:                        for(JeuRelation relation : valideRelations) {
  231:                                result += createRelationString(relation, true, "",
 ""); //"  Objectif : " + relation.objectifStructurel + " - Moyen : " + relation.moyenStr
ucturel + "<br/>"; //$NON-NLS-1$ //$NON-NLS-2$
  232:                        }
```

```
233:                 }
234:
235:                 result += "</html>"; //$NON-NLS-1$
236:                 return result;
237:         }
238:
239:         public void removeJeuRelation(int index) {
240:                 ensembleRelations.removeElementAt(index);
241:
242:         }
243:
244:         static public RelationComplete createNoRelation()
245:         {
246:                 RelationComplete result = new RelationComplete(true);
247:                 return result;
248:
249:         }
250: }
```

```
    1: package main;
    2:
    3: import translation.Messages;
    4:
    5: public enum Statut {
    6:         nonDef(0, Messages.getString("TriadeKernel.Statut.0")), directeurGeneral(1
, Messages.getString("TriadeKernel.Statut.1"), 1), directeur( //$NON-NLS-1$ //$NON-NLS-2$
    7:                         2, Messages.getString("TriadeKernel.Statut.2"), 2), chefDe
Service(3, Messages.getString("TriadeKernel.Statut.3"), 3), chefEquipe( //$NON-NLS-1$ //$
NON-NLS-2$
    8:                         4, Messages.getString("TriadeKernel.Statut.4"), 4), cadreI
ntermediaire(5, //$NON-NLS-1$
    9:                         Messages.getString("TriadeKernel.Statut.5"), 5), technicie
nSuperieur(6, //$NON-NLS-1$
   10:                         Messages.getString("TriadeKernel.Statut.6"), 6), ouvrierSp
ecialise(7, //$NON-NLS-1$
   11:                         Messages.getString("TriadeKernel.Statut.7"), 7), deleguePe
rsonnel(8, //$NON-NLS-1$
   12:                         Messages.getString("TriadeKernel.Statut.8")), client(9, Me
ssages.getString("TriadeKernel.Statut.9")), auditeur(10, //$NON-NLS-1$ //$NON-NLS-2$
   13:                         Messages.getString("TriadeKernel.Statut.10")), psychologue
(11, Messages.getString("TriadeKernel.Statut.11")), conseilJuridique(12, //$NON-NLS-1$ //
$NON-NLS-2$
   14:                         Messages.getString("TriadeKernel.Statut.12")), orgaSyndica
le(13, Messages.getString("TriadeKernel.Statut.13")), partenaire( //$NON-NLS-1$ //$NON-NL
S-2$
   15:                         14, Messages.getString("TriadeKernel.Statut.14")), formate
ur(15, Messages.getString("TriadeKernel.Statut.15")), medecinTravail(16, //$NON-NLS-1$ //
$NON-NLS-2$
   16:                         Messages.getString("TriadeKernel.Statut.16")), chefProjet(
17, Messages.getString("TriadeKernel.Statut.17")); //$NON-NLS-1$ //$NON-NLS-2$
   17:
   18:         public String nom;
   19:         public int rang;
   20:         public Integer id;
   21:
   22:         private Statut(int _id, String _nom) {
   23:                 this(_id, _nom, -1);
   24:         }
   25:
   26:         private Statut(int _id, String _nom, int _rang) {
   27:                 nom = _nom;
   28:                 rang = _rang;
   29:                 id = new Integer(_id);
   30:         }
   31:
   32:         public String toString() {
   33:                 return nom;
   34:         }
   35:
   36: }
```

```
    1: package main;
    2:
    3: import translation.Messages;
    4:
    5: public enum ObjectifLienHierarchique implements Objectif {
    6:
    7:         defRegles(Messages.getString("TriadeKernel.ObjectifLienHierarchique.0")),
evalCompetences( //$NON-NLS-1$
    8:                         Messages.getString("TriadeKernel.ObjectifLienHierarchique.
1")), donnerConsigne(Messages.getString("TriadeKernel.ObjectifLienHierarchique.2")), eval
utationPolitique( //$NON-NLS-1$ //$NON-NLS-2$
    9:                         Messages.getString("TriadeKernel.ObjectifLienHierarchique.
3")), definitionPolitique(Messages.getString("TriadeKernel.ObjectifLienHierarchique.4")),
 delegationActivite( //$NON-NLS-1$ //$NON-NLS-2$
   10:                         Messages.getString("TriadeKernel.ObjectifLienHierarchique.
5")), delegationMission(Messages.getString("TriadeKernel.ObjectifLienHierarchique.6")), c
ontroleExecutionActivite( //$NON-NLS-1$ //$NON-NLS-2$
   11:                         Messages.getString("TriadeKernel.ObjectifLienHierarchique.
7")), controleUsageOutils( //$NON-NLS-1$
   12:                         Messages.getString("TriadeKernel.ObjectifLienHierarchique.
8")), developpementCompetences( //$NON-NLS-1$
   13:                         Messages.getString("TriadeKernel.ObjectifLienHierarchique.
9")), resolutionConflit( //$NON-NLS-1$
   14:                         Messages.getString("TriadeKernel.ObjectifLienHierarchique.
10")), resolutionProbleme(Messages.getString("TriadeKernel.ObjectifLienHierarchique.11"))
; //$NON-NLS-1$ //$NON-NLS-2$
   15:
   16:         public String nom;
   17:
   18:         private ObjectifLienHierarchique(String _nom) {
   19:                 nom = _nom;
   20:         }
   21:
   22:         public String toString() {
   23:                 return nom;
   24:         }
   25:
   26: }
```

```
 1: package main;
 2:
 3: import java.io.Serializable;
 4:
 5: import edu.uci.ics.jung.graph.impl.SimpleSparseVertex;
 6:
 7: public class Pole extends SimpleSparseVertex implements Serializable{
 8:
 9:         /**
10:          *
11:          */
12:         private static final long serialVersionUID = 5185453429018761925L;
13:         public static int ACTEUR = 1;
14:         public static int OBJET = 2;
15:         protected static int idMax = 1;
16:         protected int id;
17:
18:         public Pole()
19:         {
20:                 super();
21:         }
22:
23:         public int getType()
24:         {
25:                 return 0;
26:         }
27:
28:         public void setId(int newId)
29:         {
30:                 id = newId;
31:                 if (newId > idMax)
32:                         idMax = newId;
33:         }
34:
35:         public int getId()
36:         {
37:                 return id;
38:         }
39:
40:         public static int getIdMax()
41:         {
42:                 return idMax;
43:         }
44:
45:
46: }
```

```
    1: package main;
    2:
    3: import translation.Messages;
    4:
    5: public enum ObjectifRetourLienHierarchique implements Objectif {
    6:
    7:         negociation(
    8:                         Messages.getString("TriadeKernel.ObjectifRetourLienHierarc
hique.0")), compteRendu( //$NON-NLS-1$
    9:                         Messages.getString("TriadeKernel.ObjectifRetourLienHierarc
hique.1")); //$NON-NLS-1$
   10:
   11:         public String nom;
   12:
   13:         private ObjectifRetourLienHierarchique(String _nom) {
   14:                 nom = _nom;
   15:         }
   16:
   17:         public String toString() {
   18:                 return nom;
   19:         }
   20:
   21: }
```

```
1: package main;
2:
3: public enum Ecart {
4:         gouffre, ravin, cheveux;
5: }
```

```
    1: package main;
    2:
    3: import java.util.*;
    4:
    5: public class LienActeurObjet {
    6:
    7:
    8:         protected static Hashtable<String,String[]> moyens;
    9:
   10:
   11:         protected ObjectifLienPartenaire objectif;
   12:         protected String moyen;
   13:
   14:         protected static void initTable()
   15:         {
   16:                 moyens = new Hashtable<String,String[]>();
   17:
   18:                 String[] concep = {"Analyse"};
   19:                 String[] util = {"Application de connaissance", "DÃ©couverte de l'
objet"};
   20:                 String[] recup = {"Observation instrumentÃ©e", "Observation non in
strumentÃ©e"};
   21:                 String[] eval = {"Normes"};
   22:
   23:                 moyens.put(ObjectifActeurObjet.conception.toString(), concep);
   24:                 moyens.put(ObjectifActeurObjet.utilisation.toString(), util);
   25:                 moyens.put(ObjectifActeurObjet.recupInfo.toString(),recup );

   26:                 moyens.put(ObjectifActeurObjet.eval.toString(), eval);
   27:         }
   28:
   29:
   30:         public static String[] getStrings(String key)
   31:         {
   32:                 if (moyens == null)
   33:                         initTable();
   34:                 return (moyens.get(key));
   35:         }
   36:
   37:
   38:
   39:
   40: }
```

```
    1: package main;
    2:
    3: import translation.Messages;
    4:
    5: public enum ObjectifObjetActeur implements Objectif {
    6:         evalCompetences(Messages.getString("TriadeKernel.ObjectifObjetActeur.0"));
//$NON-NLS-1$
    7:
    8:         public String nom;
    9:
   10:         private ObjectifObjetActeur(String _nom) {
   11:                 nom = _nom;
   12:         }
   13:
   14:         public String toString() {
   15:                 return nom;
   16:         }
   17:
   18: }
```

```
    1: package main;
    2:
    3: public enum ObjectifObjetObjet implements Objectif
    4: {
    5:                 permetRea("Permet la rÃ©alisation"),fiabiliseRea("Fiabilise la rÃ©
alisation"),controleRea("Controle la rÃ©alisation"),transforme("Transformer"),controleEta
t("ContrÃ´ler l'Ã©tat"),eval("Evaluation");
    6:
    7:                 public String nom;
    8:
    9:                 private ObjectifObjetObjet(String _nom)
   10:                 {
   11:                         nom = _nom;
   12:                 }
   13:
   14:                 public String toString()
   15:                 {
   16:                         return nom;
   17:                 }
   18:
   19:
   20:
   21: }
```

```
    1: package main;
    2:
    3: import translation.Messages;
    4:
    5: public enum ObjectifActeurObjet implements Objectif {
    6:
    7:         conception(Messages.getString("TriadeKernel.ObjectifActeurObjet.0")), util
isation(Messages.getString("TriadeKernel.ObjectifActeurObjet.1")), recupInfo( //$NON-NLS-
1$ //$NON-NLS-2$
    8:                         Messages.getString("TriadeKernel.ObjectifActeurObjet.2")),
 eval(Messages.getString("TriadeKernel.ObjectifActeurObjet.3")); //$NON-NLS-1$ //$NON-NLS
-2$
    9:
   10:         public String nom;
   11:
   12:         private ObjectifActeurObjet(String _nom) {
   13:                 nom = _nom;
   14:         }
   15:
   16:         public String toString() {
   17:                 return nom;
   18:         }
   19:
   20: }
```

```
    1: package main;
    2:
    3: import translation.Messages;
    4:
    5: public enum ObjectifMoyenObjet implements Objectif {
    6:
    7:         estAdapteEff(Messages.getString("TriadeKernel.ObjectifMoyenObjet.0")), fac
iliRea(Messages.getString("TriadeKernel.ObjectifMoyenObjet.1")), cadreRea( //$NON-NLS-1$
//$NON-NLS-2$
    8:                         Messages.getString("TriadeKernel.ObjectifMoyenObjet.2")),
securiseRea(Messages.getString("TriadeKernel.ObjectifMoyenObjet.3")); //$NON-NLS-1$ //$NO
N-NLS-2$
    9:
   10:         String nom;
   11:
   12:         ObjectifMoyenObjet(String _nom) {
   13:                 nom = _nom;
   14:         }
   15:
   16:         @Override
   17:         public String toString() {
   18:                 return nom;
   19:         }
   20: }
```

```
    1: package main;
    2:
    3: import translation.Messages;
    4:
    5: public enum ObjectifObjetMoyen implements Objectif {
    6:
    7:         evaluePerti(Messages.getString("TriadeKernel.ObjectifObjetMoyen.0")), vali
deProc(Messages.getString("TriadeKernel.ObjectifObjetMoyen.1")); //$NON-NLS-1$ //$NON-NLS
-2$
    8:
    9:         public String nom;
   10:
   11:         private ObjectifObjetMoyen(String _nom) {
   12:                 nom = _nom;
   13:         }
   14:
   15:         public String toString() {
   16:                 return nom;
   17:         }
   18: }
```

```
    1: package main;
    2:
    3: import java.io.Serializable;
    4: import java.util.Vector;
    5:
    6: public class ActionTimeListe implements Serializable {
    7:
    8:         /**
    9:          *
   10:          */
   11:         private static final long serialVersionUID = 1062344209989758490L;
   12:         protected Vector<ActionTime> actionTimes;
   13:
   14:         //TODO supprimer cette ligne lorsque l'on aura plus besoin de rÃ©cupÃ©rer
les anciens temps d'actions.
   15:         protected Vector<ActionTime>actionsTime;
   16:
   17:         public ActionTimeListe() {
   18:                 actionTimes = new Vector<ActionTime>();
   19:         }
   20:
   21:         public ActionTime addActionTime(String name) {
   22:                 ActionTime newActionTime = new ActionTime(name, actionTimes.size()
);
   23:                 actionTimes.add(newActionTime);
   24:                 return newActionTime;
   25:         }
   26:
   27:         public Vector<ActionTime> values() {
   28:                 return actionTimes;
   29:         }
   30:
   31:         public void removeActionTime(ActionTime aT) {
   32:                 actionTimes.remove(aT);
   33:         }
   34:
   35:         public ActionTime getActionTimeByName(String name)
   36:         {
   37:                 for (ActionTime aT : actionTimes)
   38:                 {
   39:                         if (aT.toString().equals(name))
   40:                                 return aT;
   41:                 }
   42:                 return null;
   43:         }
   44:
   45:         public void restoreFromOldVersion() {
   46:
   47:                 actionTimes = actionsTime;
   48:                 actionsTime = null;
   49:
   50:         }
   51:
   52: }
```

```
    1: package main;
    2:
    3: import translation.Messages;
    4:
    5: public enum ObjectifLienPartenaire implements Objectif {
    6:         atteindreAccord(Messages.getString("TriadeKernel.ObjectifLienPartenaire.0"
)), appliquerAccord( //$NON-NLS-1$
    7:                         Messages.getString("TriadeKernel.ObjectifLienPartenaire.1"
)), controlerAccord(Messages.getString("TriadeKernel.ObjectifLienPartenaire.2")); //$NON-
NLS-1$ //$NON-NLS-2$
    8:
    9:         public String nom;
   10:         public String[] moyens;
   11:
   12:         private ObjectifLienPartenaire(String _nom) {
   13:                 nom = _nom;
   14:         }
   15:
   16:         public String toString() {
   17:                 return nom;
   18:         }
   19: }
```

```java
package main;

import graphicalUserInterface.Program;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.io.StreamCorruptedException;

import relations.Goal;
import relations.Mean;
import relations.RelationPossibility;
import translation.Messages;

public class JeuRelation implements Serializable {
	/**
	 *
	 */
	private static final long serialVersionUID = 1102146748579962 54L;
	public Goal objectifStructurel;
	public Goal objectifReel;

	public Mean moyenReel;
	public Mean moyenStructurel;

	public JeuRelation() {
		objectifStructurel = RelationPossibility.UNDEFINED_GOAL;
		objectifReel = RelationPossibility.UNDEFINED_GOAL;
		moyenReel = RelationPossibility.UNDEFINED_MEAN;
		moyenStructurel = RelationPossibility.UNDEFINED_MEAN;
	}

	public JeuRelation(boolean noRelationFlag)
	{

		objectifStructurel = RelationPossibility.NORELATION_GOAL;
		objectifReel = RelationPossibility.NORELATION_GOAL;
		moyenReel = RelationPossibility.NORELATION_MEAN;
		moyenStructurel = RelationPossibility.NORELATION_MEAN;
	}

	public JeuRelation(JeuRelation jeuRelation) {
		objectifStructurel = jeuRelation.objectifStructurel;
		objectifReel = jeuRelation.objectifReel;
		moyenStructurel = jeuRelation.moyenStructurel;
		moyenReel = jeuRelation.moyenReel;
	}

	public boolean etatInitial() {
		return (objectifStructurel.equals(RelationPossibility.UNDEFINED_GOAL)
				&& objectifReel.equals(RelationPossibility.UNDEFINED_GOAL)
				&& moyenReel.equals(RelationPossibility.UNDEFINED_MEAN) && moyenStructurel.equals(RelationPossibility.UNDEFINED_MEAN));
	}

	public void print() {
		System.out.println(Messages.getString("JeuRelation.0") + objectifStructurel); //$NON-NLS-1$
		System.out.println(Messages.getString("JeuRelation.1") + objectifReel); //$NON-NLS-1$
		System.out.println(Messages.getString("JeuRelation.2") + moyenStructurel); //$NON-NLS-1$
		System.out.println(Messages.getString("JeuRelation.3") + moyenReel); //$NON-NLS-1$

	}

	public boolean isEmpty(int mode) {
		if (mode == RelationPossibility.RELATIONSTRUCTURELLE)
			return objectifStructurel.equals(RelationPossibility.UNDEFINED_GOAL)
			&& moyenStructurel.equals(RelationPossibility.UNDEFINED_MEAN);
		else
			return moyenReel.equals(RelationPossibility.UNDEFINED_MEAN)
			&& objectifReel.equals(RelationPossibility.UNDEFINED_GOAL);

	}

	public boolean isNoRelation()
	{
		return (objectifStructurel.equals(RelationPossibility.NORELATION_GOAL)
				&& objectifReel.equals(RelationPossibility.NORELATION_GOAL)
				&& moyenReel.equals(RelationPossibility.NORELATION_MEAN) && moyenStructurel.equals(RelationPossibility.NORELATION_MEAN));

	}

	public int getState()
	{

		if (Program.isTriades())
		{
			if (objectifStructurel.equals(objectifReel) && moyenStructurel.equals(moyenReel))
				return RelationComplete.RELATION_OK;

			if (objectifReel.equals(RelationPossibility.UNDEFINED_GOAL))
				return RelationComplete.RELATION_INCOMPLETE;

			else if(objectifStructurel.equals(objectifReel) == false)
				return RelationComplete.RELATION_ECART_OBJECTIF;
			else
				return RelationComplete.RELATION_ECART_MOYEN;
		}
		else
		{
			if (objectifStructurel.equals(RelationPossibility.UNDEFINED_GOAL))
				return RelationComplete.RELATION_INCOMPLETE;
			else
				return RelationComplete.RELATION_OK;
		}
	}

	private void writeObject(ObjectOutputStream out) throws IOException
	{
		out.defaultWriteObject();
	}

	private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException
	{
		ObjectInputStream.GetField fields = in.readFields();
		Object oS = fields.get("objectifStructurel", RelationPossibility.UNDEFINED_GOAL);
```

```
  116:                 if (oS instanceof String)
  117:                 {
  118:                         objectifStructurel = new Goal((String)oS);
  119:                         objectifReel = new Goal((String)fields.get("objectifReel",
 RelationPossibility.UNDEFINED_STRING));
  120:                         moyenStructurel = new Mean((String)fields.get("moyenStruct
urel", RelationPossibility.UNDEFINED_STRING));
  121:                         moyenReel = new Mean((String)fields.get("moyenReel", Relat
ionPossibility.UNDEFINED_STRING));
  122:                 }
  123:                 else if (oS instanceof Goal)
  124:
  125:                 {
  126:                         objectifStructurel = (Goal)oS;
  127:                         objectifReel = (Goal)fields.get("objectifReel", RelationPo
ssibility.UNDEFINED_GOAL);
  128:                         moyenStructurel = (Mean)fields.get("moyenStructurel", Rela
tionPossibility.UNDEFINED_MEAN);
  129:                         moyenReel = (Mean)fields.get("moyenReel", RelationPossibil
ity.UNDEFINED_MEAN);
  130:                 }
  131:                 else
  132:                 {
  133:                         System.err.println("Error while loading a JeuRelation, the
 first object is neither a String, neither a Goal");
  134:                         throw new StreamCorruptedException();
  135:                 }
  136:
  137:         }
  138: }
```

```
    1: package main;
    2:
    3: import translation.Messages;
    4:
    5: public enum ObjectifMoyenActeur implements Objectif {
    6:
    7:         evalComp(Messages.getString("TriadeKernel.ObjectifMoyenActeur.0")), defNor
mes( //$NON-NLS-1$
    8:                         Messages.getString("TriadeKernel.ObjectifMoyenActeur.1")),
 securiseAction( //$NON-NLS-1$
    9:                         Messages.getString("TriadeKernel.ObjectifMoyenActeur.2"));
 //$NON-NLS-1$
   10:
   11:         public String nom;
   12:
   13:         private ObjectifMoyenActeur(String _nom) {
   14:                 nom = _nom;
   15:         }
   16:
   17:         public String toString() {
   18:                 return nom;
   19:         }
   20: }
```

```
    1: package main;
    2:
    3: import translation.Messages;
    4:
    5: public enum StatutObjet {
    6:         nonDef(Messages.getString("TriadeKernel.StatutObjet.0"), new Integer(0)),
outils(Messages.getString("TriadeKernel.StatutObjet.1"), new Integer(1)), methode( //$NON
-NLS-1$ //$NON-NLS-2$
    7:                         Messages.getString("TriadeKernel.StatutObjet.2"), new Inte
ger(2)), stocks(Messages.getString("TriadeKernel.StatutObjet.3"), //$NON-NLS-1$ //$NON-NL
S-2$
    8:                         new Integer(3));
    9:
   10:         public String nom;
   11:         public Integer id;
   12:
   13:         private StatutObjet(String _nom, Integer _id) {
   14:                 nom = _nom;
   15:                 id = _id;
   16:         }
   17:
   18:         public String toString() {
   19:                 return nom;
   20:         }
   21:
   22: }
```

```
    1: package main;
    2:
    3: import java.util.*;
    4:
    5: public class LienHierarchique {
    6:
    7:
    8:         public static Hashtable<String,String[]> moyens;
    9:
   10:         //Moyens(Objectif) pour relation de type N->N-1
   11:
   12:         static protected void initTable()
   13:         {
   14:         moyens = new Hashtable<String,String[]>();
   15:         String[] defReg = {"Ennoncer les rÃ¨gles","Faire dÃ©couvrir les rÃ¨gles","
Rappel d'un retour d'expÃ©rience"};
   16:         String[] evalComp = {"Restitution active du passÃ©","Formes dÃ©cisionnelle
s du sujet acteur","Evaluation 360Â°"};
   17:         String[] donConsigne = {"Donner un ordre","Faire participer Ã  la dÃ©cisio
n","SuggÃ©rer une consigne","Suggestion implicite de la consigne", "Autre"};
   18:         String[] defPoli = {"Ennonciation de la politique"};
   19:         String[] evalPoli = {"Evalutation de la satisfaction du client","ContrÃ´le
 des rÃ©sultats"};
   20:         String[] delegation = {"Lettre de mission Ã©crite","DÃ©lÃ©gation orale","D
Ã©lÃ©gation avec mode d'Ã©valuation","DÃ©lÃ©gation avec procÃ©dure"};
   21:         String[] controle = {"Observation des rÃ©sultats", "Observation de l'Ã©tat
", "Formes dÃ©cisionnelles du sujet acteur","Etude de satisfaction"};
   22:         String[] devComp = {"Formation,stage,...","Forme dÃ©cisionnelle du sujet a
cteur"};
   23:         //String[] resProb = {"Brainstorming", "Triades", "Outils usuels"};
   24:         String[] resConflit = {"NÃ©gociation","MÃ©diation rationnelle","Questionne
ment sur l'action","Formes dÃ©cisionnelles du sujet acteur","Outils de rÃ©solution de pro
blÃ¨me"};
   25:         //String[] vide = {""};
   26:
   27:         String[] nego = {"Questionnement sur la conception des outils et/ou des ac
tivitÃ©s", "Evaluation des compÃ©tences"};
   28:         String[] compRend = {"Description des activitÃ©s", "Description de l'usage
 des outils", "Description des procÃ©dures"};
   29:
   30:
   31:         moyens.put(ObjectifLienHierarchique.defRegles.toString(),defReg);
   32:         moyens.put(ObjectifLienHierarchique.evalCompetences.toString(),evalComp);
   33:         moyens.put(ObjectifLienHierarchique.donnerConsigne.toString(),donConsigne)
;
   34:         moyens.put(ObjectifLienHierarchique.definitionPolitique.toString(),defPoli
);
   35:         moyens.put(ObjectifLienHierarchique.evalutationPolitique.toString(),evalPo
li);
   36:         moyens.put(ObjectifLienHierarchique.delegationActivite.toString(),delegati
on);
   37:         moyens.put(ObjectifLienHierarchique.delegationMission.toString(),delegatio
n);
   38:         moyens.put(ObjectifLienHierarchique.controleExecutionActivite.toString(),c
ontrole);
   39:         moyens.put(ObjectifLienHierarchique.controleUsageOutils.toString(),control
e);
   40:         moyens.put(ObjectifLienHierarchique.developpementCompetences.toString(),de
vComp);
   41:         moyens.put(ObjectifLienHierarchique.resolutionConflit.toString(),resConfli
t);
   42:         moyens.put(ObjectifLienHierarchique.resolutionProbleme.toString(),resConfl
it);
   43:
   44:         moyens.put(ObjectifRetourLienHierarchique.negociation.toString(), nego);
   45:         moyens.put(ObjectifRetourLienHierarchique.compteRendu.toString(), compRend
);
   46:
   47:
   48:
   49:         }
   50:
   51:         //Moyens(Objectif) retour
   52:
   53:
   54:
   55:
   56:         protected ObjectifLienHierarchique objectif;
   57:         protected ObjectifRetourLienHierarchique objectifRetour;
   58:         protected String moyen;
   59:
   60:         public static String[] getStrings(String key)
   61:         {
   62:                 if (moyens == null)
   63:                         initTable();
   64:                 return moyens.get(key);
   65:         }
   66:
   67: }
```

```
1: package main;
2:
3: public enum RelationReelle {
4:         Absente, Incomprehension, Rejet, Emulation;
5: }
```

```
    1: package main;
    2:
    3: import java.util.*;
    4:
    5: public class FormesDecisionnelles {
    6:
    7:         protected static Hashtable<String,String[]> moyens;
    8:
    9:         protected static void initTable()
   10:         {
   11:                 moyens = new Hashtable<String,String[]>();
   12:
   13:                 String[] sujetAct = { "Rechercher plus dâ\200\231informations sur
lâ\200\231acteur", "Demander de dÃ©crire avec plus de prÃ©cision une sÃ©quence dâ\200\231
Ã©vÃ©nements","Rechercher les dÃ©cisions de la personne qui raconte une sÃ©quence dâ\200
\231actions","Demander le type dâ\200\231effet dâ\200\231une action","Sâ\200\231enquÃ©rir
 de la cohÃ©rence entre objectif et moyen utilisÃ© pour lâ\200\231atteindre","Interroger
ce quâ\200\231elle a pris en compte pour les dÃ©cisions","Demander quels sont les importa
nts qui dirigent lâ\200\231action","Demande de la hiÃ©rarchisation faite pour prendre sa
dÃ©cision"};
   14:                 String[] sujetDec =  {"Ordonner un acte","Enoncer la loi ou une rÃ
¨gle convenue","Donner des informations","Donner des consignes","Annoncer une stratÃ©gie
dâ\200\231action"};
   15:                 String[] sujetRai = {"Ecouter","Appropriation du problÃ¨me de lâ
\200\231autre en les rapportant Ã  son expÃ©rience","InterprÃ©ter ce qui est dit, fait (a
ctions ou comportements)","Evaluer dires et faire en les comparant Ã  une norme (sociale
ou individuelle)","Reformuler","Rechercher plus dâ\200\231informations sur la situation",
"Enoncer un constat ou un effet","Faire des hypothÃ¨ses de causes (qui expliquent les Ã©v
Ã©nements)","Emettre des hypothÃ¨ses de solution Ã  un problÃ¨me ou difficultÃ©","Recherc
her une causalitÃ©","Explique (ce quâ\200\231il faut faire ou pourquoi les Ã©vÃ©nements s
e sont passÃ©s de la sorte)","Demander des explications pour comprendre une idÃ©e ou une
situation","Donner un conseil"};
   16:                 String[] def =  {"ProfÃ©rer une menace ou mettre en garde", "Se me
ttre en dÃ©fense (justification,fuite,passivitÃ©,mise en Å\223uvre dâ\200\231un mÃ©canism
e de dÃ©fense psychique)"};
   17:                 String[] univRel = { "Entrer en relation selon un des quatre objec
tifs de relation","Partager toutes les informations ou en cacher quelques unes.","Prendre
 lâ\200\231autre dans sa totalitÃ© ou le considÃ©rer selon une partie de son identitÃ©.",
"Soumettre, dominer, rechercher une relation dâ\200\231Ã©galitÃ©"};
   18:
   19:                 moyens.put(ObjectifFormesDecisionnelles.sujetActeur.toString(),suj
etAct);
   20:                 moyens.put(ObjectifFormesDecisionnelles.sujetDecideur.toString(),s
ujetDec);
   21:                 moyens.put(ObjectifFormesDecisionnelles.sujetRaisonneur.toString()
,sujetRai);
   22:                 moyens.put(ObjectifFormesDecisionnelles.defensive.toString(),def);
   23:                 moyens.put(ObjectifFormesDecisionnelles.universRelation.toString()
,univRel);
   24:
   25:
   26:         }
   27:
   28:         public static String[] getStrings(String key)
   29:         {
   30:                 if (moyens == null)
   31:                         initTable();
   32:                 return (moyens.get(key));
   33:         }
   34:
   35: }
```

```
 1: package main;
 2:
 3:
 4: public class Objet extends Pole{
 5:
 6:         /**
 7:          *
 8:          */
 9:         private static final long serialVersionUID = 5593038862364253170L;
10:         public String nom;
11:         public StatutObjet statut;
12:
13:         public Objet(String _nom,StatutObjet _statut)
14:         {
15:                 super();
16:                 nom = _nom;
17:                 statut = _statut;
18:         }
19:
20:         public Objet()
21:         {
22:                 this("Non dÃ©fini",StatutObjet.nonDef);
23:         }
24:
25:         public String toString()
26:         {
27:                 return nom;
28:         }
29:
30:         public int getType()
31:         {
32:                 return Pole.OBJET;
33:         }
34:
35: }
```

```
    1: package main;
    2:
    3: import java.util.*;
    4:
    5: public class LienPartenaire {
    6:
    7:
    8:         protected static Hashtable<String,String[]> moyens;
    9:
   10:
   11:         protected ObjectifLienPartenaire objectif;
   12:         protected String moyen;
   13:
   14:         protected static void initTable()
   15:         {
   16:                 moyens = new Hashtable<String,String[]>();
   17:                 String[] attAccord = {"NÃ©gociation"};
   18:                 String[] appAccord = {"ExÃ©cution"};
   19:                 String[] conAccord = {"Observation de l'activitÃ©"};
   20:
   21:                 moyens.put(ObjectifLienPartenaire.appliquerAccord.toString(), appA
ccord);
   22:                 moyens.put(ObjectifLienPartenaire.atteindreAccord.toString(), attA
ccord);
   23:                 moyens.put(ObjectifLienPartenaire.controlerAccord.toString(), conA
ccord);
   24:
   25:         }
   26:
   27:
   28:         public static String[] getStrings(String key)
   29:         {
   30:                 if (moyens == null)
   31:                         initTable();
   32:                 return (moyens.get(key));
   33:         }
   34:
   35:
   36:
   37:
   38: }
```

```
    1: package main;
    2:
    3: import java.util.ArrayList;
    4: import java.util.Hashtable;
    5:
    6: import relations.Mean;
    7: import translation.Messages;
    8:
    9: public class Liens {
   10:
   11:         protected static Hashtable<Objectif, String[]> moyens;
   12:
   13:         protected static void initTable() {
   14:
   15:                 moyens = new Hashtable<Objectif, String[]>();
   16:
   17:                 // Lien hierarchique
   18:                 String[] defReg = { Messages.getString("TriadeKernel.Liens.0"), //
$NON-NLS-1$
   19:                                 Messages.getString("TriadeKernel.Liens.1"), Messag
es.getString("TriadeKernel.Liens.2") }; //$NON-NLS-1$ //$NON-NLS-2$
   20:                 String[] evalComp = { Messages.getString("TriadeKernel.Liens.3"),
//$NON-NLS-1$
   21:                                 Messages.getString("TriadeKernel.Liens.4"), Messag
es.getString("TriadeKernel.Liens.5") }; //$NON-NLS-1$ //$NON-NLS-2$
   22:                 String[] donConsigne = { Messages.getString("TriadeKernel.Liens.6"
), //$NON-NLS-1$
   23:                                 Messages.getString("TriadeKernel.Liens.7"), Messag
es.getString("TriadeKernel.Liens.8"), //$NON-NLS-1$ //$NON-NLS-2$
   24:                                 Messages.getString("TriadeKernel.Liens.9"), Messag
es.getString("TriadeKernel.Liens.10") }; //$NON-NLS-1$ //$NON-NLS-2$
   25:                 String[] defPoli = { Messages.getString("TriadeKernel.Liens.11") }
; //$NON-NLS-1$
   26:                 String[] evalPoli = { Messages.getString("TriadeKernel.Liens.12"),
 //$NON-NLS-1$
   27:                                 Messages.getString("TriadeKernel.Liens.13") }; //$
NON-NLS-1$
   28:                 String[] delegation = { Messages.getString("TriadeKernel.Liens.14"
), Messages.getString("TriadeKernel.Liens.15"), //$NON-NLS-1$ //$NON-NLS-2$
   29:                                 Messages.getString("TriadeKernel.Liens.16"), //$NO
N-NLS-1$
   30:                                 Messages.getString("TriadeKernel.Liens.17") }; //$
NON-NLS-1$
   31:                 String[] controle = { Messages.getString("TriadeKernel.Liens.18"),
 //$NON-NLS-1$
   32:                                 Messages.getString("TriadeKernel.Liens.19"), //$NO
N-NLS-1$
   33:                                 Messages.getString("TriadeKernel.Liens.20"), //$NO
N-NLS-1$
   34:                                 Messages.getString("TriadeKernel.Liens.21") }; //$
NON-NLS-1$
   35:                 String[] devComp = { Messages.getString("TriadeKernel.Liens.22"),
//$NON-NLS-1$
   36:                                 Messages.getString("TriadeKernel.Liens.23") }; //$
NON-NLS-1$
   37:                 // String[] resProb = {"Brainstorming", "Triades", "Outils usuels"
};
   38:                 String[] resConflit = { Messages.getString("TriadeKernel.Liens.24"
), Messages.getString("TriadeKernel.Liens.25"), //$NON-NLS-1$ //$NON-NLS-2$
   39:                                 Messages.getString("TriadeKernel.Liens.26"), //$NO
N-NLS-1$
   40:                                 Messages.getString("TriadeKernel.Liens.27"), //$NO
N-NLS-1$
   41:                                 Messages.getString("TriadeKernel.Liens.28") }; //$
NON-NLS-1$
   42:                 // String[] vide = {""};
   43:
   44:                 String[] nego = {
   45:                                 Messages.getString("TriadeKernel.Liens.29"), //$NO
N-NLS-1$
   46:                                 Messages.getString("TriadeKernel.Liens.30") }; //$
NON-NLS-1$
   47:                 String[] compRend = { Messages.getString("TriadeKernel.Liens.31"),
 //$NON-NLS-1$
   48:                                 Messages.getString("TriadeKernel.Liens.32"), //$NO
N-NLS-1$
   49:                                 Messages.getString("TriadeKernel.Liens.33") }; //$
NON-NLS-1$
   50:
   51:                 moyens.put(ObjectifLienHierarchique.defRegles, defReg);
   52:                 moyens.put(ObjectifLienHierarchique.evalCompetences, evalComp);
   53:                 moyens.put(ObjectifLienHierarchique.donnerConsigne, donConsigne);
   54:                 moyens.put(ObjectifLienHierarchique.definitionPolitique, defPoli);
   55:                 moyens.put(ObjectifLienHierarchique.evalutationPolitique, evalPoli
);
   56:                 moyens.put(ObjectifLienHierarchique.delegationActivite, delegation
);
   57:                 moyens.put(ObjectifLienHierarchique.delegationMission, delegation)
;
   58:                 moyens
   59:                                 .put(ObjectifLienHierarchique.controleExecutionAct
ivite,
   60:                                                 controle);
   61:                 moyens.put(ObjectifLienHierarchique.controleUsageOutils, controle)
;
   62:                 moyens.put(ObjectifLienHierarchique.developpementCompetences, devC
omp);
   63:                 moyens.put(ObjectifLienHierarchique.resolutionConflit, resConflit)
;
   64:                 moyens.put(ObjectifLienHierarchique.resolutionProbleme, resConflit
);
   65:
   66:                 moyens.put(ObjectifRetourLienHierarchique.negociation, nego);
   67:                 moyens.put(ObjectifRetourLienHierarchique.compteRendu, compRend);
   68:
   69:                 // Lien partenaire
   70:
   71:                 String[] attAccord = { Messages.getString("TriadeKernel.Liens.34")
 }; //$NON-NLS-1$
   72:                 String[] appAccord = { Messages.getString("TriadeKernel.Liens.35")
 }; //$NON-NLS-1$
   73:                 String[] conAccord = { Messages.getString("TriadeKernel.Liens.36")
 }; //$NON-NLS-1$
   74:
   75:                 moyens.put(ObjectifLienPartenaire.appliquerAccord, appAccord);
   76:                 moyens.put(ObjectifLienPartenaire.atteindreAccord, attAccord);
   77:                 moyens.put(ObjectifLienPartenaire.controlerAccord, conAccord);
   78:
   79:                 // Formes dÃ©cisionnelles
   80:                 /*String[] sujetAct = {
   81:                                 Messages.getString("TriadeKernel.Liens.37"), //$NO
N-NLS-1$
   82:                                 Messages.getString("TriadeKernel.Liens.38"), //$NO
N-NLS-1$
   83:                                 Messages.getString("TriadeKernel.Liens.39"), //$NO
N-NLS-1$
   84:                                 Messages.getString("TriadeKernel.Liens.40"), //$NO
N-NLS-1$
   85:                                 Messages.getString("TriadeKernel.Liens.41"), //$NO
N-NLS-1$
   86:                                 Messages.getString("TriadeKernel.Liens.42"), //$NO
N-NLS-1$
   87:                                 Messages.getString("TriadeKernel.Liens.43"), //$NO
N-NLS-1$
```

```
   88:                                 Messages.getString("TriadeKernel.Liens.44") }; //$
NON-NLS-1$
   89:                 String[] sujetDec = { Messages.getString("TriadeKernel.Liens.45"),
 //$NON-NLS-1$
   90:                                 Messages.getString("TriadeKernel.Liens.46"), //$NO
N-NLS-1$
   91:                                 Messages.getString("TriadeKernel.Liens.47"), Messa
ges.getString("TriadeKernel.Liens.48"), //$NON-NLS-1$ //$NON-NLS-2$
   92:                                 Messages.getString("TriadeKernel.Liens.49") }; //$
NON-NLS-1$
   93:                 String[] sujetRai = {
   94:                                 Messages.getString("TriadeKernel.Liens.50"), //$NO
N-NLS-1$
   95:                                 Messages.getString("TriadeKernel.Liens.51"), //$NO
N-NLS-1$
   96:                                 Messages.getString("TriadeKernel.Liens.52"), //$NO
N-NLS-1$
   97:                                 Messages.getString("TriadeKernel.Liens.53"), //$NO
N-NLS-1$
   98:                                 Messages.getString("TriadeKernel.Liens.54"), //$NO
N-NLS-1$
   99:                                 Messages.getString("TriadeKernel.Liens.55"), //$NO
N-NLS-1$
  100:                                 Messages.getString("TriadeKernel.Liens.56"), //$NO
N-NLS-1$
  101:                                 Messages.getString("TriadeKernel.Liens.57"), //$NO
N-NLS-1$
  102:                                 Messages.getString("TriadeKernel.Liens.58"), //$NO
N-NLS-1$
  103:                                 Messages.getString("TriadeKernel.Liens.59"), //$NO
N-NLS-1$
  104:                                 Messages.getString("TriadeKernel.Liens.60"), //$NO
N-NLS-1$
  105:                                 Messages.getString("TriadeKernel.Liens.61"), //$NO
N-NLS-1$
  106:                                 Messages.getString("TriadeKernel.Liens.62") }; //$
NON-NLS-1$
  107:                 String[] def = {
  108:                                 Messages.getString("TriadeKernel.Liens.63"), //$NO
N-NLS-1$
  109:                                 Messages.getString("TriadeKernel.Liens.64") }; //$
NON-NLS-1$
  110:                 String[] univRel = {
  111:                                 Messages.getString("TriadeKernel.Liens.65"), //$NO
N-NLS-1$
  112:                                 Messages.getString("TriadeKernel.Liens.66"), //$NO
N-NLS-1$
  113:                                 Messages.getString("TriadeKernel.Liens.67"), //$NO
N-NLS-1$
  114:                                 Messages.getString("TriadeKernel.Liens.68") }; //$
NON-NLS-1$
  115:                  */
  116:
  117:                 String[] seCentrer = {
  118:                                 Messages.getString("TriadeKernel.ObjectifFormesDec
isionnelles.0"), //$NON-NLS-1$
  119:                                 Messages.getString("TriadeKernel.ObjectifFormesDec
isionnelles.1"), //$NON-NLS-1$
  120:                                 Messages.getString("TriadeKernel.ObjectifFormesDec
isionnelles.2"), //$NON-NLS-1$
  121:                                 Messages.getString("TriadeKernel.ObjectifFormesDec
isionnelles.3"), //$NON-NLS-1$
  122:                                 Messages.getString("TriadeKernel.ObjectifFormesDec
isionnelles.4")};
  123:
  124:                 moyens.put(ObjectifFormesDecisionnelles.sujetActeur, seCentrer);
  125:
  126: /*              moyens.put(ObjectifFormesDecisionnelles.sujetActeur, sujetAct);
  127:                 moyens.put(ObjectifFormesDecisionnelles.sujetDecideur, sujetDec);
  128:                 moyens.put(ObjectifFormesDecisionnelles.sujetRaisonneur, sujetRai)
;
  129:                 moyens.put(ObjectifFormesDecisionnelles.defensive, def);
  130:                 moyens.put(ObjectifFormesDecisionnelles.universRelation, univRel);
  131: */
  132:                 // Lien Acteur objet
  133:                 String[] concep = { Messages.getString("TriadeKernel.Liens.69") };
 //$NON-NLS-1$
  134:                 String[] util = { Messages.getString("TriadeKernel.Liens.70"), //$
NON-NLS-1$
  135:                                 Messages.getString("TriadeKernel.Liens.71") }; //$
NON-NLS-1$
  136:                 String[] recup = { Messages.getString("TriadeKernel.Liens.72"), //
$NON-NLS-1$
  137:                                 Messages.getString("TriadeKernel.Liens.73") }; //$
NON-NLS-1$
  138:                 String[] eval = { Messages.getString("TriadeKernel.Liens.74") }; /
/$NON-NLS-1$
  139:
  140:                 moyens.put(ObjectifActeurObjet.conception, concep);
  141:                 moyens.put(ObjectifActeurObjet.utilisation, util);
  142:                 moyens.put(ObjectifActeurObjet.recupInfo, recup);
  143:                 moyens.put(ObjectifActeurObjet.eval, eval);
  144:
  145:                 // Lien Acteur Moyen
  146:                 String[] appl = { Messages.getString("TriadeKernel.Liens.75"), Mes
sages.getString("TriadeKernel.Liens.76") }; //$NON-NLS-1$ //$NON-NLS-2$
  147:                 String[] utilise = { Messages.getString("TriadeKernel.Liens.77") }
; //$NON-NLS-1$
  148:                 moyens.put(ObjectifActeurMoyen.applique, appl);
  149:                 moyens.put(ObjectifActeurMoyen.utilise, utilise);
  150:
  151:                 // Lien Moyen Acteur
  152:                 String[] compUtil = { Messages.getString("TriadeKernel.Liens.78")
}; //$NON-NLS-1$
  153:                 String[] donneInfo = { Messages.getString("TriadeKernel.Liens.79")
 }; //$NON-NLS-1$
  154:
  155:                 moyens.put(ObjectifMoyenActeur.evalComp, compUtil);
  156:                 moyens.put(ObjectifMoyenActeur.defNormes, donneInfo);
  157:                 moyens.put(ObjectifMoyenActeur.securiseAction, donneInfo);
  158:
  159:                 // Lien Moyen Moyen
  160:                 String[] pasMoyen = { Messages.getString("TriadeKernel.Liens.80")
}; //$NON-NLS-1$
  161:
  162:                 moyens.put(ObjectifMoyenMoyen.determineNorme, pasMoyen);
  163:                 moyens.put(ObjectifMoyenMoyen.evaluePerti, donneInfo);
  164:
  165:                 // Lien Moyen Objet
  166:
  167:                 String[] defNormes = { Messages.getString("TriadeKernel.Liens.81")
 }; //$NON-NLS-1$
  168:
  169:                 moyens.put(ObjectifMoyenObjet.estAdapteEff, pasMoyen);
  170:                 moyens.put(ObjectifMoyenObjet.cadreRea, defNormes);
  171:                 moyens.put(ObjectifMoyenObjet.faciliRea, defNormes);
  172:                 moyens.put(ObjectifMoyenObjet.securiseRea, defNormes);
  173:
  174:                 // Lien Objet Acteur
  175:                 String[] evalCompObj = { Messages.getString("TriadeKernel.Liens.82
") }; //$NON-NLS-1$
  176:
  177:                 moyens.put(ObjectifObjetActeur.evalCompetences, evalCompObj);
  178:
```

```
179:                // Lien Objet Moyen
180:
181:                moyens.put(ObjectifObjetMoyen.evaluePerti, donneInfo);
182:                moyens.put(ObjectifObjetMoyen.valideProc, donneInfo);
183:
184:                // Lien Objet objet
185:                // Aucune relation de ce type
186:
187:        }
188:
189:        public static ArrayList<Mean> getStrings(Objectif obj) {
190:                if (moyens == null)
191:                        initTable();
192:
193:                ArrayList<Mean> result = new ArrayList<Mean>();
194:
195:                for(String moyen : moyens.get(obj)) {
196:                        result.add(new Mean(moyen));
197:                }
198:
199:                return result;
200:        }
201:
202: }
```

```
    1: package main;
    2:
    3: import translation.Messages;
    4:
    5: public enum ObjectifMoyenMoyen implements Objectif {
    6:
    7:         determineNorme(Messages.getString("TriadeKernel.ObjectifMoyenMoyen.0")), e
valuePerti( //$NON-NLS-1$
    8:                         Messages.getString("TriadeKernel.ObjectifMoyenMoyen.1"));
//$NON-NLS-1$
    9:
   10:         public String nom;
   11:
   12:         private ObjectifMoyenMoyen(String _nom) {
   13:                 nom = _nom;
   14:         }
   15:
   16:         public String toString() {
   17:                 return nom;
   18:         }
   19: }
```

```
    1: package main;
    2:
    3: public class Triade {
    4:         /*
    5:         private Pole[] poles;
    6:         private RelationStructurelle relationStruct;
    7:         private RelationReelle relationReelle;
    8:
    9:         public Triade()
   10:         {
   11:                 poles = new Pole[3];
   12:                 relationStruct = RelationStructurelle.NonDefinie;
   13:                 relationReelle = RelationReelle.NonDefinie;
   14:         }
   15:
   16:         public Triade(Acteur acteur1, Acteur acteur2, String _objet)
   17:         {
   18:                 poles = new Pole[3];
   19:                 poles[0] = acteur1;
   20:                 poles[1] = acteur2;
   21:                 poles[2] = new Objet(_objet);
   22:                 relationReelle = RelationReelle.NonDefinie;
   23:         }
   24:
   25:
   26:         private void evaluerRelationStructurelle()
   27:         {
   28:                 if (acteur1.statut == null || acteur2.statut == null)
   29:                 {
   30:                         System.out.println("Les statuts des 2 acteurs doivent Ãªtr
e dÃ©finis pour pouvoir Ã©valuer la " +
   31:                                                 "relation structurelle entre ces deux acte
urs");
   32:                         relationStruct = null;
   33:                 }
   34:                 else
   35:                 {
   36:                         relationStruct = Data.getRelationStructurelle(acteur1.stat
ut.toString()+acteur2.statut.toString());
   37:
   38:                 }
   39:
   40:
   41:         }
   42:
   43:         public Pole[] getPoles()
   44:         {
   45:                 return poles;
   46:         }
   47:         */
   48: }
   49:
   50:
   51:
   52:
```

```
1: package main;
2:
3: public class RelationStructurelle{
4:
5: }
```

```
    1: package main;
    2:
    3: import java.util.Vector;
    4:
    5: import relations.EnsembleRelation;
    6: import relations.RelationDescription;
    7:
    8: import dataPack.Acteur;
    9: import dataPack.DataPack;
   10:
   11: public class RelationsPossibles {
   12:
   13:         public Vector<String> objectifs;
   14:         public Vector<String[]> moyens;
   15:
   16:
   17:         protected RelationsPossibles(Vector<String> objStruct,
   18:                         Vector<String[]> moyStruct) {
   19:                 objectifs = objStruct;
   20:                 moyens = moyStruct;
   21:
   22:         }
   23:
   24:         public static RelationsPossibles getRelationsPossibles(Integer aArg,
   25:                         Integer bArg, DataPack datapack, int typeRelation) {
   26:                 EnsembleRelation relationSet = datapack.getRelations();
   27:                 Vector<String> objectif = new Vector<String>();
   28:                 Vector<Vector<String>> moyens = new Vector<Vector<String>>();
   29:
   30:                 for (RelationDescription description : relationSet.getRelationsPos
sibles(a, b))
   31:
   32:                         RelationsPossibles result = new RelationsPossibles(null, n
ull);
   33:         }
   34:
   35:
   36:         System.out.println("Relation possible avant traitement : " + aArg +
   37:                         "   -   " + bArg); Integer statutA; if (aArg.intValue() <=
 -10 &&
   38:                                         aArg.intValue() > -50) statutA = new Integ
er(6); else if (aArg.intValue()
   39:                                                         <= -50) statutA = aArg; el
se if (aArg.intValue() >= 0) statutA = new
   40:                                                         Integer(Statut.values()[((
Acteur) datapack
   41:                                                                         .getActor(
aArg)).getIdStatut().intValue()].rang); else statutA = new
   42:                                                                         Integer(1)
;
   43:
   44:                         Integer statutB; if (bArg.intValue() <= -10 && bArg.intVal
ue() > -50)
   45:                                 statutB = new Integer(6); else if (bArg.intValue()
 <= -50) statutB =
   46:                                         bArg; else if (bArg.intValue() >= 0) statu
tB = new
   47:                                         Integer(Statut.values()[((Acteur) datapack
   48:                                                         .getActor(bArg)).getIdStat
ut().intValue()].rang); else statutB = new
   49:                                                         Integer(1);
   50:
   51:                         System.out .println("RelationPossibles : " + statutA + "
-  " +
   52:                                         statutB); Vector<String[]> moyens = new Ve
ctor<String[]>();
   53:                                         Vector<String> objectif = new Vector<Strin
g>(); String[] undefArray = {
   54:                                                                                 UNDEFINED }; objectif.add(
UNDEFINED); moyens.add(undefArray);
   55:                                                                                 RelationsPossibles result;
   56:
   57:                                                                                 if (statutA.intValue() > -
10 && statutB.intValue() > -10)
   58:                                                                                 {
   59:                                                                                         int rangA = statut
A.intValue();
   60:                                                                                         int rangB = statut
B.intValue();
   61:
   62:                                                                                         // Les deux poles
dont des acteurs
   63:                                                                                         for (ObjectifLienP
artenaire obj :ObjectifLienPartenaire.values())
   64:                                                                                         {
   65:                                                                                                 objectif.a
dd(obj.toString());
   66:                                                                                                 moyens.add
(Liens.getStrings(obj));
   67:                                                                                         }
   68:
   69:                                                                                         objectif.add(Objec
tifFormesDecisionnelles.sujetActeur.toString());
   70:                                                                                         moyens.add(Liens.g
etStrings(ObjectifFormesDecisionnelles.sujetActeur));
   71:
   72:                                                                                         if (rangA != -1 &&
 rangB != -1) { if (rangA <= rangB) {
   73:                                                                                                 for (Objec
tifLienHierarchique obj : ObjectifLienHierarchique .values()) {
   74:                                                                                                         ob
jectif.add(obj.toString()); moyens.add(Liens.getStrings(obj));
   75:                                                                                                 }
   76:                                                                                         }
   77:                                                                                         else
   78:                                                                                         {
   79:                                                                                                 for (Objec
tifRetourLienHierarchique obj : ObjectifRetourLienHierarchique .values())
   80:                                                                                                 {
   81:                                                                                                         ob
jectif.add(obj.toString());
   82:                                                                                                         mo
yens.add(Liens.getStrings(obj));
   83:                                                                                                 }
   84:                                                                                         }
   85:                                                                                         }
   86:                                                                                 }
   87:                                                                                 else
   88:                                                                                 {
   89:                                                                                         if (statutA.intVal
ue()<= -50 && statutB.intValue() <= -50)
   90:                                                                                         {
   91:                                                                                                 if (statut
A <= -100)
   92:                                                                                                 {
   93:                                                                                                         fo
r (ObjectifObjetMoyen obj : ObjectifObjetMoyen.values())
   94:                                                                                                         {
   95:
objectif.add(obj.toString());
   96:
moyens.add(Liens.getStrings(obj));
   97:                                                                                                         }
   98:
```

```java
                        } else if (statutB <= -100)
                        {
                            for (ObjectifMoyenObjet obj :ObjectifMoyenObjet.values())
                            {
                                objectif.add(obj.toString());
                                moyens.add(Liens.getStrings(obj));
                            }
                        }
                        else
                        {
                            for (ObjectifMoyenMoyen obj: ObjectifMoyenMoyen.values())
                            {
                                objectif.add(obj.toString());
                                moyens.add(Liens.getStrings(obj));
                            }
                        }
                    }
                    else
                    {
                        if (statutA.intValue() <=-50)
                        {
                            if (statutA.intValue() <= -100)
                            {
                                for (ObjectifObjetActeur obj :ObjectifObjetActeur .values())
                                {
                                    objectif.add(obj.toString());
                                    moyens.add(Liens.getStrings(obj));
                                }
                            }
                            else
                            {
                                for (ObjectifMoyenActeur obj : ObjectifMoyenActeur .values())
                                {
                                    objectif.add(obj.toString());
                                    moyens.add(Liens.getStrings(obj));
                                }
                            }
                        }
                        else
                        {
                            if (statutB <= -100)
                            {
                                for (ObjectifActeurObjet obj : ObjectifActeurObjet .values()) {
                                    objectif.add(obj.toString());
                                    moyens.add(Liens.getStrings(obj));
                                }
                            }
                            else
                            {
                                for (ObjectifActeurMoyen obj : ObjectifActeurMoyen .values())
                                {
                                    objectif.add(obj.toString());
                                    moyens.add(Liens.getStrings(obj));
                                }
                            }
                        }
                    }

                }

                if (typeRelation == RELATIONREELLE) { for (ObjectifFormesDecisionnelles
                        forme : ObjectifFormesDecisionnelles.values()) {
                    objectif.add(forme.toString()); moyens.add(Liens.getStrings(forme)); } }

                result = new RelationsPossibles(objectif, moyens);

                return result;

        }


}
```

```
    1: package main;
    2:
    3:
    4: public enum ObjectifFormesDecisionnelles implements Objectif {
    5:
    6: /*      sujetRaisonneur(Messages.getString("TriadeKernel.ObjectifFormesDecisionnel
les.0")), sujetActeur( //$NON-NLS-1$
    7:                         Messages.getString("TriadeKernel.ObjectifFormesDecisionnel
les.1")), sujetDecideur( //$NON-NLS-1$
    8:                         Messages.getString("TriadeKernel.ObjectifFormesDecisionnel
les.2")), defensive( //$NON-NLS-1$
    9:                         Messages.getString("TriadeKernel.ObjectifFormesDecisionnel
les.3")), universRelation( //$NON-NLS-1$
   10:                         Messages.getString("TriadeKernel.ObjectifFormesDecisionnel
les.4")); //$NON-NLS-1$
   11: */
   12:         sujetActeur("Se centrer sur l'acteur");
   13:
   14:         protected String nom;
   15:
   16:         private ObjectifFormesDecisionnelles(String _nom) {
   17:                 nom = _nom;
   18:         }
   19:
   20:         @Override
   21:         public String toString() {
   22:                 return nom;
   23:         }
   24: }
```

```
    1: package main;
    2:
    3: import translation.Messages;
    4:
    5: public enum ObjectifActeurMoyen implements Objectif {
    6:
    7:         utilise(Messages.getString("TriadeKernel.ObjectifActeurMoyen.0")), appliqu
e(Messages.getString("TriadeKernel.ObjectifActeurMoyen.1")); //$NON-NLS-1$ //$NON-NLS-2$
    8:
    9:         public String nom;
   10:
   11:         private ObjectifActeurMoyen(String _nom) {
   12:                 nom = _nom;
   13:         }
   14:
   15:         public String toString() {
   16:                 return nom;
   17:         }
   18: }
```

```
   1: package client;
   2:
   3: import graphicalUserInterface.Program;
   4:
   5: import java.awt.Component;
   6: import java.awt.event.MouseEvent;
   7: import java.awt.event.MouseListener;
   8: import java.util.Hashtable;
   9: import java.util.Vector;
  10:
  11: import javax.swing.JLabel;
  12: import javax.swing.JPanel;
  13: import javax.swing.JTree;
  14: import javax.swing.tree.DefaultMutableTreeNode;
  15: import javax.swing.tree.DefaultTreeModel;
  16: import javax.swing.tree.TreeCellRenderer;
  17: import javax.swing.tree.TreePath;
  18: import javax.swing.tree.TreeSelectionModel;
  19:
  20: import models.Brick;
  21: import models.BrickEdge;
  22: import models.BrickVertex;
  23: import models.BrickView;
  24: import client.export.ExportModel;
  25: import client.export.ExportsContextualMenu;
  26: import dataPack.Content;
  27: import dataPack.Groupe;
  28: import dataPack.MyDefaultMutableTreeNode;
  29: import dataPack.MyMutableJTree;
  30: import dataPack.TreeActorsCellRendere;
  31: import dataPack.TreeListener;
  32:
  33: public class NavigationTree extends MyMutableJTree implements TreeCellRenderer , M
ouseListener {
  34:
  35:         /**
  36:          *
  37:          */
  38:         private static final long serialVersionUID = 2949464319099111849L;
  39:
  40:         protected DefaultTreeModel treeModel;
  41:         protected JPanel schemaView;
  42:         protected String currentStep;
  43:         protected BrickView<?, ?> currentBrickView;
  44:         protected Hashtable<String, BrickView<?,?>> brickViews;
  45:         final protected Hashtable<Content, DefaultMutableTreeNode> exportGroups;
  46:
  47:         protected Session session;
  48:
  49:         public NavigationTree() {
  50:                 //TODO a virer une fois ces arbre ne seront plus sauvÃ©
  51:                 super();
  52:                 exportGroups = null;
  53:         }
  54:
  55:         public NavigationTree(Session session, JPanel schemaView) {
  56:                 System.out.println("NavigationTree.NavigationTree()");
  57:                 this.schemaView = schemaView;
  58:                 this.session = session;
  59:                 brickViews = new Hashtable<String, BrickView<?,?>>();
  60:                 exportGroups = new Hashtable<Content, DefaultMutableTreeNode>();
  61:
  62:                 session.setNavigationTree(this);
  63:
  64:                 setEditable(false);
  65:
  66:                 buildTree();
  67:
  68:                 getModel().addTreeModelListener(this);
  69:         }
  70:
  71:         private void buildTree() {
  72:                 currentStep = ""; //$NON-NLS-1$
  73:
  74:                 DefaultMutableTreeNode root = new Groupe("Root"); //$NON-NLS-1$
  75:
  76:                 treeModel = new DefaultTreeModel(root);
  77:                 getSelectionModel().setSelectionMode(
  78:                                 TreeSelectionModel.SINGLE_TREE_SELECTION);
  79:
  80:                 setModel(treeModel);
  81:                 setCellRenderer(this);
  82:                 addMouseListener(this);
  83:                 setRootVisible(false);
  84:
  85:                 /* construction du model avec la seesion */
  86:                 /* ajoute les schemas */
  87:                 for(int i = 0 ; i < session.datapack.getSteps().size() ; i++) {
  88:                         Brick navigationSchema = session.getBrickList().elementAt(
i);
  89:
  90:                         if(navigationSchema.getVertices().size() != 0) {
  91:                                 // ajoute les dossiers d'Ã©tape
  92:                                 DefaultMutableTreeNode stepGroup = new Groupe(sess
ion.datapack.getSteps().elementAt(i));
  93: //                              stepGroup.setUserObject(navigationSchema);
  94:                                 root.add(stepGroup);
  95:                                 expandPath(new TreePath(stepGroup.getPath()));
  96:
  97:                                 //ajouter le schema de navigation
  98:                                 //DefaultMutableTreeNode navigationSchemaNode = ne
w DefaultMutableTreeNode(navigationSchema);
  99:                                 //stepGroup.add(navigationSchemaNode);
 100:
 101:                                 Groupe exportGroup = new Groupe("ExportÃ©s"); //$N
ON-NLS-1$
 102:                                 exportGroups.put(navigationSchema, exportGroup);
 103:                                 //ajouter les sous schema
 104:                                 for(BrickVertex vertex : navigationSchema.getVerti
ces()) {
 105:                                         Brick brick = (Brick)vertex.getContent();
 106:                                         stepGroup.add(new DefaultMutableTreeNode(b
rick));
 107:                                         exportGroups.put(brick, exportGroup);
 108: //                                      System.out.println("Put : " + brick);
 109:
 110:                                         Vector<ExportModel> brickExported = sessio
n.getExports(brick);
 111:                                         if (brickExported != null) {
 112:                                                 for (ExportModel exportModel : bri
ckExported) {
 113:                                                         // ajoute les schemas axpo
rtÃ©s dans le dossier
 114:                                                         DefaultMutableTreeNode exp
ort = new DefaultMutableTreeNode(
 115:                                                                         exportMode
l);
 116:                                                         exportGroup.add(export);
 117:                                                         export.setUserObject(expor
tModel);
 118:                                                 }
 119:                                         }
 120:                                 }
 121:
```

```
  122:                                /* Ajoute le dossier des exports */
  123:                                stepGroup.add(exportGroup);
  124:                        }
  125:                }
  126:
  127:                treeModel.reload();
  128:                validate();
  129:                repaint();
  130:        }
  131:
  132:        public void addExport(ExportModel newExportModel, Brick schema) {
  133:                DefaultMutableTreeNode exportGroup = getExportGroup(schema);
  134:                DefaultMutableTreeNode node = new DefaultMutableTreeNode(newExport
Model);
  135:                exportGroup.add(node);
  136:                makeVisible(new TreePath(node.getPath()));
  137:
  138:                treeModel.reload();
  139:                validate();
  140:        }
  141:
  142:        public DefaultMutableTreeNode getExportGroup(Content content) {
  143: //                System.out.println("Get " + schemaId);
  144:                return exportGroups.get(content);
  145:        }
  146:
  147:        protected DefaultMutableTreeNode getSelectedNode() {
  148:                TreePath path = getSelectionPath();
  149:                if (path == null) {
  150:                        return null;
  151:                }
  152:
  153:                return (DefaultMutableTreeNode) path.getLastPathComponent();
  154:        }
  155:
  156:        protected Brick getSelectedSchema() {
  157:                try{
  158:                        return (Brick)getSelectedNode().getUserObject();
  159:                } catch(Exception e) {
  160:                        return null;
  161:                }
  162:        }
  163:
  164:        protected ExportModel getSelectedExportModel() {
  165:                try{
  166:                        return (ExportModel)getSelectedNode().getUserObject();
  167:                } catch (Exception e) {
  168:                        return null;
  169:                }
  170:        }
  171:
  172:        protected String getSelectedStep() {
  173:                try{
  174:                        DefaultMutableTreeNode selectedNode = getSelectedNode();
  175:
  176:                        if(((DefaultMutableTreeNode)getModel().getRoot()).isNodeCh
ild(selectedNode)) {
  177:                                return selectedNode.toString();
  178:                        }
  179:
  180:                        return selectedNode.getParent().toString();
  181:                } catch(Exception e) {
  182:                        e.printStackTrace();
  183:                        return ""; //$NON-NLS-1$
  184:                }
  185:        }
  186:
  187:        protected int getSelectedStepIndex() {
  188:                String stepString = getSelectedStep();
  189:                for(int i = 0 ; i < session.datapack.getSteps().size() ; i++) {
  190:                        if(session.datapack.getSteps().elementAt(i).equalsIgnoreCa
se(stepString))
  191:                                return i;
  192:                }
  193:
  194:                return -1;
  195:        }
  196:
  197:        @Override
  198:        public Component getTreeCellRendererComponent(JTree tree, Object value,
  199:                        boolean selected, boolean expanded, boolean leaf, int row,
  200:                        boolean hasFocus) {
  201:                try{
  202:                        MyDefaultMutableTreeNode myNode = (MyDefaultMutableTreeNod
e)value;
  203:                        return myNode.getJComponent(selected, expanded, leaf, row,
 hasFocus);
  204:                } catch(Exception e) {
  205:                        DefaultMutableTreeNode myNode = (DefaultMutableTreeNode)va
lue;
  206:                        JLabel label = TreeActorsCellRendere.createDefaultLabel(my
Node.getUserObject().toString(), null, selected);
  207:                        if(treeModel.getRoot() == myNode.getParent().getParent() &
& myNode.getParent().getIndex(myNode) == 0 && leaf) {
  208:                                //label.setBackground(Color.lightGray);
  209:                        }
  210:
  211:                        return label;
  212:                }
  213:        }
  214:
  215:        @Override
  216:        public void mouseClicked(MouseEvent e) {
  217:                if (e.getButton() == MouseEvent.BUTTON3) {
  218:                        JTree navigationTree = (NavigationTree)e.getSource();
  219:
  220:                        TreePath nodePath = navigationTree.getPathForLocation(e.ge
tX(), e.getY());
  221:                        if(nodePath == null) {
  222:                                return;
  223:                        }
  224:
  225:                        DefaultMutableTreeNode selectedNode = (DefaultMutableTreeN
ode)nodePath.getLastPathComponent();
  226:                        if(selectedNode != null) {
  227:                                navigationTree.setSelectionPath(new TreePath(selec
tedNode.getPath()));
  228:                        }
  229:                }
  230:
  231:                Brick schema = getSelectedSchema();
  232:                if (schema != null) {
  233:                        if (e.getButton() == MouseEvent.BUTTON3) {
  234:                                ExportsContextualMenu menu = new ExportsContextual
Menu(schema, this);
  235:                                menu.show(e.getComponent(), e.getX(), e.getY());
  236:                        } else if (e.getClickCount() == 1) {
  237:                                if (getSelectedStep() != currentStep) {
  238:                                        BrickView<?,?> bV = brickViews.get(getSele
ctedStep());
  239:                                        if (bV == null) {
  240:                                                // TODO Faire communiquer le navig
ationTree avec la
  241:                                                // brickView pour gerer la sÃ©lect
```

```
ion rÃ©ciproque
  242:                                                TreeListener treeListener = new Tr
eeListener(this);
  243:                                                bV = new BrickView<BrickVertex, Br
ickEdge>(session
  244:                                                                .getBrickList().el
ementAt(
  245:                                                                ge
tSelectedStepIndex()), null,
  246:                                                                Pr
ogram.myMainFrame.getDataPack(), treeListener);
  247:                                                brickViews.put(getSelectedStep(),
bV);
  248:                                        }
  249:
  250:                                        schemaView.removeAll();
  251:                                        schemaView.add(bV);
  252:                                        schemaView.validate();
  253:                                        schemaView.repaint();
  254:                                        currentStep = getSelectedStep();
  255:                                        currentBrickView = bV;
  256:                                }
  257:                        }else if (e.getClickCount() > 1) {
  258:                                if (!schema.isNavigationBrick())
  259:                                        Program.myMainFrame.addTab(schema);
  260:                        }
  261:                } else if(e.getClickCount() > 1 && e.getButton() == MouseEvent.BUT
TON1 && getSelectedNode().getUserObject() instanceof ExportModel) {
  262:                        Program.myMainFrame.addTab(getSelectedNode().getUserObject
());
  263:                }
  264:        }
  265:
  266:        @Override
  267:        public void mouseEntered(MouseEvent e) {
  268:        }
  269:
  270:        @Override
  271:        public void mouseExited(MouseEvent e) {
  272:        }
  273:
  274:        @Override
  275:        public void mousePressed(MouseEvent e) {
  276:        }
  277:
  278:        @Override
  279:        public void mouseReleased(MouseEvent e) {
  280:        }
  281:
  282:        public void setSelectedContent(Content content) {
  283:                // TODO Auto-generated method stub
  284:        }
  285: }
```

```
 1: package client.export;
 2:
 3: import java.io.Serializable;
 4:
 5: public class Pair implements Serializable {
 6:
 7:         /**
 8:          *
 9:          */
10:         private static final long serialVersionUID = -3912777682364629562L;
11:         protected final Integer a;
12:         protected final Integer b;
13:
14:         public Pair(Integer a, Integer b) {
15:                 this.a = a;
16:                 this.b = b;
17:         }
18:
19:         public Pair(Pair other) {
20:                 a = other.a;
21:                 b = other.b;
22:         }
23:
24:         @Override
25:         public boolean equals(Object object) {
26:                 {
27:                         if (object instanceof Pair) {
28:                                 Pair other = (Pair) object;
29:
30:                                 return a.equals(other.a) && b.equals(other.b);
31:                         }
32:                 }
33:                 return false;
34:         }
35:
36:         public int hashCode()
37:         {
38:                 return 100000 * a.intValue() + b.intValue();
39:         }
40: }
```

```
    1: package client.export;
    2:
    3: import graphicalUserInterface.IconDatabase;
    4:
    5: import java.io.Serializable;
    6:
    7: import javax.swing.Icon;
    8:
    9: public class ExportImageData implements Serializable {
   10:         private static final long serialVersionUID = -8232182755856294608L;
   11:
   12:         private int imageIndex;
   13:         private String imagePath;
   14:
   15:         public ExportImageData() {
   16:                 imageIndex = -1;
   17:                 imagePath = null;
   18:         }
   19:
   20:         public ExportImageData(ExportImageData imageData) {
   21:                 imageIndex = imageData.imageIndex;
   22:                 imagePath = imageData.imagePath;
   23:         }
   24:
   25:         public Icon getIcon() {
   26:                 if(imagePath !=null) {
   27:                         return IconDatabase.getIcon(imagePath);
   28:                 }
   29:
   30:                 if(imageIndex >= 0) {
   31:                         return IconDatabase.vectorExportedIcons.elementAt(imageInd
ex);
   32:                 }
   33:
   34:                 return null;
   35:         }
   36:
   37:         public void setImageIndex(int index) {
   38:                 imageIndex = index;
   39:         }
   40:
   41:         public int getImageIndex() {
   42:                 return imageIndex;
   43:         }
   44:
   45:         public void setImagePath(String path) {
   46:                 imagePath = path;
   47:         }
   48:
   49:         public String getImagePath() {
   50:                 return imagePath;
   51:         }
   52:
   53:         public void clear() {
   54:                 imageIndex = -1;
   55:                 imagePath = null;
   56:         }
   57:
   58:         public void set(ExportImageData data) {
   59:                 imageIndex = data.imageIndex;
   60:                 imagePath = data.imagePath;
   61:         }
   62:
   63:         @Override
   64:         public boolean equals(Object obj) {
   65:                 if(obj != null && obj instanceof ExportImageData) {
   66:                         ExportImageData other = (ExportImageData)obj;
   67:                         return (imageIndex == other.imageIndex) && (imagePath == n
ull ? other.imagePath == null : imagePath.equals(other.imagePath));
   68:                 }
   69:
   70:                 return false;
   71:         }
   72:
   73:         @Override
   74:         public String toString()
   75:         {
   76:                 return "ImageData : path = "+imagePath+" & index = "+imageIndex;
   77:         }
   78: }
```

```
1: package client.export;
2:
3: public class ExportControler {
4:
5: }
```

```
    1: package client.export;
    2:
    3: import java.io.Serializable;
    4:
    5: /*
    6:  * enregistre les modification a effectuer sur un sommet pour un temps d’action do
nnÃ©e
    7:  */
    8:
    9: public class ExportTimeVertexData implements Serializable {
   10:
   11:         /**
   12:          *
   13:          */
   14:         private static final long serialVersionUID = -2513120406212602925L;
   15:
   16:         protected Integer actionTime;
   17:
   18:         /*
   19:          * info sur les changement
   20:          */
   21:
   22:         public ExportTimeVertexData(Integer actionTime) {
   23:                 this.actionTime = actionTime;
   24:         }
   25: }
```

```
1: package client.export;
2:
3: public interface DataSelectionListener {
4:
5:         void updateSelection(ExportDataInterface object);
6:
7: }
```

```
   1: package client.export;
   2:
   3: import java.awt.event.ActionEvent;
   4: import java.awt.event.ActionListener;
   5: import java.awt.event.MouseEvent;
   6: import java.awt.geom.Point2D;
   7:
   8: import javax.swing.JMenuItem;
   9: import javax.swing.JPopupMenu;
  10:
  11: import translation.Messages;
  12:
  13: import models.Brick;
  14: import models.TriadeEditingMousePlugin;
  15: import dataPack.Content;
  16: import dataPack.TreeListener;
  17: import edu.uci.ics.jung.algorithms.layout.GraphElementAccessor;
  18: import edu.uci.ics.jung.algorithms.layout.Layout;
  19: import edu.uci.ics.jung.visualization.VisualizationViewer;
  20: import graphicalUserInterface.DialogHandlerFrame;
  21:
  22: public class ExportingMousePlugin extends
  23:                 TriadeEditingMousePlugin<ExportVertexData, ExportEdgeData> {
  24:
  25:         // protected Model editedModel;
  26:         protected ExportModel currentExport;
  27:         protected ExportPopUp popUp;
  28:
  29:         public ExportingMousePlugin(TreeListener _treeListener,
  30:                         ExportModel _curentShema,
  31:                         ExportPopUp popUp, Layout<ExportVertexData, ExportEdgeData
> vertexPosition) {
  32:                 super(vertexPosition);
  33:                 this.popUp = popUp;
  34:                 currentExport = _curentShema;
  35:                 treeListener = _treeListener;
  36:                 treeListener.setGrapheListener(this);
  37:         }
  38:
  39:         @Override
  40:         public void selectEdge(ExportEdgeData newSelection) {
  41:                 if (newSelection != null)
  42:                         popUp.selectNewObject(newSelection);
  43:                 else
  44:                         popUp.unselectEdge();
  45:
  46:                 super.selectEdge(newSelection);
  47:         }
  48:
  49:         @Override
  50:         public void selectVertex(ExportVertexData vertex) {
  51:
  52:                 if (vertex != null)
  53:                         popUp.selectNewObject(vertex);
  54:                 else
  55:                         popUp.unselectVertex();
  56:                 super.selectVertex(vertex);
  57:         }
  58:
  59:         @Override
  60:         public void mousePressed(MouseEvent e) {
  61:                 if (checkModifiers(e)) {
  62:                         final VisualizationViewer<ExportVertexData, ExportEdgeData
> vv = (VisualizationViewer<ExportVertexData, ExportEdgeData>) e
  63:                                         .getSource();
  64:                         GraphElementAccessor<ExportVertexData, ExportEdgeData> pic
kSupport = vv
  65:                                         .getPickSupport();
  66:                         if (pickSupport != null) {
  67:                                 final ExportVertexData vertex = pickSupport.getVer
tex(vv
  68:                                                 .getGraphLayout(), e.getPoint().ge
tX(), e.getPoint()
  69:                                                 .getY());
  70:
  71:
  72:                                 ExportEdgeData edge = pickSupport.getEdge(vertexLo
cations, e.getPoint().getX(), e.getPoint().getY());
  73:
  74:                                 if (vertex == null && edge != null)
  75:                                 {
  76:                                         selectEdge(edge);
  77:
  78:                                 }
  79:                                 else
  80:                                 {
  81:                                         selectEdge(null);
  82:                                 }
  83:
  84:                                 if (vertex != null) { // get ready to make an edge
  85:                                         startVertex = vertex;
  86:                                         down = e.getPoint();
  87:                                         transformEdgeShape(down, down);
  88:                                         vv.addPostRenderPaintable(edgePaintable);
  89:                                         edgeIsDirected = true;
  90:                                         transformArrowShape(down, e.getPoint());
  91:                                         vv.addPostRenderPaintable(arrowPaintable);
  92:                                 }
  93:                                 else
  94:                                         selectVertex((ExportVertexData)null);
  95:                         }
  96:                 }
  97:         }
  98:
  99:         /**
 100:          * If startVertex is non-null, and the mouse is released over an existing
 101:          * vertex, create an undirected edge from startVertex to the vertex under
 102:          * the mouse pointer. If shift was also pressed, create a directed edge
 103:          * instead.
 104:          */
 105:         @Override
 106:         public void mouseReleased(MouseEvent e) {
 107:                 if (checkModifiers(e)) {
 108:                         final VisualizationViewer<ExportVertexData, ExportEdgeData
> vv = (VisualizationViewer<ExportVertexData, ExportEdgeData>) e
 109:                                         .getSource();
 110:                         final Point2D p = e.getPoint();
 111:                         GraphElementAccessor<ExportVertexData, ExportEdgeData> pic
kSupport = vv
 112:                                         .getPickSupport();
 113:                         if (pickSupport != null) {
 114:                                 final ExportVertexData vertex = pickSupport.getVer
tex(vv
 115:                                                 .getGraphLayout(), p.getX(),
 116:                                                 p.getY());
 117:                                 if (vertex != null && startVertex != null) {
 118:                                         if (vertex != startVertex) {
 119:                                                 if (vv.getGraphLayout().getGraph()
.isSuccessor(
 120:                                                                 startVertex, verte
x)) {
 121:
 122:                                                         ExportEdgeData edge = vv.g
```

```
etGraphLayout()
  123:                                                                    .getGraph(
).findEdge(startVertex, vertex);
  124:                                                            selectEdge(edge);
  125:
  126:                                                    } else {
  127:                                                            DialogHandlerFrame
  128:                                                                    .showError
Dialog(Messages.getString("ExportingMousePlugin.0")); //$NON-NLS-1$
  129:                                                    }
  130:                                            } else {
  131:                                                    if (vertex != selectedVertex) {
  132:                                                            selectVertex(vertex);
  133:                                                    }
  134:                                            }
  135:                                    } else if (vertex == null && startVertex == null)
  136:                                            selectVertex((ExportVertexData)null);
  137:                            }
  138:                            startVertex = null;
  139:                            down = null;
  140:                            edgeIsDirected = false;
  141:                            vv.removePostRenderPaintable(edgePaintable);
  142:                            vv.removePostRenderPaintable(arrowPaintable);
  143:                            vv.repaint();
  144:                    }
  145:                    else if (e.getButton() == MouseEvent.BUTTON3) {
  146:                            if (modelView instanceof ExportView) {
  147:                                    final ExportView view = (ExportView) modelView;
  148:                                    JPopupMenu menu = JpegGenerator.getGenerationMenu(
view);
  149:                                    menu.addSeparator();
  150:                                    JMenuItem previousStep = new JMenuItem(Messages.ge
tString("ExportingMousePlugin.1")); //$NON-NLS-1$
  151:                                    previousStep.addActionListener(new ActionListener(
) {
  152:
  153:                                            @Override
  154:                                            public void actionPerformed(ActionEvent e)
 {
  155:                                                    if (view.getCurrentApparitionStep(
) > 1)
  156:                                                            view.setCurrentApparitionS
tep(view.getCurrentApparitionStep() - 1);
  157:                                                    else
  158:                                                            DialogHandlerFrame.showErr
orDialog(Messages.getString("ExportingMousePlugin.2")); //$NON-NLS-1$
  159:                                            }
  160:                                    });
  161:
  162:                                    JMenuItem nextStep = new JMenuItem(Messages.getStr
ing("ExportingMousePlugin.3")); //$NON-NLS-1$
  163:                                    nextStep.addActionListener(new ActionListener() {
  164:
  165:                                            @Override
  166:                                            public void actionPerformed(ActionEvent e)
 {
  167:                                                    if (view.getCurrentApparitionStep(
) < currentExport.getApparitionStepCount())
  168:                                                            view.setCurrentApparitionS
tep(view.getCurrentApparitionStep() + 1);
  169:                                                    else
  170:                                                    DialogHandlerFrame.showErrorDialog
(Messages.getString("ExportingMousePlugin.4")); //$NON-NLS-1$
  171:                                            }
  172:
  173:                                    });
  174:                                    menu.add(previousStep);
  175:                                    menu.add(nextStep);
  176:
  177:                                    menu.show(view, e.getX(), e.getY());
  178:                            }
  179:                    }
  180:            }
  181:
  182:            @Override
  183:            public Brick getEditedAbstractSchema() {
  184:                    System.err
  185:                                    .println("!!! Abstract schema indisponible dans un
 shemaExportingMousePlugin !!!"); //$NON-NLS-1$
  186:                    return null;
  187:            }
  188:
  189:            @Override
  190:            public void removeSelectedVertex() {
  191:                    System.err
  192:                                    .println("!!! Impossible de supprimer un sommet de
puis un ShemaView !!!"); //$NON-NLS-1$
  193:            }
  194:
  195:            @Override
  196:            public void selectVertex(Content content) {
  197:                    selectVertex(currentExport.getExportVertexDataByContent(content));
  198:            }
  199:
  200:            @Override
  201:            public void notifyTree(Content content) {
  202:                    treeListener.setSelectedContent(content);
  203:
  204:
  205:            }
  206:
  207:            public void setExportView(ExportView theView)
  208:            {
  209:                    popUp.setView(theView);
  210:                    super.setModelView(theView);
  211:            }
  212:
  213: }
  214:
  215: // cacher/afficher
  216: //
  217:
```

```java
package client.export;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Vector;

import models.Brick;
import models.BrickEdge;
import models.BrickVertex;
import dataPack.Content;
import edu.uci.ics.jung.visualization.annotations.Annotation;

/*
 * cette classe enregiste tout les changement visuel effectuÃ© par l'utilisateur l
ors de l'exporte d'un schema.
 * Elle possÃ©de pour chaque element du graphe un dataExporte qui contient les cha
ngement
 */

public class ExportModel implements Serializable {

	/**
	 *
	 */
	private static final long serialVersionUID = 8579249503567790228L;

	protected Vector<ExportVertexData> vertexData; // regroupe les
	                                               // modifications
	                                               // Ã  effectuer
	                                               // sur les
	                                               // sommets
	protected Vector<ExportEdgeData> edgeData; // modification sur
	                                           // les arretes

	protected Vector<Annotation<String>> annotations;

	protected String name;
	protected Brick baseSchema;
	protected int apparitionStepCount;

	public ExportModel(String name, Brick schema) {
		this.name = name;
		this.baseSchema = schema;
		vertexData = new Vector<ExportVertexData>();
		edgeData = new Vector<ExportEdgeData>();
		apparitionStepCount = 1;

		HashMap<BrickVertex, ExportVertexData> oldNew = new HashMap<BrickV
ertex, ExportVertexData>();

		for (BrickVertex vertex : schema.getVertices()) {
			ExportVertexData newVertex = new ExportVertexData(vertex);
			oldNew.put(vertex, newVertex);
			vertexData.add(newVertex);
		}

		for(BrickEdge edge : schema.getEdges()) {
			edgeData.add(new ExportEdgeData(oldNew.get(edge.getSource(
)), oldNew.get(edge.getDestination()), edge.getCompleteRelation()));
		}
	}

	public ExportModel(ExportModel modelExport) {
		vertexData = new Vector<ExportVertexData>(
				modelExport.vertexData);
		edgeData = new Vector<ExportEdgeData>(modelExport.edgeData);

		this.name = modelExport.name;
		this.baseSchema = modelExport.baseSchema;
	}

	public ExportVertexData addVertexData(BrickVertex newVertex) {

		if (newVertex == null)
			return null;

		ExportVertexData data = new ExportVertexData(newVertex);
		vertexData.add(data);
		return data;
	}

	public ExportEdgeData addEdgeData(BrickEdge newEdge) {
		ExportEdgeData data = new ExportEdgeData(newEdge);
		edgeData.add(data);

		return data;
	}


// public boolean isVertexVisible(Integer actorId) {
	// //TODO Utile ?
	// if (actorId == null)
	// return false;
	//
	// ExportVertexData data = vertexData.get(actorId);
	//
	// if (data == null)
	// return true;
	// else
	// return data.isVisible();
	// }

	// public boolean isEdgeVisible(Integer srcId, Integer dstId) {
	// //TODO Utile ?
	// ExportEdgeData data = getExportEdgeData(srcId, dstId);
	//
	// if (data == null)
	// return true;
	// else
	// return data.isVisible();
	// }

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Brick getBaseSchema() {
		return baseSchema;
	}

	public ExportVertexData getExportVertexDataByContent(Content content) {
		for(ExportVertexData exportVertexData : vertexData) {
			Content currentContent = exportVertexData.getContent();
			if(content == currentContent) {
				return exportVertexData;
```

```
126:                         }
127:                 }
128:
129:                 return null;
130:         }
131:
132:         public Vector<ExportVertexData> getVertexData() {
133:                 return vertexData;
134:         }
135:
136:         public void setVertexData(Vector<ExportVertexData> vertexData) {
137:                 this.vertexData = vertexData;
138:         }
139:
140:         public Vector<ExportEdgeData> getEdgeData() {
141:                 return edgeData;
142:         }
143:
144:         public void setEdgeData(Vector<ExportEdgeData> edgeData) {
145:                 this.edgeData = edgeData;
146:         }
147:
148:         public int getApparitionStepCount() {
149:                 return apparitionStepCount;
150:         }
151:
152:         public void setApparitionStepCount(int apparitionStepCount) {
153:                 this.apparitionStepCount = apparitionStepCount;
154:         }
155:
156:         @Override
157:         public String toString() {
158:                 return name;
159:         }
160: }
```

```
package client.export;

import graphicalUserInterface.DialogHandlerFrame;
import graphicalUserInterface.PopUpView;

import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.util.ArrayList;
import java.util.Vector;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.DefaultComboBoxModel;
import javax.swing.Icon;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSpinner;
import javax.swing.JTextField;
import javax.swing.SpinnerNumberModel;
import javax.swing.event.CaretEvent;

import tools.ConfigTriades;
import translation.Messages;
import dataPack.Activite;

public class ExportPopUp extends PopUpView {

	/**
	 *
	 */
	private static final long serialVersionUID = -4896702758779246432L;

	protected JPanel edgePanel;
	protected JPanel vertexPanel;
	protected JPanel textPanel;
	protected JPanel forAllPanel;

	protected ExportModel model;
	protected ExportView view;

	protected ExportDataInterface currentData;

	protected ArrayList<DataSelectionListener> edgeListeners;
	protected ArrayList<DataSelectionListener> vertexListeners;
	protected ArrayList<DataSelectionListener> textListeners;
	protected ArrayList<DataSelectionListener> forAllListeners;

	protected JPanel contentPanel;
	protected CardLayout contentLayout;

	final protected JPanel mainPanel;


	protected final ExportPopUp access;


	public ExportPopUp(ExportModel model, JPanel mainPanel) {

		access = this;
		this.mainPanel = mainPanel;
		this.model = model;

		edgeListeners = new ArrayList<DataSelectionListener>();
		vertexListeners = new ArrayList<DataSelectionListener>();
		textListeners = new ArrayList<DataSelectionListener>();
		forAllListeners = new ArrayList<DataSelectionListener>();

		edgePanel = buildEdgePanel();
		vertexPanel = buildVertexPanel();
		textPanel = buildTextPanel();
		forAllPanel = buildForAllPanel();

		contentPanel = new JPanel();
		contentLayout = new CardLayout();
		contentPanel.setLayout(contentLayout);

		currentData = null;

		contentPanel.add(edgePanel, "edge"); //$NON-NLS-1$
		contentPanel.add(vertexPanel, "vertex"); //$NON-NLS-1$
		contentPanel.add(textPanel, "text"); //$NON-NLS-1$

		panelMax.add(contentPanel, BorderLayout.CENTER);
		panelMax.add(forAllPanel, BorderLayout.NORTH);
		JButton masquer = new JButton(Messages.getString("ExportPopUp.3"))
; //$NON-NLS-1$
		masquer.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				access.panelMax.setVisible(false);
				access.panelMin.setVisible(true);
				access.validate();
			}
		});

		panelMax.add(masquer, BorderLayout.SOUTH);
	}

	public void setView(ExportView theView)
	{
		view = theView;
	}

	public void selectNewObject(ExportDataInterface newSelection)
	{
		currentData = newSelection;


		if (newSelection != null) {
			for (DataSelectionListener allListener : forAllListeners)
{
				allListener.updateSelection(newSelection);
			}

			if (newSelection instanceof ExportEdgeData) {
				for (DataSelectionListener listener : edgeListener
s)
					listener.updateSelection(newSelection);
				contentLayout.show(contentPanel, "edge"); //$NON-N
LS-1$
			} else if (newSelection instanceof ExportVertexData) {
				for (DataSelectionListener listener : vertexListen
ers)
					listener.updateSelection(newSelection);
				contentLayout.show(contentPanel, "vertex"); //$NON
-NLS-1$
```

```
129:                        } else if (newSelection instanceof ExportTextData) {
130:                                for (DataSelectionListener listener : textListener
s)
131:                                        listener.updateSelection(newSelection);
132:                                contentLayout.show(contentPanel, "text"); //$NON-N
LS-1$
133:                        } else {
134:                                panelMax.setVisible(false);
135:                                return;
136:                        }
137:                        panelMax.setVisible(true);
138:                        panelMin.setVisible(false);
139:                        this.validate();
140:
141:                } else {
142:                        panelMax.setVisible(false);
143:                        panelMin.setVisible(false);
144:                }
145:
146:        }
147:
148:        public void unselectVertex()
149:        {
150:                if (currentData != null && currentData instanceof ExportVertexData
)
151:                        selectNewObject(null);
152:        }
153:
154:        public void unselectEdge()
155:        {
156:                if (currentData != null && currentData instanceof ExportEdgeData)
157:                        selectNewObject(null);
158:        }
159:
160:        abstract class ListenerLabel extends JLabel implements
161:        DataSelectionListener {
162:                private static final long serialVersionUID = -8048108358182093783L
;
163:        };
164:
165:        abstract class ListenerCheckBox extends JCheckBox implements
166:        DataSelectionListener {
167:                private static final long serialVersionUID = -8649477080670733830L
;
168:
169:                public ListenerCheckBox(String text) {
170:                        super(text);
171:                }
172:        };
173:
174:        abstract class ListenerTextField extends JTextField implements DataSelecti
onListener {
175:                private static final long serialVersionUID = 7905579799396427169L;
176:
177:        }
178:
179:        abstract class ListenerJSpinner extends JSpinner implements DataSelectionL
istener{
180:                private static final long serialVersionUID = -3180229401654302906L
;
181:
182:                public ListenerJSpinner()
183:                {
184:                        super(new SpinnerNumberModel(1, 1, 25, 1));
185:                }
186:        }
187:
188:        abstract class ListenerJCombotBox extends JComboBox implements DataSelecti
onListener {
189:                private static final long serialVersionUID = 8976617494377440512L;
190:        }
191:
192:        private JPanel buildEdgePanel() {
193:                JPanel result = new JPanel();
194:                result.setLayout(new BorderLayout());
195:
196:
197:                return result;
198:        }
199:
200:        private JPanel buildVertexPanel() {
201:                JPanel result = new JPanel();
202:                result.setLayout(new BoxLayout(result, BoxLayout.Y_AXIS));
203:
204:                final ListenerJSpinner spinner = new ListenerJSpinner() {
205:
206:                        private static final long serialVersionUID = -529891602022
228620L;
207:
208:                        @Override
209:                        public void updateSelection(ExportDataInterface object) {
210:                                if (object instanceof ExportVertexData)
211:                                {
212:
213:                                        ExportVertexData data = (ExportVertexData)
object;
214:                                        setValue(data.getApparitionStep());
215:
216:                                }
217:                        }
218:
219:                        @Override
220:                        protected void fireStateChanged()
221:                        {
222:                                if (access.currentData instanceof ExportVertexData
)
223:                                {
224:                                        ExportVertexData data = (ExportVertexData)
access.currentData;
225:                                        data.setApparitionStep(((Integer)getValue(
)).intValue());
226:
227:                                        if (((Integer)getValue()).intValue() > mod
el.getApparitionStepCount())
228:                                                model.setApparitionStepCount(((Int
eger)getValue()).intValue());
229:                                        mainPanel.repaint();
230:                                }
231:
232:                                super.fireStateChanged();
233:                        }
234:                };
235:                vertexListeners.add(spinner);
236:
237:                spinner.setBorder(BorderFactory.createTitledBorder(Messages.getStr
ing("ExportPopUp.7"))); //$NON-NLS-1$
238:                result.add(spinner,BorderLayout.CENTER);
239:
240:                final ListenerJCombotBox lastImages = new ListenerJCombotBox() {
241:                        private static final long serialVersionUID = -347625976821
4720516L;
242:
243:                        @Override
244:                        public void updateSelection(ExportDataInterface object) {
```

```
245:                                if(ConfigTriades.getInstance().getLastImages().get
LastObjects().size() <= 0) {
246:                                        setVisible(false);
247:                                        return;
248:                                }
249:
250:                                setVisible(true);
251:
252:                                Vector<Icon> images = new Vector<Icon>();
253:                                for (ExportImageData imageData : ConfigTriades.get
Instance().getLastImages().getLastObjects()) {
254:                                        if (imageData.getIcon() != null)
255:                                                images.add(imageData.getIcon());
256:                                }
257:
258:                                setModel(new DefaultComboBoxModel(images));
259:                                if (object instanceof ExportVertexData)
260:                                {
261:                                        ExportVertexData data = (ExportVertexData)
object;
262:
263:                                        setSelectedItem(data.getIcon());
264:                                }
265:
266:                                mainPanel.repaint();
267:
268:                        }
269:
270:                        @Override
271:                        protected void fireItemStateChanged(ItemEvent e)
272:                        {
273:                                if(e.getStateChange() == ItemEvent.DESELECTED || g
etSelectedItem() == null) {
274:                                        return;
275:                                }
276:
277:
278:                                int index = getSelectedIndex();
279:
280:                                ExportImageData imageData = ConfigTriades.getInsta
nce().getLastImages().getLastObjects().elementAt(index);
281:                                ((ExportVertexData)access.currentData).setImageDat
a(imageData);
282:                                ConfigTriades.getInstance().getLastImages().addLas
tObject(new ExportImageData(imageData));
283:
284:                                Vector<Icon> images = new Vector<Icon>();
285:                                for (ExportImageData imgData : ConfigTriades.getIn
stance().getLastImages().getLastObjects()) {
286:                                        if (imgData.getIcon() != null)
287:                                                images.add(imgData.getIcon());
288:                                }
289:                                setModel(new DefaultComboBoxModel(images));
290:
291:
292:                                repaint();
293:
294:                                mainPanel.repaint();
295:                                super.fireItemStateChanged(e);
296:
297:                        }
298:                };
299:
300:
301:                vertexListeners.add(lastImages);
302:
303:                JButton selectImageButton = new JButton(Messages.getString("Export
PopUp.8")); //$NON-NLS-1$
304:
305:                selectImageButton.addActionListener(new ActionListener() {
306:                        @Override
307:                        public void actionPerformed(ActionEvent e) {
308:                                ExportImagesView.getSingleton().display((ExportVer
texData)access.currentData, mainPanel, lastImages);
309:                        }
310:                });
311:
312:                JPanel imagePanel = new JPanel();
313:                imagePanel.setToolTipText("Selection de l'image associÃ©e au somme
t.");
314:                imagePanel.setLayout(new BoxLayout(imagePanel, BoxLayout.Y_AXIS));
315:                imagePanel.setBorder(BorderFactory.createTitledBorder(Messages.get
String("ExportPopUp.9"))); //$NON-NLS-1$
316:
317:                imagePanel.add(selectImageButton, BorderLayout.CENTER);
318:                imagePanel.add(lastImages, BorderLayout.CENTER);
319:
320:                result.add(imagePanel);
321:
322:                return result;
323:        }
324:
325:        private JPanel buildTextPanel() {
326:                JPanel result = new JPanel();
327:
328:                return result;
329:        }
330:
331:        private JPanel buildForAllPanel() {
332:                JPanel result = new JPanel();
333:                result.setLayout(new BoxLayout(result, BoxLayout.Y_AXIS));
334:
335:                ListenerCheckBox lcheckBox = new ListenerCheckBox(Messages.getStri
ng("ExportPopUp.10")) { //$NON-NLS-1$
336:                        private static final long serialVersionUID = 8305867850610
117203L;
337:
338:                        @Override
339:                        public void updateSelection(ExportDataInterface object) {
340:
341:                                if (access.currentData != null) {
342:                                        setSelected(access.currentData.getExportDa
ta().isVisible());
343:                                        if(object instanceof ExportVertexData && (
(ExportVertexData)object).getContent() instanceof Activite)
344:                                                setEnabled(false);
345:                                        else
346:                                                setEnabled(true);
347:                                } else
348:                                        setEnabled(false);
349:
350:                        }
351:
352:                        @Override
353:                        protected void fireItemStateChanged(ItemEvent e) {
354:                                if (access.currentData != null) {
355:                                        access.currentData.getExportData()
356:                                        .setVisible(e.getStateChange() == ItemEven
t.SELECTED);
357:                                        mainPanel.repaint();
358:                                }
359:                                super.fireItemStateChanged(e);
360:
361:                        }
```

```
362:                };
363:
364:                ListenerTextField nameField = new ListenerTextField() {
365:
366:                        private static final long serialVersionUID = 1L;
367:
368:
369:                        @Override
370:                        public void updateSelection(ExportDataInterface object) {
371:                                if (access.currentData != null
372:                                                && access.currentData.getExportDat
a().getText() != null) {
373:                                        setText(access.currentData.getExportData()
.getText());
374:                                } else {
375:                                        setText(""); //$NON-NLS-1$
376:                                }
377:                        }
378:
379:                        @Override
380:                        protected void fireCaretUpdate(CaretEvent e)
381:                        {
382:                                if (access.currentData != null) {
383:                                        access.currentData.getExportData().setText
(getText());
384:                                        mainPanel.repaint();
385:                                }
386:                                super.fireCaretUpdate(e);
387:                        }
388:
389:                };
390:
391:                nameField.setBorder(BorderFactory.createTitledBorder("Texte associ
Ã©"));
392:                nameField.setToolTipText("Texte affichÃ© Ã  cÃ´tÃ© du sommet. Si v
ous laissez ce champs vide le texte par dÃ©faut sera affichÃ©.");
393:                result.add(nameField);
394:                result.add(lcheckBox);
395:
396:                forAllListeners.add(nameField);
397:                forAllListeners.add(lcheckBox);
398:
399:                JPanel stepControl = new JPanel();
400:                stepControl.setBorder(BorderFactory.createTitledBorder(Messages.ge
tString("ExportPopUp.12"))); //$NON-NLS-1$
401:
402:
403:                stepControl.setLayout(new BoxLayout(stepControl, BoxLayout.X_AXIS)
);
404:
405:                JButton minus = new JButton(" - "); //$NON-NLS-1$
406:                JButton plus = new JButton(" + "); //$NON-NLS-1$
407:                final JTextField count = new JTextField(2);
408:                count.setText(model.getApparitionStepCount()+""); //$NON-NLS-1$
409:
410:                minus.addActionListener(new ActionListener() {
411:
412:                        @Override
413:                        public void actionPerformed(ActionEvent e) {
414:                                if (view.getCurrentApparitionStep() > 1)
415:                                        view.setCurrentApparitionStep(view.getCurr
entApparitionStep()-1);
416:                                else
417:                                        DialogHandlerFrame.showErrorDialog(Message
s.getString("ExportPopUp.16")); //$NON-NLS-1$
418:                                count.setText(view.getCurrentApparitionStep()+"");
 //$NON-NLS-1$
419:                                count.repaint();
420:                                view.repaint();
421:                        }
422:                });
423:
424:                plus.addActionListener(new ActionListener() {
425:
426:                        @Override
427:                        public void actionPerformed(ActionEvent e) {
428:                                if (view.getCurrentApparitionStep() < model.getApp
aritionStepCount())
429:                                        view.setCurrentApparitionStep(view.getCurr
entApparitionStep()+1);
430:                                else
431:                                        DialogHandlerFrame.showErrorDialog(Message
s.getString("ExportPopUp.18")); //$NON-NLS-1$
432:                                count.setText(view.getCurrentApparitionStep()+"");
 //$NON-NLS-1$
433:                                count.repaint();
434:                                view.repaint();
435:                        }
436:                });
437:
438:                stepControl.setToolTipText("ContrÃ´le de l'Ã©tape affichÃ©e. Cela
permet de vÃ©rifier les ordres d'apparitions des sommets");
439:                stepControl.add(Box.createGlue());
440:                stepControl.add(minus);
441:                stepControl.add(count);
442:                stepControl.add(plus);
443:                stepControl.add(Box.createGlue());
444:                result.add(stepControl);
445:
446:                return result;
447:        }
448:
449: }
450:
```

```
 1: package client.export;
 2:
 3: import java.util.Hashtable;
 4:
 5: import main.RelationComplete;
 6: import models.BrickEdge;
 7:
 8: /*
 9:  * contient les modification d'Ã©xportation d'une arrete
10:  */
11:
12: public class ExportEdgeData extends BrickEdge implements ExportDataInterface {
13:
14:
15:         /**
16:          *
17:          */
18:         private static final long serialVersionUID = -3158403916349233462L;
19:
20:         protected Hashtable<Integer, ExportTimeEdgeData> changeByActionTime;
21:         protected ExportData exportData;
22:
23:         public ExportEdgeData(BrickEdge baseEdge) {
24:                 super(baseEdge, true);
25:                 exportData = new ExportData();
26:         }
27:
28:
29:
30:         public ExportEdgeData(ExportVertexData source,
31:                         ExportVertexData destination,
32:                         RelationComplete completeRelation) {
33:                 super(source, destination, completeRelation, true);
34:                 exportData = new ExportData();
35:         }
36:
37:
38:
39:         public ExportTimeEdgeData getChangeForActionTime(Integer actionTime) {
40:                 return changeByActionTime.get(actionTime);
41:         }
42:
43:
44:         @Override
45:         public ExportData getExportData() {
46:
47:                 return exportData;
48:         }
49:
50: }
```

```
    1: package client.export;
    2:
    3: import graphicalUserInterface.Program;
    4:
    5: import java.awt.event.ActionEvent;
    6: import java.awt.event.ActionListener;
    7: import java.text.SimpleDateFormat;
    8: import java.util.Date;
    9: import java.util.Vector;
   10:
   11: import javax.swing.JMenuItem;
   12: import javax.swing.JOptionPane;
   13: import javax.swing.JPopupMenu;
   14: import javax.swing.JSeparator;
   15:
   16: import translation.Messages;
   17:
   18: import models.Brick;
   19: import client.NavigationTree;
   20: import client.Session;
   21:
   22: public class ExportsContextualMenu extends JPopupMenu{
   23:
   24:         private static final long serialVersionUID = 6895482137669411024L;
   25:         protected Brick baseSchema;
   26:         NavigationTree navigationTree;
   27:
   28:         public ExportsContextualMenu(Brick brick) {
   29:                 super(Messages.getString("ExportsContextualMenu.0")); //$NON-NLS-1
$
   30:                 baseSchema = brick;
   31:                 fillMenu();
   32:                 navigationTree = Program.myMainFrame.datapack.getCurrentSession().
getNavigationTree();
   33:         }
   34:
   35:         public ExportsContextualMenu(Brick schema, NavigationTree navigationTree)
{
   36:                 this(schema);
   37:
   38:                 this.navigationTree = navigationTree;
   39:         }
   40:
   41:         protected void fillMenu() {
   42:                 final Session session = Program.myMainFrame.getDataPack()
   43:                 .getCurrentSession();
   44:
   45:                 JMenuItem newExport = new JMenuItem(Messages.getString("ExportsCon
textualMenu.1")); //$NON-NLS-1$
   46:                 newExport.addActionListener(new ActionListener() {
   47:
   48:                         @Override
   49:                         public void actionPerformed(ActionEvent e) {
   50:                                 String name = JOptionPane.showInputDialog(Program.
myMainFrame,
   51:                                                 Messages.getString("ExportsContext
ualMenu.2"), //$NON-NLS-1$
   52:                                                 Messages.getString("ExportsContext
ualMenu.3") //$NON-NLS-1$
   53:                                                                 + baseSchema.getNa
me()
   54:                                                                 + " (" //$NON-NLS-
1$
   55:                                                                 + new SimpleDateFo
rmat("dd MMM yyyy kk'h'mm") //$NON-NLS-1$
   56:                                 .format(new Date()) + ")"); //$NON-NLS-1$
   57:                                 if(name != null && name.compareTo("") != 0) { //$N
ON-NLS-1$
   58:                                                 ExportModel newExport = new ExportModel(na
me, baseSchema);
   59:                                                 session.addExport(baseSchema, newExport);
   60:                                                 if(navigationTree != null)
   61:                                                         navigationTree.addExport(newExport
, baseSchema);
   62:                                                 else
   63:                                                         System.out.println("NavigtionTree
null dans ExportsContextualMenu, c'est surement pour ca que l'arbre ne ce met pas a jour
quand on creer une nouvelle image depuis ici."); //$NON-NLS-1$
   64:                                                 Program.myMainFrame.addTab(newExport);
   65:                                         }
   66:                                 }
   67:                         });
   68:
   69:                         add(newExport);
   70:                         add(new JSeparator());
   71:                         final JPopupMenu baseMenu = this;
   72:                         Vector<ExportModel> exportList = session.getExports(baseSchema);
   73:                         if (exportList != null)
   74:                                 for (final ExportModel export : exportList) {
   75:                                         JMenuItem oldExport = new JMenuItem(export.getName
());
   76:                                         oldExport.addActionListener(new ActionListener() {
   77:
   78:                                                 @Override
   79:                                                 public void actionPerformed(ActionEvent ar
g0) {
   80:                                                         Program.myMainFrame.addTab(export)
;
   81:
   82:                                                 }
   83:                                         });
   84:                                         baseMenu.add(oldExport);
   85:
   86:                                 }
   87:
   88:         }
   89:
   90:
   91: }
```

```
    1: package client.export;
    2:
    3: import graphicalUserInterface.Program;
    4:
    5: import java.awt.Color;
    6: import java.awt.Component;
    7: import java.awt.event.MouseEvent;
    8: import java.awt.event.MouseListener;
    9:
   10: import javax.swing.JLabel;
   11: import javax.swing.JTree;
   12: import javax.swing.tree.TreeCellRenderer;
   13: import javax.swing.tree.TreeModel;
   14: import javax.swing.tree.TreePath;
   15:
   16: import models.Brick;
   17: import models.BrickVertex;
   18: import models.TriadeEditingMousePlugin;
   19: import translation.Messages;
   20: import dataPack.Content;
   21: import dataPack.JTreeActors;
   22: import dataPack.MyDefaultMutableTreeNode;
   23:
   24: public class ExportTree extends JTreeActors implements TreeCellRenderer, MouseList
ener{
   25:
   26:         /**
   27:          *
   28:          */
   29:         private static final long serialVersionUID = -1870470942564510295L;
   30:
   31:         TreeModel treeModel;
   32:         ExportModel exportModel;
   33:         ExportPopUp popup;
   34:
   35:         public ExportTree() {
   36:                 //TODO a virer une fois ces arbre ne seront plus sauvÃ©
   37:                 super();
   38:                 setEditable(false);
   39:         }
   40:
   41:         public ExportTree(ExportModel exportModel, ExportPopUp popup, TriadeEditin
gMousePlugin triadeEditingMousePlugin) {
   42:                 super(Program.myMainFrame.datapack);
   43:
   44:                 this.popup = popup;
   45:                 this.exportModel = exportModel;
   46:
   47:                 setTriadeEditingMousePlugin(triadeEditingMousePlugin);
   48:
   49:                 addMouseListener(this);
   50:                 setCellRenderer(this);
   51:                 setEditable(false);
   52:
   53:                 /* cache les sommets qui ne sont pas dans le schema Ã  exporter */
   54:                 simpleHideAll();
   55: //              DataPack dataPack = Program.myMainFrame.datapack;
   56:                 brick = exportModel.getBaseSchema();
   57:                 for(BrickVertex vertex : brick.getVertices()) {
   58:                         if(vertex.getContent() instanceof Brick) {
   59:                                 //TODO gerer les graphe de navigation qui ont des
schema
   60:                         }
   61:                         else {
   62:                                 Content content = vertex.getContent();
   63:                                 if(content instanceof MyDefaultMutableTreeNode) {
   64:                                         MyDefaultMutableTreeNode actor = (MyDefaul
tMutableTreeNode)content;
   65:                                         simpleShowActor(actor);
   66:                                 } else {
   67:                                         //addContent(content);
   68:                                         //TODO ici pour ajouter les activitÃ© dans
 l'arbre des export
   69:                                 }
   70: //                              saveVertex(actor);
   71: //                              saveAssociatedActor(vertex, actor);
   72:                         }
   73:                 }
   74:
   75:                 expendAllNode();
   76:         }
   77:
   78:         @Override
   79:         protected MyDefaultMutableTreeNode getSelectedNode() {
   80:                 TreePath path = getSelectionPath();
   81:                 if (path == null) {
   82:                         return null;
   83:                 }
   84:
   85:                 return (MyDefaultMutableTreeNode) path.getLastPathComponent();
   86:         }
   87:
   88:         @Override
   89:         public Component getTreeCellRendererComponent(JTree tree, Object value,
   90:                         boolean selected, boolean expanded, boolean leaf, int row,
   91:                         boolean hasFocus) {
   92:                 MyDefaultMutableTreeNode myNode = (MyDefaultMutableTreeNode) value
;
   93:                 JLabel label = myNode.getJComponent(selected, expanded, leaf, row,
 hasFocus);
   94:
   95:                 if(MyDefaultMutableTreeNode.isGroupe(myNode) == false) {
   96:                         ExportVertexData vertexData = exportModel.getExportVertexD
ataByContent(myNode);
   97:                         if(vertexData != null && vertexData.getExportData().isVisi
ble() == false) {
   98:                                 if(selected == false) {
   99:                                         label.setBackground(Color.lightGray);
  100:                                 }
  101:                                 label.setText(label.getText() + Messages.getString
("ExportTree.0")); //$NON-NLS-1$
  102:                         } else {
  103:                                 String labelText = label.getText();
  104:                                 //TODO trouver comment afficher correctement les l
ables sans devoir utiliser les espaces
  105:                                 label.setText(labelText + "
   "); //$NON-NLS-1$
  106:                         }
  107:                 }
  108:
  109:                 return label;
  110:         }
  111:
  112:         @Override
  113:         public void mouseClicked(MouseEvent e) {
  114:                 MyDefaultMutableTreeNode selectedNode = getSelectedNode();
  115:                 if(selectedNode != null && MyDefaultMutableTreeNode.isGroupe(selec
tedNode) == false) {
  116:                         ExportVertexData vertexData = exportModel.getExportVertexD
ataByContent(selectedNode);
  117:                         //TODO ajouter le schema view (ou le listener) et l'inform
er qu'un nouveau vertex est selectonnÃ©
  118:                         setVertexSelected(selectedNode);
  119:                         if(vertexData != null) {
```

```
120:                                popup.selectNewObject(vertexData);
121:                        } else {
122:                                popup.selectNewObject(null);
123:                        }
124:                } else {
125:                        setVertexSelected(null);
126:                        popup.selectNewObject(null);
127:                }
128:        }
129:
130:
131:        @Override
132:        public void mouseEntered(MouseEvent e) {
133:        }
134:
135:        @Override
136:        public void mouseExited(MouseEvent e) {
137:        }
138:
139:        @Override
140:        public void mousePressed(MouseEvent e) {
141:        }
142:
143:        @Override
144:        public void mouseReleased(MouseEvent e) {
145:        }
146: }
```

```
    1: package client.export;
    2:
    3: import graphicalUserInterface.Program;
    4:
    5: import java.awt.Color;
    6: import java.awt.Font;
    7: import java.awt.Paint;
    8:
    9: import javax.swing.Icon;
   10:
   11: import models.BrickView;
   12:
   13: import org.apache.commons.collections15.Predicate;
   14:
   15:
   16: import dataPack.TreeListener;
   17: import edu.uci.ics.jung.algorithms.layout.KKLayout;
   18: import edu.uci.ics.jung.algorithms.layout.Layout;
   19: import edu.uci.ics.jung.algorithms.layout.StaticLayout;
   20: import edu.uci.ics.jung.graph.Graph;
   21: import edu.uci.ics.jung.graph.util.Context;
   22: import edu.uci.ics.jung.visualization.VisualizationViewer;
   23:
   24: public class ExportView extends BrickView<ExportVertexData, ExportEdgeData> {
   25:         private static final long serialVersionUID = -765230156911470630 2L;
   26:
   27:         protected ExportModel exportModel;
   28:         protected int currentApparitionStep;
   29:
   30:         public ExportView(ExportModel exportModel, TreeListener treeListener,
   31:                         ExportPopUp pUV) {
   32:                 super(pUV, Program.myMainFrame.datapack, treeListener);
   33:
   34:                 brick = exportModel.getBaseSchema();
   35:                 currentApparitionStep = exportModel.apparitionStepCount;
   36:
   37:                 KKLayout<ExportVertexData, ExportEdgeData> kkLayout = null;
   38:                 if(exportModel.getBaseSchema().isNavigationBrick()) {
   39:                         kkLayout = new KKLayout<ExportVertexData, ExportEdgeData>(
graph);
   40:                         kkLayout.setLengthFactor(1);
   41:                         vertexLocations = kkLayout;
   42:                 } else {
   43:                         vertexLocations = new StaticLayout<ExportVertexData, Expor
tEdgeData>(graph);
   44:                 }
   45:
   46:                 mousePlugin = new ExportingMousePlugin(treeListener, exportModel,
pUV, vertexLocations);
   47:                 buildComponent();
   48:
   49:                 buildGraph(exportModel);
   50:
   51:                 if(kkLayout != null) {
   52:                         kkLayout.initialize();
   53:                         kkLayout.adjustForGravity();
   54:                 }
   55:
   56:                 this.exportModel = exportModel;
   57:         }
   58:
   59:         public VisualizationViewer<ExportVertexData, ExportEdgeData> getVisualizat
ionViewer() {
   60:                 return vv;
   61:         }
   62:
   63:         @Override
   64:         protected void buildComponent() {
   65:                 super.buildComponent();
   66:                 final ExportView access = this;
   67:                 //ajouter les predicate
   68:                 pr.setVertexIncludePredicate(new Predicate<Context<Graph<ExportVer
texData,ExportEdgeData>,ExportVertexData>>() {
   69:
   70:                         @Override
   71:                         public boolean evaluate(Context<Graph<ExportVertexData, Ex
portEdgeData>, ExportVertexData> arg0) {
   72:                                 return access.getVertexIncludePredicate(ar
g0);
   73:                         }
   74:                 });
   75:
   76:                 pr.setEdgeIncludePredicate(new Predicate<Context<Graph<ExportVerte
xData,ExportEdgeData>,ExportEdgeData>>() {
   77:
   78:                         @Override
   79:                         public boolean evaluate(
   80:                                         Context<Graph<ExportVertexData, ExportEdge
Data>, ExportEdgeData> arg0) {
   81:                                 return access.getEdgeIncludePredicate(arg0);
   82:
   83:                         }
   84:                 });
   85:         }
   86:
   87:         protected void buildGraph(ExportModel exportModel) {
   88:
   89:                 Layout<ExportVertexData, ExportEdgeData> layout = vv.getGraphLayou
t();
   90:                 boolean restartModel = false;
   91:
   92:                 /* ajoute les donneÃ©s d'export des sommets */
   93:                 for (ExportVertexData vertexData : exportModel.getVertexData()) {
   94:                         graph.addVertex(vertexData);
   95:                         vertexData.setSelected(false);
   96:
   97:                                 restartModel = true;
   98:
   99:                         if(exportModel.getBaseSchema().isNavigationBrick() == fals
e) {
  100:                                 layout.lock(vertexData, true);
  101:
  102:                                 /* place les sommets correctement suivant les vale
urs de l'export */
  103:
  104:                                 vertexLocations.setLocation(vertexData, vertexData
.getLocation());
  105:
  106:                                 layout.lock(vertexData, false);
  107:                         }
  108:                 }
  109:
  110:                 if(restartModel) {
  111:                         vv.repaint();
  112:                 }
  113:
  114:                 /* ajoute les donnÃ©es d'export des arÃªte */
  115:                 for (ExportEdgeData edgeData : exportModel.getEdgeData()) {
  116:                         edgeData.setSelected(false);
  117:                         graph.addEdge(edgeData, (ExportVertexData) edgeData.getSou
rce(),
  118:                                         (ExportVertexData) edgeData.getDestination
());
  119:
```

```
  120:                }
  121:        }
  122:
  123:        /******************/
  124:        /** Vertex export **/
  125:        /******************/
  126:        /* Icon du sommet */
  127:        @Override
  128:        public Icon getVertexIcon(ExportVertexData vertexData) {
  129:                Icon icon = null;
  130:                if(vertexData == null || (icon = vertexData.getIcon()) == null) {
  131:                        return super.getVertexIcon(vertexData);
  132:                }
  133:
  134:                return icon;
  135:        }
  136:
  137:        /* Texte du sommet */
  138:        @Override
  139:        public String getVertexLabel(ExportVertexData vertexData) {
  140:                String label;
  141:                if(vertexData != null && (label = vertexData.getExportData().getTe
xt()) !=null && label.compareTo("") != 0) //$NON-NLS-1$
  142:                        return label;
  143:                else
  144:                        return super.getVertexLabel(vertexData);
  145:        }
  146:
  147:        @Override
  148:        public String getEdgeLabel(ExportEdgeData edgeData) {
  149:                String text = edgeData.getExportData().getText();
  150:                if(text != null && !text.isEmpty()) {
  151:                        return text;
  152:                }
  153:
  154:                return super.getEdgeLabel(edgeData);
  155:        }
  156:
  157:        @Override
  158:        public Font getVertexFont(ExportVertexData vertexData) {
  159:                return super.getVertexFont(vertexData);//new Font("Helvetica", Fon
t.BOLD, 23);
  160:        }
  161:
  162:
  163:        public boolean getVertexIncludePredicate(Context<Graph<ExportVertexData, E
xportEdgeData>, ExportVertexData> arg0) {
  164:                return arg0.element.getApparitionStep() <= currentApparitionStep &
& arg0.element.getExportData().isVisible() && super.getVertexIncludePredicate(arg0);
  165:        }
  166:
  167:        /*****************/
  168:        /** edge export **/
  169:        /*****************/
  170:        /* Couleur de l'arÃªte */
  171:        @Override
  172:        public Paint getEdgeDrawPaint(ExportEdgeData edgeData) {
  173:                Color color;
  174:                if(edgeData != null && (color = edgeData.getExportData().getColor(
)) !=null)
  175:                        return color;
  176:                else
  177:                        return super.getEdgeDrawPaint(edgeData);
  178:        }
  179:
  180:        public boolean getEdgeIncludePredicate(Context<Graph<ExportVertexData, Exp
ortEdgeData>, ExportEdgeData> arg0)
  181:        {
  182:                return arg0.element.getExportData().isVisible() && super.getEdgeIn
cludePredicate(arg0);
  183:        }
  184:
  185:        /* Type de flÃ©che Ã  afficher */
  186:        //TODO Ã  coder en trouvant oÃ¹ le modifier et si on le fait
  187:
  188:
  189:        public ExportModel getExportModel() {
  190:                return exportModel;
  191:        }
  192:
  193:        public int getCurrentApparitionStep() {
  194:                return currentApparitionStep;
  195:        }
  196:
  197:        public void setCurrentApparitionStep(int currentApparitionStep) {
  198:                this.currentApparitionStep = currentApparitionStep;
  199:        }
  200: }
```

```
    1: package client.export;
    2:
    3: import graphicalUserInterface.DialogHandlerFrame;
    4: import graphicalUserInterface.Program;
    5:
    6: import java.awt.event.ActionEvent;
    7: import java.awt.event.ActionListener;
    8: import java.awt.image.BufferedImage;
    9: import java.io.File;
   10: import java.io.IOException;
   11: import java.util.Vector;
   12:
   13: import javax.imageio.ImageIO;
   14: import javax.swing.JFileChooser;
   15: import javax.swing.JMenuItem;
   16: import javax.swing.JOptionPane;
   17: import javax.swing.JPopupMenu;
   18: import javax.swing.filechooser.FileNameExtensionFilter;
   19:
   20: import translation.Messages;
   21: import client.Session;
   22: import dataPack.TreeListener;
   23: import edu.uci.ics.jung.visualization.VisualizationViewer;
   24:
   25: public class JpegGenerator {
   26:
   27:         private static JpegGenerator singleton;
   28:         private String path;
   29:
   30:         protected JpegGenerator() {
   31:
   32:         }
   33:
   34:         static public JpegGenerator getSingleton() {
   35:                 if (singleton == null) {
   36:                         singleton = new JpegGenerator();
   37:                 }
   38:                 return singleton;
   39:         }
   40:
   41:         public void exportSession(Session session) {
   42:                 JFileChooser chooser = new JFileChooser(path);
   43:                 chooser.setDialogTitle(Messages.getString("JpegGenerator.0")); //$
NON-NLS-1$
   44:                 chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
   45:                 int returnValue = chooser.showSaveDialog(Program.myMainFrame);
   46:                 if (returnValue == JFileChooser.APPROVE_OPTION)
   47:                 {
   48:                         String folder = chooser.getSelectedFile().getPath();
   49:                         for (ExportModel export : session.getExports())
   50:                         {
   51:
   52:                                 File file = new File(folder+export.getName()+".png
"); //$NON-NLS-1$
   53:                                 //TODO vÃ©rifier l'existence d'un fichier et deman
der si on remplace ou on renomme
   54:
   55:                                 generateJpeg(new ExportView(export, new TreeListen
er(), null),
   56:                                                 file);
   57:
   58:                         }
   59:                 }
   60:
   61:         }
   62:
   63:         public void generateJpeg(ExportView view) {
   64:                 generateJpeg(view, null);
   65:         }
   66:
   67:         public void generateJpeg(ExportView view, File file)
   68:         {
   69:                 String extension = "png"; //$NON-NLS-1$
   70:                 String unumberedFileName = ""; //$NON-NLS-1$
   71:                 if (file == null)
   72:                 {
   73:                         boolean isAllowedToSave = false;
   74:                         String nomFichier;
   75:                         JFileChooser chooser = new JFileChooser(path);
   76:                         FileNameExtensionFilter filter = new FileNameExtensionFilt
er(
   77:                                                         Messages.getString("JpegGenerator.4"), ext
ension, "jpg", "bmp"); //$NON-NLS-1$ //$NON-NLS-2$ //$NON-NLS-3$
   78:                         chooser.setFileFilter(filter);
   79:                         while (!isAllowedToSave) {
   80:                                 int returnVal = chooser.showSaveDialog(view);
   81:                                 if (returnVal == JFileChooser.APPROVE_OPTION) {
   82:                                         file = chooser.getSelectedFile();
   83:                                         nomFichier = file.getPath();
   84:
   85:                                         int pointIndex = nomFichier.lastIndexOf('.
');
   86:                                         if(pointIndex >= 0) {
   87:                                                 extension = nomFichier.substring(p
ointIndex+1, nomFichier.length());
   88:                                                 nomFichier = nomFichier.substring(
0, pointIndex);
   89:                                         }
   90:
   91:                                         if (view.getExportModel().getApparitionSte
pCount() > 1)
   92:                                         {
   93:
   94:                                                 nomFichier += Messages.getString("
JpegGenerator.7"); //$NON-NLS-1$
   95:                                                 unumberedFileName = nomFichier;
   96:                                                 nomFichier += "1."+extension; //$N
ON-NLS-1$
   97:                                         }
   98:                                         else
   99:                                         {
  100:                                                 nomFichier+= "."+extension;
  101:                                                 System.out.println("Nom du fichier
 : "+nomFichier);
  102:                                         }
  103:                                         file = new File(nomFichier);
  104:
  105:                                         if (file.exists()) {
  106:                                                 int returnV = DialogHandlerFrame
  107:                                                 .showYesNoDialog(
  108:                                                                 chooser,
  109:                                                                 Messages.getString
("JpegGenerator.9") //$NON-NLS-1$
  110:                                                                 + file.getName()
  111:                                                                 + Messages.getStri
ng("JpegGenerator.10")); //$NON-NLS-1$
  112:                                                 if (returnV == JOptionPane.YES_OPT
ION) {
  113:                                                         isAllowedToSave = true;
  114:                                                 }
  115:                                         } else {
  116:                                                 isAllowedToSave = true;
  117:                                         }
  118:                                 } else if (returnVal == JFileChooser.CANCEL_OPTION
```

```
)
                                        return;
                                }
                        }
                        else if(view.getExportModel().getApparitionStepCount() > 1)
                        {
                                int pointIndex = file.getPath().lastIndexOf('.');
                                unumberedFileName = file.getPath().substring(0, pointIndex
);
                                unumberedFileName += Messages.getString("JpegGenerator.11"
); //$NON-NLS-1$
                                extension = file.getPath().substring(pointIndex+1);

                        }
                        path = file.getParent();
                        if (view.getExportModel().getApparitionStepCount() > 1)
                        {
                                Vector<Integer> usefullSteps = scanForApparitionStep(view.
getExportModel());
                                int j = 0;
                                for (Integer i : usefullSteps)
                                {
                                        //GÃ©nÃ©rer tous les exports qui vont bien

                                        VisualizationViewer vV = view.getVisualizationView
er();
                                        view.setCurrentApparitionStep(i.intValue());
                                        file = new File(unumberedFileName+ (++j) +"."+exte
nsion); //$NON-NLS-1$

                                        BufferedImage bI = new BufferedImage(vV.getWidth()
, vV.getHeight(),
                                                        BufferedImage.TYPE_INT_RGB);
                                        // capture: create a BufferedImage
                                        // create the Graphics2D object that paints to it
                                        vV.paint( bI.getGraphics() );
                                        // and save out the BufferedImage
                                        try {
                                                ImageIO.write(bI, extension, file);
                                                System.out.println(Messages.getString("Jpe
gGenerator.13") + file + " - " + extension); //$NON-NLS-1$ //$NON-NLS-2$
                                        } catch (IOException e) {
                                                DialogHandlerFrame
                                                .showErrorDialog(Messages.getString("JpegG
enerator.15")); //$NON-NLS-1$
                                                e.printStackTrace();
                                        }
                                }
                        }
                        else
                        {
                                VisualizationViewer vV = view.getVisualizationViewer();

                                BufferedImage bI = new BufferedImage(vV.getWidth(), vV.get
Height(),
                                                BufferedImage.TYPE_INT_RGB);
                                // capture: create a BufferedImage
                                // create the Graphics2D object that paints to it
                                vV.paint( bI.getGraphics() );
                                // and save out the BufferedImage
                                try {
                                        ImageIO.write(bI, extension, file);
                                        System.out.println(Messages.getString("JpegGenerat
or.16") + file + " - " + extension); //$NON-NLS-1$ //$NON-NLS-2$
                                } catch (IOException e) {
                                        DialogHandlerFrame
                                        .showErrorDialog(Messages.getString("JpegGenerator
.18")); //$NON-NLS-1$
                                        e.printStackTrace();
                                }
                        }

                }

        }

        public static JPopupMenu getGenerationMenu(final ExportView view) {
                JPopupMenu result = new JPopupMenu();
                JMenuItem item = new JMenuItem(Messages.getString("JpegGenerator.1
9")); //$NON-NLS-1$
                item.addActionListener(new ActionListener() {

                        @Override
                        public void actionPerformed(ActionEvent e) {
                                getSingleton().generateJpeg(view);
                        }
                });
                result.add(item);
                return result;
        }

        protected Vector<Integer> scanForApparitionStep(ExportModel model)
        {
                Vector<Integer> result = new Vector<Integer>();
                for (int i = 1; i <= model.apparitionStepCount; i++)
                {
                        boolean usefullStep = false;
                        for (ExportVertexData data : model.getVertexData())
                        {
                                if (data.getApparitionStep() == i)
                                        usefullStep = true;
                        }
                        if (usefullStep)
                                result.add(i);

                }
                return result;

        }


}
```

```
 1: package client.export;
 2:
 3: import java.awt.Color;
 4: import java.io.Serializable;
 5:
 6: public class ExportData implements Serializable {
 7:
 8:         /**
 9:          *
10:          */
11:         private static final long serialVersionUID = -5832206653696002697L;
12:         boolean visible;
13:         protected String text; // texte Ã  afficher
14:         protected Color color; // nouvelle couleur de l'arrete
15:
16:         public ExportData() {
17:                 visible = true;
18:         }
19:
20:         public boolean isVisible() {
21:                 return visible;
22:         }
23:
24:         public void setVisible(boolean visible) {
25:                 this.visible = visible;
26:         }
27:
28:         public void setText(String text) {
29:                 this.text = text;
30:         }
31:
32:         public String getText() {
33:                 return text;
34:         }
35:
36:
37:         public Color getColor() {
38:                 return color;
39:         }
40:
41:         public void setColor(Color color) {
42:                 this.color = color;
43:         }
44: }
```

```
    1: package client.export;
    2:
    3: import java.io.Serializable;
    4:
    5: /*
    6:  * enregistre les modification a effectuer sur une arrete pour un temps d’action d
onnÃ©e
    7:  */
    8:
    9: public class ExportTimeEdgeData implements Serializable {
   10:
   11:         /**
   12:          *
   13:          */
   14:         private static final long serialVersionUID = 2157472866785528461L;
   15:
   16:         Integer actionTime; // id du temps de l’action
   17:
   18:         /*
   19:          * ajouter les variable pour effectuer les modif
   20:          */
   21:
   22:         public ExportTimeEdgeData(Integer actionTime) {
   23:                 this.actionTime = actionTime;
   24:         }
   25:
   26:         public Integer getActionTime() {
   27:                 return actionTime;
   28:         }
   29:
   30: }
```

```java
package client.export;

import graphicalUserInterface.Program;

import java.util.Hashtable;

import javax.swing.Icon;

import models.BrickVertex;
import dataPack.Content;

/*
 * contient les modification a effectuer sur un sommet
 */

public class ExportVertexData extends BrickVertex implements
                ExportDataInterface {

        /**
         *
         */
        private static final long serialVersionUID = 6976012649418320957L;

        protected Hashtable<Integer, ExportTimeVertexData> changeByActionTime;

        protected ExportData exportData;
        protected int apparitionStep;
        protected ExportImageData image;
        protected Content content;

        public ExportVertexData(BrickVertex baseVertex) {
                super(baseVertex);
                // TODO quel constructeur appeler pour BrickVertex depuis
                // ExportVertexData?
                exportData = new ExportData();
                exportData.visible = true;
                exportData.text = null;
                apparitionStep = 1;
                image = new ExportImageData();
                content = baseVertex.getContent();
        }

        public Icon getIcon() {
                Icon icon = null;
                if (image != null)
                        icon = image.getIcon();

                if(icon != null) {
                        return icon;
                }

                return Program.myMainFrame.getDataPack().getCurrentSession().getDe
faultImage(content);
        }

        public ExportTimeVertexData getChangeByActionTime(Integer actionTime) {
                return changeByActionTime.get(actionTime);
        }

        @Override
        public ExportData getExportData() {
                return exportData;
        }

        @Override
        public boolean equals(Object other) {
                if(other instanceof BrickVertex) {
                        return ((BrickVertex)other).getContent().equals(content);
                }

                return false;
        }

        public void setApparitionStep(int newStep)
        {
                apparitionStep = newStep;
        }

        public int getApparitionStep()
        {
                return apparitionStep;
        }

        public void setImageData(ExportImageData data) {
                image.set(data);
        }
}
```

```
    1: package client.export;
    2:
    3: import graphicalUserInterface.DialogHandlerFrame;
    4: import graphicalUserInterface.IconDatabase;
    5: import graphicalUserInterface.Program;
    6:
    7: import java.awt.Color;
    8: import java.awt.Component;
    9: import java.awt.Dimension;
   10: import java.awt.GridLayout;
   11: import java.awt.event.ActionEvent;
   12: import java.awt.event.ActionListener;
   13: import java.awt.event.ItemEvent;
   14: import java.awt.event.ItemListener;
   15: import java.awt.event.MouseEvent;
   16: import java.awt.event.MouseListener;
   17: import java.io.File;
   18: import java.util.Vector;
   19:
   20: import javax.swing.BorderFactory;
   21: import javax.swing.BoxLayout;
   22: import javax.swing.ImageIcon;
   23: import javax.swing.JButton;
   24: import javax.swing.JCheckBox;
   25: import javax.swing.JComponent;
   26: import javax.swing.JFrame;
   27: import javax.swing.JLabel;
   28: import javax.swing.JPanel;
   29: import javax.swing.JScrollPane;
   30: import javax.swing.border.Border;
   31:
   32: import tools.ConfigTriades;
   33: import translation.Messages;
   34:
   35: public class ExportImagesView extends JFrame {
   36:         static private final long serialVersionUID = -7353684973355867933L;
   37:
   38:         private static int borderSize = 3;
   39:
   40:         static private final Dimension windowSize = new Dimension(800, 600);
   41:
   42:         static private Border EmptyBorder = BorderFactory.createEmptyBorder(
   43:                         borderSize, borderSize, borderSize, borderSize);
   44:         static private Border SelectedBorder = BorderFactory.createLineBorder(
   45:                         Color.GREEN, borderSize);
   46:         static private Border OverlapingBorder = BorderFactory.createLineBorder(
   47:                         Color.BLUE, borderSize);
   48:
   49:         static public final String imagesDirectoryPath = System.getProperty("user.
home") + File.separatorChar + "Triade" + File.separatorChar + "pics"; //$NON-NLS-1$
   50:         private boolean init;
   51:
   52:         private JLabel selectedImage;
   53:         private final ExportImageData exportImageData;
   54:
   55:         private ExportVertexData exportedVertex;
   56:
   57:         private JComponent mainView;
   58:
   59:         private final ExportImagesView access = this;
   60:
   61:         private boolean isGlobalImage;
   62:         private JCheckBox globalCheckBox;
   63:
   64:         DataSelectionListener dataSelectionListener;
   65:
   66:         static private ExportImagesView singleton = null;
   67:
   68:         private ExportImagesView() {
   69:                 super();
   70:
   71:                 exportImageData = new ExportImageData();
   72:                 dataSelectionListener = null;
   73:                 this.exportedVertex = null;
   74:
   75:                 setSize(windowSize);
   76:                 setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
   77:
   78:                 init();
   79:
   80:                 setResizable(false);
   81:                 setVisible(true);
   82:                 validate();
   83:         }
   84:
   85:         private void init() {
   86:                 if (!init) {
   87:                         File imagesDirectoryFile = new File(imagesDirectoryPath);
   88:
   89:                         if (imagesDirectoryFile.exists()) {
   90:                                 if (imagesDirectoryFile.isDirectory() == false) {
   91:                                         DialogHandlerFrame
   92:                                                         .showInformationDialog(
   93:                                                                         Program.my
MainFrame,
   94:                                                                         Messages.g
etString("ExportImagesView.1"), //$NON-NLS-1$
   95:                                                                         Messages.g
etString("ExportImagesView.2") //$NON-NLS-1$
   96:
+ imagesDirectoryPath
   97:
+ Messages.getString("ExportImagesView.3"), //$NON-NLS-1$
   98:                                                                         null);
   99:                                         init = false;
  100:                                         setVisible(false);
  101:                                         return;
  102:                                 }
  103:                         } else {
  104:                                 imagesDirectoryFile.mkdir();
  105:                         }
  106:
  107:                         add(createView(imagesDirectoryFile));
  108:                         validate();
  109:
  110:                         init = true;
  111:                 }
  112:         }
  113:
  114:         private JComponent createView(File imagesDirectoryFile) {
  115:                 JButton okButton = new JButton(Messages.getString("ExportImagesVie
w.4")); //$NON-NLS-1$
  116:                 okButton.setSize(300, 35);
  117:                 okButton.setVisible(true);
  118:                 okButton.addActionListener(new ActionListener() {
  119:                         @Override
  120:                         public void actionPerformed(ActionEvent e) {
  121:                                 access.onButtonOk();
  122:                         }
  123:                 });
  124:
  125:                 JButton cancelButton = new JButton(Messages.getString("ExportImage
sView.5")); //$NON-NLS-1$
  126:                 cancelButton.setSize(300, 35);
```

```
  127:                 cancelButton.setVisible(true);
  128:                 cancelButton.addActionListener(new ActionListener() {
  129:                         @Override
  130:                         public void actionPerformed(ActionEvent e) {
  131:                                 access.hideView();
  132:                         }
  133:                 });
  134:
  135:                 JPanel buttonsPanel = new JPanel();
  136:                 buttonsPanel.add(cancelButton);
  137:                 buttonsPanel.add(okButton);
  138:
  139:                 buttonsPanel.setLayout(new BoxLayout(buttonsPanel, BoxLayout.X_AXI
S));
  140:
  141:                 globalCheckBox = new JCheckBox(Messages.getString("ExportImagesVie
w.6")); //$NON-NLS-1$
  142:                 globalCheckBox.addItemListener(new ItemListener() {
  143:                         @Override
  144:                         public void itemStateChanged(ItemEvent e) {
  145:                                 isGlobalImage = (e.getStateChange() == ItemEvent.S
ELECTED);
  146:                         }
  147:                 });
  148:
  149:                 JPanel view = new JPanel();
  150:                 view.setLayout(new BoxLayout(view, BoxLayout.Y_AXIS));
  151:
  152:                 addImagesViewFromVector(IconDatabase.vectorExportedIcons, Messages
.getString("ExportImagesView.7"), //$NON-NLS-1$
  153:                                 view);
  154:                 addImagesViewFromFile(imagesDirectoryFile, view);
  155:
  156:                 JScrollPane scrollPaneView = new JScrollPane(view);
  157:                 scrollPaneView.getVerticalScrollBar().setUnitIncrement(20);
  158:                 scrollPaneView.setVisible(true);
  159:
  160:                 JPanel globalView = new JPanel();
  161:                 globalView.setLayout(new BoxLayout(globalView, BoxLayout.Y_AXIS));
  162:                 globalView.add(scrollPaneView);
  163:                 globalView.add(globalCheckBox);
  164:                 globalView.add(buttonsPanel);
  165:
  166:                 return globalView;
  167:         }
  168:
  169:         protected void onButtonOk() {
  170:                 ConfigTriades.getInstance().getLastImages()
  171:                                 .addLastObject(new ExportImageData(exportImageData
));
  172:
  173:                 if (dataSelectionListener != null) {
  174:                         dataSelectionListener.updateSelection(null);
  175:                 }
  176:
  177:                 if (isGlobalImage) {
  178:                         Program.myMainFrame
  179:                                         .getDataPack()
  180:                                         .getCurrentSession()
  181:                                         .setExportedImageData(exportImageData,
  182:                                                         exportedVertex.getContent(
));
  183:                         exportedVertex.setImageData(new ExportImageData());
  184:                 } else {
  185:                         exportedVertex.setImageData(exportImageData);
  186:                 }
  187:
  188:                 hideView();
  189:         }
  190:
  191:         private void addImagesViewFromFile(File file, JComponent view) {
  192:                 File files[] = file.listFiles();
  193:                 Vector<JLabel> iconsVector = new Vector<JLabel>();
  194:                 int maxWidth = -1;
  195:
  196:                 for (int i = 0; i < files.length; i++) {
  197:                         final File subFile = files[i];
  198:
  199:                         if (subFile.isHidden()) {
  200:                                 continue;
  201:                         }
  202:
  203:                         if (subFile.isDirectory()) {
  204:                                 addImagesViewFromFile(subFile, view);
  205:                         } else if (subFile.isFile()) {
  206:                                 ImageIcon imageIcon = null;
  207:                                 try {
  208:                                 //TODO Modifier l'import des pics (redimenssioner
tranparence fond)
  209:                                         //      BufferedImage bufImage = new Buffe
redI
  210:                                         imageIcon = new ImageIcon(subFile.getPath(
));
  211:
  212:                                 } catch (Exception e) {
  213:                                         continue;
  214:                                 }
  215:
  216:                                 if ((imageIcon.getIconHeight() + imageIcon.getIcon
Width()) <= 0) {
  217:                                         continue;
  218:                                 }
  219:
  220:                                 final JLabel image = new JLabel(imageIcon);
  221:                                 image.setBorder(EmptyBorder);
  222:                                 image.addMouseListener(new MouseListener() {
  223:                                         @Override
  224:                                         public void mouseReleased(MouseEvent e) {
  225:                                         }
  226:
  227:                                         @Override
  228:                                         public void mousePressed(MouseEvent e) {
  229:                                         }
  230:
  231:                                         @Override
  232:                                         public void mouseExited(MouseEvent e) {
  233:                                                 if (selectedImage == image) {
  234:                                                         image.setBorder(SelectedBo
rder);
  235:                                                 } else {
  236:                                                         image.setBorder(EmptyBorde
r);
  237:                                                 }
  238:                                         }
  239:
  240:                                         @Override
  241:                                         public void mouseEntered(MouseEvent e) {
  242:                                                 image.setBorder(OverlapingBorder);
  243:                                         }
  244:
  245:                                         @Override
  246:                                         public void mouseClicked(MouseEvent e) {
  247:                                                 access.setSelectedPath(subFile.get
Path(), image);
```

```
248:                                                    if(e.getClickCount() > 1) {
249:                                                            access.onButtonOk();
250:                                                    }
251:                                            }
252:                                    });
253:
254:                                    maxWidth = Math.max(maxWidth, imageIcon.getIconWid
th());
255:
256:                                    iconsVector.add(image);
257:                            }
258:                    }
259:
260:                    if (!iconsVector.isEmpty()) {
261:                            view.add(createImagesView(iconsVector, maxWidth, file.getN
ame()));
262:                    }
263:            }
264:
265:            private void addImagesViewFromVector(Vector<ImageIcon> icons, String name,
266:                            JComponent view) {
267:                    Vector<JLabel> images = new Vector<JLabel>();
268:                    int maxWidth = -1;
269:
270:                    for (int i = 0; i < icons.size(); i++) {
271:                            final int index = i;
272:                            final JLabel image = new JLabel(icons.elementAt(i));
273:                            image.setBorder(EmptyBorder);
274:
275:                            image.addMouseListener(new MouseListener() {
276:                                    @Override
277:                                    public void mouseReleased(MouseEvent arg0) {
278:                                    }
279:
280:                                    @Override
281:                                    public void mousePressed(MouseEvent arg0) {
282:                                    }
283:
284:                                    @Override
285:                                    public void mouseExited(MouseEvent e) {
286:                                            if (selectedImage == image) {
287:                                                    image.setBorder(SelectedBorder);
288:                                            } else {
289:                                                    image.setBorder(EmptyBorder);
290:                                            }
291:                                    }
292:
293:                                    @Override
294:                                    public void mouseEntered(MouseEvent e) {
295:                                            image.setBorder(OverlapingBorder);
296:                                    }
297:
298:                                    @Override
299:                                    public void mouseClicked(MouseEvent arg0) {
300:                                            access.setSelectedIndex(index, image);
301:
302:                                            if(arg0.getClickCount() > 1) {
303:                                                    access.onButtonOk();
304:                                            }
305:                                    }
306:                            });
307:
308:                            maxWidth = Math.max(maxWidth, icons.elementAt(i).getIconWi
dth());
309:                            images.add(image);
310:                    }
311:
312:
313:                    if (!images.isEmpty()) {
314:                            view.add(createImagesView(images, maxWidth, name));
315:                    }
316:            }
317:
318:            private Component createImagesView(Vector<JLabel> iconsVector,
319:                            int maxWidth, String name) {
320:                    if (iconsVector.isEmpty()) {
321:                            throw new RuntimeException("Empty vector !"); //$NON-NLS-1
$
322:                    }
323:
324:                    JPanel directoryView = new JPanel();
325:                    directoryView.setLayout(new BoxLayout(directoryView, BoxLayout.Y_A
XIS));
326:
327:                    int colsCount = windowSize.width / (maxWidth + 2 * borderSize + 10
) - 1;
328:                    JPanel imagesView = new JPanel(new GridLayout(0, colsCount));
329:
330:                    for (JComponent image : iconsVector) {
331:                            imagesView.add(image);
332:                    }
333:
334:                    directoryView.add(new JLabel(name));
335:                    directoryView.add(imagesView);
336:
337:                    return directoryView;
338:            }
339:
340:            protected void hideView() {
341:                    if (selectedImage != null) {
342:                            selectedImage.setBorder(EmptyBorder);
343:                    }
344:
345:                    selectedImage = null;
346:                    exportedVertex = null;
347:
348:                    setVisible(false);
349:                    setEnabled(false);
350:                    // view.validate();
351:                    mainView.repaint();
352:            }
353:
354:            public void display(ExportVertexData exportedVertex, JComponent mainView,
DataSelectionListener dataListener) {
355:                    this.exportedVertex = exportedVertex;
356:                    this.mainView = mainView;
357:                    exportImageData.clear();
358:                    isGlobalImage = false;
359:                    globalCheckBox.setSelected(isGlobalImage);
360:                    dataSelectionListener = dataListener;
361:
362:                    setEnabled(true);
363:                    setVisible(true);
364:            }
365:
366:            protected void setSelectedIcon(JLabel image) {
367:                    if (selectedImage != null) {
368:                            selectedImage.setBorder(BorderFactory.createEmptyBorder(bo
rderSize,
369:                                            borderSize, borderSize, borderSize));
370:                    }
371:
372:                    selectedImage = image;
373:                    selectedImage.setBorder(SelectedBorder);
```

```
374:         }
375:
376:         protected void setSelectedIndex(int index, JLabel image) {
377:                 exportImageData.setImageIndex(index);
378:                 setSelectedIcon(image);
379:         }
380:
381:         protected void setSelectedPath(String path, JLabel image) {
382:                 exportImageData.setImagePath(path);
383:                 setSelectedIcon(image);
384:         }
385:
386:         static public ExportImagesView getSingleton() {
387:                 if (singleton == null) {
388:                         singleton = new ExportImagesView();
389:                 }
390:
391:                 singleton.init();
392:
393:                 return singleton;
394:         }
395: }
```

```
 1: package client.export;
 2:
 3: public class ExportTextData extends ExportData {
 4:
 5:         /**
 6:          *
 7:          */
 8:         private static final long serialVersionUID = 2987937624530906669L;
 9:
10: }
```

```
1: package client.export;
2:
3: public interface ExportDataInterface {
4:
5:         public ExportData getExportData();
6:
7: }
```

```
    1: package client.export;
    2:
    3: import java.io.Serializable;
    4: import java.util.ArrayList;
    5: import java.util.HashMap;
    6: import java.util.Map;
    7: import java.util.TreeMap;
    8:
    9: import models.BrickVertex.VerticeRank;
   10: import translation.Messages;
   11: import client.Session;
   12: import dataPack.Content;
   13: import dataPack.DataPack;
   14: import dataPack.MyDefaultMutableTreeNode;
   15:
   16: public class ExportedDatapackModule implements Serializable {
   17:
   18:         private static final long serialVersionUID = -120958692936379007L;
   19:         protected Session fullSession;
   20:         protected ArrayList<Session> sessions;
   21:         protected DataPack datapack;
   22:
   23:         public ExportedDatapackModule(DataPack datapack)
   24:         {
   25:
   26:                 this.datapack = datapack;
   27:                 sessions = new ArrayList<Session>();
   28:
   29:                 HashMap<Content, VerticeRank> allActorSet = new HashMap<Content, V
erticeRank>();
   30:                 for (MyDefaultMutableTreeNode a : datapack.getAllActors())
   31:                 {
   32:                         allActorSet.put(a, VerticeRank.primary);
   33:                 }
   34:
   35:                 fullSession = new Session(allActorSet, Messages.getString("Exporte
dDatapackModule.0"), datapack); //$NON-NLS-1$
   36:
   37:         }
   38:
   39:         public Session addNewSession(String name, Map<Content, VerticeRank> actors
List) {
   40:                 Session newSession = new Session(actorsList, name, datapack);
   41:                 sessions.add(newSession);
   42:                 return newSession;
   43:         }
   44:
   45:         public ArrayList<Session> getSessionList() {
   46:                 return sessions;
   47:         }
   48:
   49:         public Session getFullSession() {
   50:
   51:                 return fullSession;
   52:         }
   53:
   54:         public void upSession(Session selectedSession) {
   55:                 int index = sessions.indexOf(selectedSession);
   56:                 if (index > 0) {
   57:                         sessions.remove(index);
   58:                         sessions.add(0, selectedSession);
   59:                 }
   60:         }
   61:
   62: }
```

```
    1: package client;
    2:
    3: import java.awt.event.MouseEvent;
    4:
    5: import translation.Messages;
    6:
    7: import main.RelationComplete;
    8: import models.Brick;
    9: import models.BrickEdge;
   10: import models.BrickVertex;
   11: import models.TriadeEditingMousePlugin;
   12: import client.export.ExportsContextualMenu;
   13: import dataPack.Content;
   14: import dataPack.TreeListener;
   15: import edu.uci.ics.jung.algorithms.layout.GraphElementAccessor;
   16: import edu.uci.ics.jung.algorithms.layout.Layout;
   17: import edu.uci.ics.jung.graph.Graph;
   18: import edu.uci.ics.jung.visualization.VisualizationViewer;
   19: import graphicalUserInterface.DialogHandlerFrame;
   20: import graphicalUserInterface.Program;
   21: import graphicalUserInterface.RelationChooserPopUp;
   22:
   23: public class SchemaEditingMousePlugin extends
   24:                 TriadeEditingMousePlugin<BrickVertex, BrickEdge> {
   25:
   26:         protected Brick editedSchema;
   27:         protected NavigationTree navigationTree;
   28:
   29:         public SchemaEditingMousePlugin(TreeListener _treeListener,
   30:                         Brick _editedSchema, Layout<BrickVertex, BrickEdge> vertex
Position) {
   31:                 super(vertexPosition);
   32:                 this.navigationTree = Program.myMainFrame.datapack.getCurrentSessi
on().getNavigationTree();
   33:                 editedSchema = _editedSchema;
   34:                 setTreeListener(_treeListener);
   35:         }
   36:
   37:         @Override
   38:         public void removeSelectedVertex() {
   39:                 if (selectedVertex != null) {
   40:                         DialogHandlerFrame
   41:                                         .showErrorDialog(Messages.getString("Schem
aEditingMousePlugin.0")); //$NON-NLS-1$
   42:
   43:                 } else if (selectedEdge != null) {
   44:                         DialogHandlerFrame
   45:                                         .showErrorDialog(Messages.getString("Schem
aEditingMousePlugin.1")); //$NON-NLS-1$
   46:
   47:                 }
   48:         }
   49:
   50:         @SuppressWarnings("unchecked")
   51:         @Override
   52:         public void mousePressed(MouseEvent e) {
   53:                 if (checkModifiers(e)) {
   54:
   55:                         final VisualizationViewer<BrickVertex, BrickEdge> vv = (Vi
sualizationViewer<BrickVertex, BrickEdge>) e
   56:                                         .getSource();
   57:                         GraphElementAccessor<BrickVertex, BrickEdge> pickSupport =
 vv
   58:                                         .getPickSupport();
   59:                         if (pickSupport != null) {
   60:                                 final BrickVertex vertex = pickSupport.getVertex(v
v
   61:                                                 .getGraphLayout(), e
   62:                                                 .getPoint().getX(), e.getPoint().g
etY());
   63:
   64:                                 BrickEdge edge = pickSupport.getEdge(vertexLocatio
ns, e.getPoint().getX(), e.getPoint().getY());
   65:
   66:                                 if (vertex == null && edge != null)
   67:                                 {
   68:                                         if (((RelationChooserPopUp) modelView.getP
opUp()).showRelationChooserView(edge, this,Messages.getString("SchemaEditingMousePlugin.2
")) != -1) //$NON-NLS-1$
   69:                                                 selectEdge(edge);
   70:
   71:                                 }
   72:                                 else
   73:                                 {
   74:                                         selectEdge(null);
   75:                                 }
   76:
   77:                                 if (vertex != null) { // get ready to make an edge
   78:                                         startVertex = vertex;
   79:                                         down = e.getPoint();
   80:                                         transformEdgeShape(down, down);
   81:                                         vv.addPostRenderPaintable(edgePaintable);
   82:                                         edgeIsDirected = true;
   83:                                         transformArrowShape(down, e.getPoint());
   84:                                         vv.addPostRenderPaintable(arrowPaintable);
   85:
   86:                                 }
   87:                                 else
   88:                                 {
   89:                                         selectVertex((BrickVertex)null);
   90:                                 }
   91:                         }
   92:                 }
   93:         }
   94:
   95:         /**
   96:          * If startVertex is non-null, and the mouse is released over an existing
   97:          * vertex, create an undirected edge from startVertex to the vertex under
   98:          * the mouse pointer. If shift was also pressed, create a directed edge
   99:          * instead.
  100:          */
  101:         @SuppressWarnings("unchecked")
  102:         @Override
  103:         public void mouseReleased(MouseEvent e) {
  104:                 if (checkModifiers(e)) {
  105:                         final VisualizationViewer<BrickVertex, BrickEdge> vv = (Vi
sualizationViewer<BrickVertex, BrickEdge>) e
  106:                                         .getSource();
  107:                         GraphElementAccessor<BrickVertex, BrickEdge> pickSupport =
 vv
  108:                                         .getPickSupport();
  109:                         if (pickSupport != null) {
  110:                                 final BrickVertex vertex = pickSupport.getVertex(v
v
  111:                                                 .getGraphLayout(), e
  112:                                                 .getPoint().getX(), e
  113:                                                 .getPoint().getY());
  114:                                 if (vertex != null && startVertex != null) {
  115:                                         if (vertex != startVertex) {
  116:                                                 if (vv.getGraphLayout().getGraph()
.isSuccessor(
  117:                                                                 startVertex, verte
x)) {
```

```
                                                        BrickEdge edge = vv.getGra
phLayout().getGraph()
                                                                        .findEdge(
                                                                        startVerte
x, vertex);
                                                        if (((RelationChooserPopUp
) modelView.getPopUp())
                                                                        .showRelat
ionChooserView(
                                                edge, this,
                                                Messages.getString("SchemaEditingMousePlugin.3")) != -1) //$NON-NLS-1$
                                                                selectEdge(edge);

                                                } else {
                                                        Graph<BrickVertex, BrickEd
ge> graph = vv.getGraphLayout().getGraph();
                                                        BrickVertex source = start
Vertex;
                                                        BrickVertex destination =
vertex;
                                                        BrickEdge mEdge = editedSc
hema.addEdge(new BrickEdge(source, destination,RelationComplete.createNoRelation()));

                                                        startVertex = null;
                                                        down = null;

                                                        boolean ok = false;
                                                        if (((RelationChooserPopUp
) modelView.getPopUp())
                                                                        .showRelat
ionChooserView(mEdge, this,
                                                Messages.getString("SchemaEditingMousePlugin.4")) != -1) //$NON-NLS-1$
                                                                ok = true;



                                                        if (ok) {


                                                                graph.addEdge(mEdg
e, source, destination);
                                                                selectEdge(mEdge);
                                                        }

                                                }
                                        } else {
                                                if (vertex != selectedVertex) {
                                                        selectVertex(vertex);

                                                }
                                        }

                                }
                        }
                        startVertex = null;
                        down = null;
                        edgeIsDirected = false;
                        vv.removePostRenderPaintable(edgePaintable);
                        vv.removePostRenderPaintable(arrowPaintable);
                        vv.repaint();
                }
                if (e.getButton() == MouseEvent.BUTTON3) {
                        ExportsContextualMenu menu = new ExportsContextualMenu(
                                        editedSchema, navigationTree);
                        menu.show(e.getComponent(), e.getX(), e.getY());
                }

        }

        @Override
        public Brick getEditedAbstractSchema() {
                return editedSchema;
        }

        @Override
        public void selectVertex(Content content) {
                selectVertex(editedSchema.getBrickVertexByContent(content));
        }

        @Override
        public void notifyTree(Content content) {
                treeListener.setSelectedContent(content);
        }

        @Override
        public void setTreeListener(TreeListener treeListener) {
                super.setTreeListener(treeListener);
                if(treeListener != null)
                        treeListener.setGrapheListener(this);
        }

        @Override
        public void setContent(Content content) {
                selectVertex(content);
        }
}
```

```
    1: package client;
    2:
    3: import models.Brick;
    4: import models.BrickEdge;
    5: import models.BrickVertex;
    6: import models.TriadeEditingMousePlugin;
    7: import dataPack.JTreeActors;
    8: import dataPack.MyDefaultMutableTreeNode;
    9:
   10: public class SchemaTree extends JTreeActors {
   11:
   12:         /**
   13:          *
   14:          */
   15:         private static final long serialVersionUID = 8772042230835489826L;
   16:
   17:         public SchemaTree() {
   18:                 //TODO a virer une fois ces arbre ne seront plus sauvÃ©
   19:                 super();
   20:         }
   21:
   22:         public SchemaTree(Brick schema, TriadeEditingMousePlugin<BrickVertex, Bric
kEdge> triadeEditingMousePlugin) {
   23:                 super(schema.getDatapack());
   24:
   25:                 this.brick = schema;
   26:
   27:                 setEditable(false);
   28:                 setTriadeEditingMousePlugin(triadeEditingMousePlugin);
   29:
   30:                 /* cache les sommets qui ne sont pas dans le schema */
   31:                 simpleHideAll();
   32:
   33:                 for(BrickVertex vertex : schema.getVertices()) {
   34:                         if(vertex.getContent() instanceof MyDefaultMutableTreeNode
)
   35:                                 simpleShowActor(vertex.getContent());
   36:                 }
   37:
   38:                 expendAllNode();
   39:         }
   40: }
```

```
1: package client.stringTranslator;
2:
3: import graphicalUserInterface.IconDatabase;
4: import graphicalUserInterface.Program;
5:
6: import java.awt.BorderLayout;
7: import java.awt.Color;
8: import java.awt.Dimension;
9: import java.awt.GridLayout;
10: import java.awt.event.MouseEvent;
11: import java.awt.event.MouseListener;
12: import java.awt.event.WindowEvent;
13: import java.awt.event.WindowListener;
14: import java.util.Enumeration;
15: import java.util.Hashtable;
16: import java.util.Vector;
17:
18: import javax.swing.BorderFactory;
19: import javax.swing.Box;
20: import javax.swing.BoxLayout;
21: import javax.swing.Icon;
22: import javax.swing.JComponent;
23: import javax.swing.JFrame;
24: import javax.swing.JLabel;
25: import javax.swing.JPanel;
26: import javax.swing.JScrollPane;
27: import javax.swing.JTextField;
28: import javax.swing.ScrollPaneConstants;
29: import javax.swing.event.CaretEvent;
30: import javax.swing.event.CaretListener;
31: import javax.swing.tree.DefaultMutableTreeNode;
32:
33: import main.ActionTime;
34: import relations.Goal;
35: import relations.Mean;
36: import relations.RelationDescription;
37: import dataPack.Activite;
38: import dataPack.DataPack;
39: import dataPack.Groupe;
40: import dataPack.JeuActeur;
41:
42: public class TranslatorView extends JPanel {
43:         private static final long serialVersionUID = -1617618019555022483L;
44:
45:         private final Icon hidenIcon;
46:         private final Icon displayIcon;
47:         private final Icon bulletIcon;
48:
49:         static private int VerticalStructSize = 10;
50:
51:         private final DataPack dataPack;
52:         private final Hashtable<String, Vector<JPanel>> titles;
53:         final JPanel access;
54:
55:         private final Vector<JPanel> allPanels;
56:         private final Vector<JTextField> allTextFields;
57:         private final Vector<String> allLabels;
58:
59:         public TranslatorView(DataPack dataPack) {
60:                 super(new BorderLayout());
61:                 Program.setIsTriade(true);
62:                 setTranslucentBackGroundColor(this);
63:                 allPanels = new Vector<JPanel>();
64:                 allTextFields = new Vector<JTextField>();
65:                 allLabels = new Vector<String>();
66:
67:                 hidenIcon = IconDatabase.iconMinimize;
68:                 displayIcon = IconDatabase.iconMaximize;
69:                 bulletIcon = IconDatabase.iconClose;
70:                 access = this;
71:                 this.dataPack = dataPack;
72:                 this.setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
73:                 titles = new Hashtable<String, Vector<JPanel>>();
74:                 initTitles();
75:                 initView();
76:         }
77:
78:         private void initTitles() {
79:                 // //actors
80:
81:                 Vector<JPanel> panels = new Vector<JPanel>();
82:
83:                 // activies
84:                 panels = new Vector<JPanel>();
85:                 for (Activite activity : dataPack.getActivities().getActivities())
 {
86:
87:                         panels.add(createStringView(activity, dataPack
88:                                         .getStringTranslator().activities, createB
ulletPanel(setTranslucentBackGroundColor(new JLabel(
89:                                                         StringTranslator.getNotTra
nslatedString(activity)))), StringTranslator.getNotTranslatedString(activity)));
90:                 }
91:                 titles.put("ActivitÃ©s", panels);
92:
93:                 // Staps
94:                 panels = new Vector<JPanel>();
95:                 for (String step : dataPack.getSteps()) {
96:                         panels.add(createStringView(step,
97:                                         dataPack.getStringTranslator().steps, crea
teBulletPanel(setTranslucentBackGroundColor(new JLabel(
98:                                                         StringTranslator.getNotTra
nslatedString(step))))
99:                                                         , StringTranslator.getNotT
ranslatedString(step)));
100:                 }
101:                 titles.put("Etapes", panels);
102:
103:                 // Action Times
104:                 panels = new Vector<JPanel>();
105:                 for (ActionTime actionTime : dataPack.getActionTimeList()) {
106:                         panels.add(createStringView(actionTime, dataPack
107:                                         .getStringTranslator().actionTimes, create
BulletPanel(setTranslucentBackGroundColor(new JLabel(
108:                                                         StringTranslator.getNotTra
nslatedString(actionTime)))), StringTranslator.getNotTranslatedString(actionTime)));
109:                 }
110:                 titles.put("Temps d'action", panels);
111:
112:                 //Relations
113:                 panels = new Vector<JPanel>();
114:                 for (RelationDescription relationDescription : dataPack.getRelatio
ns().getRelationDescriptions()) {
115:                         JPanel goalsPanel = createDefautlTranslucentJPanel(BoxLayo
ut.Y_AXIS);
116:                         for(Goal goal : relationDescription.getPossibility().getMa
p().keySet()) {
117:                                 JPanel meansPanel = createDefautlTranslucentJPanel
(BoxLayout.Y_AXIS);
118:                                 for(Mean mean : relationDescription.getPossibility
().getMap().get(goal)) {
119:                                         JPanel meanTitleView = createDefautlTransl
ucentJPanel(BoxLayout.X_AXIS);
120:                                         meanTitleView.add(Box.createHorizontalStru
```

```
t(VerticalStructSize*3));
  121:                                        meanTitleView.add(setTranslucentBackGround
Color(createBulletPanel(setTranslucentBackGroundColor(new JLabel(StringTranslator.getNotT
ranslatedString(mean))))));
  122:                                        meansPanel.add(createStringView(mean, data
Pack.getStringTranslator().moyens, meanTitleView, StringTranslator.getNotTranslatedString
(mean)));
  123:                                }
  124:
  125:                                JPanel goalTitleView = createDefautlTranslucentJPa
nel(BoxLayout.X_AXIS);
  126:                                goalTitleView.add(Box.createHorizontalStrut(Vertic
alStructSize*2));
  127:                                goalTitleView.add(createExpendedPanel(setTranslucе
ntBackGroundColor(new JLabel(StringTranslator.getNotTranslatedString(goal))), meansPanel)
);
  128:
  129:                                goalsPanel.add(createStringView(goal, dataPack.get
StringTranslator().objectifs, goalTitleView, StringTranslator.getNotTranslatedString(goal
)));
  130:                                goalsPanel.add(meansPanel);
  131:                        }
  132:
  133:                        JPanel relationTitleView = createDefautlTranslucentJPanel(
BoxLayout.X_AXIS);
  134:                        relationTitleView.add(Box.createHorizontalStrut(VerticalSt
ructSize));
  135:                        relationTitleView.add(createExpendedPanel(setTranslucentBa
ckGroundColor(new JLabel(StringTranslator.getNotTranslatedString(relationDescription))),
goalsPanel));
  136:
  137:                        JPanel relationView = createDefautlTranslucentJPanel(BoxLa
yout.Y_AXIS);
  138:                        relationView.add(createStringView(relationDescription, dat
aPack.getStringTranslator().groups, relationTitleView, StringTranslator.getNotTranslatedS
tring(relationDescription)));
  139:                        relationView.add(goalsPanel);
  140:
  141:                        panels.add(relationView);
  142:                }
  143:                titles.put("Relations", panels);
  144:        }
  145:
  146:        private JPanel createDefautlTranslucentJPanel(int axe) {
  147:                JPanel panel = new JPanel();
  148:                setTranslucentBackGroundColor(panel);
  149:                panel.setLayout(new BoxLayout(panel, axe));
  150:                return panel;
  151:        }
  152:
  153:        private void initView() {
  154:                JPanel allViews = new JPanel();
  155:                allViews.setLayout(new BoxLayout(allViews, BoxLayout.Y_AXIS));
  156:                setTranslucentBackGroundColor(allViews);
  157:
  158:                // Actors
  159:                JPanel actorsPanel = new JPanel();
  160:                JPanel verticalStruct = new JPanel();
  161:                verticalStruct.add(Box.createHorizontalStrut(1));
  162:                verticalStruct.setOpaque(false);
  163: //             actorsPanel.add(verticalStruct);
  164:                actorsPanel.setLayout(new BoxLayout(actorsPanel, BoxLayout.Y_AXIS)
);
  165:                actorsPanel.add(createGroupPanel(dataPack.getActorsModel().getGrou
peActeurs(), dataPack.getStringTranslator().actors));
  166:                actorsPanel.setBackground(Color.lightGray);
  167:                setDefaultTitleBorder(actorsPanel, "Acteurs");
  168:
  169:                // jeux acteurs
  170: //             JPanel jaView = new JPanel();
  171: //             jaView.setBackground(Color.blue);
  172: //             jaView.setLayout(new BoxLayout(jaView, BoxLayout.Y_AXIS));
  173: //             setDefaultTitleBorder(jaView, "Jeux d'acteurs");
  174:
  175:                for (JeuActeur jeuActeur : dataPack.getJeuxActeur()) {
  176:                        actorsPanel.add(createJeuxActeurPanel(jeuActeur));
  177:                }
  178:
  179:                allViews.add(actorsPanel);
  180: //             allViews.add(jaView);
  181:
  182:                // Moyens
  183:                JComponent moyensPanel = setDefaultTitleBorder(createGroupPanel(da
taPack.getActorsModel().getGroupeMoyens(), dataPack.getStringTranslator().moyens), "Moyen
s");
  184:                moyensPanel.setBackground(Color.pink);
  185:                moyensPanel.setOpaque(true);
  186:                allViews.add(moyensPanel);
  187:
  188:                // Others
  189:                allViews.setLayout(new BoxLayout(allViews, BoxLayout.Y_AXIS));
  190:                Color[] colors = new Color[]{Color.gray, new Color(210, 90, 90), n
ew Color(90, 210, 90), new Color(90, 90, 210)};
  191:                int colorIndex = 0;
  192:                for (String title : titles.keySet()) {
  193:                        Vector<JPanel> stringViews = titles.get(title);
  194:                        JComponent otherPanel = setDefaultTitleBorder(createTitleV
iew(title, stringViews), title);
  195:                        otherPanel.setOpaque(true);
  196:                        otherPanel.setBackground(colors[(colorIndex++)%colors.leng
th]);
  197:                        allViews.add(otherPanel);
  198:                }
  199:
  200:                JPanel globalView = new JPanel(new BorderLayout());
  201:
  202:                globalView.add(createSearchView(), BorderLayout.NORTH);
  203:                globalView.add(allViews, BorderLayout.CENTER);
  204:
  205:                JScrollPane scrollPane = new JScrollPane(globalView);
  206:                scrollPane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZO
NTAL_SCROLLBAR_NEVER);
  207:                scrollPane.getVerticalScrollBar().setUnitIncrement(20);
  208:                add(scrollPane);
  209:        }
  210:
  211: //     private JComponent createRelationPanel() {
  212: //             JPanel globalPanel = new JPanel();
  213: //             setTranslucentBackGroundColor(globalPanel);
  214: //             globalPanel.setLayout(new BoxLayout(globalPanel, BoxLayout.Y_AXIS)
);
  215: //
  216: //             for (RelationDescription relationDescription : dataPack.getRelatio
ns().getRelationDescriptions()) {
  217: //                     JPanel relationPanel = new JPanel();
  218: //                     setTranslucentBackGroundColor(relationPanel);
  219: //                     relationPanel.setLayout(new BoxLayout(relationPanel, BoxLa
yout.Y_AXIS));
  220: //
  221: //                     JPanel titleRelation = new JPanel();
  222: //                     setTranslucentBackGroundColor(titleRelation);
  223: //                     titleRelation.setLayout(new BoxLayout(titleRelation, BoxLa
yout.X_AXIS));
  224: //
```

```
  225: //                      globalPanel.add(createStringView(object, translation, left
View, label))
  226: //              }
  227: //
  228: //              return globalPanel;
  229: //      }
  230:
  231:          private JComponent setTranslucentBackGroundColor(JComponent panel) {
  232:                  panel.setOpaque(false);
  233:                  panel.setBackground(Color.yellow);
  234:                  return panel;
  235:          }
  236:
  237:          private JComponent setDefaultTitleBorder(JComponent view, String text) {
  238:                  view.setBorder(BorderFactory.createBevelBorder(1));
  239:                  return view;
  240:          }
  241:
  242:          private JComponent createBulletPanel(JComponent view) {
  243:                  JPanel titlePanel = new JPanel();
  244:                  setTranslucentBackGroundColor(titlePanel);
  245:                  titlePanel.setLayout(new BoxLayout(titlePanel, BoxLayout.X_AXIS));
  246:
  247: //               titlePanel.setBorder(BorderFactory.createTitledBorder("Test"));
  248:                  titlePanel.add(setTranslucentBackGroundColor(new JLabel(bulletIcon
)));
  249:                  titlePanel.add(view);
  250:
  251:                  return titlePanel;
  252:          }
  253:
  254:          private JPanel createSearchView() {
  255:                  JPanel panel = new JPanel(new GridLayout());
  256:
  257:                  JTextField searchText = new JTextField();
  258:                  searchText.addCaretListener(new CaretListener() {
  259:                          @Override
  260:                          public void caretUpdate(CaretEvent e) {
  261:                                  if(e.getSource() instanceof JTextField) {
  262:                                          String searchString = ((JTextField)e.getSo
urce()).getText();
  263:                                          for(int i = 0 ; i < allPanels.size() ; i++
){
  264:                                                  boolean display = false;
  265:                                                  if(searchString.compareTo("") == 0
) {
  266:                                                          display = true;
  267:                                                  }else if(allLabels.elementAt(i).co
ntains(searchString) || allTextFields.elementAt(i).getText().contains(searchString)) {
  268:                                                          display = true;
  269:                                                  }
  270:
  271:                                                  allPanels.elementAt(i).setVisible(
display);
  272:                                          }
  273:                                  }
  274:                          }
  275:                  });
  276:
  277:                  panel.add(new JLabel("Recherche"));
  278:                  panel.add(searchText);
  279:                  panel.setBorder(BorderFactory.createBevelBorder(5));
  280:
  281:                  JPanel result = createDefautlTranslucentJPanel(BoxLayout.Y_AXIS);
  282:
  283:                  result.add(Box.createVerticalStrut(10));
  284:                  result.add(panel);
  285:                  result.add(Box.createVerticalStrut(10));
  286:
  287:                  return result;
  288:          }
  289:
  290:          private JPanel createStringView(Object object,
  291:                          Hashtable<Object, String> translation, JComponent leftView
, String label) {
  292:                  JPanel panel = new JPanel(new GridLayout());
  293:                  setTranslucentBackGroundColor(panel);
  294:
  295:                  JTextField textField = new JTextField();
  296:                  textField.setSize(new Dimension(20, 35));
  297:
  298:                  if (translation.get(object) != null) {
  299:                          textField.setText(translation.get(object));
  300:                  }
  301:
  302:                  textField
  303:                  .addCaretListener(createTextFieldListener(object, translation));
  304:
  305:                  JPanel labelPanel = new JPanel();
  306:                  setTranslucentBackGroundColor(labelPanel);
  307:                  labelPanel.setLayout(new BoxLayout(labelPanel, BoxLayout.X_AXIS));
  308:
  309:                  if (leftView != null) {
  310:                          labelPanel.add(leftView);
  311:                  }
  312:
  313:                  // labelPanel.add(new
  314:                  // JLabel(StringTranslator.getNotTranslatedString(object)));
  315:                  labelPanel.add(Box.createGlue());
  316:
  317:                  panel.add(labelPanel);
  318:                  panel.add(textField);
  319:                  textField.validate();
  320:                  textField.repaint();
  321:
  322:                  allLabels.add(label);
  323:                  allTextFields.add(textField);
  324:                  allPanels.add(panel);
  325:
  326:                  return panel;
  327:          }
  328:
  329:          private JPanel createJeuxActeurPanel(JeuActeur jeuActeur) {
  330:                  JPanel panel = new JPanel();
  331:                  setTranslucentBackGroundColor(panel);
  332:                  panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
  333:
  334:                  JPanel subView = new JPanel();
  335:                  setTranslucentBackGroundColor(subView);
  336:                  subView.setLayout(new BoxLayout(subView, BoxLayout.Y_AXIS));
  337:                  for (String corp : jeuActeur.getListeCorps()) {
  338:                          subView.add(createGroupPanel(jeuActeur.getGroupeCorps(corp
),
  339:                                          dataPack.getStringTranslator().actors, 1))
;
  340:                  }
  341:
  342:                  JPanel titlePanel = new JPanel();
  343:                  setTranslucentBackGroundColor(titlePanel);
  344:                  titlePanel.setLayout(new BoxLayout(titlePanel, BoxLayout.X_AXIS));
  345:
  346:                  JLabel expendeButton = createExpendedImage(subView);
  347:                  setTranslucentBackGroundColor(expendeButton);
  348:
```

```
  349:                 titlePanel.add(expendeButton);
  350:                 titlePanel.add(setTranslucentBackGroundColor(new JLabel(StringTran
slator
  351:                                 .getNotTranslatedString(jeuActeur.getGroupeJeuActe
ur())));
  352:                 titlePanel.add(Box.createGlue());
  353:
  354:                 // JPanel expendedButton =
  355:                 // createExpendedPanel(jeuActeur.getGroupeJeuActeur(),
  356:                 // dataPack.getStringTranslator().groups, subView);
  357:                 // createExpendedPanel(createStringView(jeuActeur.getGroupeJeuActe
ur(),
  358:                 // dataPack.getStringTranslator().groups), subView)
  359:                 panel.add(createStringView(jeuActeur.getGroupeJeuActeur(), dataPac
k.getStringTranslator().groups, titlePanel, StringTranslator.getNotTranslatedString(jeuAc
teur.getGroupeJeuActeur())));
  360:                 panel.add(subView);
  361:
  362: //               panel.setBorder(BorderFactory.createLineBorder(Color.blue, 1));
  363:
  364:                 return panel;
  365:         }
  366:
  367:         private JLabel createExpendedImage(final JPanel subView) {
  368:                 final JLabel label = new JLabel(hidenIcon);
  369:                 setTranslucentBackGroundColor(label);
  370:                 label.addMouseListener(new MouseListener() {
  371:                         @Override
  372:                         public void mouseReleased(MouseEvent e) {
  373:                                 subView.setVisible(!subView.isVisible());
  374:                                 label.setIcon(subView.isVisible() ? hidenIcon : di
splayIcon);
  375:                         }
  376:
  377:                         @Override
  378:                         public void mousePressed(MouseEvent e) {}
  379:
  380:                         @Override
  381:                         public void mouseExited(MouseEvent e) {}
  382:
  383:                         @Override
  384:                         public void mouseEntered(MouseEvent e) {}
  385:
  386:                         @Override
  387:                         public void mouseClicked(MouseEvent e) {}
  388:                 });
  389:
  390:                 return label;
  391:         }
  392:
  393:         private CaretListener createTextFieldListener(final Object object,
  394:                         final Hashtable<Object, String> translation) {
  395:                 return new CaretListener() {
  396:                         @Override
  397:                         public void caretUpdate(CaretEvent e) {
  398:                                 if (e.getSource() instanceof JTextField) {
  399:                                         String text = ((JTextField) e.getSource())
.getText();
  400:
  401:                                         if (text.compareTo("") != 0) {
  402:                                                 translation.put(object, text);
 - " + translation);
  403:                                                 System.out.println("Put traduction
  404:                                         } else {
  405:                                                 translation.remove(object);
  406:                                         }
  407:                                 }
  408:                         };
  409:         }
  410:
  411:         private JPanel createTitleView(String title, Vector<JPanel> stringViews) {
  412:                 JPanel globalPanel = createDefautlTranslucentJPanel(BoxLayout.Y_AX
IS);
  413:                 JLabel titleView = new JLabel(title);
  414:                 setTranslucentBackGroundColor(titleView);
  415:
  416:                 JPanel stringView = createDefautlTranslucentJPanel(BoxLayout.Y_AXI
S);
  417:                 for (JPanel jPanel : stringViews) {
  418:                         setTranslucentBackGroundColor(jPanel);
  419:                         stringView.add(jPanel);
  420:                 }
  421:
  422:                 globalPanel.add(createExpendedPanel(titleView, stringView));
  423:                 globalPanel.add(stringView);
  424:                 globalPanel.setBorder(BorderFactory.createLineBorder(Color.black,
1));
  425:                 return globalPanel;
  426:         }
  427:
  428:         private JPanel createExpendedPanel(JComponent view, final JComponent subVi
ew) {
  429:                 JPanel panel = new JPanel(new BorderLayout());
  430:                 setTranslucentBackGroundColor(panel);
  431:
  432:                 final JLabel button = new JLabel(hidenIcon);
  433:                 setTranslucentBackGroundColor(button);
  434:                 button.addMouseListener(new MouseListener() {
  435:                         @Override
  436:                         public void mouseReleased(MouseEvent e) {
  437:                                 subView.setVisible(!subView.isVisible());
  438:                                 button.setIcon(subView.isVisible() ? hidenIcon : d
isplayIcon);
  439:                         }
  440:
  441:                         @Override
  442:                         public void mousePressed(MouseEvent e) {}
  443:
  444:                         @Override
  445:                         public void mouseExited(MouseEvent e) {}
  446:
  447:                         @Override
  448:                         public void mouseEntered(MouseEvent e) {}
  449:
  450:                         @Override
  451:                         public void mouseClicked(MouseEvent e) {}
  452:                 });
  453:
  454:                 panel.add(button, BorderLayout.WEST);
  455:                 panel.add(view, BorderLayout.CENTER);
  456:
  457:                 return panel;
  458:         }
  459:
  460:         private JPanel createGroupPanel(DefaultMutableTreeNode group,
  461:                         Hashtable<Object, String> table) {
  462:                 return createGroupPanel(group, table, 0);
  463:         }
  464:
  465:         private JPanel createGroupPanel(DefaultMutableTreeNode group,
  466:                         Hashtable<Object, String> table, int depth) {
  467:                 JPanel panel = createDefautlTranslucentJPanel(BoxLayout.Y_AXIS);
  468:                 JPanel titlePanel = createDefautlTranslucentJPanel(BoxLayout.X_AXI
S);
```

```
  469:
  470:                 if (group instanceof Groupe) {
  471:                         // create children panel
  472:                         titlePanel.add(Box.createHorizontalStrut(VerticalStructSiz
e*depth));
  473:                         final JPanel childrenPanels = createDefautlTranslucentJPan
el(BoxLayout.Y_AXIS);
  474:
  475:                         for (Enumeration<DefaultMutableTreeNode> nodes = group.chi
ldren(); nodes
  476:                         .hasMoreElements();) {
  477:                                 DefaultMutableTreeNode node = nodes.nextElement();
  478:                                 childrenPanels.add(createGroupPanel(node, table, d
epth+1));
  479:                         }
  480:
  481:                         JLabel expendeButton = createExpendedImage(childrenPanels)
;
  482:
  483:                         titlePanel.add(expendeButton);
  484:                         titlePanel.add(setTranslucentBackGroundColor(setTranslucen
tBackGroundColor(new JLabel(StringTranslator
  485:                                         .getNotTranslatedString(group)))));
  486:
  487:                         panel.add(createStringView(group,
  488:                                         dataPack.getStringTranslator().groups, tit
lePanel, StringTranslator.getNotTranslatedString(group)), StringTranslator.getNotTranslat
edString(group));
  489:                         panel.add(childrenPanels);
  490:                 } else {
  491:                         // create string panel
  492:
  493:                         titlePanel.add(Box.createHorizontalStrut(VerticalStructSiz
e*(depth-1) + VerticalStructSize/2));
  494:                         titlePanel.add(createBulletPanel(setTranslucentBackGroundC
olor(new JLabel(StringTranslator.getNotTranslatedString(group)))));
  495:
  496:                         panel.add(createStringView(group, table, titlePanel, Strin
gTranslator.getNotTranslatedString(group)));
  497:                 }
  498:
  499: //              panel.setBorder(BorderFactory.createLineBorder(Color.red, 1));
  500:
  501:                 return panel;
  502:         }
  503:
  504:         public static void main(String[] args) {
  505:                 new IconDatabase();
  506:                 final DataPack dataPack = (DataPack) Program.loadSavableObject(nul
l,
  507:                                 true);
  508:
  509:                 if(dataPack == null) {
  510:                         return;
  511:                 }
  512: //               dataPack.setStringTranslator(null);
  513:
  514:                  if(dataPack.getStringTranslator() == null) {
  515:                          dataPack.setStringTranslator(new StringTranslator());
  516:                  }
  517:
  518:                 JFrame frame = new JFrame();
  519:                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  520:                 frame.addWindowListener(new WindowListener() {
  521:                         @Override
  522:                         public void windowOpened(WindowEvent e) {
  523:                                 // TODO Auto-generated method stub
  524:
  525:                         }
  526:
  527:                         @Override
  528:                         public void windowIconified(WindowEvent e) {
  529:                                 // TODO Auto-generated method stub
  530:
  531:                         }
  532:
  533:                         @Override
  534:                         public void windowDeiconified(WindowEvent e) {
  535:                                 // TODO Auto-generated method stub
  536:
  537:                         }
  538:
  539:                         @Override
  540:                         public void windowDeactivated(WindowEvent e) {
  541:                                 Program.setIsTriade(false);
  542:                                 Program.save(dataPack, true);
  543:                         }
  544:
  545:                         @Override
  546:                         public void windowClosing(WindowEvent e) {
  547:                                 Program.setIsTriade(false);
  548:                                 Program.save(dataPack, true);
  549:                         }
  550:
  551:                         @Override
  552:                         public void windowClosed(WindowEvent e) {
  553:                                 Program.setIsTriade(false);
  554:                                 Program.save(dataPack, true);
  555:                         }
  556:
  557:                         @Override
  558:                         public void windowActivated(WindowEvent e) {
  559:                                 Program.setIsTriade(false);
  560:                                 Program.save(dataPack, true);
  561:                         }
  562:                 });
  563:                 frame.setSize(new Dimension(800, 600));
  564:
  565:                 frame.add(new TranslatorView(dataPack));
  566:
  567:                 frame.validate();
  568:                 frame.repaint();
  569:                 frame.setVisible(true);
  570:         }
  571: }
```

```
    1: package client.stringTranslator;
    2:
    3: import graphicalUserInterface.Program;
    4:
    5: import java.io.Serializable;
    6: import java.util.Hashtable;
    7:
    8: public class StringTranslator implements Serializable {
    9:         private static final long serialVersionUID = -536675760096045711 7L;
   10:
   11:         public enum StringType{actorType, groupType, activityType, objectifType, m
oyenType, brickType, stepsType, actionTimesType, actorSetType, statusType}
   12:
   13:         final public Hashtable<Object, String> actors;
   14:         final public Hashtable<Object, String> groups;
   15:         final public Hashtable<Object, String> activities;
   16:         final public Hashtable<Object, String> objectifs;
   17:         final public Hashtable<Object, String> moyens;
   18:         final public Hashtable<Object, String> bricks;
   19:         final public Hashtable<Object, String> steps;
   20:         final public Hashtable<Object, String> actionTimes;
   21: //      final public Hashtable<Translatable, String> actorsSet;
   22: //      final private Hashtable<String, String> status;
   23:
   24:         public StringTranslator() {
   25:                 actors = new Hashtable<Object, String>();
   26:                 groups = new Hashtable<Object, String>();
   27:                 activities = new Hashtable<Object, String>();
   28:                 objectifs = new Hashtable<Object, String>();
   29:                 moyens = new Hashtable<Object, String>();
   30:                 bricks = new Hashtable<Object, String>();
   31:                 steps = new Hashtable<Object, String>();
   32:                 actionTimes = new Hashtable<Object, String>();
   33: //              actorsSet = new Hashtable<Translatable, String>();
   34: //              status = new Hashtable<String, String>();
   35:         }
   36:
   37:         private String getTranslatedString(Object object, Hashtable<Object, String
> hashTables) {
   38:                 if(hashTables == objectifs) {
   39:                         System.out.println("");
   40:                 } else if(hashTables == moyens) {
   41:                         System.out.println("");
   42:                 }
   43:                 return hashTables.get(object);
   44:         }
   45:
   46:         public static String getNotTranslatedString(Object object) {
   47:                 String noTranlatedString;
   48:
   49:                 try {
   50:                         java.lang.reflect.Method method = object.getClass().getMet
hod("getNoTranslatedString");
   51:                         noTranlatedString = method.invoke(object).toString();
   52:                 } catch (Exception e) {
   53:                         if(object instanceof String) {
   54:                                 noTranlatedString = (String)object;
   55:                         } else {
   56:                                 System.out.println("Impossible de trouver le metho
de \"getNotTranslatedString\" dans l'objet + " + object.getClass().getName());
   57:                                 e.printStackTrace();
   58:                                 noTranlatedString = object.getClass().getName();
   59:                         }
   60:                 }
   61:
   62:                 return noTranlatedString;
   63:         }
   64:
   65:         private String getString(Object object, StringType type) {
   66:                 switch(type) {
   67:                         case actorType : return getTranslatedString(object, actors
);
   68:                         case groupType : return getTranslatedString(object, groups
);
   69:                         case moyenType : return getTranslatedString(object, moyens
);
   70:                         case activityType : return getTranslatedString(object, act
ivities);
   71:                         case objectifType : return getTranslatedString(object, obj
ectifs);
   72:                         case brickType : return getTranslatedString(object, bricks
);
   73:                         case stepsType : return getTranslatedString(object, steps)
;
   74:                         case actionTimesType : return getTranslatedString(object,
actionTimes);
   75: //                      case actorSetType : return getTranslatedString(translatabl
e, actorsSet);
   76: //                      case statusType : return getTranslatedString(string, statu
s);
   77:                 }
   78:
   79:                 return null;
   80:         }
   81:
   82:         public static String getTranslatedString(Object object, StringType type, S
tringTranslator translator) {
   83:                 if(translator == null) {
   84:                         return getNotTranslatedString(object);
   85:                 } else {
   86:                         return translator.getString(object, type);
   87:                 }
   88:         }
   89:
   90:         public static String getTranslatedString(Object object, StringType type) {
   91:                 if(Program.myMainFrame == null) {
   92:                         return getNotTranslatedString(object);
   93:                 } else {
   94:                         StringTranslator translator = Program.myMainFrame.getDataP
ack().getStringTranslator();
   95:                         String result = getTranslatedString(object, type, translat
or);
   96:                         if(result == null) {
   97:                                 System.out.println(getNotTranslatedString(object)
+ "n'a pas detraduction");
   98:                                 return getNotTranslatedString(object);
   99:                         }
  100:                         return result;
  101:                 }
  102:         }
  103:
  104:         private void printTable(String string, Hashtable<? extends Object, String>
 table) {
  105:                 for (Object object : table.keySet()) {
  106:                         System.out.println(string + " : " + getNotTranslatedString
(object) + " -> " + table.get(object));
  107:                 }
  108:         }
  109: }
```

```java
package client;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;
import java.util.Vector;

import javax.swing.Icon;

import models.Brick;
import models.BrickVertex.VerticeRank;
import models.SchemaGenerator;
import client.export.ExportImageData;
import client.export.ExportModel;
import dataPack.Content;
import dataPack.DataPack;

public class Session implements Serializable {

	private static final long serialVersionUID = -8314347987491588105L;
	protected HashMap<Content, VerticeRank> actorsList;
	protected String name;
	protected transient SchemaGenerator schemaGenerator;
	protected transient NavigationTree navigationTree;
	protected DataPack datapack;
	protected Vector<Brick> Bricks;

	protected HashMap<Brick, Vector<ExportModel>> exports;
	protected HashMap<Content, ExportImageData> defaultImages;

	protected Session() {
		actorsList = null;
		name = null;
		schemaGenerator = null;
		datapack = null;
		Bricks = null;
		exports = new HashMap<Brick, Vector<ExportModel>>();
		defaultImages = new HashMap<Content, ExportImageData>();
	}

	public Session(Map<Content, VerticeRank> actorsList, String name,
			DataPack datapack) {
		this();

		this.actorsList = new HashMap<Content, VerticeRank>(actorsList);
		this.name = name;
		this.datapack = datapack;
		schemaGenerator = new SchemaGenerator(datapack, actorsList);
	}

	public Vector<Brick> getBrickList() {
		if (Bricks == null)
  {
			schemaGenerator = new SchemaGenerator(datapack, actorsList
);
			Bricks = schemaGenerator.generateBricks();

		}
		return Bricks;
	}

	public ExportModel addExport(Brick baseBrick, ExportModel newExport) {
		Vector<ExportModel> vertexExports;
		if ((vertexExports = exports.get(baseBrick)) == null) {
			vertexExports = new Vector<ExportModel>();
			exports.put(baseBrick, vertexExports);
		}
		vertexExports.add(newExport);

		return newExport;
	}

	public Vector<ExportModel> getExports() {
		Vector<ExportModel> result = new Vector<ExportModel>();

		for(Vector<ExportModel> vertexExports : exports.values())
			result.addAll(vertexExports);

		return result;
	}

	public Vector<ExportModel> getExports(Brick baseBrick) {
		return exports.get(baseBrick);
	}

	public HashMap<Content, VerticeRank> getActorsList() {
		return actorsList;
	}

	public void setActorsList(HashMap<Content, VerticeRank> actorsList) {
		this.actorsList = actorsList;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public void setNavigationTree(NavigationTree navigationTree) {
		this.navigationTree = navigationTree;
	}

	public NavigationTree getNavigationTree() {
		return navigationTree;
	}

	public Icon getDefaultImage(Content content) {

		ExportImageData data = defaultImages.get(content);

		if(data != null) {
			return data.getIcon();
		}

		return null;
	}

	@Override
	public String toString() {
		return name;
	}

	public void setExportedImageData(ExportImageData exportImageData, Content
content) {
		defaultImages.put(content, new ExportImageData(exportImageData));
	}

}
```

```
 1: package graphicalUserInterface;
 2:
 3: import translation.Messages;
 4:
 5:
 6: public class CreatorLauncher {
 7:
 8:         /**
 9:          * @param args
10:          */
11:         public static void main(String[] args) {
12:                 if (args.length == 0) {
13:                         Logger.getInstance().setLoggerCreator(true);
14:                 }
15:
16:                 String[] arg = { "creator" }; //$NON-NLS-1$
17:                 Program.main(arg);
18:         }
19: }
```

```
 1: package graphicalUserInterface;
 2:
 3: public class TriadeLauncher {
 4:
 5:         protected int test;
 6:
 7:         public static void main(String[] args) {
 8:                 if (args.length == 0) {
 9:                         Logger.getInstance().setLoggerTriade(true);
10:                 }
11:
12:                 String[] arg = null;
13:                 Program.main(arg);
14:         }
15: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.util.HashMap;
    4: import java.util.Vector;
    5:
    6: import javax.swing.Icon;
    7: import javax.swing.ImageIcon;
    8:
    9: public final class IconDatabase {
   10:
   11:         static public String defaultIconsDirectory = "/Icones/16x16/configure.png"
;
   12:
   13:         // 16x16 icons
   14:         public static ImageIcon iconNew;
   15:
   16:         public static ImageIcon iconOpen;
   17:         public static ImageIcon iconSave;
   18:         public static ImageIcon iconSaveAs;
   19:         public static ImageIcon iconExport;
   20:         public static ImageIcon iconExportPopUp;
   21:         public static ImageIcon iconClose;
   22:         public static ImageIcon iconExit;
   23:         public static ImageIcon iconUndo;
   24:         public static ImageIcon iconRedo;
   25:         public static ImageIcon iconCopy;
   26:         public static ImageIcon iconCut;
   27:         public static ImageIcon iconPaste;
   28:         public static ImageIcon iconDelete;
   29:         public static ImageIcon iconSearch;
   30:         public static ImageIcon iconTree;
   31:         public static ImageIcon iconSchema;
   32:         public static ImageIcon iconAnalysis;
   33:         public static ImageIcon iconResolution;
   34:         public static ImageIcon iconAudit;
   35:         public static ImageIcon iconHelp;
   36:         public static ImageIcon iconUpdate;
   37:         public static ImageIcon iconAbout;
   38:         public static ImageIcon iconMinimize;
   39:         public static ImageIcon iconMaximize;
   40:         public static ImageIcon iconLeftArrow;
   41:         public static ImageIcon iconRightArrow;
   42:         public static ImageIcon iconDuplicateArrow;
   43:         public static ImageIcon iconConfigure;
   44:         public static ImageIcon iconArrowDownLeft;
   45:         public static ImageIcon iconArrowDownRight;
   46:         public static ImageIcon iconArrowUpLeft;
   47:         public static ImageIcon iconArrowUpRight;
   48:         public static ImageIcon iconUnfold;
   49:         public static ImageIcon iconAdd;
   50:         public static ImageIcon iconButtonOk;
   51:         public static ImageIcon iconButtonCancel;
   52:         public static ImageIcon iconRemoveActor0;
   53:         public static ImageIcon iconRemoveActor1;
   54:         public static ImageIcon iconRemoveActor2;
   55:         public static Vector<ImageIcon> vectorIconActorsMin;
   56:         public static ImageIcon iconMoyen;
   57:         public static ImageIcon iconBrickMin;
   58:         public static ImageIcon iconFileClose;
   59:         public static ImageIcon iconFileOpen;
   60:         public static ImageIcon iconFileEmpty;
   61:         public static Vector<ImageIcon> vectorIconActivitiesMin;
   62:         public static Vector<ImageIcon> vectorIconMoyensMin;
   63:         public static ImageIcon iconRename;
   64:
   65:         // 22x22 icons
   66:         public static ImageIcon iconRemoveTab;
   67:         public static ImageIcon iconAddActor;
   68:         public static ImageIcon iconAddGroup;
   69:         public static ImageIcon iconAddActivity;
   70:         public static ImageIcon iconAddBrick;
   71:         public static ImageIcon iconAddActionTime;
   72:         public static ImageIcon iconAddMoyen;
   73:         public static ImageIcon iconAddStep;
   74:
   75:         // 32x32 icons
   76:         public static ImageIcon iconDatapack;
   77:         public static ImageIcon iconModel;
   78:         public static ImageIcon iconBrick;
   79:         public static ImageIcon iconNewFile;
   80:         public static ImageIcon iconOpenFile;
   81:         public static ImageIcon iconUpgrade;
   82:         public static ImageIcon iconActorSet;
   83:
   84:         // 48x48 icons
   85:         public static ImageIcon iconTriad;
   86:         public static ImageIcon iconHumanActor;
   87:         public static ImageIcon iconHumanActorSelected;
   88:         public static ImageIcon iconArrowOut;
   89:         public static ImageIcon iconArrowOutSelected;
   90:         public static ImageIcon iconActivity;
   91:         public static ImageIcon iconActivitySelected;
   92:         public static ImageIcon iconGroup;
   93:         public static ImageIcon iconGroupSelected;
   94:         public static ImageIcon iconLeftArrowMin;
   95:         public static ImageIcon iconRightArrowMin;
   96:         public static ImageIcon iconLeftArrowMax;
   97:         public static ImageIcon iconRightArrowMax;
   98:         public static Vector<ImageIcon> vectorIconActivities;
   99:         public static Vector<ImageIcon> vectorIconMoyens;
  100:         public static Vector<ImageIcon> vectorIconActorsBig;
  101:
  102:         // other
  103:         public static ImageIcon iconTitleNew;
  104:         public static ImageIcon iconTitleDataPack;
  105:         public static ImageIcon iconTitleNewBrick;
  106:         public static ImageIcon iconTitleModel;
  107:         public static ImageIcon iconTitleTriad;
  108:         public static ImageIcon iconAssistant;
  109:         public static ImageIcon iconCorpsBases;
  110:
  111:         // Exported icons
  112:         public static Vector<ImageIcon> vectorExportedIcons;
  113:         public static HashMap<String, ImageIcon> icons;
  114:
  115:         public IconDatabase() {
  116:
  117:                 if (iconNew == null) {
  118:                         iconNew = new ImageIcon(getClass().getResource(
  119:                                         "/Icones/16x16/nouveau.png")); //$NON-NLS-
1$
  120:
  121:                         iconOpen = new ImageIcon(getClass().getResource(
  122:                                         "/Icones/16x16/ouvrir.png")); //$NON-NLS-1
$
  123:                         iconSave = new ImageIcon(getClass().getResource(
  124:                                         "/Icones/16x16/enregistrer.png")); //$NON-
NLS-1$
  125:                         iconSaveAs = new ImageIcon(getClass().getResource(
  126:                                         "/Icones/16x16/enregistrer_sous.png")); //
$NON-NLS-1$
  127:                         iconUnfold = new ImageIcon(getClass().getResource(
  128:                                         "/Icones/16x16/derouler.png")); //$NON-NLS
-1$
```

```
  129:                      iconClose = new ImageIcon(getClass().getResource(
  130:                                  "/Icones/16x16/fermer.png")); //$NON-NLS-1
$
  131:                      iconExit = new ImageIcon(getClass().getResource(
  132:                                  "/Icones/16x16/quitter.png")); //$NON-NLS-
1$
  133:                      iconDelete = new ImageIcon(getClass().getResource(
  134:                                  "/Icones/16x16/supprimer.png")); //$NON-NL
S-1$
  135:                      iconTree = new ImageIcon(getClass().getResource(
  136:                                  "/Icones/16x16/arborescence.png")); //$NON
-NLS-1$
  137:                      iconSchema = new ImageIcon(getClass().getResource(
  138:                                  "/Icones/16x16/schema.png")); //$NON-NLS-1
$
  139:                      iconHelp = new ImageIcon(getClass().getResource(
  140:                                  "/Icones/16x16/aide.png")); //$NON-NLS-1$
  141:                      iconUpdate = new ImageIcon(getClass().getResource(
  142:                                  "/Icones/16x16/maj.png")); //$NON-NLS-1$
  143:                      iconAbout = new ImageIcon(getClass().getResource(
  144:                                  "/Icones/16x16/a_propos.png")); //$NON-NLS
-1$
  145:                      iconMinimize = new ImageIcon(getClass().getResource(
  146:                                  "/Icones/16x16/minimiser.png")); //$NON-NL
S-1$
  147:                      iconMaximize = new ImageIcon(getClass().getResource(
  148:                                  "/Icones/16x16/maximiser.png")); //$NON-NL
S-1$
  149:                      iconLeftArrow = new ImageIcon(getClass().getResource(
  150:                                  "/Icones/16x16/flecheGauche.png")); //$NON
-NLS-1$
  151:                      iconRightArrow = new ImageIcon(getClass().getResource(
  152:                                  "/Icones/16x16/flecheDroite.png")); //$NON
-NLS-1$
  153:                      iconDuplicateArrow = new ImageIcon(getClass().getResource(
  154:                                  "/Icones/16x16/flecheDupliquer.png")); //$
NON-NLS-1$
  155:                      iconConfigure = new ImageIcon(getClass().getResource(
  156:                                  "/Icones/16x16/configure.png")); //$NON-NL
S-1$
  157:                      iconArrowDownLeft = new ImageIcon(getClass().getResource(
  158:                                  "/Icones/16x16/flecheBasGauche.png")); //$
NON-NLS-1$
  159:                      iconArrowDownRight = new ImageIcon(getClass().getResource(
  160:                                  "/Icones/16x16/flecheBasDroite.png")); //$
NON-NLS-1$
  161:                      iconArrowUpLeft = new ImageIcon(getClass().getResource(
  162:                                  "/Icones/16x16/flecheHautGauche.png")); //
$NON-NLS-1$
  163:                      iconArrowUpRight = new ImageIcon(getClass().getResource(
  164:                                  "/Icones/16x16/flecheHautDroite.png")); //
$NON-NLS-1$
  165:                      iconRemoveActor0 = new ImageIcon(getClass().getResource(
  166:                                  "/Icones/16x16/removeActor0.png")); //$NON
-NLS-1$
  167:                      iconRemoveActor1 = new ImageIcon(getClass().getResource(
  168:                                  "/Icones/16x16/removeActor1.png")); //$NON
-NLS-1$
  169:                      iconRemoveActor2 = new ImageIcon(getClass().getResource(
  170:                                  "/Icones/16x16/removeActor2.png")); //$NON
-NLS-1$
  171:                      vectorIconActorsMin = new Vector<ImageIcon>();
  172:                      vectorIconActorsMin.add(new ImageIcon(getClass().getResour
ce(
  173:                                  "/Icones/16x16/userBlack.png"))); //$NON-N
LS-1$
  174:                      vectorIconActorsMin.add(new ImageIcon(getClass().getResour
ce(
  175:                                  "/Icones/16x16/userRed.png"))); //$NON-NLS
-1$
  176:                      vectorIconActorsMin.add(new ImageIcon(getClass().getResour
ce(
  177:                                  "/Icones/16x16/userGreen.png"))); //$NON-N
LS-1$
  178:                      vectorIconActorsMin.add(new ImageIcon(getClass().getResour
ce(
  179:                                  "/Icones/16x16/userYellow.png"))); //$NON-
NLS-1$
  180:                      vectorIconActorsMin.add(new ImageIcon(getClass().getResour
ce(
  181:                                  "/Icones/16x16/userPurple.png"))); //$NON-
NLS-1$
  182:                      vectorIconActorsMin.add(new ImageIcon(getClass().getResour
ce(
  183:                                  "/Icones/16x16/userBrown.png"))); //$NON-N
LS-1$
  184:                      iconMoyen = new ImageIcon(getClass().getResource(
  185:                                  "/Icones/16x16/moyen.png")); //$NON-NLS-1$
  186:                      iconBrickMin = new ImageIcon(getClass().getResource(
  187:                                  "/Icones/16x16/brick.png")); //$NON-NLS-1$
  188:                      iconFileClose = new ImageIcon(getClass().getResource(
  189:                                  "/Icones/16x16/fichierFerme.png")); //$NON
-NLS-1$
  190:                      iconFileOpen = new ImageIcon(getClass().getResource(
  191:                                  "/Icones/16x16/fichierOuvert.png")); //$NO
N-NLS-1$
  192:                      iconFileEmpty = new ImageIcon(getClass().getResource(
  193:                                  "/Icones/16x16/fichierVide.png")); //$NON-
NLS-1$
  194:                      iconRename = new ImageIcon(getClass().getResource(
  195:                                  "/Icones/16x16/contrat.png")); //$NON-NLS-
1$
  196:                      vectorIconActivitiesMin = new Vector<ImageIcon>();
  197:                      vectorIconActivitiesMin.add(new ImageIcon(getClass().getRe
source(
  198:                                  "/Icones/16x16/activite.png"))); //$NON-NL
S-1$
  199:                      vectorIconActivitiesMin.add(new ImageIcon(getClass().getRe
source(
  200:                                  "/Icones/16x16/contact.png"))); //$NON-NLS
-1$
  201:                      vectorIconActivitiesMin.add(new ImageIcon(getClass().getRe
source(
  202:                                  "/Icones/16x16/contrat.png"))); //$NON-NLS
-1$
  203:                      vectorIconActivitiesMin.add(new ImageIcon(getClass().getRe
source(
  204:                                  "/Icones/16x16/processus.png"))); //$NON-N
LS-1$
  205:                      vectorIconActivitiesMin.add(new ImageIcon(getClass().getRe
source(
  206:                                  "/Icones/16x16/compas.png"))); //$NON-NLS-
1$
  207:                      vectorIconActivitiesMin.add(new ImageIcon(getClass().getRe
source(
  208:                                  "/Icones/16x16/note.png"))); //$NON-NLS-1$
  209:                      vectorIconActivitiesMin.add(new ImageIcon(getClass().getRe
source(
  210:                                  "/Icones/16x16/reunion.png"))); //$NON-NLS
-1$
  211:                      vectorIconActivitiesMin.add(new ImageIcon(getClass().getRe
source(
  212:                                  "/Icones/16x16/aide.png"))); //$NON-NLS-1$
  213:                      vectorIconMoyensMin = new Vector<ImageIcon>();
```

```
  214:                         vectorIconMoyensMin.add(new ImageIcon(getClass().getResour
ce(
  215:                                 "/Icones/16x16/moyenGenerique.png"))); //$
NON-NLS-1$
  216:                         vectorIconMoyensMin.add(new ImageIcon(getClass().getResour
ce(
  217:                                 "/Icones/16x16/outils.png"))); //$NON-NLS-
1$
  218:                         vectorIconMoyensMin.add(new ImageIcon(getClass().getResour
ce(
  219:                                 "/Icones/16x16/methode.png"))); //$NON-NLS
-1$
  220:                         vectorIconMoyensMin.add(new ImageIcon(getClass().getResour
ce(
  221:                                 "/Icones/16x16/stock.png"))); //$NON-NLS-1
$
  222:
  223:                         // 22x22 icons
  224:                         iconRemoveTab = new ImageIcon(getClass().getResource(
  225:                                 "/Icones/22x22/removeTab.png")); //$NON-NL
S-1$
  226:                         iconAddActor = new ImageIcon(getClass().getResource(
  227:                                 "/Icones/22x22/addActor.png")); //$NON-NLS
-1$
  228:                         iconAddGroup = new ImageIcon(getClass().getResource(
  229:                                 "/Icones/22x22/addGroup.png")); //$NON-NLS
-1$
  230:                         iconAddActivity = new ImageIcon(getClass().getResource(
  231:                                 "/Icones/22x22/addActivity.png")); //$NON-
NLS-1$
  232:                         iconAddBrick = new ImageIcon(getClass().getResource(
  233:                                 "/Icones/22x22/addBrick.png")); //$NON-NLS
-1$
  234:                         iconAddActionTime = new ImageIcon(getClass().getResource(
  235:                                 "/Icones/22x22/addActionTime.png")); //$NO
N-NLS-1$
  236:                         iconAddMoyen = new ImageIcon(getClass().getResource(
  237:                                 "/Icones/22x22/addMoyen.png")); //$NON-NLS
-1$
  238:                         iconAddStep = new ImageIcon(getClass().getResource(
  239:                                 "/Icones/22x22/step.png")); //$NON-NLS-1$
  240:
  241:                         // 32x32 icons
  242:                         iconDatapack = new ImageIcon(getClass().getResource(
  243:                                 "/Icones/32x32/datapack.png")); //$NON-NLS
-1$
  244:                         iconModel = new ImageIcon(getClass().getResource(
  245:                                 "/Icones/32x32/model.png")); //$NON-NLS-1$
  246:                         iconBrick = new ImageIcon(getClass().getResource(
  247:                                 "/Icones/32x32/brick.png")); //$NON-NLS-1$
  248:                         iconNewFile = new ImageIcon(getClass().getResource(
  249:                                 "/Icones/32x32/newFile.png")); //$NON-NLS-
1$
  250:                         iconOpenFile = new ImageIcon(getClass().getResource(
  251:                                 "/Icones/32x32/openFile.png")); //$NON-NLS
-1$
  252:                         iconUpgrade = new ImageIcon(getClass().getResource(
  253:                                 "/Icones/32x32/upgrade.png")); //$NON-NLS-
1$
  254:
  255:                         // 48x48
  256:                         iconTriad = new ImageIcon(getClass().getResource(
  257:                                 "/Icones/48x48/Triade.png")); //$NON-NLS-1
$
  258:                         iconHumanActor = new ImageIcon(getClass().getResource(
  259:                                 "/Icones/48x48/acteurHumainNoir.png")); //
$NON-NLS-1$
  260:
rce(
  261:
; //$NON-NLS-1$
  262:
  263:
 //$NON-NLS-1$
  264:
e(
  265:
.png")); //$NON-NLS-1$
  266:
  267:
-1$
  268:
e(
  269:
$NON-NLS-1$
  270:
  271:
  272:
  273:
N-NLS-1$
  274:
  275:
NON-NLS-1$
  276:
  277:
NON-NLS-1$
  278:
  279:
NON-NLS-1$
  280:
  281:
NON-NLS-1$
  282:
  283:
  284:
  285:
rce(
  286:
-1$
  287:
rce(
  288:
$NON-NLS-1$
  289:
rce(
  290:
-1$
  291:
rce(
  292:
$NON-NLS-1$
  293:
rce(
  294:
LS-1$
  295:
rce(
  296:
//$NON-NLS-1$
  297:
rce(
  298:
1$
  299:
                         iconHumanActorSelected = new ImageIcon(getClass().getResou
                                 "/Icones/48x48/acteurHumainSelected.png"))
                         iconArrowOut = new ImageIcon(getClass().getResource(
                                 "/Icones/48x48/flecheVersExterieur.png"));
                         iconArrowOutSelected = new ImageIcon(getClass().getResourc
                                 "/Icones/48x48/flecheVersExterieurSelected
.png")); //$NON-NLS-1$
                         iconActivity = new ImageIcon(getClass().getResource(
                                 "/Icones/48x48/activite.png")); //$NON-NLS
-1$
                         iconActivitySelected = new ImageIcon(getClass().getResourc
e(
                                 "/Icones/48x48/activiteSelected.png")); //
$NON-NLS-1$
                         iconGroup = new ImageIcon(getClass().getResource(
                                 "/Icones/48x48/group.png")); //$NON-NLS-1$
                         iconGroupSelected = new ImageIcon(getClass().getResource(
                                 "/Icones/48x48/groupSelected.png")); //$NO
N-NLS-1$
                         iconLeftArrowMin = new ImageIcon(getClass().getResource(
                                 "/Icones/48x48/FlecheGaucheMin.png")); //$
NON-NLS-1$
                         iconRightArrowMin = new ImageIcon(getClass().getResource(
                                 "/Icones/48x48/FlecheDroiteMin.png")); //$
NON-NLS-1$
                         iconLeftArrowMax = new ImageIcon(getClass().getResource(
                                 "/Icones/48x48/FlecheGaucheMax.png")); //$
NON-NLS-1$
                         iconRightArrowMax = new ImageIcon(getClass().getResource(
                                 "/Icones/48x48/FlecheDroiteMax.png")); //$
NON-NLS-1$
                         vectorIconActivities = new Vector<ImageIcon>();
                         vectorIconActivities.add(iconActivity);
                         vectorIconActivities.add(iconActivitySelected);
                         vectorIconActivities.add(new ImageIcon(getClass().getResou
                                 "/Icones/48x48/contact.png"))); //$NON-NLS
-1$
                         vectorIconActivities.add(new ImageIcon(getClass().getResou
rce(
                                 "/Icones/48x48/contactSelected.png"))); //
$NON-NLS-1$
                         vectorIconActivities.add(new ImageIcon(getClass().getResou
rce(
                                 "/Icones/48x48/contrat.png"))); //$NON-NLS
-1$
                         vectorIconActivities.add(new ImageIcon(getClass().getResou
rce(
                                 "/Icones/48x48/contratSelected.png"))); //
$NON-NLS-1$
                         vectorIconActivities.add(new ImageIcon(getClass().getResou
rce(
                                 "/Icones/48x48/processus.png"))); //$NON-N
LS-1$
                         vectorIconActivities.add(new ImageIcon(getClass().getResou
rce(
                                 "/Icones/48x48/processusSelected.png")));
//$NON-NLS-1$
                         vectorIconActivities.add(new ImageIcon(getClass().getResou
rce(
                                 "/Icones/48x48/compas.png"))); //$NON-NLS-
1$
                         vectorIconActivities.add(new ImageIcon(getClass().getResou
```

```
rce(
  300:                                  "/Icones/48x48/compasSelected.png"))); //$
NON-NLS-1$
  301:                      vectorIconActivities.add(new ImageIcon(getClass().getResou
rce(
  302:                                  "/Icones/48x48/note.png"))); //$NON-NLS-1$
  303:                      vectorIconActivities.add(new ImageIcon(getClass().getResou
rce(
  304:                                  "/Icones/48x48/noteSelected.png"))); //$NO
N-NLS-1$
  305:                      vectorIconActivities.add(new ImageIcon(getClass().getResou
rce(
  306:                                  "/Icones/48x48/reunion.png"))); //$NON-NLS
-1$
  307:                      vectorIconActivities.add(new ImageIcon(getClass().getResou
rce(
  308:                                  "/Icones/48x48/reunionSelected.png"))); //
$NON-NLS-1$
  309:                      vectorIconActivities.add(new ImageIcon(getClass().getResou
rce(
  310:                                  "/Icones/48x48/idee.png"))); //$NON-NLS-1$
  311:                      vectorIconActivities.add(new ImageIcon(getClass().getResou
rce(
  312:                                  "/Icones/48x48/ideeSelected.png"))); //$NO
N-NLS-1$
  313:                      vectorIconMoyens = new Vector<ImageIcon>();
  314:                      vectorIconMoyens.add(new ImageIcon(getClass().getResource(
  315:                                  "/Icones/48x48/moyenGenerique.png"))); //$
NON-NLS-1$
  316:                      vectorIconMoyens.add(new ImageIcon(getClass().getResource(
  317:                                  "/Icones/48x48/moyenGeneriqueSelected.png"
))); //$NON-NLS-1$
  318:                      vectorIconMoyens.add(new ImageIcon(getClass().getResource(
  319:                                  "/Icones/48x48/outils.png"))); //$NON-NLS-
1$
  320:                      vectorIconMoyens.add(new ImageIcon(getClass().getResource(
  321:                                  "/Icones/48x48/outilsSelected.png"))); //$
NON-NLS-1$
  322:                      vectorIconMoyens.add(new ImageIcon(getClass().getResource(
  323:                                  "/Icones/48x48/methode.png"))); //$NON-NLS
-1$
  324:                      vectorIconMoyens.add(new ImageIcon(getClass().getResource(
  325:                                  "/Icones/48x48/methodeSelected.png"))); //
$NON-NLS-1$
  326:                      vectorIconMoyens.add(new ImageIcon(getClass().getResource(
  327:                                  "/Icones/48x48/stock.png"))); //$NON-NLS-1
$
  328:                      vectorIconMoyens.add(new ImageIcon(getClass().getResource(
  329:                                  "/Icones/48x48/stockSelected.png"))); //$N
ON-NLS-1$
  330:                      vectorIconActorsBig = new Vector<ImageIcon>();
  331:                      vectorIconActorsBig.add(new ImageIcon(getClass().getResour
ce(
  332:                                  "/Icones/48x48/acteurHumainSelected.png"))
); //$NON-NLS-1$
  333:                      vectorIconActorsBig.add(new ImageIcon(getClass().getResour
ce(
  334:                                  "/Icones/48x48/acteurHumainNoir.png"))); /
/$NON-NLS-1$
  335:                      vectorIconActorsBig.add(new ImageIcon(getClass().getResour
ce(
  336:                                  "/Icones/48x48/acteurHumainRouge.png")));
//$NON-NLS-1$
  337:                      vectorIconActorsBig.add(new ImageIcon(getClass().getResour
ce(
  338:                                  "/Icones/48x48/acteurHumainVert.png"))); /
/$NON-NLS-1$
  339:                      vectorIconActorsBig.add(new ImageIcon(getClass().getResour
ce(
  340:                                  "/Icones/48x48/acteurHumainJaune.png")));
//$NON-NLS-1$
  341:                      vectorIconActorsBig.add(new ImageIcon(getClass().getResour
ce(
  342:                                  "/Icones/48x48/acteurHumainViolet.png")));
 //$NON-NLS-1$
  343:                      vectorIconActorsBig.add(new ImageIcon(getClass().getResour
ce(
  344:                                  "/Icones/48x48/acteurHumainMarron.png")));
 //$NON-NLS-1$
  345:
  346:                      // other
  347:                      iconTitleNew = new ImageIcon(getClass().getResource(
  348:                                  "/Icones/big/TitreNouveau.png")); //$NON-N
LS-1$
  349:                      iconTitleDataPack = new ImageIcon(getClass().getResource(
  350:                                  "/Icones/big/TitreDataPack.png")); //$NON-
NLS-1$
  351:                      iconTitleNewBrick = new ImageIcon(getClass().getResource(
  352:                                  "/Icones/big/TitreNouvelleBrique.png")); /
/$NON-NLS-1$
  353:                      iconTitleModel = new ImageIcon(getClass().getResource(
  354:                                  "/Icones/big/TitreModel.png")); //$NON-NLS
-1$
  355:                      iconTitleTriad = new ImageIcon(getClass().getResource(
  356:                                  "/Icones/big/TitreNouveauTriade.png")); //
$NON-NLS-1$
  357:                      iconActorSet = new ImageIcon(getClass().getResource(
  358:                                  "/Icones/big/TitreSelectionActeurs.png"));
 //$NON-NLS-1$
  359:                      iconAssistant = new ImageIcon(getClass().getResource(
  360:                                  "/Icones/big/TitreAssistant.png")); //$NON
-NLS-1$
  361:                      iconCorpsBases = new ImageIcon(getClass().getResource(
  362:                                  "/Icones/big/corps_bases.png")); //$NON-NL
S-1$
  363:
  364:                      // Exported icons
  365:                      vectorExportedIcons = new Vector<ImageIcon>();
  366:                      vectorExportedIcons.addAll(vectorIconActivities);
  367:                      vectorExportedIcons.addAll(vectorIconActorsMin);
  368:                      vectorExportedIcons.addAll(vectorIconMoyens);
  369:
  370:                      icons = new HashMap<String, ImageIcon>();
  371:              }
  372:      }
  373:
  374:      public static Icon getIcon(String imagePath) {
  375:              ImageIcon result = icons.get(imagePath);
  376:
  377:              if(result == null) {
  378:                      result = new ImageIcon(imagePath);
  379:
  380:                      if(result.getIconHeight() == 0) {
  381:                              return null;
  382:                      }
  383:
  384:                      icons.put(imagePath, result);
  385:              }
  386:
  387:              return result;
  388:      }
  389:
  390: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.awt.BorderLayout;
    4: import java.awt.event.ActionEvent;
    5: import java.awt.event.ActionListener;
    6: import java.awt.event.MouseEvent;
    7: import java.awt.event.MouseListener;
    8: import java.util.Vector;
    9:
   10: import javax.swing.Box;
   11: import javax.swing.BoxLayout;
   12: import javax.swing.JButton;
   13: import javax.swing.JLabel;
   14: import javax.swing.JPanel;
   15:
   16: import models.Brick;
   17: import models.BrickView;
   18:
   19: public class SchemaSelectionView extends JPanel {
   20:
   21:         private static final long serialVersionUID = -921539312712929027L;
   22:
   23:         private Brick brick;
   24:         private int index;
   25:
   26:         public SchemaSelectionView(final MainFrameTriades mf,
   27:                         final Vector<BrickView> vect, final Vector<Brick> vectSche
ma) {
   28:                 brick = vectSchema.firstElement();
   29:                 index = 0;
   30:                 setLayout(new BorderLayout());
   31:
   32:                 JPanel jpNorth = new JPanel();
   33:                 JPanel jpSouth = new JPanel();
   34:                 final JLabel jlLeftArrow = new JLabel(IconDatabase.iconLeftArrowMi
n);
   35:                 final JLabel jlRightArrow = new JLabel(IconDatabase.iconRightArrow
Min);
   36:                 final JLabel jlTitle = new JLabel();
   37:                 JButton jbSelection = new JButton("SÃ©lectionner");
   38:
   39:                 // Nord
   40:                 jpNorth.setLayout(new BoxLayout(jpNorth, BoxLayout.X_AXIS));
   41:                 jpNorth.add(Box.createHorizontalGlue());
   42:                 jlTitle.setText("SchÃ©ma nÂ°" + (index + 1));
   43:                 jpNorth.add(jlTitle);
   44:                 jpNorth.add(Box.createHorizontalGlue());
   45:                 add(jpNorth, BorderLayout.NORTH);
   46:
   47:                 // Ouest
   48:                 jlLeftArrow.addMouseListener(new MouseListener() {
   49:
   50:                         @Override
   51:                         public void mouseClicked(MouseEvent e) {
   52:                                 if (index > 0) {
   53:                                         remove(vect.elementAt(index));
   54:                                         index--;
   55:                                         jlTitle.setText("SchÃ©ma nÂ°" + (index + 1
));
   56:                                         add(vect.elementAt(index));
   57:                                 } else
   58:                                         jlTitle.setText("SchÃ©ma nÂ°" + (index + 1
) + " (premier)");
   59:                         }
   60:
   61:                         @Override
   62:                         public void mouseEntered(MouseEvent e) {
   63:                                 jlLeftArrow.setIcon(IconDatabase.iconLeftArrowMax)
;
   64:                         }
   65:
   66:                         @Override
   67:                         public void mouseExited(MouseEvent e) {
   68:                                 jlLeftArrow.setIcon(IconDatabase.iconLeftArrowMin)
;
   69:                         }
   70:
   71:                         @Override
   72:                         public void mousePressed(MouseEvent e) {
   73:
   74:                         }
   75:
   76:                         @Override
   77:                         public void mouseReleased(MouseEvent e) {
   78:
   79:                         }
   80:                 });
   81:                 add(jlLeftArrow, BorderLayout.WEST);
   82:
   83:                 // Est
   84:                 jlRightArrow.addMouseListener(new MouseListener() {
   85:
   86:                         @Override
   87:                         public void mouseClicked(MouseEvent e) {
   88:                                 if (index < vect.size() - 1) {
   89:                                         remove(vect.elementAt(index));
   90:                                         index++;
   91:                                         jlTitle.setText("SchÃ©ma nÂ°" + (index + 1
));
   92:                                         add(vect.elementAt(index));
   93:                                 } else
   94:                                         jlTitle.setText("SchÃ©ma nÂ°" + (index + 1
) + " (dernier)");
   95:                         }
   96:
   97:                         @Override
   98:                         public void mouseEntered(MouseEvent e) {
   99:                                 jlRightArrow.setIcon(IconDatabase.iconRightArrowMa
x);
  100:                         }
  101:
  102:                         @Override
  103:                         public void mouseExited(MouseEvent e) {
  104:                                 jlRightArrow.setIcon(IconDatabase.iconRightArrowMi
n);
  105:                         }
  106:
  107:                         @Override
  108:                         public void mousePressed(MouseEvent e) {
  109:
  110:                         }
  111:
  112:                         @Override
  113:                         public void mouseReleased(MouseEvent e) {
  114:
  115:                         }
  116:                 });
  117:                 add(jlRightArrow, BorderLayout.EAST);
  118:
  119:                 // Centre
  120:                 add(vect.elementAt(index));
  121:
  122:                 // Sud
  123:                 jpSouth.setLayout(new BoxLayout(jpSouth, BoxLayout.X_AXIS));
```

```
  124:                 jpSouth.add(Box.createHorizontalGlue());
  125:                 jbSelection.addActionListener(new ActionListener() {
  126:                         @Override
  127:                         public void actionPerformed(ActionEvent e) {
  128:                                 mf. = vectSchema.elementAt(index);
  129:                                 mf.tabbedPane.removeAll();
  130:                                 mf.tabbedPane.add("SchÃ©ma sÃ©lectionnÃ©", mf.myPa
nel(vect
  131:                                                 .elementAt(index)));
  132:                         }
  133:                 });
  134:                 jpSouth.add(jbSelection);
  135:                 jpSouth.add(Box.createHorizontalGlue());
  136:                 add(jpSouth, BorderLayout.SOUTH);
  137:         }
  138:
  139:         public void setSchema(Schema schema) {
  140:                 this.brick = schema;
  141:         }
  142:
  143:         public Schema getSchema() {
  144:                 return brick;
  145:         }
  146:
  147: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.awt.BorderLayout;
    4:
    5: import javax.swing.JOptionPane;
    6:
    7: import models.BrickEdge;
    8: import models.BrickVertex;
    9: import models.TriadeEditingMousePlugin;
   10: import relations.RelationPossibility;
   11: import translation.Messages;
   12:
   13: public class RelationChooserPopUp
   14:                 extends PopUpView {
   15:
   16:         /**
   17:          *
   18:          */
   19:         private static final long serialVersionUID = 6942148785999726898L;
   20:
   21:         protected RelationChooserView relationChooser;
   22:         protected BrickEdge editedModelEdge;
   23:
   24:         public int showRelationChooserView(BrickEdge edge,
   25:                         TriadeEditingMousePlugin<BrickVertex, BrickEdge> mousePlug
in,
   26:                         String title) {
   27:                 boolean reelle = Program.isTriades();
   28:                 labelTitle.setText(title);
   29:                 if (editedModelEdge == null || !edge.equals(editedModelEdge)) {
   30:                         if (relationChooser != null) {
   31:                                 int choice = DialogHandlerFrame
   32:                                                 .showYesNoCancelDialog(
   33:                                                                 Program.myMainFram
e,
   34:                                                                 Messages.getString
("RelationChooserPopUp.0")); //$NON-NLS-1$
   35:                                 switch (choice) {
   36:                                 case JOptionPane.YES_OPTION:
   37:                                         relationChooser.saveState();
   38:                                 case JOptionPane.NO_OPTION:
   39:                                         removeRelationChooserView(mousePlugin);
   40:                                         break;
   41:                                 default:
   42:                                         return -1;
   43:                                 }
   44:
   45:                                 removeRelationChooserView();
   46:                         }
   47:
   48:                         BrickVertex source = edge.getSource();
   49:                         BrickVertex destination = edge.getDestination();
   50:
   51:
   52:
   53:                         editedModelEdge = edge;
   54:                         if (!reelle)
   55:                                 relationChooser = new RelationChooserView(edge,
   56:                                                 mousePlugin, Program.myMainFrame.d
atapack
   57:                                                                 .getRelations().ge
tRelationPossibility(
   58:                                                                         so
urce.getContent(),
   59:                                                                         de
stination.getContent(),
   60:                                                                         fa
lse),
   61:                                                                 RelationPossibility.RELATIONSTRUCT
URELLE, this);
   62:                         else
   63:                                 relationChooser = new RelationChooserView(edge,
   64:                                                 mousePlugin, Program.myMainFrame.d
atapack
   65: .getRelations()
   66:                                                                 .getRelationPossib
ility(source.getContent(),
   67:                                                                         de
stination.getContent(),
   68:                                                                         tr
ue),
   69:                                                                 RelationPossibility.RELATIONREELLE
, this);
   70:
   71:                         contentPane.add(relationChooser, BorderLayout.CENTER);
   72:                         panelMax.setVisible(true);
   73:                         this.validate();
   74:                         return 0;
   75:                 }
   76:                 return -1;
   77:         }
   78:
   79:         protected void removeRelationChooserView() {
   80:                 removeRelationChooserView(null);
   81:         }
   82:
   83:         protected void removeRelationChooserView(
   84:                         TriadeEditingMousePlugin<BrickVertex, BrickEdge> mousePlug
in) {
   85:                 removeRelationChooserView(mousePlugin, editedModelEdge);
   86:         }
   87:
   88:         protected void removeRelationChooserView(
   89:                         TriadeEditingMousePlugin<BrickVertex, BrickEdge> mousePlug
in,
   90:                         BrickEdge modelEdge) {
   91:                 if (relationChooser != null && editedModelEdge == modelEdge) {
   92:                         contentPane.remove(relationChooser);
   93:                         if (mousePlugin != null
   94:                                         && editedModelEdge.getCompleteRelation().i
sEmpty(
   95:                                                                         RelationPossibility.RELATI
ONSTRUCTURELLE)) {
   96:                                 mousePlugin.getEditedAbstractSchema().removeModelE
dge(
   97:                                                                         editedModelEdge);
   98:                         }
   99:                         relationChooser = null;
  100:                         editedModelEdge = null;
  101:                 }
  102:                 panelMax.setVisible(false);
  103:                 panelMin.setVisible(false);
  104:         }
  105:
  106: }
```

```java
package graphicalUserInterface;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSeparator;
import javax.swing.JToolBar;

import translation.Messages;

public class PopUpHelpView extends JPanel {

	private static final long serialVersionUID = 980015872109265231L;

	private JPanel panelMin;
	private JPanel panelMax;

	public PopUpHelpView() {
		setLayout(new BorderLayout());
		setBorder(BorderFactory.createLineBorder(Color.gray));
		build();
	}

	private void build() {
		// DÃ©clarations + initialisations
		panelMax = new JPanel(new BorderLayout());
		panelMin = new JPanel();
		JPanel pTitle = new JPanel();
		BoxLayout layoutTitle = new BoxLayout(pTitle, BoxLayout.X_AXIS);
		BoxLayout layoutMin = new BoxLayout(panelMin, BoxLayout.Y_AXIS);
		JToolBar jtbMax = new JToolBar(JToolBar.HORIZONTAL);
		jtbMax.setFloatable(false);
		JToolBar jtbMin = new JToolBar(JToolBar.HORIZONTAL);
		jtbMin.setFloatable(false);
		JButton jbMinimize = new JButton(IconDatabase.iconArrowDownRight);
		jbMinimize.setFocusable(false);
		JButton jbMaximize = new JButton(IconDatabase.iconArrowUpLeft);
		jbMaximize.setFocusable(false);

		// Construction barre de titre max
		pTitle.setLayout(layoutTitle);
		pTitle.setBackground(Color.LIGHT_GRAY);
		pTitle.add(Box.createHorizontalStrut(3));
		pTitle.add(new JLabel(IconDatabase.iconHelp));
		pTitle.add(Box.createHorizontalStrut(5));
		pTitle.add(new JLabel(Messages.getString("PopUpHelpView.0"))); //$
NON-NLS-1$
		pTitle.add(Box.createHorizontalStrut(5));
		jtbMax.setFloatable(false);
		jbMinimize.setBackground(Color.LIGHT_GRAY);
		jbMinimize.setFocusable(false);
		jbMinimize.setToolTipText(Messages.getString("PopUpHelpView.1"));
//$NON-NLS-1$
		jbMinimize.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				panelMin.setVisible(true);
				panelMax.setVisible(false);
			}

		});
		jtbMax.add(jbMinimize);
		jtbMax.setBackground(Color.LIGHT_GRAY);
		pTitle.add(jtbMax);

		// Construction panel max
		panelMax.add(pTitle, BorderLayout.NORTH);
		panelMax.setVisible(false);

		// Construction panel min
		panelMin.setVisible(true);
		jtbMin.add(Box.createHorizontalStrut(2));
		jtbMin.add(new JLabel(IconDatabase.iconHelp));
		jbMaximize.setBackground(Color.LIGHT_GRAY);
		jbMaximize.setFocusable(false);
		jbMaximize.setToolTipText(Messages.getString("PopUpHelpView.2"));
//$NON-NLS-1$
		jbMaximize.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				panelMin.setVisible(false);
				panelMax.setVisible(true);
			}

		});
		jtbMin.add(jbMaximize);
		jtbMin.setBackground(Color.LIGHT_GRAY);
		panelMin.setLayout(layoutMin);
		panelMin.add(Box.createGlue());
		panelMin.add(jtbMin);

		// Construction panel total
		add(panelMax, BorderLayout.CENTER);
		add(panelMin, BorderLayout.EAST);
	}

	public void setView(JPanel panel) {
		panelMax.add(panel, BorderLayout.CENTER);
	}

	public void setBig(boolean b) {
		panelMax.setVisible(b);
		panelMin.setVisible(!b);
	}

	public boolean isBig() {
		return panelMax.isVisible();
	}

	public static JPanel showText(String title, String message) {
		// DÃ©clarations et initialisations
		JPanel panel = new JPanel();
		String[] strSet = message.split("\n"); //$NON-NLS-1$

		// Layout + border
		panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
		if (title != null)
			panel.setBorder(BorderFactory.createTitledBorder(title));
		else
			panel.setBorder(BorderFactory.createCompoundBorder(BorderF
actory
.createLineBorder(panel.getBackground(), 2
```

```
), BorderFactory
  131:                                        .createCompoundBorder(BorderFactory
  132:                                                .createLineBorder(new Colo
r(169, 211, 247)),
  133:                                                BorderFactory.createEmptyB
order(3, 4, 3, 4))));
  134:
  135:                // Construction panel
  136:                for (int i = 0; i < strSet.length; i++) {
  137:                        panel.add(new JLabel(strSet[i]));
  138:                }
  139:
  140:                return panel;
  141:        }
  142:
  143:        public static JPanel showDataPackHelp() {
  144:                // Declarations et initialisations
  145:                JPanel panel = new JPanel();
  146:                panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
  147:                JPanel pAddActors = getQuestionAnswerPanel(
  148:                                Messages.getString("PopUpHelpView.4"), //$NON-NLS-
1$
  149:                                Messages.getString("PopUpHelpView.5") //$NON-NLS-1
$
  150:                                        + Messages.getString("PopUpHelpVie
w.6") //$NON-NLS-1$
  151:                                        + Messages.getString("PopUpHelpVie
w.7") //$NON-NLS-1$
  152:                                        + Messages.getString("PopUpHelpVie
w.8")); //$NON-NLS-1$
  153:                JPanel pDeleteActors = getQuestionAnswerPanel(
  154:                                Messages.getString("PopUpHelpView.9"), //$NON-NLS-
1$
  155:                                Messages.getString("PopUpHelpView.10") //$NON-NLS-
1$
  156:                                        + Messages.getString("PopUpHelpVie
w.11") //$NON-NLS-1$
  157:                                        + Messages.getString("PopUpHelpVie
w.12") //$NON-NLS-1$
  158:                                        + Messages.getString("PopUpHelpVie
w.13") + Messages.getString("PopUpHelpView.14")); //$NON-NLS-1$ //$NON-NLS-2$
  159:                JPanel pAddGroups = getQuestionAnswerPanel(
  160:                                Messages.getString("PopUpHelpView.15"), //$NON-NLS
-1$
  161:                                Messages.getString("PopUpHelpView.16") //$NON-NLS-
1$
  162:                                        + Messages.getString("PopUpHelpVie
w.17")); //$NON-NLS-1$
  163:                JPanel pDeleteGroups = getQuestionAnswerPanel(
  164:                                Messages.getString("PopUpHelpView.18"), //$NON-NLS
-1$
  165:                                Messages.getString("PopUpHelpView.19") //$NON-NLS-
1$
  166:                                        + Messages.getString("PopUpHelpVie
w.20") //$NON-NLS-1$
  167:                                        + Messages.getString("PopUpHelpVie
w.21") //$NON-NLS-1$
  168:                                        + Messages.getString("PopUpHelpVie
w.22") + Messages.getString("PopUpHelpView.23")); //$NON-NLS-1$ //$NON-NLS-2$
  169:                JPanel pAddActivities = getQuestionAnswerPanel(
  170:                                Messages.getString("PopUpHelpView.24"), //$NON-NLS
-1$
  171:                                Messages.getString("PopUpHelpView.25") //$NON-NLS-
1$
  172:                                        + Messages.getString("PopUpHelpVie
w.26") //$NON-NLS-1$
  173:                                        + Messages.getString("PopUpHelpVie
w.27")); //$NON-NLS-1$
  174:                JPanel pAddBrickTypes = getQuestionAnswerPanel(
  175:                                Messages.getString("PopUpHelpView.28"), //$NON-NLS
-1$
  176:                                Messages.getString("PopUpHelpView.29") //$NON-NLS-
1$
  177:                                        + Messages.getString("PopUpHelpVie
w.30")); //$NON-NLS-1$
  178:                JPanel pAddActionTime = getQuestionAnswerPanel(
  179:                                Messages.getString("PopUpHelpView.31"), //$NON-NLS
-1$
  180:                                Messages.getString("PopUpHelpView.32") //$NON-NLS-
1$
  181:                                        + Messages.getString("PopUpHelpVie
w.33")); //$NON-NLS-1$
  182:                JPanel pBricksAndActivities = getQuestionAnswerPanel(
  183:                                Messages.getString("PopUpHelpView.34"), //$NON-NLS
-1$
  184:                                Messages.getString("PopUpHelpView.35") //$NON-NLS-
1$
  185:                                        + Messages.getString("PopUpHelpVie
w.36") //$NON-NLS-1$
  186:                                        + Messages.getString("PopUpHelpVie
w.37")); //$NON-NLS-1$
  187:
  188:                // Ajouts au panel
  189:                panel.add(getTitle());
  190:                panel.add(pAddActors);
  191:                panel.add(pDeleteActors);
  192:                panel.add(pAddGroups);
  193:                panel.add(pDeleteGroups);
  194:                panel.add(pAddActivities);
  195:                panel.add(pAddBrickTypes);
  196:                panel.add(pAddActionTime);
  197:                panel.add(pBricksAndActivities);
  198:
  199:                return panel;
  200:        }
  201:
  202:        public static JPanel showBrickHelp() {
  203:                // Declarations et initialisations
  204:                JPanel panel = new JPanel();
  205:                panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
  206:                JPanel pAddActors = getQuestionAnswerPanel(
  207:                                Messages.getString("PopUpHelpView.38"), //$NON-NLS
-1$
  208:                                Messages.getString("PopUpHelpView.39") //$NON-NLS-
1$
  209:                                        + Messages.getString("PopUpHelpVie
w.40")); //$NON-NLS-1$
  210:                JPanel pAddRelation = getQuestionAnswerPanel(
  211:                                Messages.getString("PopUpHelpView.41"), //$NON-NLS
-1$
  212:                                Messages.getString("PopUpHelpView.42") //$NON-NLS-
1$
  213:                                        + Messages.getString("PopUpHelpVie
w.43") //$NON-NLS-1$
  214:                                        + Messages.getString("PopUpHelpVie
w.44") //$NON-NLS-1$
  215:                                        + Messages.getString("PopUpHelpVie
w.45") //$NON-NLS-1$
  216:                                        + Messages.getString("PopUpHelpVie
w.46") //$NON-NLS-1$
  217:                                        + Messages.getString("PopUpHelpVie
w.47") //$NON-NLS-1$
  218:                                        + Messages.getString("PopUpHelpVie
w.48")); //$NON-NLS-1$
```

```
  219:                 JPanel pEditRelation = getQuestionAnswerPanel(
  220:                                 Messages.getString("PopUpHelpView.49"), //$NON-NLS
-1$
  221:                                 Messages.getString("PopUpHelpView.50") //$NON-NLS-
1$
  222:                                                 + Messages.getString("PopUpHelpVie
w.51") //$NON-NLS-1$
  223:                                                 + Messages.getString("PopUpHelpVie
w.52") //$NON-NLS-1$
  224:                                                 + Messages.getString("PopUpHelpVie
w.53") //$NON-NLS-1$
  225:                                                 + Messages.getString("PopUpHelpVie
w.54") //$NON-NLS-1$
  226:                                                 + Messages.getString("PopUpHelpVie
w.55")); //$NON-NLS-1$
  227:
  228:                 // Ajouts au panel
  229:                 panel.add(pAddActors);
  230:                 panel.add(pAddRelation);
  231:                 panel.add(pEditRelation);
  232:
  233:                 return panel;
  234:         }
  235:
  236:         public static JPanel showModelHelp() {
  237:                 // Declarations et initialisations
  238:                 JPanel panel = new JPanel();
  239:                 panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
  240:                 JPanel pAddActors = getQuestionAnswerPanel(
  241:                                 Messages.getString("PopUpHelpView.56"), //$NON-NLS
-1$
  242:                                 Messages.getString("PopUpHelpView.57") //$NON-NLS-
1$
  243:                                                 + Messages.getString("PopUpHelpVie
w.58")); //$NON-NLS-1$
  244:                 JPanel pAddRelation = getQuestionAnswerPanel(
  245:                                 Messages.getString("PopUpHelpView.59"), //$NON-NLS
-1$
  246:                                 Messages.getString("PopUpHelpView.60") //$NON-NLS-
1$
  247:                                                 + Messages.getString("PopUpHelpVie
w.61") //$NON-NLS-1$
  248:                                                 + Messages.getString("PopUpHelpVie
w.62") //$NON-NLS-1$
  249:                                                 + Messages.getString("PopUpHelpVie
w.63") //$NON-NLS-1$
  250:                                                 + Messages.getString("PopUpHelpVie
w.64") //$NON-NLS-1$
  251:                                                 + Messages.getString("PopUpHelpVie
w.65") //$NON-NLS-1$
  252:                                                 + Messages.getString("PopUpHelpVie
w.66")); //$NON-NLS-1$
  253:                 JPanel pEditRelation = getQuestionAnswerPanel(
  254:                                 Messages.getString("PopUpHelpView.67"), //$NON-NLS
-1$
  255:                                 Messages.getString("PopUpHelpView.68") //$NON-NLS-
1$
  256:                                                 + Messages.getString("PopUpHelpVie
w.69") //$NON-NLS-1$
  257:                                                 + Messages.getString("PopUpHelpVie
w.70") //$NON-NLS-1$
  258:                                                 + Messages.getString("PopUpHelpVie
w.71") //$NON-NLS-1$
  259:                                                 + Messages.getString("PopUpHelpVie
w.72") //$NON-NLS-1$
  260:                                                 + Messages.getString("PopUpHelpVie
w.73")); //$NON-NLS-1$
  261:
  262:                 // Ajouts au panel
  263:                 panel.add(pAddActors);
  264:                 panel.add(pAddRelation);
  265:                 panel.add(pEditRelation);
  266:
  267:                 return panel;
  268:         }
  269:
  270:         public static JPanel showSelectionActorsHelp() {
  271:                 // Declarations et initialisations
  272:                 JPanel panel = new JPanel();
  273:                 panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
  274:                 JPanel pAddActors = getQuestionAnswerPanel(
  275:                                 Messages.getString("PopUpHelpView.74"), //$NON-NLS
-1$
  276:                                 Messages.getString("PopUpHelpView.75") //$NON-NLS-
1$
  277:                                                 + Messages.getString("PopUpHelpVie
w.76")); //$NON-NLS-1$
  278:                 JPanel pAddGroup = getQuestionAnswerPanel(
  279:                                 Messages.getString("PopUpHelpView.77"), //$NON-NLS
-1$
  280:                                 Messages.getString("PopUpHelpView.78") //$NON-NLS-
1$
  281:                                                 + Messages.getString("PopUpHelpVie
w.79")); //$NON-NLS-1$
  282:
  283:                 // Ajouts au panel
  284:                 panel.add(pAddActors);
  285:                 panel.add(pAddGroup);
  286:
  287:                 return panel;
  288:         }
  289:
  290:         public static JPanel showSchemaGlobalHelp() {
  291:                 // Declarations et initialisations
  292:                 JPanel panel = new JPanel();
  293:                 panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
  294:                 JPanel pEditRelation = getQuestionAnswerPanel(
  295:                                 Messages.getString("PopUpHelpView.80"), //$NON-NLS
-1$
  296:                                 Messages.getString("PopUpHelpView.81") //$NON-NLS-
1$
  297:                                                 + Messages.getString("PopUpHelpVie
w.82") //$NON-NLS-1$
  298:                                                 + Messages.getString("PopUpHelpVie
w.83") //$NON-NLS-1$
  299:                                                 + Messages.getString("PopUpHelpVie
w.84") //$NON-NLS-1$
  300:                                                 + Messages.getString("PopUpHelpVie
w.85") //$NON-NLS-1$
  301:                                                 + Messages.getString("PopUpHelpVie
w.86")); //$NON-NLS-1$
  302:                 JPanel pMove = getQuestionAnswerPanel(
  303:                                 Messages.getString("PopUpHelpView.87"), //$NON-NLS
-1$
  304:                                 Messages.getString("PopUpHelpView.88")); //$NON-NL
S-1$
  305:                 JPanel pMoveElement = getQuestionAnswerPanel(
  306:                                 Messages.getString("PopUpHelpView.89"), //$NON-NLS
-1$
  307:                                 Messages.getString("PopUpHelpView.90")); //$NON-NL
S-1$
  308:
  309:                 // Ajouts au panel
  310:                 panel.add(pEditRelation);
```

```
311:                 panel.add(pMove);
312:                 panel.add(pMoveElement);
313:
314:                 return panel;
315:         }
316:
317:         private static JPanel getQuestionAnswerPanel(String strQuestion,
318:                         String strAnswer) {
319:                 // Declarations et initialisations
320:                 final JPanel panel = new JPanel(new BorderLayout());
321:                 final JPanel pQuestion = new JPanel();
322:                 final JPanel pAnswers = new JPanel();
323:                 final JLabel icon = new JLabel(IconDatabase.iconMaximize);
324:                 JLabel question = new JLabel(strQuestion);
325:                 String[] strSet = strAnswer.split("\n"); //$NON-NLS-1$
326:
327:                 pQuestion.setLayout(new BoxLayout(pQuestion, BoxLayout.X_AXIS));
328:                 pAnswers.setLayout(new BoxLayout(pAnswers, BoxLayout.Y_AXIS));
329:                 pAnswers.setVisible(false);
330:                 pAnswers.setBackground(Color.WHITE);
331:                 pAnswers.setBorder(BorderFactory.createCompoundBorder(BorderFactor
y
332:                                 .createLineBorder(panel.getBackground(), 2), Borde
rFactory
333:                                 .createCompoundBorder(BorderFactory.createLineBord
er(new Color(
334:                                                 169, 211, 247)), BorderFactory.cre
ateEmptyBorder(3, 4,
335:                                                 3, 4))));
336:
337:                 // Question
338:                 pQuestion.add(Box.createHorizontalStrut(3));
339:                 pQuestion.add(icon);
340:                 pQuestion.add(Box.createHorizontalStrut(3));
341:                 pQuestion.add(question);
342:                 pQuestion.addMouseListener(new MouseListener() {
343:                         @Override
344:                         public void mouseClicked(MouseEvent e) {
345:                                 if (pAnswers.isVisible()) {
346:                                         icon.setIcon(IconDatabase.iconMaximize);
347:                                         pAnswers.setVisible(false);
348:                                 } else {
349:                                         icon.setIcon(IconDatabase.iconUnfold);
350:                                         pAnswers.setVisible(true);
351:                                 }
352:                                 pQuestion.setBackground(panel.getBackground());
353:                         }
354:
355:                         @Override
356:                         public void mouseEntered(MouseEvent e) {
357:                                 pQuestion.setBackground(new Color(169, 211, 247));
358:
359:                         }
360:
361:                         @Override
362:                         public void mouseExited(MouseEvent e) {
363:                                 pQuestion.setBackground(panel.getBackground());
364:                         }
365:
366:                         @Override
367:                         public void mousePressed(MouseEvent e) {
368:                         }
369:
370:                         @Override
371:                         public void mouseReleased(MouseEvent e) {
372:                         }
373:
374:                 });
375:
376:                 // Answers
377:                 for (int i = 0; i < strSet.length; i++) {
378:                         pAnswers.add(new JLabel(strSet[i]));
379:                 }
380:
381:                 // Total
382:                 panel.add(pQuestion, BorderLayout.NORTH);
383:                 panel.add(pAnswers, BorderLayout.CENTER);
384:
385:                 return panel;
386:         }
387:
388:         private static JPanel getTitle() {
389:                 JPanel pTitle = new JPanel(new BorderLayout());
390:                 JLabel labelTitle = new JLabel(
391:                                 Messages.getString("PopUpHelpView.92")); //$NON-NL
S-1$
392:                 labelTitle.setBorder(BorderFactory.createEmptyBorder(2, 2, 2, 2));
393:
394:                 pTitle.setBackground(Color.white);
395:                 pTitle.add(
396:                                 new JLabel(Messages.getString("PopUpHelpView.93"))
, //$NON-NLS-1$
397:                                 BorderLayout.CENTER);
398:                 pTitle.add(new JSeparator(), BorderLayout.SOUTH);
399:
400:                 return pTitle;
401:         }
402: }
```

```java
package graphicalUserInterface;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JPanel;
import javax.swing.JTextField;

import translation.Messages;

import dataPack.Activite;

import models.Brick;

public class BrickParameterPopUp extends JPanel {

	/**
	 *
	 */
	private static final long serialVersionUID = -1782702240360015741L;

	protected Brick brick;
	protected JTextField name;
	protected JComboBox step;
	protected JComboBox activity;
	protected JPanel mainPanel;

	public BrickParameterPopUp(Brick theBrick)
	{
		brick = theBrick;
		buildPanel(this);
		setVisible(false);
	}
	private void buildPanel(JPanel result)
	{
		final BrickParameterPopUp access = this;

		result.setLayout(new BorderLayout());
		result.setBorder(BorderFactory.createRaisedBevelBorder());

		JPanel fieldPanel = new JPanel();
		fieldPanel.setLayout(new BoxLayout(fieldPanel,BoxLayout.Y_AXIS));
		JPanel buttonPanel = new JPanel();
		buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.X_AXIS));




		name = new JTextField(35);
		if (brick != null)
			name.setText(brick.getName());
		name.setBorder(BorderFactory.createTitledBorder(Messages.getString("BrickParameterPopUp.0"))); //$NON-NLS-1$
		//name.setAlignmentX(LEFT_ALIGNMENT);
		name.addFocusListener(new FocusListener() {

			@Override
			public void focusLost(FocusEvent e) {
				if (name.getText().trim().length() == 0)

					name.setText(brick.getName());
			}

			@Override
			public void focusGained(FocusEvent e) {

			}
		});

		step = new JComboBox(Program.myMainFrame.getDataPack().getSteps());
		if (brick != null)
			step.setSelectedItem(brick.getStep());
		step.setBorder(BorderFactory.createTitledBorder(Messages.getString("BrickParameterPopUp.1"))); //$NON-NLS-1$
		//	step.setAlignmentX(LEFT_ALIGNMENT);
		activity = new JComboBox(Program.myMainFrame.getDataPack().getActivities().getActivities());
		activity.setBorder(BorderFactory.createTitledBorder(Messages.getString("BrickParameterPopUp.2"))); //$NON-NLS-1$


		JPanel tempNamePanel = new JPanel();
		tempNamePanel.setLayout(new FlowLayout());
		tempNamePanel.add(name);


		fieldPanel.add(Box.createVerticalGlue());
		fieldPanel.add(tempNamePanel);
		fieldPanel.add(Box.createRigidArea(new Dimension(250,5)));
		fieldPanel.add(step);
		fieldPanel.add(activity);
		fieldPanel.add(Box.createGlue());

		JButton ok = new JButton(Messages.getString("BrickParameterPopUp.3")); //$NON-NLS-1$
		JButton cancel = new JButton(Messages.getString("BrickParameterPopUp.4")); //$NON-NLS-1$

		ok.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				boolean change = false;
				String oldName = brick.toString();
				String newTitle = name.getText().trim();
				if (newTitle.length() > 0)
				{
					if (!newTitle.equals(brick.getName()))
					{
						brick.setName(newTitle);
						change = true;
					}
				}
				else
				{
					name.setText(brick.getName());

				}

				if (!step.getSelectedItem().toString().equals(brick.getStep()))
				{
```

```
  126:                                        brick.setStep(step.getSelectedItem().toStr
ing());
  127:                                        change = true;
  128:                                }
  129:
  130:                                if (!activity.getSelectedItem().equals(brick.getAc
tivity()))
  131:                                {
  132:                                        brick.switchActivity(brick.getActivity(),
(Activite) activity.getSelectedItem());
  133:                                        change = true;
  134:                                }
  135:
  136:                                if (change)
  137:                                {
  138:                                        if (Program.myMainFrame.closeTab(oldName))
  139:                                        {
  140:                                                Program.myMainFrame.addTab(brick);
  141:                                                Program.myMainFrame.showMainTab();
  142:                                        }
  143:
  144:                                }
  145:
  146:
  147:                                        access.setVisible(false);
  148:
  149:                        }
  150:                });
  151:
  152:                cancel.addActionListener(new ActionListener() {
  153:
  154:                        @Override
  155:                        public void actionPerformed(ActionEvent e) {
  156:                                access.setVisible(false);
  157:                        }
  158:                });
  159:
  160:                buttonPanel.add(Box.createGlue());
  161:                buttonPanel.add(cancel);
  162:                buttonPanel.add(ok);
  163:                buttonPanel.add(Box.createGlue());
  164:
  165:                result.add(fieldPanel,BorderLayout.CENTER);
  166:                result.add(buttonPanel, BorderLayout.SOUTH);
  167:
  168:
  169:
  170:        }
  171:
  172:        public void showMeTheBrick(Brick theBrick)
  173:        {
  174:                if (step.getItemCount() != theBrick.getDatapack().getSteps().size(
))
  175:                {
  176:                        step.removeAllItems();
  177:                        for (String s : theBrick.getDatapack().getSteps())
  178:                        {
  179:                                step.addItem(s);
  180:                        }
  181:                }
  182:
  183:
  184:
  185:                brick = theBrick;
  186:                name.setText(brick.getName());
  187:                step.setSelectedItem(brick.getStep());
  188:                activity.setSelectedItem(brick.getActivity());
  189:                setVisible(true);
  190:        }
  191:
  192: }
```

```java
package graphicalUserInterface;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.util.Vector;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;

import main.Statut;
import translation.Messages;
import dataPack.ActeurBase;
import dataPack.JeuActeur;

public class ActorSetView extends JPanel {

	private static final long serialVersionUID = -8042407423948049392L;

	private final JeuActeur ja;
	private final JPanel panelTable = new JPanel();
	private final Vector<JPanel> rowsView = new Vector<JPanel>();

	// Utiles pour les FocusListener
	private int indexCol = 0;
	private int indexRow = 0;

	public ActorSetView(JeuActeur ja) {
		super(new BorderLayout());
		this.ja = ja;
		buildAll();
	}

	/**
	 * Construction du panel global
	 */
	private void buildAll() {

		panelTable.setLayout(new BoxLayout(panelTable, BoxLayout.Y_AXIS));

		JPanel firstRow = createFirstRow();
		rowsView.add(firstRow);
		panelTable.add(firstRow);
		// Remplissage si jeu d'acteur non vide
		for (String name : ja.getListeCorps()) {
			addRow(name);
		}

		setBorder(BorderFactory.createTitledBorder(Messages.getString("ActorSetView.0") + ja.toString())); //$NON-NLS-1$

		JPanel panel = new JPanel(new BorderLayout());
		panel.add(panelTable, BorderLayout.WEST);
		panel.add(Box.createGlue(), BorderLayout.CENTER);
		panel.add(Box.createGlue(), BorderLayout.SOUTH);
		JScrollPane jsp = new JScrollPane(panel);
		jsp
				.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
		jsp.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
		add(createButtonsPanel(), BorderLayout.NORTH);
		add(jsp, BorderLayout.CENTER);
	}

	/**
	 * Construction panel 1ere ligne (contenant le nom des bases)
	 */
	private JPanel createFirstRow() {

		JPanel panel = new JPanel();
		panel.setLayout(new BoxLayout(panel, BoxLayout.X_AXIS));

		panel.add(new JLabel(IconDatabase.iconCorpsBases));
		for (ActeurBase ab : ja.getListeBase()) {
			panel.add(createJTFCombo(ab.toString()));
			indexCol++;
		}

		return panel;

	}

	/**
	 * Construction d'un panel contenant uniquement l'entÃªte de ligne
	 * (jtextfield + jcombobox)
	 */
	private JPanel createRowTitle(String name) {

		JPanel panel = new JPanel();
		panel.setLayout(new BoxLayout(panel, BoxLayout.X_AXIS));

		JTextField jtf = createJTF(name);
		jtf.setName("C"); //$NON-NLS-1$
		jtf.addFocusListener(generateFocusListener(jtf, indexRow, null, 0));
		panel.add(jtf);

		return panel;

	}

	/**
	 * Remplissage de la ligne corps de nom "name" pour toutes les bases
	 */
	private void generateRowContent(JPanel rowPanel, String name) {

		Vector<ActeurBase> vect = ja.getCorps(name);
		for (int i = rowPanel.getComponentCount() - 1; i < vect.size(); i++) {

			JTextField jtf = createJTF(vect.elementAt(i).toString());
			jtf.setName("P"); //$NON-NLS-1$
			jtf.addFocusListener(generateFocusListener(jtf, i, null, rowsView
					.indexOf(rowPanel) - 1));
			rowPanel.add(jtf);
		}
		rowPanel.validate();
	}

	/**
	 * Update des produits d'une ligne
	 */
	private void updateRowContent(JPanel rowPanel, String name) {
```

```
129:
130:                 int i = 1;
131:                 for (ActeurBase ab : ja.getCorps(name)) {
132:                         ((JTextField) rowPanel.getComponent(i)).setText(ab.toStrin
g());
133:                         i++;
134:                 }
135:                 rowPanel.validate();
136:         }
137:
138:         /**
139:          * Update des produits de toutes les lignes
140:          */
141:         private void updateAllRowContent() {
142:
143:                 int i = 1;
144:                 for (String name : ja.getListeCorps()) {
145:                         updateRowContent(rowsView.elementAt(i), name);
146:                         i++;
147:                 }
148:
149:         }
150:
151:         /**
152:          * Ajout d'une ligne AVEC REMPLISSAGE AUTOMATIQUE DES PRODUITS (Corps)
153:          */
154:         private void addRow(String rowName) {
155:
156:                 JPanel rowPanel = createRowTitle(rowName);
157:                 rowsView.add(rowPanel);
158:                 if (rowsView.firstElement().getComponentCount() > 1) {
159:                         generateRowContent(rowPanel, rowName);
160:                 }
161:                 panelTable.add(rowPanel);
162:                 validate();
163:                 indexRow++;
164:         }
165:
166:         /**
167:          * Ajout d'une colonne (Base)
168:          */
169:         private void addCol(String colName) {
170:
171:                 rowsView.firstElement().add(createJTFCombo(colName));
172:                 int i = 1;
173:                 for (String s : ja.getListeCorps()) {
174:                         generateRowContent(rowsView.elementAt(i), s);
175:                         i++;
176:                 }
177:                 validate();
178:                 indexCol++;
179:         }
180:
181:         /**
182:          * Fonction de crÃ©ation d'un panel cellule de type jtextfield (Base ou
183:          * Produit)
184:          */
185:         private JTextField createJTF(String name) {
186:
187:                 JTextField jtf = new JTextField(name);
188:                 jtf.setMinimumSize(new Dimension(200, 60));
189:                 jtf.setPreferredSize(new Dimension(200, 60));
190:                 jtf.setMaximumSize(new Dimension(200, 60));
191:
192:                 return jtf;
193:         }
194:
195:         /**
196:          * Fonction de crÃ©ation d'un panel cellule de type jtextfield/jcombobox
197:          * (Corps)
198:          */
199:         private JPanel createJTFCombo(String name) {
200:
201:                 JPanel panel = new JPanel();
202:                 panel.setMinimumSize(new Dimension(200, 60));
203:                 panel.setPreferredSize(new Dimension(200, 60));
204:                 panel.setMaximumSize(new Dimension(200, 60));
205:                 panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
206:                 JTextField jtf = new JTextField(name);
207:                 jtf.setName("B"); //$NON-NLS-1$
208:                 jtf.addFocusListener(generateFocusListener(jtf, indexCol, null, 0)
);
209:                 JComboBox jcb = new JComboBox(Statut.values());
210:                 jcb.addFocusListener(generateFocusListener(jcb, 0, jtf, 0));
211:                 jcb.setSelectedIndex(ja.getListeBase().elementAt(
212:                                 ja.getBaseIndexByName(name)).getIdStatut());
213:                 panel.add(jtf);
214:                 panel.add(jcb);
215:
216:                 return panel;
217:
218:         }
219:
220:         /**
221:          * CrÃ©ation panel contenant les deux boutons pour ajouter lignes et colon
nes
222:          */
223:         private JPanel createButtonsPanel() {
224:
225:                 JPanel panelButtons = new JPanel();
226:                 panelButtons.setLayout(new BoxLayout(panelButtons, BoxLayout.X_AXI
S));
227:                 JButton col = new JButton(Messages.getString("ActorSetView.4")); /
/$NON-NLS-1$
228:                 col.addActionListener(new ActionListener() {
229:                         @Override
230:                         public void actionPerformed(ActionEvent e) {
231:                                 ja.ajouterBase(Messages.getString("ActorSetView.5"
) + (indexCol + 1), new Integer(0)); //$NON-NLS-1$
232:                                 addCol(Messages.getString("ActorSetView.6") + (ind
exCol + 1)); //$NON-NLS-1$
233:                         }
234:                 });
235:                 col.setFocusable(false);
236:                 JButton row = new JButton(Messages.getString("ActorSetView.7")); /
/$NON-NLS-1$
237:                 row.addActionListener(new ActionListener() {
238:                         @Override
239:                         public void actionPerformed(ActionEvent e) {
240:                                 ja.ajouterCorps(Messages.getString("ActorSetView.8
") + (indexRow + 1)); //$NON-NLS-1$
241:                                 addRow(Messages.getString("ActorSetView.9") + (ind
exRow + 1)); //$NON-NLS-1$
242:                         }
243:                 });
244:                 row.setFocusable(false);
245:                 JButton close = new JButton(Messages.getString("ActorSetView.10"))
; //$NON-NLS-1$
246:                 close.addActionListener(Program.myMainFrame.controller.actionRemov
eTab);
247:                 close.setFocusable(false);
248:                 JButton apply = new JButton(Messages.getString("ActorSetView.11"))
; //$NON-NLS-1$
249:                 panelButtons.add(col);
```

```
  250:                 panelButtons.add(row);
  251:                 panelButtons.add(Box.createHorizontalGlue());
  252:                 panelButtons.add(apply);
  253:                 panelButtons.add(close);
  254:
  255:                 return panelButtons;
  256:         }
  257:
  258:         /**
  259:          * GÃ©nÃ©ration d'un FocusListener Ã  destination de (au choix) - Cellule
Base
  260:          * "B" - Cellule Corps "C" - Cellule Produit "P"
  261:          */
  262:         private FocusListener generateFocusListener(final Object o,
  263:                         final int index, final JTextField aux, final int indexBis)
 {
  264:
  265:                 return new FocusListener() {
  266:
  267:                         String temp;
  268:
  269:                         @Override
  270:                         public void focusLost(FocusEvent e) {
  271:                                 if (o.getClass() == JTextField.class) {
  272:                                         JTextField jtf = (JTextField) o;
  273:                                         if (!jtf.getText().equals(temp)) {
  274:
  275:                                                 String name = jtf.getName();
  276:                                                 if (name.equals("B")) { // Base //
$NON-NLS-1$
  277:                                                         if (ja.getBaseIndexByName(
jtf.getText()) != -1) {
  278:                                                                 DialogHandlerFrame
  279:                                                                                 .s
howErrorDialog(Messages.getString("ActorSetView.13")); //$NON-NLS-1$
  280:                                                                 jtf.setText(temp);
  281:                                                                 return;
  282:                                                         }
  283:                                                         if (jtf.getText().trim().i
sEmpty()) {
  284:                                                                 DialogHandlerFrame
  285:                                                                                 .s
howErrorDialog(Messages.getString("ActorSetView.14")); //$NON-NLS-1$
  286:                                                                 jtf.setText(temp);
  287:                                                                 return;
  288:                                                         }
  289:
  290:                                                         ja.getListeBase().elementA
t(index).setPoste(
  291:                                                                         jtf.getTex
t());
  292:                                                         ja.computeName();
  293:                                                         updateAllRowContent();
  294:                                                 } else {
  295:                                                         if (name.equals("C")) { //
 Corps //$NON-NLS-1$
  296:                                                                 if (ja.getListeCor
ps().indexOf(jtf.getText()) != -1) {
  297:                                                                         DialogHand
lerFrame
  298:
.showErrorDialog(Messages.getString("ActorSetView.16")); //$NON-NLS-1$
  299:                                                                         jtf.setTex
t(temp);
  300:                                                                         return;
  301:                                                                 }
  302:                                                                 if (jtf.getText().
trim().isEmpty()) {
  303:                                                                         DialogHand
lerFrame
  304:
.showErrorDialog(Messages.getString("ActorSetView.17")); //$NON-NLS-1$
  305:                                                                         jtf.setTex
t(temp);
  306:                                                                         return;
  307:                                                                 }
  308:
  309:                                                                 ja.modifierCorps(t
emp, jtf.getText());
  310:                                                                 ja.computeName();
  311:                                                                 updateAllRowConten
t();
  312:                                                         } else if (name.equals("P"
)) { // Produit //$NON-NLS-1$
  313:
  314:                                                                 String s = ja.getL
isteCorps().elementAt(
  315:                                                                                 in
dexBis);
  316:                                                                 ja.getCorps(s).ele
mentAt(index).setPoste(
  317:                                                                                 jt
f.getText());
  318:
  319:                                                         }
  320:                                                 }
  321:                                         }
  322:                                 } else if (o.getClass() == JComboBox.class) {
  323:                                         JComboBox jcb = (JComboBox) o;
  324:                                         if (aux != null) {
  325:                                                 Integer id = ((Statut) jcb.getSele
ctedItem()).id;
  326:                                                 System.out.println("Nom corps = "
+ aux.getText() //$NON-NLS-1$
  327:                                                                 + " index statut :
 " + id); //$NON-NLS-1$
  328:                                                 ja.getListeBase().elementAt(
  329:                                                                 ja.getBaseIndexByN
ame(aux.getText()))
  330:                                                                 .setIdStatut(id);
  331:                                         }
  332:                                 }
  333:                         }
  334:
  335:                         @Override
  336:                         public void focusGained(FocusEvent e) {
  337:                                 if (o.getClass() == JTextField.class) {
  338:                                         temp = ((JTextField) o).getText();
  339:                                 }
  340:                                 return;
  341:                         }
  342:                 };
  343:         }
  344:
  345: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.awt.event.ActionEvent;
    4: import java.awt.event.ActionListener;
    5:
    6: import javax.swing.ButtonGroup;
    7: import javax.swing.JPopupMenu;
    8: import javax.swing.JRadioButtonMenuItem;
    9:
   10: import models.Brick;
   11: import models.BrickVertex.VerticeRank;
   12: import models.BrickVertex;
   13:
   14: public class RankContextualMenu extends JPopupMenu {
   15:
   16:         /**
   17:          *
   18:          */
   19:         private static final long serialVersionUID = 920753966653532282L;
   20:
   21:
   22:
   23:         public RankContextualMenu(final BrickVertex bV, final Brick brick)
   24:         {
   25:
   26:                 JRadioButtonMenuItem primary = new JRadioButtonMenuItem("Acteur pr
imaire");
   27:                 primary.addActionListener(new ActionListener(){
   28:
   29:                         @Override
   30:                         public void actionPerformed(ActionEvent e) {
   31:                                 bV.setRank(VerticeRank.primary);
   32:                         }
   33:                 });
   34:
   35:                 JRadioButtonMenuItem secondary = new JRadioButtonMenuItem("Acteur
secondaire");
   36:                 secondary.addActionListener(new ActionListener() {
   37:
   38:                         @Override
   39:                         public void actionPerformed(ActionEvent e) {
   40:                                 bV.setRank(VerticeRank.secondary);
   41:                         }
   42:                 });
   43:
   44:                 JRadioButtonMenuItem remaining = new JRadioButtonMenuItem("Acteur
restant");
   45:                 remaining.addActionListener(new ActionListener() {
   46:
   47:                         @Override
   48:                         public void actionPerformed(ActionEvent e) {
   49:                                 bV.setRank(VerticeRank.remaining);
   50:                         }
   51:                 });
   52:
   53:                 ButtonGroup group = new ButtonGroup();
   54:                 group.add(primary);
   55:                 group.add(secondary);
   56:                 group.add(remaining);
   57:
   58:                 if (bV.getRank() == null)
   59:                         bV.setRank(VerticeRank.remaining);
   60:
   61:
   62:                 switch (bV.getRank())
   63:                 {
   64:                 case primary :
   65:                         primary.setSelected(true);
   66:                         break;
   67:                 case secondary :
   68:                         secondary.setSelected(true);
   69:                         break;
   70:                 case remaining :
   71:                         remaining.setSelected(true);
   72:                         break;
   73:                 }
   74:
   75:                 add(primary);
   76:                 add(secondary);
   77:                 add(remaining);
   78:
   79:
   80:         }
   81:
   82:
   83: }
```

```java
package graphicalUserInterface;

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Vector;

import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import models.Brick;
import models.BrickEdge;
import models.BrickVertex;
import models.BrickView;
import relations.EnsembleRelation;
import relations.RelationBrowser;
import translation.Messages;
import dataPack.AutoSaveCreator;
import dataPack.DataPack;
import dataPack.JTreeActors;

public class MainFrameDatapackCreator extends MainFrame {

	private static final long serialVersionUID = -1230397799638872519L;

	protected DataPackView dataPackView;
	protected Vector<BrickView<BrickVertex,BrickEdge>> modelViewList;
	public OptionNewFrameDatapackCreator myOptionNewFrame;

	public MainFrameDatapackCreator() {
		modelViewList = new Vector<BrickView<BrickVertex,BrickEdge>>();
		setTitle(Messages.getString("MainFrameDatapackCreator.0")); //$NON-NLS-1$
	}

	@Override
	protected JPanel myPanel(Object object, JPanel mainPanel) {
		JTreeActors jta = null;
		PopUpView popUp = new RelationChooserPopUp();
		JPanel pSchema = new JPanel(new BorderLayout());

		if (object == null) {
			if (datapack != null) {
				jta = new JTreeActors(datapack);
				mainJta = jta;
				dataPackView = new DataPackView(datapack, this);
				pSchema.add(new TitleBar(IconDatabase.iconSchema,
				Messages.getString("MainFrameDatapackCreator.1")), BorderLayout.NORTH); //$NON-NLS-1$
				pSchema.add(dataPackView, BorderLayout.CENTER);
			}
		} else if (object.getClass() == Brick.class) {
			Brick brick = (Brick) object;
			if (datapack != null) {
				jta = new JTreeActors(datapack, brick);

				BrickView<BrickVertex, BrickEdge> modelView = new BrickView<BrickVertex,BrickEdge>(
						brick, popUp, datapack, jta.getListener());
				pSchema.add(new TitleBar(IconDatabase.iconSchema,
				Messages.getString("MainFrameDatapackCreator.2")), BorderLayout.NORTH); //$NON-NLS-1$
				modelViewList.add(modelView);
				pSchema.add(modelView, BorderLayout.CENTER);
			}
		}

		return drawPanel(jta, popUp, pSchema, mainPanel);
	}

	@Override
	public BrickView<BrickVertex,BrickEdge> getActiveModelView() {

		if (tabbedPane.getSelectedIndex() > 0) {
			try {
				if (tabbedPane.getSelectedComponent() == modelViewList.elementAt(tabbedPane.getSelectedIndex() - 1)
						.getParent())
					return modelViewList.elementAt(tabbedPane
							.getSelectedIndex() - 1);
			} catch (ArrayIndexOutOfBoundsException e) {

			}
			for (BrickView<BrickVertex,BrickEdge> mV : modelViewList) {

				if (tabbedPane.getSelectedComponent() == mV.getParent()
						.getParent().getParent().getParent())
					return mV;
			}
		}
		return null;
	}

	@Override
	public void setDataPack(DataPack dtp) {
		datapack = dtp;

		setTitle(Messages.getString("MainFrameDatapackCreator.0") + " " + dtp); //$NON-NLS-1$

		if (autoSaveCreator == null) {
			autoSaveCreator = new AutoSaveCreator();
		}
		autoSaveCreator.setDataPack(datapack);
		// Fonction de vidage des brique, modÃ¨le et type de brique
		// datapack.getBrickList().clear();
		// datapack.getBrickTypeList().clear();
		// datapack.getModelList().clear();

		tabbedPane.removeAll();
		myOptionNewFrame = new OptionNewFrameDatapackCreator(this, dtp);
		if (datapack == null) {
			return;
		}
		dtp.init();
		popUpHelp = new PopUpHelpView();
		popUpHelp.setView(PopUpHelpView.showDataPackHelp());
		tabbedPane.addTab(Messages.getString("MainFrameDatapackCreator.3"), myPanel(null)); //$NON-NLS-1$
		dataPackView = new DataPackView(datapack, this);
	}

	@Override
	public void initialState(MainFrame mf) {
```

```
  123:                 myOptionNewFrame = new OptionNewFrameDatapackCreator(mf, null);
  124:                 myOptionNewFrame.setTitle(Messages.getString("MainFrameDatapackCre
ator.4")); //$NON-NLS-1$
  125:                 myOptionNewFrame.setInitialStep();
  126:                 myOptionNewFrame.setVisible(true);
  127:         }
  128:
  129:         protected void removeTab() {
  130:
  131:                 BrickView<BrickVertex,BrickEdge> bV = getActiveModelView();
  132:                 if (bV != null)
  133:                 {
  134:                         modelViewList.remove(bV);
  135:                 }
  136:                 return;
  137:         }
  138:
  139:         public void cleanTabAndGraphs()
  140:         {
  141:                 int initialTabCount = tabbedPane.getTabCount();
  142:                 for (int i = 1; i < initialTabCount; i++)
  143:                 {
  144:                         tabbedPane.remove(1);
  145:                 }
  146:                 modelViewList.removeAllElements();
  147:                 for (Brick b : datapack.getBrickList())
  148:                 {
  149:                         b.resetGraph();
  150:                 }
  151:         }
  152:
  153:         @Override
  154:         protected JMenuBar myMenuBar() {
  155:                 JMenuBar result = super.myMenuBar();
  156:                 JMenu relation = new JMenu(Messages.getString("MainFrameDatapackCr
eator.5")); //$NON-NLS-1$
  157:                 JMenuItem edit = new JMenuItem(Messages.getString("MainFrameDatapa
ckCreator.6")); //$NON-NLS-1$
  158:                 JMenuItem export = new JMenuItem(Messages.getString("MainFrameData
packCreator.7")); //$NON-NLS-1$
  159:                 JMenuItem add = new JMenuItem(Messages.getString("MainFrameDatapac
kCreator.8")); //$NON-NLS-1$
  160:                 JMenuItem replace = new JMenuItem(Messages.getString("MainFrameDat
apackCreator.9")); //$NON-NLS-1$
  161:                 JMenuItem defaut = new JMenuItem(Messages.getString("MainFrameData
packCreator.10")); //$NON-NLS-1$
  162:                 relation.add(edit);
  163:                 relation.add(export);
  164:                 relation.add(add);
  165:                 relation.add(replace);
  166:                 relation.add(defaut);
  167:                 result.add(relation);
  168:
  169:
  170:                 edit.addActionListener(new ActionListener() {
  171:
  172:                         @Override
  173:                         public void actionPerformed(ActionEvent arg0) {
  174:                                 for (int i = 0; i < tabbedPane.getTabCount(); i++)
 {
  175:                                         if (tabbedPane.getTitleAt(i).equals(Messag
es.getString("MainFrameDatapackCreator.11"))) { //$NON-NLS-1$
  176:                                                 tabbedPane.setSelectedIndex(i);
  177:                                                 return;
  178:                                         }
  179:
  180:                                 }
  181:                                 if (datapack.getRelations() == null) {
  182:                                         datapack.setRelations(new EnsembleRelation
());
  183:                                         System.out.println(Messages.getString("Mai
nFrameDatapackCreator.12")); //$NON-NLS-1$
  184:                                 }
  185:                                 tabbedPane.addTab(Messages.getString("MainFrameDat
apackCreator.13"), new RelationBrowser(datapack //$NON-NLS-1$
  186:                                                 .getRelations()));
  187:                                 tabbedPane.setSelectedIndex(tabbedPane.getTabCount
() - 1);
  188:                         }
  189:                 });
  190:
  191:                 export.addActionListener(new ActionListener() {
  192:
  193:                         @Override
  194:                         public void actionPerformed(ActionEvent arg0) {
  195:                                 if (datapack == null) {
  196:                                         DialogHandlerFrame
  197:                                         .showErrorDialog(Messages.getString("MainF
rameDatapackCreator.14")); //$NON-NLS-1$
  198:                                         return;
  199:                                 }
  200:                                 datapack.getRelations().setFilePath(null);
  201:                                 if(Program.save(datapack.getRelations(), ".dtr", /
/$NON-NLS-1$
  202:                                 Messages.getString("MainFrameDatapackCreator.16"),
 true)) { //$NON-NLS-1$
  203:                                         System.out.println("Impossible de sauvegar
der les relations");
  204:                                 }
  205:                         }
  206:                 });
  207:
  208:                 add.addActionListener(new ActionListener() {
  209:
  210:                         @Override
  211:                         public void actionPerformed(ActionEvent arg0) {
  212:                                 EnsembleRelation newRelation = (EnsembleRelation)
Program
  213:                                 .loadSavableObject(null, "dtr", //$NON-NLS-1$
  214:                                 Messages.getString("MainFrameDatapackCreator.18"),
 false); //$NON-NLS-1$
  215:                                 if (newRelation != null) {
  216:                                         if (datapack.getRelations() == null) {
  217:                                                 datapack.setRelations(newRelation)
;
  218:                                         } else {
  219:                                                 datapack.getRelations().addEnsembl
eRelation(newRelation);
  220:                                         }
  221:                                         Component tabComponent = Program.myMainFra
me.getTabByName(Messages.getString("MainFrameDatapackCreator.19")); //$NON-NLS-1$
  222:                                         if (tabComponent != null & tabComponent in
stanceof RelationBrowser)
  223:                                         {
  224:                                                 RelationBrowser panel = (RelationB
rowser)tabComponent;
  225:                                                 panel.refreshList();
  226:                                         }
  227:                                 }
  228:
  229:                         }
  230:                 });
  231:
  232:                 replace.addActionListener(new ActionListener() {
```

```
  233:
  234:                        @Override
  235:                        public void actionPerformed(ActionEvent arg0) {
  236:                                System.out.println(Messages.getString("MainFrameDa
tapackCreator.20")); //$NON-NLS-1$
  237:                                if (datapack.getRelations() != null) {
  238:                                        if (DialogHandlerFrame
  239:                                                        .showYesNoDialog(Messages.
getString("MainFrameDatapackCreator.21")) != JOptionPane.YES_OPTION) //$NON-NLS-1$
  240:                                                return;
  241:
  242:                                }
  243:                                EnsembleRelation newRelation = (EnsembleRelation)
Program
  244:                                .loadSavableObject(null, "dtr", //$NON-NLS-1$
  245:                                Messages.getString("MainFrameDatapackCreator.23"),
 true); //$NON-NLS-1$
  246:                                if (newRelation != null)
  247:                                {
  248:                                        datapack.setRelations(newRelation);
  249:
  250:                                        Component tabComponent = Program.myMainFra
me.getTabByName(Messages.getString("MainFrameDatapackCreator.24")); //$NON-NLS-1$
  251:                                        if (tabComponent != null & tabComponent in
stanceof RelationBrowser)
  252:                                        {
  253:                                                RelationBrowser panel = (RelationB
rowser)tabComponent;
  254:                                                panel.refreshList();
  255:                                        }
  256:                                }
  257:                        }
  258:                });
  259:
  260:                defaut.addActionListener(new ActionListener() {
  261:                        @Override
  262:                        public void actionPerformed(ActionEvent arg0) {
  263:                                if (DialogHandlerFrame.showYesNoDialog(Messages.ge
tString("MainFrameDatapackCreator.25")) != JOptionPane.YES_NO_CANCEL_OPTION) { //$NON-NLS
-1$
  264:                                        if (datapack.getRelations() == null) {
  265:                                                datapack.setRelations(EnsembleRela
tion.getDefaultRelations());
  266:                                        } else {
  267:                                                datapack.getRelations().addEnsembl
eRelation(EnsembleRelation.getDefaultRelations());
  268:                                        }
  269:                                        Component tabComponent = Program.myMainFra
me.getTabByName(Messages.getString("MainFrameDatapackCreator.26")); //$NON-NLS-1$
  270:                                        if (tabComponent != null & tabComponent in
stanceof RelationBrowser)
  271:                                        {
  272:                                                RelationBrowser panel = (RelationB
rowser)tabComponent;
  273:                                                panel.refreshList();
  274:                                        }
  275:
  276:                                }
  277:                        }
  278:                });
  279:
  280:                return result;
  281:        }
  282:
  283: }
```

```java
package graphicalUserInterface;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JToolBar;

import translation.Messages;

public abstract class PopUpView extends JPanel {

	private static final long serialVersionUID = -3496894332815190012L;

	protected JPanel contentPane;

	protected JPanel panelMax;
	protected JPanel panelMin;
	protected JLabel labelTitle;

	public PopUpView() {
		setLayout(new BorderLayout());
		setBorder(BorderFactory.createLineBorder(Color.gray));
		build();
	}

	private void build() {
		// DÃ©clarations + initialisations
		contentPane = new JPanel(new BorderLayout());
		panelMax = contentPane;
		panelMin = new JPanel();
		labelTitle = new JLabel("Titre"); //$NON-NLS-1$
		JPanel pTitle = new JPanel();
		BoxLayout layoutTitle = new BoxLayout(pTitle, BoxLayout.X_AXIS);
		BoxLayout layoutMin = new BoxLayout(panelMin, BoxLayout.Y_AXIS);
		JToolBar jtbMax = new JToolBar(JToolBar.HORIZONTAL);
		jtbMax.setFloatable(false);
		JToolBar jtbMin = new JToolBar(JToolBar.HORIZONTAL);
		jtbMin.setFloatable(false);
		JButton jbMinimize = new JButton(IconDatabase.iconArrowDownLeft);
		jbMinimize.setFocusable(false);
		JButton jbMaximize = new JButton(IconDatabase.iconArrowUpRight);
		jbMaximize.setFocusable(false);

		// Construction barre de titre max
		pTitle.setLayout(layoutTitle);
		pTitle.setBackground(Color.LIGHT_GRAY);
		pTitle.add(Box.createHorizontalStrut(3));
		pTitle.add(new JLabel(IconDatabase.iconConfigure));
		pTitle.add(Box.createHorizontalStrut(5));
		pTitle.add(labelTitle);
		pTitle.add(Box.createGlue());
		jtbMax.setFloatable(false);
		jbMinimize.setBackground(Color.LIGHT_GRAY);
		jbMinimize.setFocusable(false);
		jbMinimize.setToolTipText(Messages.getString("PopUpView.1")); //$N
ON-NLS-1$
		jbMinimize.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				panelMin.setVisible(true);
				panelMax.setVisible(false);
			}

		});
		jtbMax.add(jbMinimize);
		jtbMax.setBackground(Color.LIGHT_GRAY);
		pTitle.add(jtbMax);

		// Construction panel max
		panelMax.add(pTitle, BorderLayout.NORTH);
		panelMax.setVisible(false);

		// Construction panel min
		panelMin.setVisible(false);
		jtbMin.add(Box.createHorizontalStrut(2));
		jtbMin.add(new JLabel(IconDatabase.iconConfigure));
		jbMaximize.setBackground(Color.LIGHT_GRAY);
		jbMaximize.setFocusable(false);
		jbMaximize.setToolTipText(Messages.getString("PopUpView.2")); //$N
ON-NLS-1$
		jbMaximize.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				panelMin.setVisible(false);
				panelMax.setVisible(true);
			}

		});
		jtbMin.add(jbMaximize);
		jtbMin.setBackground(Color.LIGHT_GRAY);
		panelMin.setLayout(layoutMin);
		panelMin.add(Box.createGlue());
		panelMin.add(jtbMin);

		// Construction panel total
		add(panelMax, BorderLayout.CENTER);
		add(panelMin, BorderLayout.WEST);
	}


}
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.awt.Component;
    4:
    5: import javax.swing.Icon;
    6: import javax.swing.JOptionPane;
    7:
    8: import translation.Messages;
    9:
   10: public class DialogHandlerFrame {
   11:
   12:         public static void showErrorDialog(Component cmp, String message) {
   13:                 JOptionPane.showMessageDialog(cmp, message, Messages.getString("Di
alogHandlerFrame.0"), //$NON-NLS-1$
   14:                                 JOptionPane.WARNING_MESSAGE);
   15:         }
   16:
   17:         public static void showErrorDialog(String message) {
   18:                 showErrorDialog(Program.myMainFrame, message);
   19:         }
   20:
   21:         public static int showYesNoCancelDialog(Component cmp, String message) {
   22:                 return JOptionPane.showConfirmDialog(cmp, message);
   23:         }
   24:
   25:         public static int showYesNoCancelDialog(String message) {
   26:                 return showYesNoCancelDialog(Program.myMainFrame, message);
   27:         }
   28:
   29:         public static int showYesNoDialog(Component cmp, String message) {
   30:                 return JOptionPane.showConfirmDialog(cmp, message,
   31:                                 Messages.getString("DialogHandlerFrame.1"), JOptio
nPane.YES_NO_OPTION); //$NON-NLS-1$
   32:         }
   33:
   34:         public static int showYesNoDialog(String message) {
   35:                 return showYesNoDialog(Program.myMainFrame, message);
   36:         }
   37:
   38:         public static void showInformationDialog(Component cmp, String title,
   39:                         String message, Icon icon) {
   40:                 JOptionPane.showMessageDialog(cmp, message, title,
   41:                                 JOptionPane.INFORMATION_MESSAGE, icon);
   42:         }
   43: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.io.File;
    4: import java.io.FileInputStream;
    5: import java.io.FileNotFoundException;
    6: import java.io.FileOutputStream;
    7: import java.io.IOException;
    8: import java.io.ObjectInputStream;
    9: import java.io.ObjectOutputStream;
   10: import java.lang.Thread.UncaughtExceptionHandler;
   11:
   12: import javax.swing.JFileChooser;
   13: import javax.swing.JOptionPane;
   14: import javax.swing.ToolTipManager;
   15: import javax.swing.UIManager;
   16: import javax.swing.UnsupportedLookAndFeelException;
   17: import javax.swing.filechooser.FileNameExtensionFilter;
   18:
   19: import tools.ConfigCreator;
   20: import tools.Encrypting;
   21: import translation.Messages;
   22: import Mailing.MailErrorForm;
   23: import Mailing.MailView;
   24: import client.export.ExportImagesView;
   25: import dataPack.AutoSaveCreator;
   26: import dataPack.DataPack;
   27: import dataPack.SavableObject;
   28:
   29: public class Program {
   30:
   31:         public static MainFrame myMainFrame;
   32:         private static boolean isTriades;
   33:         private static boolean isTriadesLoading;
   34:
   35:         public static void main(String[] args) {
   36:
   37:                 // TODO pour sauver en cas de crash :
   38:                 // http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Thread.html#s
etDefaultUncaughtExceptionHandler%28java.lang.Thread.UncaughtExceptionHandler%29
   39:                 ToolTipManager.sharedInstance().setDismissDelay(60000);
   40:                 ToolTipManager.sharedInstance().setReshowDelay(50);
   41:
   42:                 final UncaughtExceptionHandler oldHandler = Thread.getDefaultUncau
ghtExceptionHandler();
   43:                 Thread.setDefaultUncaughtExceptionHandler(new UncaughtExceptionHan
dler() {
   44:
   45:                         @Override
   46:                         public void uncaughtException(Thread t, Throwable e) {
   47:                                 e.printStackTrace();
   48:
   49:                                 if (DialogHandlerFrame.showYesNoDialog("Une erreur
 vient de se produire, voilÃ  sa description :\n"+e.getMessage()+"\nSi vous ne savez d'oÃ
¹ vient cette erreur\n il est prÃ©fÃ©rable de redÃ©marrer l'application\nVoulez vous envo
yez un rapport d'erreur aux dÃ©veloppeurs? \n(Si c'est la premiÃ¨re fois que cette erreur
 se produit\n il est prÃ©fÃ©rable d'envoyer ce rapport).")==JOptionPane.YES_OPTION)
   50:                                 {
   51:                                         new MailView("Rapport d'erreur", new MailE
rrorForm(e));
   52:                                         myMainFrame.autoSaveCreator.stopAutoSave()
;
   53:                                 }
   54:
   55:                                 oldHandler.uncaughtException(t, e);
   56:                         }
   57:                 });
   58:
   59:                 try {
   60:                         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelCla
ssName());
   61:                 } catch (ClassNotFoundException e) {
   62:                         e.printStackTrace();
   63:                 } catch (InstantiationException e) {
   64:                         e.printStackTrace();
   65:                 } catch (IllegalAccessException e) {
   66:                         e.printStackTrace();
   67:                 } catch (UnsupportedLookAndFeelException e) {
   68:                         e.printStackTrace();
   69:                 }
   70:
   71:                 isTriades = false;
   72:                 isTriadesLoading = false;
   73:                 new IconDatabase();
   74:
   75:                 if (args != null && args.length > 0) {
   76:
   77:                         if (args[0] != null && args[0].equals("creator")) { //$NON
-NLS-1$
   78:                                 myMainFrame = new MainFrameDatapackCreator();
   79:                                 myMainFrame.initialState(myMainFrame);
   80:                         }
   81:                 } else {
   82:                         isTriadesLoading = true;
   83:
   84:                         File imagesDirectoryFile = new File(ExportImagesView.image
sDirectoryPath);
   85:                         if(imagesDirectoryFile.exists() == false) {
   86:                                 try {
   87:                                         imagesDirectoryFile.mkdir();
   88:                                 } catch (Exception e1) {
   89:                                         e1.printStackTrace();
   90:                                 }
   91:                         }
   92:
   93:                         myMainFrame = new MainFrameTriades();
   94:
   95:                         myMainFrame.setDataPack((DataPack) loadSavableObject(null,
 true));
   96:                         myMainFrame.initialState(myMainFrame);
   97:                         isTriades = true;
   98:                 }
   99:         }
  100:
  101:         public static SavableObject loadSavableObject(String filepath, boolean cry
pted) {
  102:                 String extension = null;
  103:                 String description = null;
  104:                 if (!Program.isTriades() && !Program.isTriadesLoading()) {
  105:                         description = Messages.getString("Program.1"); //$NON-NLS-
1$
  106:                         extension = "dtp"; //$NON-NLS-1$
  107:                 } else {
  108:                         description = Messages.getString("Program.3"); //$NON-NLS-
1$
  109:                         extension = "dte"; //$NON-NLS-1$
  110:                 }
  111:                 return (loadSavableObject(filepath, extension, description, crypte
d));
  112:
  113:
  114:         }
  115:
  116:         public static SavableObject loadSavableObject(String _filePath,
  117:                         String extension, String description, boolean crypted) {
```

```
  118:                 String filePath = _filePath;
  119:
  120:                 SavableObject result = null;
  121:
  122:                 if (filePath == null) {
  123:                         JFileChooser chooser = new JFileChooser();
  124:                         FileNameExtensionFilter filter;
  125:                         filter = new FileNameExtensionFilter(description, extensio
n);
  126:                         chooser.setFileFilter(filter);
  127:                         int returnVal = chooser.showOpenDialog(myMainFrame);
  128:                         if (returnVal == JFileChooser.APPROVE_OPTION) {
  129:                                 filePath = chooser.getSelectedFile().getPath();
  130:                         } else
  131:                                 return null;
  132:                 }
  133:
  134:                 filePath = AutoSaveCreator.askLoadAutoSaveFile(filePath);
  135:
  136:                 if(crypted) {
  137:                         SavableObject object = (SavableObject)Encrypting.getInstan
ce().loadEncryptedObject(filePath);
  138:                         if(object != null) {
  139:                                 result = object;
  140:                         }
  141:                 }
  142:
  143:                 if (!crypted || result == null)
  144:                 {
  145:
  146:                         FileInputStream file;
  147:                         try {
  148:                                 file = new FileInputStream(filePath);
  149:                                 ObjectInputStream ois = new ObjectInputStream(file
);
  150:                                 SavableObject temp = (SavableObject) ois.readObjec
t();
  151:
  152:                                 if (filePath.endsWith(AutoSaveCreator.extentionAut
oSave)) {
  153:                                         filePath = AutoSaveCreator
  154:                                         .getPathWithoutAutoSaveExtention(filePath)
;
  155:                                 }
  156:
  157:                                 ConfigCreator.getInstance().getLastDatapack().addL
astObject(filePath);
  158:                                 ConfigCreator.getInstance().save();
  159:                                 result = temp;
  160:                         } catch (FileNotFoundException e) {
  161:                                 // e.printStackTrace();
  162:                         } catch (IOException e) {
  163:                                 e.printStackTrace();
  164:                         } catch (ClassNotFoundException e) {
  165:                                 e.printStackTrace();
  166:                         }
  167:                 }
  168:
  169:                 result.setFilePath(filePath);
  170:                 if (result instanceof DataPack)
  171:                 {
  172:                         ((DataPack)result).checkDatapackValidity();
  173:                 }
  174:
  175:                 return result;
  176:         }
  177:
  178:         public static boolean save(SavableObject object, boolean crypted) {
  179:                 if (!Program.isTriades()) {
  180:                         return save(object, "dtp", Messages.getString("Program.8")
, crypted); //$NON-NLS-1$ //$NON-NLS-2$
  181:                 } else {
  182:                         return save(object, "dte", Messages.getString("Program.10"
), crypted); //$NON-NLS-1$ //$NON-NLS-2$
  183:                 }
  184:         }
  185:
  186:         public static boolean save(SavableObject object, String extension, String
description, boolean crypted) {
  187:                 if (object == null) {
  188:                         System.err.println("Erreur: object == null, rien Ã  sauveg
arder"); //$NON-NLS-1$
  189:                         return false;
  190:                 }
  191:                 String savePath = null;
  192:                 File selectedFile = null;
  193:                 boolean isAllowedToSave = false;
  194:
  195:                 if (object.getFilePath() == null) {
  196:                         JFileChooser chooser = new JFileChooser();
  197:                         FileNameExtensionFilter filter = new FileNameExtensionFilt
er(
  198:                                         description, extension);
  199:                         chooser.setFileFilter(filter);
  200:                         while (!isAllowedToSave) {
  201:                                 int returnVal = chooser.showSaveDialog(myMainFrame
);
  202:                                 if (returnVal == JFileChooser.APPROVE_OPTION) {
  203:                                         selectedFile = chooser.getSelectedFile();
  204:                                         savePath = selectedFile.getPath();
  205:                                         if (selectedFile.exists()) {
  206:                                                 int returnV = DialogHandlerFrame
  207:                                                                 .showYesNoDialog(
  208:                                                                                 ch
ooser,
  209:                                                                                 Me
ssages.getString("Program.12") //$NON-NLS-1$
  210:
        + savePath
  211:
        + Messages.getString("Program.13")); //$NON-NLS-1$
  212:                                                 if (returnV == JOptionPane.YES_OPT
ION) {
  213:                                                         isAllowedToSave = true;
  214:                                                 }
  215:                                         } else {
  216:                                                 isAllowedToSave = true;
  217:                                         }
  218:                                 } else if (returnVal == JFileChooser.CANCEL_OPTION
)
  219:                                         return false;
  220:                         }
  221:                 } else
  222:                         savePath = object.getFilePath();
  223:                 if (!savePath.endsWith("." + extension) //$NON-NLS-1$
  224:                                 && object.getClass() != ConfigUser.class) {
  225:                         savePath += ("." + extension); //$NON-NLS-1$
  226:                 }
  227:
  228:                 boolean result = saveObject(object, savePath, crypted);
  229:
  230:                 if(result && object.getFilePath() == null) {
  231:                         object.setFilePath(savePath);
  232:                 }
```

```
  233:
  234:                return result;
  235:        }
  236:
  237:        static public boolean saveObject(SavableObject object, String filePath, bo
olean crypted) {
  238:                boolean result;
  239:
  240:                if(crypted) {
  241:                        result = Encrypting.getInstance().saveEncryptedObject(obje
ct, filePath);
  242:                        if(!result) {
  243:                                System.out.println("Impossible de sauver " + objec
t + " dans le fichier " + filePath);
  244:                        }
  245:
  246:                        return result;
  247:                }
  248:
  249:                try {
  250:                        File file = new File(filePath);
  251:                        File tmpFile;
  252:                        boolean exist = file.exists();
  253:
  254:                        if (exist) {
  255:                                tmpFile = File.createTempFile("datapack", "tmp");
  256:                        } else {
  257:                                tmpFile = new File(filePath);
  258:                        }
  259:
  260:                        ObjectOutputStream oos = new ObjectOutputStream(new FileOu
tputStream(tmpFile));
  261:                        oos.writeObject(object);
  262:                        oos.close();
  263:
  264:                        if(Encrypting.renameFile(tmpFile, file)) {
  265:                                System.out.println("Impossible de sauver " + objec
t + " dans le fichier " + filePath);
  266:                                return false;
  267:                        }
  268:
  269:                        System.out.println("Enregistrer: success! : " + filePath);
 //$NON-NLS-1$
  270:                        if (AutoSaveCreator.isAutoSaveFile(filePath) == false) {
  271:                                AutoSaveCreator.dataPackSaved(object);
  272:                        }
  273:
  274:                        return true;
  275:                } catch (Exception e1) {
  276:                        e1.printStackTrace();
  277:                }
  278:
  279:                System.out.println("Impossible de sauver " + object + " dans le fi
chier " + filePath);
  280:                return false;
  281:        }
  282:
  283:        public static boolean isTriades() {
  284:                return isTriades;
  285:        }
  286:
  287:        static public void setIsTriade(boolean value) {
  288:                isTriades = value;
  289:        }
  290:
  291:        public static boolean isTriadesLoading() {
  292:                return isTriadesLoading;
  293:        }
  294:
  295: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.util.Vector;
    4:
    5: import translation.Messages;
    6:
    7: import dataPack.SavableObject;
    8:
    9: public class ConfigUser implements SavableObject {
   10:
   11:         private static final long serialVersionUID = 1429351747176112333L;
   12:
   13:         private String filePath;
   14:         private final Vector<String> lastFilePath;
   15:
   16:         private ConfigUser() {
   17:                 filePath = "config"; //$NON-NLS-1$
   18:                 lastFilePath = new Vector<String>();
   19:         }
   20:
   21:         public Vector<String> getLastFilePath() {
   22:                 Vector<String> copy = new Vector<String>();
   23:                 for (int i = 0; i < lastFilePath.size(); i++)
   24:                         copy.add(lastFilePath.elementAt(lastFilePath.size() - 1 -
i));
   25:
   26:                 return copy;
   27:         }
   28:
   29:         public void addLastFilePath(String path) {
   30:                 if (lastFilePath.contains(path))
   31:                         lastFilePath.remove(path);
   32:                 if (lastFilePath.size() == 5)
   33:                         lastFilePath.remove(0);
   34:                 lastFilePath.add(path);
   35:         }
   36:
   37:         @Override
   38:         public String getFilePath() {
   39:                 return filePath;
   40:         }
   41:
   42:         @Override
   43:         public void setFilePath(String path) {
   44:                 filePath = path;
   45:         }
   46:
   47: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.awt.event.ActionEvent;
    4: import java.awt.event.ActionListener;
    5:
    6: import javax.swing.JButton;
    7: import javax.swing.JList;
    8: import javax.swing.JPanel;
    9:
   10: import translation.Messages;
   11:
   12: public class TriadeIntroPanel extends JPanel {
   13:
   14:         /**
   15:          *
   16:          */
   17:         private static final long serialVersionUID = 3546597040030449553L;
   18:
   19:         protected JButton browseAll;
   20:         protected JList exportedSchema;
   21:         protected JButton openExport;
   22:         protected JButton newExport;
   23:
   24:         public TriadeIntroPanel() {
   25:                 build();
   26:         }
   27:
   28:         protected void build() {
   29:                 JButton browseAll = new JButton(Messages.getString("TriadeIntroPan
el.0")); //$NON-NLS-1$
   30:                 browseAll.addActionListener(new ActionListener() {
   31:
   32:                         @Override
   33:                         public void actionPerformed(ActionEvent e) {
   34:
   35:                         }
   36:                 });
   37:         }
   38: }
```

```java
package graphicalUserInterface;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.JButton;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.JSplitPane;
import javax.swing.JTabbedPane;
import javax.swing.JToolBar;
import javax.swing.JTree;
import javax.swing.KeyStroke;
import javax.swing.ScrollPaneConstants;

import models.Brick;
import models.BrickView;
import tools.ConfigCreator;
import tools.ConfigTriades;
import translation.Messages;
import Mailing.MailAddRelationForm;
import Mailing.MailForm;
import Mailing.MailView;
import client.export.ExportModel;
import client.export.ExportPopUp;
import client.export.ExportView;
import dataPack.AutoSaveCreator;
import dataPack.DataPack;
import dataPack.JTreeActors;
import dataPack.SavableObject;
import dataPack.TreeListener;

public abstract class MainFrame extends JFrame {

	private static final long serialVersionUID = -1230397799638872517L;

	protected Controller controller;
	protected JTabbedPane tabbedPane;
	protected JPanel pArborescence;
	protected JPanel pSchema;
	protected JMenuBar menuBar;
	public DataPack datapack;
	protected JTreeActors mainJta;
	public PopUpHelpView popUpHelp;
	protected AutoSaveCreator autoSaveCreator;

	public MainFrame() {
		this(true);
	}

	public MainFrame(boolean init) {
		super();

		if(init) {
			createAndShowGUI();
		}
	}

	private void createAndShowGUI() {
		JPanel unPanelAnodin = new JPanel(new BorderLayout());
		controller = new Controller(this);
		tabbedPane = new JTabbedPane();
		mainJta = null;
		menuBar = myMenuBar();

		Image icone = Toolkit.getDefaultToolkit().getImage(
				"Icones/16x16/triades.png"); //$NON-NLS-1$
		setIconImage(icone);
		setSize(1024, 768);
		setLocationRelativeTo(null);
		setResizable(true);
		setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
		addWindowListener(new WindowListener() {
			@Override
			public void windowClosing(WindowEvent arg0) {
				wouldYouLikeToSaveAndExit();
			}

			@Override
			public void windowClosed(WindowEvent arg0) {
			}

			@Override
			public void windowActivated(WindowEvent arg0) {
			}

			@Override
			public void windowDeactivated(WindowEvent arg0) {
			}

			@Override
			public void windowDeiconified(WindowEvent arg0) {
			}

			@Override
			public void windowIconified(WindowEvent arg0) {
			}

			@Override
			public void windowOpened(WindowEvent arg0) {
			}
		});

		setJMenuBar(menuBar);
		getContentPane().setLayout(new BorderLayout());
		unPanelAnodin.add(myToolBar(), BorderLayout.CENTER);
		unPanelAnodin.add(new JSeparator(), BorderLayout.SOUTH);
		getContentPane().add(unPanelAnodin, BorderLayout.NORTH);
		getContentPane().add(tabbedPane, BorderLayout.CENTER);

```

```
  135:                 setVisible(true);
  136:                 this.setExtendedState(Frame.MAXIMIZED_BOTH);
  137:
  138:         }
  139:
  140:         protected JMenuBar myMenuBar() {
  141:                 // DÃ©claration + Initialisations
  142:                 JMenuBar menuBar = new JMenuBar();
  143:                 JMenu mFichier = new JMenu(Messages.getString("MainFrame.1")); //$
NON-NLS-1$
  144:                 JMenu mEdition = new JMenu(Messages.getString("MainFrame.2")); //$
NON-NLS-1$
  145:                 JMenu mFenetre = new JMenu(Messages.getString("MainFrame.3")); //$
NON-NLS-1$
  146:                 JMenu mAide = new JMenu(Messages.getString("MainFrame.4")); //$NON
-NLS-1$
  147:                 //
  148:                 JMenuItem nouveau = new JMenuItem(Messages.getString("MainFrame.5"
), IconDatabase.iconNew); //$NON-NLS-1$
  149:                 JMenuItem ouvrir = new JMenuItem(Messages.getString("MainFrame.6")
, IconDatabase.iconOpen); //$NON-NLS-1$
  150:                 JMenuItem enregistrer = new JMenuItem(Messages.getString("MainFram
e.7"), //$NON-NLS-1$
  151:                                 IconDatabase.iconSave);
  152:                 JMenuItem enregistrerSous = new JMenuItem(Messages.getString("Main
Frame.8"), //$NON-NLS-1$
  153:                                 IconDatabase.iconSaveAs);
  154:                 JMenuItem exporterDatapack = new JMenuItem(Messages.getString("Mai
nFrame.9"), //$NON-NLS-1$
  155:                                 IconDatabase.iconSaveAs);
  156:                 JMenuItem fermer = new JMenuItem(Messages.getString("MainFrame.10"
), IconDatabase.iconClose); //$NON-NLS-1$
  157:                 JMenuItem quitter = new JMenuItem(Messages.getString("MainFrame.11
"), IconDatabase.iconExit); //$NON-NLS-1$
  158:                 JMenuItem supprimer = new JMenuItem(Messages.getString("MainFrame.
12"), //$NON-NLS-1$
  159:                                 IconDatabase.iconDelete);
  160:                 JMenuItem renommer = new JMenuItem(Messages.getString("MainFrame.1
3"), IconDatabase.iconRename); //$NON-NLS-1$
  161:                 JMenuItem fermerOnglet = new JMenuItem(Messages.getString("MainFra
me.14"), //$NON-NLS-1$
  162:                                 IconDatabase.iconRemoveTab);
  163:                 JMenuItem aide = new JMenuItem(Messages.getString("MainFrame.15"),
 IconDatabase.iconHelp); //$NON-NLS-1$
  164:                 JMenuItem maj = new JMenuItem(Messages.getString("MainFrame.16"),
IconDatabase.iconUpdate); //$NON-NLS-1$
  165:                 JMenuItem aPropos = new JMenuItem(Messages.getString("MainFrame.17
"), //$NON-NLS-1$
  166:                                 IconDatabase.iconAbout);
  167:                 JMenuItem sendMail = new JMenuItem(Messages.getString("MainFrame.1
8"), IconDatabase.iconRename); //$NON-NLS-1$
  168:                 //
  169:                 JCheckBoxMenuItem arborescence = new JCheckBoxMenuItem(Messages.ge
tString("MainFrame.19"), //$NON-NLS-1$
  170:                                 IconDatabase.iconTree);
  171:                 JCheckBoxMenuItem schema = new JCheckBoxMenuItem(Messages.getStrin
g("MainFrame.20"), //$NON-NLS-1$
  172:                                 IconDatabase.iconSchema);
  173:
  174:                 // Menu "Fichier"
  175:                 nouveau.addActionListener(controller.actionNew);
  176:                 nouveau.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
  177:                                 InputEvent.CTRL_DOWN_MASK));
  178:                 mFichier.add(nouveau);
  179:                 ouvrir.addActionListener(controller.actionOpen);
  180:                 ouvrir.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
  181:                                 InputEvent.CTRL_DOWN_MASK));
  182:                 mFichier.add(ouvrir);
  183:                 //
  184:                 mFichier.addSeparator();
  185:                 //
  186:                 enregistrer.addActionListener(controller.actionSave);
  187:                 enregistrer.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
  188:                                 InputEvent.CTRL_DOWN_MASK));
  189:                 mFichier.add(enregistrer);
  190:
  191:                 enregistrerSous.addActionListener(controller.actionSaveAs);
  192:                 enregistrerSous.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_
S,
  193:                                 InputEvent.CTRL_DOWN_MASK | InputEvent.ALT_DOWN_MA
SK));
  194:                 mFichier.add(enregistrerSous);
  195:
  196:                 exporterDatapack.addActionListener(controller.actionExoprterDataPa
ck);
  197:                 mFichier.add(exporterDatapack);
  198:
  199:                 //
  200:                 mFichier.addSeparator();
  201:                 //
  202:                 fermer.addActionListener(controller.actionCloseAll);
  203:                 fermer.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_W,
  204:                                 InputEvent.CTRL_DOWN_MASK));
  205:                 mFichier.add(fermer);
  206:                 //
  207:                 mFichier.addSeparator();
  208:                 //
  209:                 quitter.addActionListener(controller.actionExit);
  210:                 quitter.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q,
  211:                                 InputEvent.CTRL_DOWN_MASK));
  212:                 mFichier.add(quitter);
  213:
  214:                 // Menu "Edition"
  215:                 supprimer.addActionListener(controller.actionDelete);
  216:                 supprimer.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_DELETE
, 0));
  217:                 mEdition.add(supprimer);
  218:                 renommer.addActionListener(controller.actionRename);
  219:                 // supprimer.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F2,
 0));
  220:                 mEdition.add(renommer);
  221:                 //
  222:                 mEdition.addSeparator();
  223:                 //
  224:                 fermerOnglet.addActionListener(controller.actionRemoveTab);
  225:                 fermerOnglet.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_W,
  226:                                 InputEvent.CTRL_DOWN_MASK | InputEvent.ALT_DOWN_MA
SK));
  227:                 mEdition.add(fermerOnglet);
  228:
  229:                 // Menu "Fenetre"
  230:                 arborescence.setSelected(true);
  231:                 arborescence.addActionListener(controller.actionTreeVisibility);
  232:                 mFenetre.add(arborescence);
  233:                 schema.setSelected(true);
  234:                 schema.addActionListener(controller.actionSchemaVisibility);
  235:                 mFenetre.add(schema);
  236:
  237:                 // Menu "Aide"
  238:                 final MainFrame frame = this;
  239:                 aide.addActionListener(controller.actionHelp);
  240:                 aide.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F1, 0));
  241:                 mAide.add(aide);
  242:                 maj.addActionListener(new ActionListener() {
```

```
243:                        @Override
244:                        public void actionPerformed(ActionEvent arg0) {
245:                                DialogHandlerFrame
246:                                                .showInformationDialog(frame, Mess
ages.getString("MainFrame.21"), //$NON-NLS-1$
247:                                                                Messages.getString
("MainFrame.22"), //$NON-NLS-1$
248:                                                                IconDatabase.iconU
pgrade);
249:                        }
250:
251:                });
252:                mAide.add(maj);
253:                aPropos.addActionListener(new ActionListener() {
254:
255:                        @Override
256:                        public void actionPerformed(ActionEvent arg0) {
257:                                DialogHandlerFrame
258:                                                .showInformationDialog(
259:                                                                frame,
260:                                                                Messages.getString
("MainFrame.23"), //$NON-NLS-1$
261:                                                                Messages.getString
("MainFrame.24"), //$NON-NLS-1$
262:                                                                IconDatabase.iconT
riad);
263:                        }
264:
265:                });
266:                mAide.add(aPropos);
267:
268:                final MailForm mailForm = new MailAddRelationForm();
269:                sendMail.addActionListener(new ActionListener() {
270:                        @Override
271:                        public void actionPerformed(ActionEvent e) {
272:                                new MailView(Messages.getString("MainFrame.25"), m
ailForm); //$NON-NLS-1$
273:                        }
274:                });
275:                mAide.add(sendMail);
276:
277:                // Barre de menu
278:                menuBar.add(mFichier);
279:                menuBar.add(mEdition);
280:                menuBar.add(mFenetre);
281:                menuBar.add(mAide);
282:
283:                return menuBar;
284:        }
285:
286:        protected JPanel myPanel(Object object) {
287:                return myPanel(object, new JPanel());
288:        }
289:        protected abstract JPanel myPanel(Object object, JPanel mainPanel);
290:
291:        protected JPanel drawPanel(JTree jta, PopUpView popUp, JPanel pSchema, JPa
nel panel) {
292:                // DÃ©clarations + initialisations
293:                JSplitPane splitPane = new JSplitPane();
294:
295:                panel.setLayout(new BorderLayout());
296:                JPanel superArbo = new JPanel(new BorderLayout());
297:                JPanel superSchema = new JPanel(new BorderLayout());
298:                pArborescence = new JPanel(new BorderLayout());
299:                JLayeredPane layeredSchema = new JLayeredPane();
300:                JToolBar pArborescenceMini = new JToolBar(JToolBar.VERTICAL);
301:                JToolBar pSchemaMini = new JToolBar(JToolBar.VERTICAL);
302:                JButton agrandirArbo = new JButton(IconDatabase.iconMaximize);
303:                JButton agrandirSche = new JButton(IconDatabase.iconMaximize);
304:                JScrollPane jsp;
305:
306:                // Arborescence rÃ©duite
307:                agrandirArbo.setFocusable(false);
308:                agrandirArbo.setToolTipText(Messages.getString("MainFrame.26")); /
/$NON-NLS-1$
309:                agrandirArbo.addActionListener(controller.actionMaximize);
310:                pArborescenceMini.setFloatable(false);
311:                pArborescenceMini.addSeparator();
312:                pArborescenceMini.add(new JLabel(IconDatabase.iconTree));
313:                pArborescenceMini.add(agrandirArbo);
314:                pArborescenceMini.setVisible(false);
315:
316:                // Arborescence agrandie
317:                pArborescence.setBorder(BorderFactory.createCompoundBorder(
318:                                BorderFactory.createEmptyBorder(2, 2, 2, 2), Borde
rFactory
319:                                                .createLineBorder(Color.GRAY)));
320:                if (jta == null)
321:                        jsp = new JScrollPane();
322:                else {
323:                        jsp = new JScrollPane(jta,
324:                                        ScrollPaneConstants.VERTICAL_SCROLLBAR_ALW
AYS,
325:                                        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_A
LWAYS);
326:                }
327:                jsp.setBorder(BorderFactory.createEmptyBorder());
328:                jsp.setMaximumSize(new Dimension(100,2000));
329:                pArborescence.add(new TitleBar(IconDatabase.iconTree, Messages.get
String("MainFrame.27")), //$NON-NLS-1$
330:                                BorderLayout.NORTH);
331:                pArborescence.add(jsp, BorderLayout.CENTER);
332:                pArborescence.setMaximumSize(new Dimension(100,2000));
333:
334:                // Super arborescence
335:                superArbo.add(pArborescenceMini, BorderLayout.WEST);
336:                superArbo.add(pArborescence, BorderLayout.CENTER);
337:
338:                // SchÃ©ma rÃ©duit
339:                agrandirSche.setFocusable(false);
340:                agrandirSche.setToolTipText(Messages.getString("MainFrame.28")); /
/$NON-NLS-1$
341:                agrandirSche.addActionListener(controller.actionMaximize);
342:                pSchemaMini.setFloatable(false);
343:                pSchemaMini.addSeparator();
344:                pSchemaMini.add(new JLabel(IconDatabase.iconSchema));
345:                pSchemaMini.add(agrandirSche);
346:                pSchemaMini.setVisible(false);
347:
348:                pSchema.setBorder(BorderFactory.createCompoundBorder(BorderFactory
349:                                .createEmptyBorder(2, 2, 2, 2), BorderFactory
350:                                .createLineBorder(Color.GRAY)));
351:
352:                // Super schÃ©ma
353:                superSchema.add(pSchemaMini, BorderLayout.WEST);
354:                superSchema.add(pSchema, BorderLayout.CENTER);
355:
356:                // Layered schema
357:                GridBagLayout gridBag = new GridBagLayout();
358:                layeredSchema.setLayout(gridBag);
359:                GridBagConstraints c = new GridBagConstraints();
360:                c.gridx = 0;
361:                c.gridy = 0;
362:                c.anchor = GridBagConstraints.CENTER;
```

```
363:                 c.fill = GridBagConstraints.BOTH;
364:                 c.weightx = 1;
365:                 c.weighty = 1;
366:                 layeredSchema.add(superSchema, c);
367:                 layeredSchema.setLayer(superSchema, 1);
368:                 if (popUp != null) {
369:                         c.anchor = GridBagConstraints.LAST_LINE_START;
370:                         c.fill = GridBagConstraints.NONE;
371:                         layeredSchema.add(popUp, c);
372:                         layeredSchema.setLayer(popUp, 2);
373:                 }
374:                 c.anchor = GridBagConstraints.LAST_LINE_END;
375:                 c.fill = GridBagConstraints.NONE;
376:                 layeredSchema.add(popUpHelp, c);
377:                 layeredSchema.setLayer(popUpHelp, 3);
378:
379:                 // Panel total
380:         //      panel.add(superArbo, BorderLayout.WEST);
381: //              panel.add(layeredSchema, BorderLayout.CENTER);
382:                 splitPane.setLeftComponent(superArbo);
383:                 splitPane.setRightComponent(layeredSchema);
384:                 splitPane.setDividerLocation(0.3);
385:                 panel.add(splitPane, BorderLayout.CENTER);
386:                 return panel;
387:         }
388:
389:         private JToolBar myToolBar() {
390:                 // DÃ©clarations + initialisations
391:                 //
392:                 JToolBar toolBar = new JToolBar(JToolBar.HORIZONTAL);
393:                 JButton nouveau = new JButton(IconDatabase.iconNew);
394:                 JButton ouvrir = new JButton(IconDatabase.iconOpen);
395:                 JButton enregistrer = new JButton(IconDatabase.iconSave);
396:                 JButton enregistrerSous = new JButton(IconDatabase.iconSaveAs);
397:                 JButton delete = new JButton(IconDatabase.iconDelete);
398:                 JButton aide = new JButton(IconDatabase.iconHelp);
399:                 JButton deleteTab = new JButton(IconDatabase.iconRemoveTab);
400:
401:                 // Construction de la tool bar
402:                 //
403:                 nouveau.setFocusable(false);
404:                 nouveau.setToolTipText(Messages.getString("MainFrame.29")); //$NON
-NLS-1$
405:                 nouveau.addActionListener(controller.actionNew);
406:                 toolBar.add(nouveau);
407:                 ouvrir.setFocusable(false);
408:                 ouvrir.setToolTipText(Messages.getString("MainFrame.30")); //$NON-
NLS-1$
409:                 ouvrir.addActionListener(controller.actionOpen);
410:                 toolBar.add(ouvrir);
411:                 //
412:                 toolBar.addSeparator();
413:                 //
414:                 enregistrer.setFocusable(false);
415:                 enregistrer.setToolTipText(Messages.getString("MainFrame.31")); //
$NON-NLS-1$
416:                 enregistrer.addActionListener(controller.actionSave);
417:                 toolBar.add(enregistrer);
418:
419:                 enregistrerSous.setFocusable(false);
420:                 enregistrerSous.setToolTipText(Messages.getString("MainFrame.32"))
; //$NON-NLS-1$
421:                 enregistrerSous.addActionListener(controller.actionSaveAs);
422:                 toolBar.add(enregistrerSous);
423:                 //
424:                 toolBar.addSeparator();
425:                 //
426:                 delete.setFocusable(false);
427:                 delete.setToolTipText(Messages.getString("MainFrame.33")); //$NON-
NLS-1$
428:                 delete.addActionListener(controller.actionDelete);
429:                 toolBar.add(delete);
430:                 //
431:                 toolBar.addSeparator();
432:                 //
433:                 aide.setFocusable(false);
434:                 aide.setToolTipText(Messages.getString("MainFrame.34")); //$NON-NL
S-1$
435:                 aide.addActionListener(controller.actionHelp);
436:                 toolBar.add(aide);
437:                 //
438:                 toolBar.add(Box.createGlue());
439:                 //
440:                 deleteTab.setFocusable(false);
441:                 deleteTab.setToolTipText(Messages.getString("MainFrame.35")); //$N
ON-NLS-1$
442:                 deleteTab.addActionListener(controller.actionRemoveTab);
443:                 toolBar.add(deleteTab);
444:
445:                 return toolBar;
446:         }
447:
448:         public Object addTab(Object object) {
449:                 for (int i = 0; i < tabbedPane.getTabCount(); i++) {
450:                         if (tabbedPane.getTitleAt(i).equals(object.toString())) {
451:                                 tabbedPane.setSelectedIndex(i);
452:                                 return object;
453:                         }
454:                 }
455:
456:                 if (object.getClass() == Brick.class) {
457:                         popUpHelp = new PopUpHelpView();
458:                         popUpHelp.setView(PopUpHelpView.showBrickHelp());
459:                         tabbedPane.addTab(object.toString(), myPanel(object));
460:                         tabbedPane.setSelectedIndex(tabbedPane.getTabCount() - 1);
461:                 }  else if (object.getClass() == ExportModel.class) {
462:                         ExportModel exportModel = (ExportModel)object;
463:
464:
465:                         JPanel mainPanel = new JPanel();
466:                         ExportView eVTemp = new ExportView(exportModel, new TreeLi
stener(),
467:                                         new ExportPopUp(exportModel, mainPanel));
468:
469:                         tabbedPane.add(exportModel.getName(), myPanel(eVTemp, main
Panel));
470:                         tabbedPane.setSelectedIndex(tabbedPane.getTabCount() - 1);

471:                 } else {
472:                         DialogHandlerFrame
473:                                         .showErrorDialog(Messages.getString("MainF
rame.36")); //$NON-NLS-1$
474:                 }
475:                 return object;
476:         }
477:
478:         public Component getTabByName(String name)
479:         {
480:                 for (int i = 0; i < tabbedPane.getTabCount(); i++)
481:                 {
482:                         if (tabbedPane.getTitleAt(i).equals(name))
483:                         {
484:                                 return tabbedPane.getComponentAt(i);
485:                         }
```

```
  486:                 }
  487:                 return null;
  488:         }
  489:
  490:         public boolean closeTab(String name)
  491:         {
  492:
  493:                 for (int i = 0; i < tabbedPane.getTabCount(); i++)
  494:                 {
  495:                         if (tabbedPane.getTitleAt(i).equals(name))
  496:                         {
  497:                                 tabbedPane.removeTabAt(i);
  498:                                 System.out.println("Fermeture d'un onglet"); //$NO
N-NLS-1$
  499:                                 return true;
  500:                         }
  501:                         else
  502:                         {
  503:                                 System.out.println("Titre de l'onglet : "+tabbedPa
ne.getTitleAt(i)+" et titre recherchÃ© : "+name); //$NON-NLS-1$ //$NON-NLS-2$
  504:                         }
  505:                 }
  506:                 return false;
  507:
  508:         }
  509:
  510:         public void clearTabsAndShow(String title, JPanel panel)
  511:         {
  512:                 tabbedPane.removeAll();
  513:                 tabbedPane.addTab(title, panel);
  514:         }
  515:
  516:         protected void wouldYouLikeToSaveAndExit() {
  517:                 SavableObject obj = datapack;
  518:
  519:                 if(Program.isTriades()) {
  520:                         ConfigTriades.getInstance().save();
  521:                 } else {
  522:                         ConfigCreator.getInstance().save();
  523:                 }
  524:
  525:                 if (obj != null) {
  526:                         setAlwaysOnTop(true);
  527:                         int choice = DialogHandlerFrame.showYesNoCancelDialog(this
,
  528:                                         Messages.getString("MainFrame.40")); //$NO
N-NLS-1$
  529:                         if (choice == JOptionPane.YES_OPTION) {
  530:                                 if(!Program.isTriades()) {
  531:                                         Program.save(obj, true);
  532:                                         ConfigCreator.getInstance().getLastDatapac
k().addLastObject(obj.getFilePath());
  533:                                         ConfigCreator.getInstance().save();
  534:                                 } else {
  535:                                         Program.save(obj, true);
  536:                                 }
  537:                                 System.exit(1);
  538:                         } else if (choice == JOptionPane.NO_OPTION) {
  539:                                 System.exit(1);
  540:                         }
  541:                         setAlwaysOnTop(false);
  542:                 } else
  543:                         System.exit(1);
  544:         }
  545:
  546:         public JTreeActors getMainJTree() {
  547:                 return mainJta;
  548:         }
  549:
  550:         public Integer getTabLength() {
  551:                 return new Integer(tabbedPane.getComponentCount());
  552:         }
  553:
  554:         public DataPack getDataPack() {
  555:                 return datapack;
  556:         }
  557:
  558:         public abstract void initialState(MainFrame mf);
  559:
  560:         public abstract void setDataPack(DataPack dtp);
  561:
  562:         public abstract BrickView getActiveModelView();
  563:
  564:         public void showMainTab() {
  565:                 tabbedPane.setSelectedIndex(0);
  566:
  567:         }
  568:
  569:         public boolean showTabByName(String name) {
  570:                 for (int i = 0; i < tabbedPane.getTabCount(); i++)
  571:                 {
  572:                         if (tabbedPane.getTitleAt(i).equals(name))
  573:                         {
  574:                                 tabbedPane.setSelectedIndex(i);
  575:                                 return true;
  576:                         }
  577:                 }
  578:                 return false;
  579:         }
  580: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.awt.BorderLayout;
    4: import java.awt.Color;
    5: import java.awt.GridLayout;
    6: import java.awt.Image;
    7: import java.awt.Toolkit;
    8: import java.awt.event.ActionEvent;
    9: import java.awt.event.ActionListener;
   10: import java.awt.event.MouseEvent;
   11: import java.awt.event.MouseListener;
   12: import java.io.File;
   13: import java.util.Vector;
   14:
   15: import javax.swing.BorderFactory;
   16: import javax.swing.Box;
   17: import javax.swing.BoxLayout;
   18: import javax.swing.ButtonGroup;
   19: import javax.swing.JButton;
   20: import javax.swing.JComboBox;
   21: import javax.swing.JFrame;
   22: import javax.swing.JLabel;
   23: import javax.swing.JList;
   24: import javax.swing.JOptionPane;
   25: import javax.swing.JPanel;
   26: import javax.swing.JRadioButton;
   27: import javax.swing.JScrollPane;
   28: import javax.swing.JSeparator;
   29: import javax.swing.JTextArea;
   30: import javax.swing.JTextField;
   31:
   32: import models.Brick;
   33: import tools.ConfigCreator;
   34: import translation.Messages;
   35: import dataPack.Activite;
   36: import dataPack.DataPack;
   37:
   38: public class OptionNewFrameDatapackCreator extends JFrame {
   39:
   40:         private static final long serialVersionUID = -3406404251988354757L;
   41:
   42:         protected JRadioButton datapack;
   43:         protected JRadioButton brick;
   44:         protected JRadioButton newFile;
   45:         protected JRadioButton openFile;
   46:         protected JRadioButton lastFile;
   47:         protected JTextArea description;
   48:         protected JTextArea descriptionBis;
   49:         private boolean isStepTwo;
   50:         private final JPanel panelMain;
   51:         private final JPanel panelInit;
   52:         private JList pathsList;
   53:
   54:         public OptionNewFrameDatapackCreator(MainFrame mf, DataPack dtp) {
   55:                 isStepTwo = false;
   56:                 setAlwaysOnTop(true);
   57:                 setTitle(Messages.getString("OptionNewFrameDatapackCreator.0")); /
/ titre //$NON-NLS-1$
   58:                 Image icone = Toolkit.getDefaultToolkit().getImage(
   59:                 "Icones/16x16/triades.png"); //$NON-NLS-1$
   60:                 setIconImage(icone);
   61:                 setSize(400, 400); // taille
   62:                 setLocationRelativeTo(mf); // fenÃªtre centrÃ©e sur la main frame
   63:                 setResizable(true); // non redimensionnable
   64:                 setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE); // la croix quitte
   65:                 // l'application
   66:
   67:                 panelMain = this.myPanel(mf, dtp);
   68:                 panelInit = this.getInitialStep(mf);
   69:                 getContentPane().add(this.myPanel(mf, dtp));
   70:
   71:         }
   72:
   73:         private JPanel myPanel(final MainFrame mf, final DataPack dtp) {
   74:                 // DÃ©clarations + initialisations
   75:                 final JPanel panel = new JPanel(new BorderLayout());
   76:                 JPanel north = new JPanel(new BorderLayout());
   77:                 final JPanel center = new JPanel(new BorderLayout());
   78:                 JPanel south = new JPanel(new BorderLayout());
   79:                 JPanel pTitle = new JPanel();
   80:                 final JPanel pChoices = new JPanel(new GridLayout(3, 2));
   81:                 JPanel pButtons = new JPanel();
   82:                 JPanel p2Buttons = new JPanel(new GridLayout(1, 2));
   83:                 description = new JTextArea(
   84:                                 Messages.getString("OptionNewFrameDatapackCreator.
2"), //$NON-NLS-1$
   85:                                 4, 0);
   86:                 BoxLayout layoutButtons = new BoxLayout(pButtons, BoxLayout.X_AXIS
);
   87:                 BoxLayout layoutTitle = new BoxLayout(pTitle, BoxLayout.X_AXIS);
   88:                 JButton ok = new JButton(Messages.getString("OptionNewFrameDatapac
kCreator.3")); //$NON-NLS-1$
   89:                 JButton cancel = new JButton(Messages.getString("OptionNewFrameDat
apackCreator.4")); //$NON-NLS-1$
   90:                 datapack = new JRadioButton(Messages.getString("OptionNewFrameData
packCreator.5"), true); //$NON-NLS-1$
   91:                 brick = new JRadioButton(Messages.getString("OptionNewFrameDatapac
kCreator.6"), false); //$NON-NLS-1$
   92:                 final ButtonGroup group = new ButtonGroup();
   93:                 final JLabel iconTitle = new JLabel(IconDatabase.iconTitleNew);
   94:                 final JTextField newBrickName = new JTextField();
   95:                 JComboBox brickSteps = null;
   96:                 JComboBox activity = null;
   97:                 JPanel pStepTwoBrick = null;
   98:                 final JPanel pStepTwoDataPack = new JPanel();
   99:                 JPanel pDataPack = new JPanel();
  100:                 final JTextField newDataPackName = new JTextField(20);
  101:                 JLabel iconBrick = new JLabel(IconDatabase.iconBrick);
  102:
  103:                 // Suivant si on a un datapack
  104:                 if (dtp == null) {
  105:
  106:                         brick.setEnabled(false);
  107:                         iconBrick.setEnabled(false);
  108:                 } else {
  109:                         brickSteps = new JComboBox(mf.datapack.getSteps());
  110:                         activity = new JComboBox(mf.datapack.getActivities().getAc
tivities());
  111:                         pStepTwoBrick = getStepTwoBrick(mf, brickSteps, activity,
  112:                                         newBrickName);
  113:                 }
  114:
  115:                 // Etape 2 de DATAPACK
  116:                 pDataPack.add(Box.createHorizontalStrut(20));
  117:                 pDataPack.add(new JLabel(Messages.getString("OptionNewFrameDatapac
kCreator.7"))); //$NON-NLS-1$
  118:                 pDataPack.add(Box.createHorizontalStrut(20));
  119:                 pDataPack.add(newDataPackName);
  120:                 pDataPack.add(Box.createHorizontalStrut(20));
  121:                 pStepTwoDataPack.add(Box.createVerticalStrut(175));
  122:                 pStepTwoDataPack.add(pDataPack);
  123:                 pStepTwoDataPack.add(Box.createVerticalStrut(170));
  124:
  125:                 // Barre de titre
```

```
  126:                 pTitle.setBackground(Color.WHITE);
  127:                 pTitle.setLayout(layoutTitle);
  128:                 pTitle.add(Box.createHorizontalStrut(10));
  129:                 pTitle.add(new JLabel(Messages.getString("OptionNewFrameDatapackCr
eator.8"))); //$NON-NLS-1$
  130:                 pTitle.add(Box.createGlue());
  131:                 pTitle.add(iconTitle);
  132:                 north.add(pTitle, BorderLayout.CENTER);
  133:                 north.add(new JSeparator(), BorderLayout.SOUTH);
  134:
  135:                 // Zone des choix
  136:                 datapack.addItemListener(mf.controller.itemListener);
  137:                 brick.addItemListener(mf.controller.itemListener);
  138:                 group.add(datapack);
  139:                 group.add(brick);
  140:                 pChoices.add(new JLabel(IconDatabase.iconDatapack));
  141:                 pChoices.add(datapack);
  142:                 pChoices.add(iconBrick);
  143:                 pChoices.add(brick);
  144:                 description.setBackground(this.getBackground());
  145:                 description.setBorder(BorderFactory.createCompoundBorder(BorderFac
tory
  146:                                 .createEmptyBorder(0, 0, 2, 0), BorderFactory
  147:                                 .createTitledBorder(Messages.getString("OptionNewF
rameDatapackCreator.9")))); //$NON-NLS-1$
  148:                 description.setEditable(false);
  149:                 center.add(pChoices, BorderLayout.CENTER);
  150:                 center.add(description, BorderLayout.SOUTH);
  151:
  152:                 final OptionNewFrameDatapackCreator frame = this;
  153:                 final JComboBox brickStepsFin = brickSteps;
  154:                 final JComboBox activityFin = activity;
  155:                 final JPanel pStepTwoBrickFin = pStepTwoBrick;
  156:
  157:                 // Barre des boutons
  158:                 ok.addActionListener(new ActionListener() {
  159:
  160:                         @Override
  161:                         public void actionPerformed(ActionEvent arg0) {
  162:                                 if (isStepTwo) {
  163:                                         boolean isAllowedToProceed = true;
  164:                                         if (brick.isSelected()) {
  165:                                                 if (newBrickName.getText().length(
) > 0
  166:                                                                 && brickStepsFin.g
etSelectedItem() != null
  167:                                                                 && activityFin.get
SelectedItem() != null) {
  168:                                                         Brick b = new Brick((Strin
g) brickStepsFin
  169:                                                                         .getSelect
edItem(), newBrickName.getText(),
  170:                                                                         mf.datapac
k, (Activite) activityFin
  171:                                                                         .getSelect
edItem());
  172:                                                         mf.datapack.getBrickList()
.add(b);
  173:                                                         mf.addTab(b);
  174:                                                 } else {
  175:                                                         DialogHandlerFrame.showErr
orDialog(frame,
  176:                                                                         Messages.g
etString("OptionNewFrameDatapackCreator.10")); //$NON-NLS-1$
  177:                                                         isAllowedToProceed = false
;
  178:                                                 }
  179:                                         } else if (datapack.isSelected()) {
  180:                                                 if (newDataPackName.getText().leng
th() > 0) {
  181:                                                         if (mf.datapack != null) {
  182:                                                                 int choice = Dialo
gHandlerFrame
  183:                                                                         .showYesNoCancelDi
alog(frame,
  184:                                                                                 Me
ssages.getString("OptionNewFrameDatapackCreator.11")); //$NON-NLS-1$
  185:                                                                 switch (choice) {
  186:                                                                 case JOptionPane.Y
ES_OPTION:
  187:                                                                         if (!Progr
am.save(Program.myMainFrame.datapack, true)) {
  188:                                                                                 Sy
stem.out
  189:
        .println("OptionNewFrameDatapackCreator.myPanel(...).new ActionListener() {...}.a
ctionPerformed()");
  190:                                                                                 Sy
stem.out.println("Impossible de sauvegarder le datapack");
  191:                                                                                 re
turn;
  192:                                                                         }
  193:                                                                         break;
  194:                                                                 case JOptionPane.C
ANCEL_OPTION:
  195:                                                                         return;
  196:                                                                 default:
  197:                                                                         break;
  198:
  199:                                                                 }
  200:                                                                 Program.myMainFram
e.setDataPack(null);
  201:                                                                 Program.myMainFram
e.tabbedPane.removeAll();
  202:                                                         }
  203:                                                         Program.myMainFrame.setDat
aPack(new DataPack(
  204:                                                                         newDataPac
kName.getText()));
  205:                                                 } else {
  206:                                                         DialogHandlerFrame.showErr
orDialog(frame,
  207:                                                                         Messages.g
etString("OptionNewFrameDatapackCreator.12")); //$NON-NLS-1$
  208:                                                         isAllowedToProceed = false
;
  209:                                                 }
  210:                                         }
  211:
  212:                                         if (isAllowedToProceed) {
  213:                                                 setVisible(false);
  214:                                                 isStepTwo = false;
  215:                                                 getStepOne(iconTitle, center, pCho
ices);
  216:                                         }
  217:                                 } else {
  218:                                         center.removeAll();
  219:                                         getContentPane().repaint();
  220:                                         if (brick.isSelected()) {
  221:                                                 if (dtp.getSteps().size() > 0
  222:                                                                 && dtp.getActiviti
es().getNbActivites() > 0) {
  223:                                                         iconTitle.setIcon(IconData
base.iconTitleNewBrick);
```

```
224:                                                    center.add(pStepTwoBrickFi
n, BorderLayout.CENTER);
225:                                                    if (activityFin.getItemCou
nt() < Program.myMainFrame.datapack
226:                                                            .getActivi
ties().getNbActivites()) {
227:                                                            activityFin.remove
AllItems();
228:                                                            for (Activite act
: Program.myMainFrame.datapack
229:                                                                    .g
etActivities().getActivities()) {
230:                                                                    activityFi
n.addItem(act);
231:                                                            }
232:                                                    }
233:                                                    newBrickName.setText(null)
;
234:                                                    brickStepsFin.setSelectedI
ndex(0);
235:                                                    activityFin.setSelectedInd
ex(0);
236:                                                    description
237:                                                    .setText(Messages.getStrin
g("OptionNewFrameDatapackCreator.13")); //$NON-NLS-1$
238:                                            } else {
239:                                                    // Il faut au moins avoir
une activitÃ© et une Ã©tape
240:                                                    // Ce n'est pas le cas don
c on ferme l'assistant de
241:                                                    // crÃ©ation pour laisser
l'utilisateur rÃ©ctifier ce
242:                                                    // problÃ¨me
243:
244:                                                    DialogHandlerFrame
245:                                                    .showErrorDialog(Messages.
getString("OptionNewFrameDatapackCreator.14")); //$NON-NLS-1$
246:
247:                                                    setVisible(false);
248:                                                    isStepTwo = false;
249:                                                    getStepOne(iconTitle, cent
er, pChoices);
250:                                                    return;
251:                                            }
252:                                    } else if (datapack.isSelected()) {
253:                                            iconTitle.setIcon(IconDatabase.ico
nTitleDataPack);
254:                                            center.add(pStepTwoDataPack, Borde
rLayout.CENTER);
255:                                            newDataPackName.setText(null);
256:                                            description
257:                                            .setText(Messages.getString("Optio
nNewFrameDatapackCreator.15")); //$NON-NLS-1$
258:
259:                                    }
260:                                    center.add(description, BorderLayout.SOUTH
);
261:                                    getContentPane().validate();
262:                                    isStepTwo = true;
263:                            }
264:
265:                    }
266:
267:            });
268:            cancel.addActionListener(new ActionListener() {
269:
270:                    @Override
271:                    public void actionPerformed(ActionEvent e) {
272:                            if (!isStepTwo) {
273:                                    setVisible(false);
274:                            } else {
275:                                    getStepOne(iconTitle, center, pChoices);
276:                            }
277:                    }
278:
279:            });
280:            p2Buttons.add(ok);
281:            p2Buttons.add(cancel);
282:            pButtons.setLayout(layoutButtons);
283:            pButtons.add(Box.createHorizontalStrut(100));
284:            pButtons.add(p2Buttons);
285:            pButtons.add(Box.createHorizontalStrut(100));
286:            south.add(new JSeparator(), BorderLayout.NORTH);
287:            south.add(pButtons, BorderLayout.CENTER);
288:
289:            // Panel global
290:            panel.add(north, BorderLayout.NORTH);
291:            panel.add(center, BorderLayout.CENTER);
292:            panel.add(south, BorderLayout.SOUTH);
293:
294:            return panel;
295:    }
296:
297:    private JPanel getStepTwoBrick(MainFrame mf, JComboBox brickSteps,
298:                    JComboBox activity, JTextField newBrickName) {
299:            setSize(500,500);
300:            JPanel panel = new JPanel();
301:            JPanel pBrickType = new JPanel();
302:            JPanel pActivity = new JPanel();
303:            JPanel pName = new JPanel();
304:
305:            brickSteps.validate();
306:
307:            panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
308:            pBrickType.setLayout(new BoxLayout(pBrickType, BoxLayout.X_AXIS));
309:            pActivity.setLayout(new BoxLayout(pActivity, BoxLayout.X_AXIS));
310:            pName.setLayout(new BoxLayout(pName, BoxLayout.X_AXIS));
311:
312:            pBrickType.add(Box.createHorizontalStrut(71));
313:            pBrickType.add(new JLabel(Messages.getString("OptionNewFrameDatapa
ckCreator.16"))); //$NON-NLS-1$
314:            pBrickType.add(Box.createHorizontalStrut(50));
315:            pBrickType.add(brickSteps);
316:            pBrickType.add(Box.createHorizontalStrut(20));
317:
318:            pActivity.add(Box.createHorizontalStrut(88));
319:            pActivity.add(new JLabel(Messages.getString("OptionNewFrameDatapac
kCreator.17"))); //$NON-NLS-1$
320:            pActivity.add(Box.createHorizontalStrut(50));
321:            pActivity.add(activity);
322:            pActivity.add(Box.createHorizontalStrut(20));
323:
324:            pName.add(Box.createHorizontalStrut(20));
325:            pName.add(new JLabel(Messages.getString("OptionNewFrameDatapackCre
ator.18"))); //$NON-NLS-1$
326:            pName.add(Box.createHorizontalStrut(50));
327:            pName.add(newBrickName);
328:            pName.add(Box.createHorizontalStrut(20));
329:
330:            panel.add(Box.createVerticalStrut(52));
331:            panel.add(pBrickType);
332:            panel.add(Box.createVerticalStrut(10));
333:            panel.add(pActivity);
334:            panel.add(Box.createVerticalStrut(10));
```

```
  335:                 panel.add(pName);
  336:                 panel.add(Box.createVerticalStrut(52));
  337:
  338:                 return panel;
  339:
  340:         }
  341:
  342:         private void getStepOne(JLabel iconTitle, JPanel center, JPanel pChoices)
{
  343:                 setSize(400,400);
  344:                 iconTitle.setIcon(IconDatabase.iconTitleNew);
  345:                 center.removeAll();
  346:                 getContentPane().repaint();
  347:                 datapack.setSelected(true);
  348:                 center.add(pChoices, BorderLayout.CENTER);
  349:                 description
  350:                 .setText(Messages.getString("OptionNewFrameDatapackCreator.19"));
//$NON-NLS-1$
  351:                 center.add(description, BorderLayout.SOUTH);
  352:                 getContentPane().validate();
  353:                 isStepTwo = false;
  354:         }
  355:
  356:         private String getSelectedLastPath() {
  357:                 if(pathsList.getSelectedIndex() >= 0) {
  358:                         return ConfigCreator.getInstance().getLastDatapack().getLa
stObjects().elementAt(pathsList.getSelectedIndex());
  359:                 } else {
  360:                         return null;
  361:                 }
  362:         }
  363:
  364:         private JPanel getInitialStep(MainFrame mf) {
  365:                 // Declarations
  366:                 final JPanel panel = new JPanel(new BorderLayout());
  367:                 JPanel north = new JPanel(new BorderLayout());
  368:                 JPanel center = new JPanel(new BorderLayout());
  369:                 JPanel south = new JPanel(new BorderLayout());
  370:                 JPanel pTitle = new JPanel();
  371:                 JPanel pChoices = new JPanel(new GridLayout(0, 2));
  372:                 JPanel pButtons = new JPanel();
  373:                 JPanel p2Buttons = new JPanel(new GridLayout(1, 2));
  374:                 ButtonGroup group = new ButtonGroup();
  375:                 JButton ok = new JButton(Messages.getString("OptionNewFrameDatapac
kCreator.20")); //$NON-NLS-1$
  376:                 JButton cancel = new JButton(Messages.getString("OptionNewFrameDat
apackCreator.21")); //$NON-NLS-1$
  377:                 JPanel descriptionBisPanel = new JPanel(new BorderLayout());
  378:                 newFile = new JRadioButton(Messages.getString("OptionNewFrameDatap
ackCreator.22")); //$NON-NLS-1$
  379:                 openFile = new JRadioButton(Messages.getString("OptionNewFrameData
packCreator.23")); //$NON-NLS-1$
  380:                 lastFile = new JRadioButton("Dernier datapck utilisÃ©s");
  381:                 descriptionBis = new JTextArea(Messages.getString("OptionNewFrameD
atapackCreator.24"), 2, 0); //$NON-NLS-1$
  382:                 descriptionBis.setEditable(false);
  383:                 descriptionBisPanel.setBorder(BorderFactory.createCompoundBorder(
  384:                                 BorderFactory.createEmptyBorder(0, 0, 2, 0), Borde
rFactory
  385:                                 .createTitledBorder(Messages.getString("OptionNewF
rameDatapackCreator.25")))); //$NON-NLS-1$
  386:                 descriptionBisPanel.setOpaque(false);
  387:                 descriptionBisPanel.add(descriptionBis);
  388:
  389:                 // Barre de titre
  390:                 pTitle.setLayout(new BoxLayout(pTitle, BoxLayout.X_AXIS));
  391:                 pTitle.setBackground(Color.WHITE);
  392:                 pTitle.add(Box.createHorizontalStrut(10));
  393:                 pTitle.add(new JLabel(Messages.getString("OptionNewFrameDatapackCr
eator.26"))); //$NON-NLS-1$
  394:                 pTitle.add(Box.createGlue());
  395:                 pTitle.add(new JLabel(IconDatabase.iconTitleTriad));
  396:                 north.add(pTitle, BorderLayout.CENTER);
  397:                 north.add(new JSeparator(), BorderLayout.SOUTH);
  398:
  399:                 // Centre
  400:                 newFile.setSelected(true);
  401:                 newFile.addItemListener(mf.controller.itemListener);
  402:                 openFile.addItemListener(mf.controller.itemListener);
  403:                 lastFile.addItemListener(mf.controller.itemListener);
  404:                 group.add(newFile);
  405:                 group.add(openFile);
  406:                 group.add(lastFile);
  407:                 pChoices.add(Box.createGlue());
  408:                 pChoices.add(Box.createGlue());
  409:                 pChoices.add(new JLabel(IconDatabase.iconNewFile));
  410:                 pChoices.add(newFile);
  411:                 pChoices.add(new JLabel(IconDatabase.iconOpenFile));
  412:                 pChoices.add(openFile);
  413:                 if(ConfigCreator.getInstance().getLastDatapack().getLastObjects().
size() > 0) {
  414:                         pChoices.add(new JLabel(IconDatabase.iconDatapack));
  415:                         pChoices.add(lastFile);
  416:                 }
  417:                 pChoices.add(Box.createGlue());
  418:                 pChoices.add(Box.createGlue());
  419:
  420:                 JPanel centerPanel = new JPanel();
  421:                 centerPanel.setLayout(new BoxLayout(centerPanel, BoxLayout.Y_AXIS)
);
  422:                 centerPanel.add(pChoices);
  423:
  424:                 Vector<String> lastPath = ConfigCreator.getInstance().getLastDatap
ack().getLastObjects();
  425:
  426:                 if(lastPath.size() > 0) {
  427:                         Vector<String> names = new Vector<String>();
  428:                         for(int i = 0 ; i < lastPath.size() ; i++) {
  429:                                 File file = new File(lastPath.elementAt(i));
  430:                                 String name = "...\\"; //$NON-NLS-1$
  431:                                 if(file.getParentFile() != null) {
  432:                                         name = file.getParentFile().getName() + "\
\"; //$NON-NLS-1$
  433:                                 }
  434:
  435:                                 if(file.getName().indexOf('.') >= 0) {
  436:                                         name += file.getName().substring(0, file.g
etName().lastIndexOf(".")); //$NON-NLS-1$
  437:                                 } else {
  438:                                         name += file.getName();
  439:                                 }
  440:
  441:                                 names.add(name);
  442:                         }
  443:
  444:                         pathsList = new JList(names);
  445:                         pathsList.setLayout(new BorderLayout());
  446:
  447:                         pathsList.addMouseListener(new MouseListener() {
  448:                                 @Override
  449:                                 public void mouseReleased(MouseEvent e) {}
  450:
  451:                                 @Override
  452:                                 public void mousePressed(MouseEvent e) {}
```

```
  453:
  454:                                @Override
  455:                                public void mouseExited(MouseEvent e) {}
  456:
  457:                                @Override
  458:                                public void mouseEntered(MouseEvent e) {}
  459:
  460:                                @Override
  461:                                public void mouseClicked(MouseEvent e) {
  462:                                        lastFile.setSelected(true);
  463:                                        if(e.getClickCount() > 1) {
  464:                                                setVisible(false);
  465:                                                String test = getSelectedLastPath(
);
  466:                                                DataPack dtp = (DataPack) Program.
loadSavableObject(getSelectedLastPath(), true);
  467:                                                Program.myMainFrame.setDataPack(dt
p);
  468:                                        }
  469:                                }
  470:                        });
  471:
  472:                        JScrollPane pathsListPanel = new JScrollPane(pathsList);
  473:                        pathsListPanel.setOpaque(false);
  474:                        pathsListPanel.setBorder(BorderFactory.createTitledBorder(
Messages.getString("OptionNewFrameDatapackCreator.30"))); //$NON-NLS-1$
  475:
  476:                        centerPanel.add(pathsListPanel);
  477:
  478:                }
  479:
  480:                center.add(centerPanel, BorderLayout.CENTER);
  481:                center.add(descriptionBisPanel, BorderLayout.SOUTH);
  482:
  483:                // Sud
  484:                ok.addActionListener(new ActionListener() {
  485:
  486:                        @Override
  487:                        public void actionPerformed(ActionEvent e) {
  488:                                if(lastFile.isSelected()) {
  489:                                        String path = getSelectedLastPath();
  490:                                        if(path != null) {
  491:                                                setVisible(false);
  492:                                                DataPack dtp = (DataPack) Program.
loadSavableObject(path, true);
  493:                                                Program.myMainFrame.setDataPack(dt
p);
  494:                                        }
  495:                                } else if (newFile.isSelected()) {
  496:                                        panel.setVisible(false);
  497:                                        getContentPane().add(panelMain);
  498:                                        // panelMain.setVisible(true);
  499:                                } else if (openFile.isSelected()) {
  500:                                        setVisible(false);
  501:                                        DataPack dtp = (DataPack) Program.loadSava
bleObject(null, true);
  502:                                        Program.myMainFrame.setDataPack(dtp);
  503:                                }
  504:                        }
  505:
  506:                });
  507:                cancel.addActionListener(new ActionListener() {
  508:
  509:                        @Override
  510:                        public void actionPerformed(ActionEvent e) {
  511:                                panel.setVisible(false);
  512:                                getContentPane().add(panelMain);
  513:                                setVisible(false);
  514:                        }
  515:
  516:                });
  517:                p2Buttons.add(ok);
  518:                p2Buttons.add(cancel);
  519:                pButtons.add(Box.createHorizontalStrut(100));
  520:                pButtons.add(p2Buttons);
  521:                pButtons.add(Box.createHorizontalStrut(100));
  522:                south.add(new JSeparator(), BorderLayout.NORTH);
  523:                south.add(pButtons, BorderLayout.CENTER);
  524:
  525:                // Panel total
  526:                panel.add(north, BorderLayout.NORTH);
  527:                panel.add(center, BorderLayout.CENTER);
  528:                panel.add(south, BorderLayout.SOUTH);
  529:
  530:                return panel;
  531:        }
  532:
  533:        public void setInitialStep() {
  534:                setSize(400,475);
  535:                getContentPane().remove(0);
  536:                getContentPane().add(panelInit);
  537:        }
  538:
  539: }
```

```java
package graphicalUserInterface;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TreeMap;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.JTextField;
import javax.swing.JToolBar;

import models.Brick;
import models.BrickVertex.VerticeRank;
import translation.Messages;
import dataPack.Acteur;
import dataPack.ActeurSelectionne;
import dataPack.Content;
import dataPack.ListeActeurSelectionne;

public class ActorListView extends JPanel {

	private static final long serialVersionUID = -8042407423948049392L;

	private JPanel panelCenter;
	private int nbActors;
	private ListeActeurSelectionne list;
	private JPanel mainList;
	private JPanel secondaryList;

	public ActorListView(MainFrame mf) {
		nbActors = 1;
		list = new ListeActeurSelectionne();
		build(mf);
	}

	private void build(final MainFrame mf) {
		// Declarations + initialisations
		JPanel panelNorth = new JPanel(new BorderLayout());
		JPanel panelSouth = new JPanel(new BorderLayout());
		JPanel panelTitle = new JPanel();
		JPanel panelButtons = new JPanel();
		JPanel panelUnusedSpace = new JPanel(new BorderLayout());
		panelCenter = new JPanel();
		mainList = new JPanel();
		mainList.setLayout(new BoxLayout(mainList, BoxLayout.Y_AXIS));
		secondaryList = new JPanel();
		secondaryList.setLayout(new BoxLayout(secondaryList, BoxLayout.Y_AXIS));
		mainList.setBorder(BorderFactory.createTitledBorder("Acteurs principaux"));
		secondaryList.setBorder(BorderFactory.createTitledBorder("Acteurs secondaires"));
		JPanel mainGluePanel = new JPanel();
		mainGluePanel.add(Box.createVerticalStrut(5));
		mainList.add(mainGluePanel);
		JPanel secondaryGluePanel = new JPanel();
		secondaryGluePanel.add(Box.createVerticalStrut(5));
		secondaryList.add(secondaryGluePanel);



		final JTextField sessionName = new JTextField(Messages.getString("ActorListView.0") //$NON-NLS-1$
				+ (new SimpleDateFormat("dd MMM yyyy kk'h'mm")) //$NON-NLS-1$
				.format(new Date()));
		sessionName.setBorder(BorderFactory
				.createTitledBorder(Messages.getString("ActorListView.2"))); //$NON-NLS-1$
		JScrollPane scrollPane = new JScrollPane(panelUnusedSpace,
				JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
				JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
		JToolBar toolBarAddActor = new JToolBar();
		JButton buttonAddMainActor = new JButton(
				Messages.getString("ActorListView.3"), //$NON-NLS-1$
				IconDatabase.iconAddActor);
		JButton buttonAddSecondaryActor = new JButton("Ajouter un acteur secondaire", IconDatabase.iconAddActor);
		JButton buttonGenerate = new JButton(Messages.getString("ActorListView.4")); //$NON-NLS-1$



		// Barre de titre
		panelTitle.setLayout(new BoxLayout(panelTitle, BoxLayout.X_AXIS));
		panelTitle.setBackground(Color.WHITE);
		panelTitle.add(Box.createHorizontalStrut(10));
		panelTitle.add(sessionName);
		panelTitle.add(Box.createGlue());
		panelTitle.add(new JLabel(IconDatabase.iconActorSet));
		panelNorth.add(new JSeparator(), BorderLayout.NORTH);
		panelNorth.add(panelTitle, BorderLayout.CENTER);
		panelNorth.add(new JSeparator(), BorderLayout.SOUTH);

		// Panel central
		buttonAddMainActor.setFocusable(false);
		buttonAddSecondaryActor.setFocusable(false);
		buttonAddMainActor.addActionListener(new ActionListener() {
			@Override
			public void actionPerformed(ActionEvent arg0) {
				if (mf.mainJta.getListener().getSelectedActors() == null) {
					DialogHandlerFrame.showInformationDialog(
							Program.myMainFrame, Messages.getString("ActorListView.5"), //$NON-NLS-1$
							Messages.getString("ActorListView.6"), null); //$NON-NLS-1$
					return;
				}
				for (Acteur actor : mf.mainJta.getListener()
						.getSelectedActors()) {
					addActorToView(new ActeurSelectionne(actor,VerticeRank.primary),
							mf);
				}
				panelCenter.getParent().validate();
			}
```

```
122:                 });
123:                 buttonAddSecondaryActor.addActionListener(new ActionListener() {
124:                         @Override
125:                         public void actionPerformed(ActionEvent arg0) {
126:                                 if (mf.mainJta.getListener().getSelectedActors() =
= null) {
127:                                         DialogHandlerFrame.showInformationDialog(
128:                                                         Program.myMainFrame, Messa
ges.getString("ActorListView.5"), //$NON-NLS-1$
129:                                                         Messages.getString("ActorL
istView.6"), null); //$NON-NLS-1$
130:                                         return;
131:                                 }
132:                                 for (Acteur actor : mf.mainJta.getListener()
133:                                                 .getSelectedActors()) {
134:                                         addActorToView(new ActeurSelectionne(actor
,VerticeRank.secondary),
135:                                                         mf);
136:                                 }
137:                                 panelCenter.getParent().validate();
138:                         }
139:                 });
140:                 toolBarAddActor.setFloatable(false);
141:                 toolBarAddActor.add(buttonAddMainActor);
142:                 toolBarAddActor.add(buttonAddSecondaryActor);
143:                 toolBarAddActor.add(new JLabel());
144:                 toolBarAddActor.add(Box.createHorizontalGlue());
145:                 panelCenter.setLayout(new BoxLayout(panelCenter, BoxLayout.Y_AXIS)
);
146:                 panelCenter.add(toolBarAddActor);
147:                 panelCenter.add(new JSeparator());
148:                 panelCenter.add(mainList);
149:                 panelCenter.add(secondaryList);
150:
151:                 // Barre de boutons
152:                 panelButtons.setLayout(new BoxLayout(panelButtons, BoxLayout.X_AXI
S));
153:                 panelButtons.add(Box.createGlue());
154:                 buttonGenerate.setFocusable(false);
155:                 buttonGenerate.addActionListener(new ActionListener() {
156:                         @Override
157:                         public void actionPerformed(ActionEvent arg0) {
158:                                 TreeMap<Content, VerticeRank> selectedActors = new
 TreeMap<Content, VerticeRank>();
159:                                 for (ActeurSelectionne aS : list.getActorsSelectio
n())
160:                                         selectedActors.put(aS.getActeur(), aS.getR
ank());
161:
162:                                 client.Session newSession = mf.getDataPack()
163:                                 .getExportModule()
164:                                 .addNewSession(sessionName.getText(), selectedActo
rs);
165:
166:                                 ((MainFrameTriades) mf).displayObject(newSession);
167:                                 mf.getDataPack().setCurrentSession(newSession);
168:                         }
169:                 });
170:                 panelButtons.add(buttonGenerate);
171:
172:                 panelButtons.add(Box.createGlue());
173:                 panelSouth.add(new JSeparator(), BorderLayout.NORTH);
174:                 panelSouth.add(panelButtons, BorderLayout.CENTER);
175:
176:                 // Panel intermÃ©diaire permettant Ã  panelCenter de ne pas prendr
e tout
177:                 // l'espace
178:                 panelUnusedSpace.add(panelCenter, BorderLayout.NORTH);
179:
180:                 // Panel total
181:                 this.setLayout(new BorderLayout());
182:                 this.add(panelNorth, BorderLayout.NORTH);
183:                 scrollPane.getVerticalScrollBar().setUnitIncrement(50);
184:                 this.add(scrollPane, BorderLayout.CENTER);
185:                 this.add(panelSouth, BorderLayout.SOUTH);
186:         }
187:
188:         public void addActorToView(final ActeurSelectionne actor, final MainFrame
mf) {
189:                 list.ajouterActeurSelectionne(actor);
190:                 Color color = nbActors % 2 == 0 ? new Color(249, 249, 249) : new C
olor(
191:                                 229, 229, 229);
192:
193:                 // DÃ©clarations et initialisations
194:                 final JPanel panelActor = new JPanel();
195:                 panelActor.setBackground(color);
196:                 final JPanel panel = new JPanel(new BorderLayout());
197:                 panel.add(panelActor, BorderLayout.CENTER);
198:                 final JLabel buttonDeleteActor = new JLabel(
199:                                 IconDatabase.iconRemoveActor0);
200:
201:                 // Construction des Ã©lÃ©ments du panel
202:                 buttonDeleteActor.setToolTipText(Messages.getString("ActorListView
.8")); //$NON-NLS-1$
203:                 buttonDeleteActor.addMouseListener(new MouseListener() {
204:                         @Override
205:                         public void mouseClicked(MouseEvent arg0) {
206:                                 buttonDeleteActor.setIcon(IconDatabase.iconRemoveA
ctor2);
207:                                 if (actor.getRank() == VerticeRank.primary)
208:                                 {
209:                                         removeMainActor(panel);
210:                                 }
211:                                 else if (actor.getRank() == VerticeRank.secondary)
212:                                         removeSecondaryActor(panel);
213:                                 panelCenter.getParent().validate();
214:                                 nbActors--;
215:                                 mf.mainJta.getListener().showActor(actor.getActeur
());
216:                         }
217:
218:                         @Override
219:                         public void mouseEntered(MouseEvent arg0) {
220:                                 buttonDeleteActor.setIcon(IconDatabase.iconRemoveA
ctor1);
221:                         }
222:
223:                         @Override
224:                         public void mouseExited(MouseEvent arg0) {
225:                                 buttonDeleteActor.setIcon(IconDatabase.iconRemoveA
ctor0);
226:                         }
227:
228:                         @Override
229:                         public void mousePressed(MouseEvent arg0) {
230:                                 buttonDeleteActor.setIcon(IconDatabase.iconRemoveA
ctor2);
231:                         }
232:
233:                         @Override
234:                         public void mouseReleased(MouseEvent arg0) {
235:
236:                         }
```

```
237:                });
238:
239:                // Construction du panel
240:                panelActor.setLayout(new BoxLayout(panelActor, BoxLayout.X_AXIS));
241:                buttonDeleteActor.setPreferredSize(new Dimension(25, 25));
242:                buttonDeleteActor
243:                .setBorder(BorderFactory.createEmptyBorder(3, 0, 0, 0));
244:                panelActor.add(buttonDeleteActor);
245:                panelActor.add(Box.createHorizontalStrut(10));
246:                JLabel label = new JLabel(actor.getActeur().toString());
247:                label.setPreferredSize(new Dimension(100, 25));
248:                panelActor.add(label);
249:                panelActor.add(Box.createHorizontalGlue());
250:                panelActor.add(new JSeparator(JSeparator.VERTICAL));
251:                panelActor.add(Box.createHorizontalGlue());
252:
253:                ButtonGroup group = new ButtonGroup();
254:                final JRadioButton mainActor = new JRadioButton("Acteur principal
  ");
255:                JRadioButton secondaryActor = new JRadioButton("Acteur secondaire"
);
256:                group.add(mainActor);
257:                group.add(secondaryActor);
258:                mainActor.addItemListener(new ItemListener() {
259:
260:                        @Override
261:                        public void itemStateChanged(ItemEvent e) {
262:
263:                                if (e.getStateChange()==ItemEvent.SELECTED && acto
r.getRank() != VerticeRank.primary)
264:                                {
265:                                        removeSecondaryActor(panel);
266:                                        addMainActor(panel);
267:                                        actor.setRank(VerticeRank.primary);
268:                                }
269:
270:                        }
271:                });
272:                secondaryActor.addItemListener(new ItemListener() {
273:
274:                        @Override
275:                        public void itemStateChanged(ItemEvent e) {
276:
277:                                if (e.getStateChange()==ItemEvent.SELECTED && acto
r.getRank() != VerticeRank.secondary)
278:                                {
279:                                        removeMainActor(panel);
280:                                        addSecondaryActor(panel);
281:                                        actor.setRank(VerticeRank.secondary);
282:                                }
283:
284:                        }
285:                });
286:                panelActor.add(mainActor);
287:                panelActor.add(secondaryActor);
288:
289:                if (actor.getRank() == VerticeRank.primary)
290:                {
291:                        mainActor.setSelected(true);
292:                        addMainActor(panel);
293:                }
294:                else if (actor.getRank() == VerticeRank.secondary)
295:                {
296:
297:                        secondaryActor.setSelected(true);
298:                        addSecondaryActor(panel);
299:                }
300:                nbActors++;
301:                mf.mainJta.getListener().hideActor(actor.getActeur());
302:        }
303:
304:        protected void removeSecondaryActor(JPanel actor)
305:        {
306:                secondaryList.remove(actor);
307:                validate();
308:        }
309:
310:        protected void removeMainActor(JPanel actor)
311:        {
312:                mainList.remove(actor);
313:                validate();
314:        }
315:
316:        protected void addSecondaryActor(JPanel actor)
317:        {
318:                secondaryList.remove(actor);
319:                secondaryList.add(actor);
320:                validate();
321:        }
322:
323:        protected void addMainActor(JPanel actor)
324:        {
325:                mainList.remove(actor);
326:                mainList.add(actor);
327:                validate();
328:
329:        }
330:
331: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.awt.event.ActionEvent;
    4: import java.lang.reflect.InvocationTargetException;
    5: import java.lang.reflect.Method;
    6:
    7: import javax.swing.AbstractAction;
    8: import javax.swing.Action;
    9: import javax.swing.Icon;
   10:
   11: /**
   12:  * The <code>TrampolineAction</code> call the command method of the controller
   13:  * when the actionPerformed method is invoked.
   14:  */
   15: public class TrampolineAction extends AbstractAction {
   16:
   17:         private static final long serialVersionUID = 1L;
   18:
   19:         private final Object controller;
   20:
   21:         private transient Method method;
   22:
   23:         /*
   24:          * @see
   25:          * java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEven
t)
   26:          */
   27:         public void actionPerformed(ActionEvent e) {
   28:                 try {
   29:                         if (method == null) {
   30:                                 method = controller.getClass().getMethod(
   31:                                                 (String) getValue(Action.ACTION_CO
MMAND_KEY),
   32:                                                 new Class[] { ActionEvent.class })
;
   33:                         }
   34:                         method.invoke(controller, new Object[] { e });
   35:                 } catch (NoSuchMethodException ex1) {
   36:                         throw new RuntimeException(ex1);
   37:                 } catch (InvocationTargetException ex2) {
   38:                         ex2.getTargetException().printStackTrace();
   39:                         throw new RuntimeException(ex2.getTargetException());
   40:                 } catch (Exception ex3) {
   41:                         throw new RuntimeException(ex3);
   42:                 }
   43:         }
   44:
   45:         /**
   46:          * TrampolineAction constructor.
   47:          *
   48:          * @param command
   49:          *             method name
   50:          * @param name
   51:          *             action name
   52:          * @param icon
   53:          *             action icon
   54:          * @param controller
   55:          *             controller object invoked on actionPerformed event.
   56:          */
   57:         public TrampolineAction(String command, String name, Icon icon,
   58:                         final Object controller) {
   59:                 super(name, icon);
   60:                 this.putValue(Action.ACTION_COMMAND_KEY, command);
   61:                 this.controller = controller;
   62:         }
   63: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import javax.swing.BorderFactory;
    4: import javax.swing.Box;
    5: import javax.swing.BoxLayout;
    6: import javax.swing.ImageIcon;
    7: import javax.swing.JButton;
    8: import javax.swing.JLabel;
    9: import javax.swing.JPanel;
   10: import javax.swing.JToolBar;
   11:
   12: import translation.Messages;
   13:
   14: public class TitleBar extends JPanel {
   15:
   16:         private static final long serialVersionUID = 8802351875147413451L;
   17:
   18:         public TitleBar(ImageIcon icon, String title) {
   19:                 JToolBar toolBar = new JToolBar();
   20:                 JButton minimiser = new JButton(IconDatabase.iconMinimize);
   21:                 JButton fermer = new JButton(IconDatabase.iconClose);
   22:
   23:                 BoxLayout layout = new BoxLayout(this, BoxLayout.X_AXIS);
   24:                 setLayout(layout);
   25:
   26:                 toolBar.setFloatable(false);
   27:                 minimiser.setFocusable(false);
   28:                 minimiser.setToolTipText(Messages.getString("TitleBar.0")); //$NON
-NLS-1$
   29:                 fermer.setFocusable(false);
   30:                 fermer.setToolTipText(Messages.getString("TitleBar.1")); //$NON-NL
S-1$
   31:
   32:                 minimiser
   33:                                 .addActionListener(Program.myMainFrame.controller.
actionMinimize);
   34:                 if (title.equals(Messages.getString("TitleBar.2"))) //$NON-NLS-1$
   35:                         fermer
   36:                                         .addActionListener(Program.myMainFrame.con
troller.actionTreeVisibility);
   37:                 else
   38:                         fermer
   39:                                         .addActionListener(Program.myMainFrame.con
troller.actionSchemaVisibility);
   40:
   41:                 add(new JLabel(icon));
   42:                 add(new JLabel(" " + title)); //$NON-NLS-1$
   43:                 add(Box.createHorizontalStrut(100)); // Afin de fixer la largeur
   44:
        // minimale de l'arborescence
   45:                 add(Box.createGlue());
   46:                 toolBar.add(minimiser);
   47:                 toolBar.add(fermer);
   48:                 add(toolBar);
   49:
   50:                 setBorder(BorderFactory.createEmptyBorder(2, 2, 2, 2));
   51:         }
   52:
   53: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.awt.BorderLayout;
    4: import java.awt.Color;
    5: import java.awt.GridLayout;
    6: import java.awt.event.ActionEvent;
    7: import java.awt.event.ActionListener;
    8: import java.util.ArrayList;
    9: import java.util.Collections;
   10: import java.util.Vector;
   11:
   12: import javax.swing.BorderFactory;
   13: import javax.swing.Box;
   14: import javax.swing.BoxLayout;
   15: import javax.swing.JButton;
   16: import javax.swing.JComboBox;
   17: import javax.swing.JLabel;
   18: import javax.swing.JPanel;
   19: import javax.swing.JSeparator;
   20: import javax.swing.JToolBar;
   21: import javax.swing.border.EtchedBorder;
   22:
   23: import main.ActionTime;
   24: import main.RelationComplete;
   25: import models.BrickEdge;
   26: import models.BrickVertex;
   27: import models.TriadeEditingMousePlugin;
   28: import relations.Goal;
   29: import relations.Mean;
   30: import relations.RelationPossibility;
   31: import translation.Messages;
   32:
   33: public class RelationChooserView extends JPanel {
   34:
   35:         private static final long serialVersionUID = 1124347489987701165L;
   36:
   37:         protected Vector<JComboBox> goal;
   38:         protected Vector<JComboBox> ressources;
   39:         protected RelationComplete relationSet;
   40:         protected RelationPossibility possibleRelationSet;
   41:         protected int mode;
   42:
   43:         public RelationChooserView(BrickEdge modelEdge,
   44:                         TriadeEditingMousePlugin<BrickVertex, BrickEdge> mousePlug
in,
   45:                         RelationPossibility _possibleRelationSet, int _mode,
   46:                         RelationChooserPopUp parent) {
   47:                 relationSet = modelEdge.getCompleteRelation();
   48:                 possibleRelationSet = _possibleRelationSet;
   49:                 mode = _mode;
   50:                 goal = new Vector<JComboBox>();
   51:                 ressources = new Vector<JComboBox>();
   52:                 final RelationChooserView chooser = this;
   53:                 int j = 0;
   54:                 Vector<Goal> sortedGoal = new Vector<Goal>(possibleRelationSet.get
Map().keySet());
   55:                 sortedGoal.remove(RelationPossibility.UNDEFINED_GOAL);
   56:                 sortedGoal.remove(RelationPossibility.NORELATION_GOAL);
   57:                 Collections.sort(sortedGoal);
   58:                 sortedGoal.add(0, RelationPossibility.NORELATION_GOAL);
   59:                 sortedGoal.add(0, RelationPossibility.UNDEFINED_GOAL);
   60:
   61:                 for (@SuppressWarnings("unused")
   62:                                 ActionTime action : Program.myMainFrame.datapack.g
etActionTimeList()) {
   63:                         JComboBox goalSingle = new JComboBox(sortedGoal);
   64:
   65:
   66:
   67:                         goal.add(goalSingle);
   68:                         for (int i = 0; i < goalSingle.getItemCount(); i++) {
   69:                                 if (mode == RelationPossibility.RELATIONSTRUCTUREL
LE) {
   70:                                         if (goalSingle.getItemAt(i).equals(
   71:                                                         relationSet.getJeuRelation
(j).objectifStructurel)) {
   72:                                                 goalSingle.setSelectedIndex(i);
   73:                                                 break;
   74:                                         }
   75:                                 } else {
   76:                                         if (goalSingle.getItemAt(i).equals(
   77:                                                         relationSet.getJeuRelation
(j).objectifReel)) {
   78:                                                 goalSingle.setSelectedIndex(i);
   79:                                                 break;
   80:                                         }
   81:                                 }
   82:
   83:                         }
   84:                         JComboBox ressourcesSingle = new JComboBox();
   85:                         ressources.add(ressourcesSingle);
   86:                         if (goalSingle.getItemCount() > 0)
   87:                                 updateRessources(j);
   88:                         for (int i = 0; i < ressourcesSingle.getItemCount(); i++)
{
   89:                                 if (mode == RelationPossibility.RELATIONSTRUCTUREL
LE) {
   90:                                         if (ressourcesSingle.getItemAt(i).equals(
   91:                                                         relationSet.getJeuRelation
(j).moyenStructurel)) {
   92:                                                 ressourcesSingle.setSelectedIndex(
i);
   93:                                                 break;
   94:                                         }
   95:                                 } else {
   96:                                         if (ressourcesSingle.getItemAt(i).equals(
   97:                                                         relationSet.getJeuRelation
(j).moyenReel)) {
   98:                                                 ressourcesSingle.setSelectedIndex(
i);
   99:                                                 break;
  100:                                         }
  101:                                 }
  102:
  103:                         }
  104:                         j++;
  105:                 }
  106:
  107:                 for (JComboBox goalSingle : goal) {
  108:                         goalSingle.addActionListener(new ActionListener() {
  109:
  110:                                 @Override
  111:                                 public void actionPerformed(ActionEvent e) {
  112:                                         int index = goal.indexOf(e.getSource());
  113:                                         chooser.updateRessources(index);
  114:                                 }
  115:
  116:                         });
  117:                 }
  118:                 setLayout(new BorderLayout());
  119:                 build(parent, modelEdge, mousePlugin);
  120:         }
  121:
  122:         protected void updateRessources(int index) {
```

```
  123:                 ressources.elementAt(index).removeAllItems();
  124:                 if (possibleRelationSet.getMap().containsKey(goal.elementAt(index)
.getSelectedItem()))
  125:                         for (Mean str : possibleRelationSet.getMap().get(goal.elem
entAt(index).getSelectedItem())) {
  126:                                 ressources.elementAt(index).addItem(str);
  127:                         }
  128:                 ressources.elementAt(index).validate();
  129:
  130:         }
  131:
  132:         private void build(final RelationChooserPopUp parent,
  133:                         final BrickEdge modelEdge,
  134:                         final TriadeEditingMousePlugin<BrickVertex, BrickEdge> mou
sePlugin) {
  135:                 // Declarations
  136:                 JPanel panelLeft = new JPanel();
  137:                 JPanel panelRight = new JPanel(new BorderLayout());
  138:                 JPanel north = new JPanel(new BorderLayout());
  139:                 JPanel center = new JPanel();
  140:                 JPanel south = new JPanel(new BorderLayout());
  141:                 JPanel pChoices = new JPanel();
  142:                 JPanel pButtons = new JPanel(new GridLayout(1, 4));
  143:                 BoxLayout layoutCenter = new BoxLayout(center, BoxLayout.Y_AXIS);
  144:                 BoxLayout layoutChoices = new BoxLayout(pChoices, BoxLayout.Y_AXIS
);
  145:                 BoxLayout layoutLeft = new BoxLayout(panelLeft, BoxLayout.Y_AXIS);
  146:                 JLabel title = new JLabel(
  147:                         Messages.getString("RelationChooserView.1")); //$N
ON-NLS-1$
  148:                 JButton ok = new JButton(Messages.getString("RelationChooserView.2
")); //$NON-NLS-1$
  149:                 ok.addActionListener(new ActionListener() {
  150:
  151:                         @Override
  152:                         public void actionPerformed(ActionEvent arg0) {
  153:                                 saveState();
  154:                                 parent.removeRelationChooserView();
  155:                         }
  156:
  157:                 });
  158:                 JButton cancel = new JButton(Messages.getString("RelationChooserVi
ew.3")); //$NON-NLS-1$
  159:                 cancel.addActionListener(new ActionListener() {
  160:
  161:                         @Override
  162:                         public void actionPerformed(ActionEvent arg0) {
  163:                                 parent.removeRelationChooserView();
  164:                                 if (mode == RelationPossibility.RELATIONSTRUCTUREL
LE
  165:                                                 && relationSet.isEmpty(mode))
  166:                                         mousePlugin.getEditedAbstractSchema().remo
veModelEdge(
  167:                                                         modelEdge);
  168:                         }
  169:
  170:                 });
  171:
  172:                 // Barre de titre
  173:                 north.setBackground(Color.white);
  174:                 title.setBorder(BorderFactory.createEmptyBorder(2, 2, 2, 2));
  175:                 north.add(title, BorderLayout.CENTER);
  176:                 north.add(new JSeparator(), BorderLayout.SOUTH);
  177:
  178:                 // Left panel
  179:                 panelLeft.setLayout(layoutLeft);
  180:                 panelLeft.add(Box.createVerticalStrut(65));
  181:
  182:                 // Choix possibles
  183:                 pChoices.setLayout(layoutChoices);
  184:                 JPanel pnGrid = new JPanel(new GridLayout(1, 2));
  185:                 JPanel pnLeft = new JPanel();
  186:                 pnLeft.setLayout(new BoxLayout(pnLeft, BoxLayout.X_AXIS));
  187:                 JPanel pnRight = new JPanel();
  188:                 pnRight.setLayout(new BoxLayout(pnRight, BoxLayout.X_AXIS));
  189:                 pnLeft.add(Box.createGlue());
  190:                 pnLeft.add(new JLabel(IconDatabase.iconLeftArrow));
  191:                 pnLeft.add(new JLabel(Messages.getString("RelationChooserView.4"))
); //$NON-NLS-1$
  192:                 pnLeft.add(new JLabel(IconDatabase.iconRightArrow));
  193:                 pnLeft.add(Box.createGlue());
  194:                 pnRight.add(Box.createGlue());
  195:                 pnRight.add(new JLabel(IconDatabase.iconLeftArrow));
  196:                 pnRight.add(new JLabel(Messages.getString("RelationChooserView.5")
)); //$NON-NLS-1$
  197:                 pnRight.add(new JLabel(IconDatabase.iconRightArrow));
  198:                 pnRight.add(Box.createGlue());
  199:                 pnGrid.add(pnLeft);
  200:                 pnGrid.add(pnRight);
  201:                 pChoices.add(pnGrid);
  202:                 int isTriade = Program.isTriades() ? 1 : 0;
  203:                 for (int i = 0; i < Program.myMainFrame.datapack.getActionTimeList
()
  204:                 .size(); i++) {
  205:                         final JPanel pnl = new JPanel(new GridLayout(1 + isTriade,
 2));
  206:                         pnl.setBorder(BorderFactory
  207:                                         .createTitledBorder(Messages.getString("Re
lationChooserView.6") //$NON-NLS-1$
  208:                                                         + Program.myMainFrame.getD
ataPack()
  209:                                                         .getActionTimeList().get(i
)));
  210:                         JLabel truc = new JLabel(" " //$NON-NLS-1$
  211:                                         + relationSet.getJeuRelation(i).objectifSt
ructurel);
  212:                         truc.setBorder(BorderFactory
  213:                                         .createEtchedBorder(EtchedBorder.LOWERED))
;
  214:                         JLabel machin = new JLabel(" " //$NON-NLS-1$
  215:                                         + relationSet.getJeuRelation(i).moyenStruc
turel);
  216:                         machin.setBorder(BorderFactory
  217:                                         .createEtchedBorder(EtchedBorder.LOWERED))
;
  218:                         if (isTriade == 1) {
  219:                                 pnl.add(truc);
  220:                                 pnl.add(machin);
  221:                         }
  222:                         pnl.add(goal.elementAt(i));
  223:                         pnl.add(ressources.elementAt(i));
  224:                         pChoices.add(pnl);
  225:                         final int _i = i;
  226:                         if (i % 2 == 1 || i > 0) {
  227:                                 JToolBar jtb = new JToolBar();
  228:                                 JButton button = new JButton(IconDatabase.iconDupl
icateArrow);
  229:                                 button.addActionListener(new ActionListener() {
  230:
  231:                                         @Override
  232:                                         public void actionPerformed(ActionEvent ar
g0) {
  233:                                                 int indexGoal = goal.elementAt(_i
- 1)
```

```
  234:                                                    .getSelectedIndex();
  235:                                                    int indexRessources = ressources.e
lementAt(_i - 1)
  236:                                                    .getSelectedIndex();
  237:                                                    goal.elementAt(_i).setSelectedInde
x(indexGoal);
  238:                                                    ressources.elementAt(_i).setSelect
edIndex(
  239:                                                                    indexRessources);
  240:                                            }
  241:
  242:                                    });
  243:                                    jtb.setFloatable(false);
  244:                                    button.setFocusable(false);
  245:                                    jtb.add(button);
  246:                                    panelLeft.add(jtb);
  247:                                    panelLeft.add(Box.createVerticalStrut(35));
  248:                            }
  249:                    }
  250:                    // panelLeft.add(Box.createVerticalStrut(50));
  251:                    center.setBorder(BorderFactory.createEmptyBorder(4, 0, 4, 2));
  252:                    center.setLayout(layoutCenter);
  253:                    center.add(Box.createGlue());
  254:                    center.add(pChoices);
  255:                    center.add(Box.createGlue());
  256:
  257:                    // Barre des boutons
  258:                    pButtons.add(Box.createGlue());
  259:                    pButtons.add(ok);
  260:                    pButtons.add(cancel);
  261:                    pButtons.add(Box.createGlue());
  262:                    pButtons.setBorder(BorderFactory.createEmptyBorder(2, 2, 2, 2));
  263:                    south.add(new JSeparator(), BorderLayout.NORTH);
  264:                    south.add(pButtons, BorderLayout.CENTER);
  265:
  266:                    // Panel right
  267:                    panelRight.add(center, BorderLayout.CENTER);
  268:
  269:                    // Panel total
  270:                    add(north, BorderLayout.NORTH);
  271:                    add(panelLeft, BorderLayout.WEST);
  272:                    add(panelRight, BorderLayout.CENTER);
  273:                    add(south, BorderLayout.SOUTH);
  274:            }
  275:
  276:            protected void saveState() {
  277:                    for (int i = 0; i < goal.size(); i++) {
  278:                            if (mode == RelationPossibility.RELATIONSTRUCTURELLE) {
  279:                                    if (goal.elementAt(i).getSelectedItem() != null)
  280:                                            relationSet.getJeuRelation(i).objectifStru
cturel = (Goal)goal
  281:                                            .elementAt(i).getSelectedItem();
  282:                                    if (ressources.elementAt(i).getSelectedItem() != n
ull)
  283:                                            relationSet.getJeuRelation(i).moyenStructu
rel = (Mean)ressources
  284:                                            .elementAt(i).getSelectedItem();
  285:                            } else {
  286:
  287:                                    if (goal.elementAt(i).getSelectedItem() != null)
  288:                                            relationSet.getJeuRelation(i).objectifReel
 = (Goal)goal.elementAt(i)
  289:                                            .getSelectedItem();
  290:                                    if (ressources.elementAt(i).getSelectedItem() != n
ull)
  291:                                            relationSet.getJeuRelation(i).moyenReel =
(Mean)ressources
  292:                                                    .elementAt(i).getSelectedItem();
  293:                            }
  294:                    }
  295:                    if (relationSet.isNoRelation())
  296:                    {
  297:                            DialogHandlerFrame.showErrorDialog(Messages.getString("Rel
ationChooserView.9")); //$NON-NLS-1$
  298:                    }
  299:
  300:                    Program.myMainFrame.repaint();
  301:
  302:            }
  303:
  304: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.awt.event.ActionEvent;
    4: import java.awt.event.ItemEvent;
    5: import java.awt.event.ItemListener;
    6: import java.beans.EventHandler;
    7:
    8: import javax.swing.AbstractButton;
    9: import javax.swing.Action;
   10: import javax.swing.Icon;
   11: import javax.swing.JButton;
   12: import javax.swing.JCheckBoxMenuItem;
   13: import javax.swing.JMenu;
   14: import javax.swing.JOptionPane;
   15: import javax.swing.JRadioButton;
   16:
   17: import models.BrickView;
   18: import tools.ConfigCreator;
   19: import tools.ConfigTriades;
   20: import translation.Messages;
   21: import client.export.JpegGenerator;
   22: import dataPack.DataPack;
   23: import dataPack.SavableObject;
   24:
   25: public final class Controller {
   26:
   27:         private final MainFrame mf;
   28:
   29:         Action actionNew = getAction("onButtonNew", "New", null); //$NON-NLS-1$ //
$NON-NLS-2$
   30:         Action actionOpen = getAction("onButtonOpen", "Open", null); //$NON-NLS-1$
 //$NON-NLS-2$
   31:         Action actionSave = getAction("onButtonSave", "Save", null); //$NON-NLS-1$
 //$NON-NLS-2$
   32:         Action actionSaveAs = getAction("onButtonSaveAs", "SaveAs", null); //$NON-
NLS-1$ //$NON-NLS-2$
   33:         Action actionExoprterDataPack = getAction("onButtonExporterDataPack", "Exp
orterDataPack", null); //$NON-NLS-1$ //$NON-NLS-2$
   34:         Action actionCloseAll = getAction("onButtonCloseAll", "CloseAll", null); /
/$NON-NLS-1$ //$NON-NLS-2$
   35:         Action actionExit = getAction("onButtonExit", "Exit", null); //$NON-NLS-1$
 //$NON-NLS-2$
   36:         Action actionHelp = getAction("onButtonHelp", "Help", null); //$NON-NLS-1$
 //$NON-NLS-2$
   37:         Action actionMinimize = getAction("onButtonMinimize", "Minimize", null); /
/$NON-NLS-1$ //$NON-NLS-2$
   38:         Action actionClose = getAction("onButtonClose", "Close", null); //$NON-NLS
-1$ //$NON-NLS-2$
   39:         Action actionTreeVisibility = getAction("onCheckButtonTree", "Tree", null)
; //$NON-NLS-1$ //$NON-NLS-2$
   40:         Action actionSchemaVisibility = getAction("onCheckButtonSchema", "Schema",
 //$NON-NLS-1$ //$NON-NLS-2$
   41:                         null);
   42:         Action actionMaximize = getAction("onButtonMaximize", "Maximize", null); /
/$NON-NLS-1$ //$NON-NLS-2$
   43:         Action actionDelete = getAction("onButtonDelete", "Delete", null); //$NON-
NLS-1$ //$NON-NLS-2$
   44:         Action actionRename = getAction("onButtonRename", "Rename", null); //$NON-
NLS-1$ //$NON-NLS-2$
   45:         Action actionRemoveTab = getAction("onButtonRemoveTab", "RemoveTab", null)
; //$NON-NLS-1$ //$NON-NLS-2$
   46:
   47:         ItemListener itemListener = getItemListener("onChange"); //$NON-NLS-1$
   48:
   49:         public Controller(MainFrame _mf) {
   50:                 mf = _mf;
   51:         }
   52:
   53:         private Action getAction(final String command, final String name, Icon ico
n) {
   54:                 return new TrampolineAction(command, name, icon, this);
   55:         }
   56:
   57:         private ItemListener getItemListener(String method) {
   58:                 return EventHandler.create(ItemListener.class, this,
   59:                                 method, ""); //$NON-NLS-1$
   60:         }
   61:
   62:         public void onButtonNew(ActionEvent e) {
   63:                 if (!Program.isTriades()) {
   64:                         ((MainFrameDatapackCreator) mf).myOptionNewFrame
   65:                                         .setTitle(Messages.getString("Controller.3
4")); //$NON-NLS-1$
   66:                         ((MainFrameDatapackCreator) mf).myOptionNewFrame
   67:                         .setLocationRelativeTo(mf);
   68:                         ((MainFrameDatapackCreator) mf).myOptionNewFrame.setVisibl
e(true);
   69:                 } else
   70:                         mf
   71:                                         .setDataPack((DataPack) Program
   72:                                                         .loadSavableObject(Message
s.getString("Controller.35"), true)); //$NON-NLS-1$
   73:         }
   74:
   75:         public void onButtonOpen(ActionEvent e) {
   76:                 SavableObject obj = mf.datapack;
   77:                 String str = Messages.getString("Controller.36"); //$NON-NLS-1$
   78:
   79:                 if (obj != null) {
   80:                         int choice = DialogHandlerFrame
   81:                                         .showYesNoCancelDialog(Messages.getString(
"Controller.37") + str //$NON-NLS-1$
   82:                                                         + Messages.getString("Cont
roller.38")); //$NON-NLS-1$
   83:                         switch (choice) {
   84:                         case JOptionPane.YES_OPTION:
   85:                                 if(!Program.isTriades()) {
   86:                                         if (!Program.save(obj, true))
   87:                                                 return;
   88:                                 } else {
   89:                                         if (!Program.save(obj, true))
   90:                                                 return;
   91:                                 }
   92:                                 break;
   93:                         case JOptionPane.CANCEL_OPTION:
   94:                                 return;
   95:                         default:
   96:                                 break;
   97:
   98:                         }
   99:                         mf.setDataPack(null);
  100:                         mf.tabbedPane.removeAll();
  101:                 }
  102:
  103:                 DataPack dtp = (DataPack) Program.loadSavableObject(null, true);
  104:                 if (dtp != null) {
  105:                         dtp.init();
  106:                         mf.setDataPack(dtp);
  107:                 }
  108:                 // TODO pour le client empecher les ouvertures
  109:         }
  110:
  111:         public void onButtonSave(ActionEvent e) {
  112:                 SavableObject obj = mf.datapack;
```

```
113:
114:                 if(Program.isTriades()) {
115:                         Program.save(obj, true);
116:                 } else {
117:                         Program.save(obj, true);
118:                 }
119:         }
120:
121:         public void onButtonSaveAs(ActionEvent e) {
122:                 SavableObject obj = mf.datapack;
123:
124:                 if (obj != null) {
125:                         String oldFilePath = obj.getFilePath();
126:                         boolean saved;
127:
128:                         obj.setFilePath(null);
129:                         if(!Program.isTriades()) {
130:                                 saved = Program.save(obj, true);
131:                                 if(saved) {
132:                                         ConfigCreator.getInstance().getLastDatapac
k().addLastObject(obj.getFilePath());
133:                                         ConfigCreator.getInstance().save();
134:                                 }
135:                         } else {
136:                                 saved = Program.save(obj, true);
137:                         }
138:
139:                         if(!saved) {
140:                                 obj.setFilePath(oldFilePath);
141:                         }
142:                 }
143:         }
144:
145:         public void onButtonExporterDataPack(ActionEvent e) {
146:                 if (Program.isTriades()) {
147:                         JpegGenerator.getSingleton().exportSession(
148:                                         mf.getDataPack().getCurrentSession());
149:                 } else {
150:                         mf.datapack.exportDataPack();
151:                 }
152:         }
153:
154:         public void onButtonCloseAll(ActionEvent e) {
155:                 SavableObject obj = mf.datapack;
156:
157:                 if(!Program.isTriades()) {
158:                         ConfigCreator.getInstance().save();
159:                 } else {
160:                         ConfigTriades.getInstance().save();
161:                 }
162:
163:                 int choice = DialogHandlerFrame
164:                                 .showYesNoCancelDialog(Messages.getString("Control
ler.39")); //$NON-NLS-1$
165:                 switch (choice) {
166:                 case JOptionPane.YES_OPTION:
167:                         if(!Program.isTriades()) {
168:                         if (!Program.save(obj, true)) {
169:                                 System.out.println("Impossible de sauvegarder le d
atapack");
170:                                 return;
171:                         }
172:                         } else {
173:                                 if (!Program.save(obj, true))
174:                                         return;
175:                         }
176:                         break;
177:                 case JOptionPane.CANCEL_OPTION:
178:                         return;
179:                 default:
180:                         break;
181:
182:                 }
183:                 if (!Program.isTriades())
184:                         mf.setDataPack(null);
185:                 mf.tabbedPane.removeAll();
186:         }
187:
188:         public void onButtonExit(ActionEvent e) {
189:                 mf.wouldYouLikeToSaveAndExit();
190:         }
191:
192:         public void onButtonHelp(ActionEvent e) {
193:                 PopUpHelpView p = mf.popUpHelp;
194:                 if (p.isBig()) {
195:                         p.setBig(false);
196:                         return;
197:                 }
198:                 p.setBig(true);
199:         }
200:
201:         public void onButtonMinimize(ActionEvent e) {
202:                 AbstractButton aBut = (AbstractButton) e.getSource();
203:                 aBut.getParent().getParent().getParent().setVisible(false);
204:                 aBut.getParent().getParent().getParent().getParent().getComponent(
0)
205:                 .setVisible(true);
206:         }
207:
208:         public void onButtonMaximize(ActionEvent e) {
209:                 AbstractButton aBut = (AbstractButton) e.getSource();
210:                 aBut.getParent().setVisible(false);
211:                 aBut.getParent().getParent().getComponent(1).setVisible(true);
212:         }
213:
214:         public void onButtonClose(ActionEvent e) {
215:                 mf.pSchema.setVisible(false);
216:         }
217:
218:         public void onCheckButtonTree(ActionEvent e) {
219:                 JCheckBoxMenuItem checkBox;
220:                 if (e.getSource().getClass() == JButton.class) {
221:                         checkBox = (JCheckBoxMenuItem) ((JMenu) mf.menuBar.getComp
onent(2))
222:                         .getMenuComponent(0);
223:                         checkBox.setSelected(false);
224:                         mf.pArborescence.setVisible(false);
225:                 } else {
226:                         checkBox = (JCheckBoxMenuItem) e.getSource();
227:                         mf.pArborescence.setVisible(checkBox.isSelected());
228:                 }
229:         }
230:
231:         public void onCheckButtonSchema(ActionEvent e) {
232:                 JCheckBoxMenuItem checkBox;
233:                 if (e.getSource().getClass() == JButton.class) {
234:                         checkBox = (JCheckBoxMenuItem) ((JMenu) mf.menuBar.getComp
onent(2))
235:                         .getMenuComponent(1);
236:                         checkBox.setSelected(false);
237:                         mf.pSchema.setVisible(false);
238:                 } else {
239:                         checkBox = (JCheckBoxMenuItem) e.getSource();
240:                         mf.pSchema.setVisible(checkBox.isSelected());
```

```
  241:                 }
  242:         }
  243:
  244:         public void onButtonRemoveTab(ActionEvent e) {
  245:                 int index = mf.tabbedPane.getSelectedIndex();
  246:                 if (index == 0) {
  247:                         onButtonCloseAll(e);
  248:                 } else if (index > 0)
  249:                 {
  250:                         if (mf.getClass().equals(MainFrameDatapackCreator.class))
{
  251:                                 ((MainFrameDatapackCreator) mf).removeTab();
  252:                         }
  253:                         mf.tabbedPane.remove(index);
  254:                         // TODO : supprimer la fuite mÃ©moire, pour cela, supprime
r le
  255:                         // VisualisationViewer
  256:                 }
  257:         }
  258:
  259:         public void onButtonDelete(ActionEvent e) {
  260:                 int index = mf.tabbedPane.getSelectedIndex();
  261:                 if (index > 0) {
  262:                         BrickView mV = mf.getActiveModelView();
  263:                         if (mV != null) {
  264:                                 mV.getMousePlugin().removeSelectedVertex();
  265:                                 mV.repaint();
  266:                         }
  267:                 } else if (index == 0) {
  268:                         mf.datapack.removeActor();
  269:                 }
  270:         }
  271:
  272:         public void onButtonRename(ActionEvent e) {
  273:                 int index = mf.tabbedPane.getSelectedIndex();
  274:                 if (index == 0) {
  275:                         mf.getMainJTree().startEditing();
  276:                 }
  277:         }
  278:
  279:         public void onChange(ItemEvent event) {
  280:                 JRadioButton button = (JRadioButton) event.getItemSelectable();
  281:                 OptionNewFrameDatapackCreator onf = ((MainFrameDatapackCreator) mf
).myOptionNewFrame;
  282:
  283:                 if (button == onf.datapack)
  284:                         onf.description
  285:                                         .setText(Messages.getString("Controller.40
")); //$NON-NLS-1$
  286:                 else if (button == onf.brick)
  287:                         onf.description
  288:                                         .setText(Messages.getString("Controller.41
")); //$NON-NLS-1$
  289:                 else if (button == onf.newFile)
  290:                         onf.descriptionBis.setText(Messages.getString("Controller.
42")); //$NON-NLS-1$
  291:                 else if (button == onf.openFile)
  292:                         onf.descriptionBis.setText(Messages.getString("Controller.
43")); //$NON-NLS-1$
  293:         }
  294: }
```

```java
package graphicalUserInterface;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.File;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.ButtonGroup;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;

import translation.Messages;
import client.Session;

public class AssistantFrameTriades extends JFrame {

	private static final long serialVersionUID = 5303866988917452819L;

	protected JList listOpen;
	protected MainFrame mf;
	protected JRadioButton jrbOpen;
	protected JRadioButton jrbAll;
	protected JRadioButton jrbNew;
	public AssistantFrameTriades(MainFrame mf) {
		this.mf = mf;
		setAlwaysOnTop(true);
		setTitle(Messages.getString("AssistantFrameTriades.0")); // titre
//$NON-NLS-1$
		Image icone = Toolkit.getDefaultToolkit().getImage(
				"Icones" + File.separatorChar + "16x16" + File.sep
aratorChar + "triades.png"); //$NON-NLS-1$
		setIconImage(icone);
		setSize(400, 400); // taille
		setLocationRelativeTo(mf); // fenÃªtre centrÃ©e sur la main frame
		setResizable(false); // non redimensionnable
		setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);

		getContentPane().add(this.myPanel(mf));
	}

	private JPanel myPanel(final MainFrame mf) {
		// DÃ©clarations + initialisations
		final JPanel panel = new JPanel(new BorderLayout());
		JPanel north = new JPanel(new BorderLayout());
		JPanel south = new JPanel(new BorderLayout());
		JPanel center = new JPanel();
		JPanel pTitle = new JPanel();
		JPanel pChoices = new JPanel(new GridLayout(3, 1));
		JPanel p2Buttons = new JPanel(new GridLayout(1, 2));
		JPanel pButtons = new JPanel();
		listOpen = new JList(mf.getDataPack().getExportModule()
				.getSessionList().toArray());
		ButtonGroup buttonGroup = new ButtonGroup();
		jrbAll = new JRadioButton(
				Messages.getString("AssistantFrameTriades.2"), tru
e); //$NON-NLS-1$
		jrbNew = new JRadioButton(
				Messages.getString("AssistantFrameTriades.3"), tru
e); //$NON-NLS-1$
		jrbOpen = new JRadioButton(
				Messages.getString("AssistantFrameTriades.4"), tru
e); //$NON-NLS-1$
		final JButton jbOk = new JButton(Messages.getString("AssistantFram
eTriades.5")); //$NON-NLS-1$
		JButton jbCancel = new JButton(Messages.getString("AssistantFrameT
riades.6")); //$NON-NLS-1$

		if (listOpen.getModel().getSize() == 0) {
			listOpen.setEnabled(false);
			jrbOpen.setEnabled(false);
		}


		// Barre de titre
		pTitle.setBackground(Color.WHITE);
		pTitle.setLayout(new BoxLayout(pTitle, BoxLayout.X_AXIS));
		pTitle.add(Box.createHorizontalStrut(10));
		pTitle.add(new JLabel(Messages.getString("AssistantFrameTriades.7"
))); //$NON-NLS-1$
		pTitle.add(Box.createGlue());
		pTitle.add(new JLabel(IconDatabase.iconAssistant));
		north.add(pTitle, BorderLayout.CENTER);
		north.add(new JSeparator(), BorderLayout.SOUTH);

		// Zone Ã  choix
		buttonGroup.add(jrbNew);
		buttonGroup.add(jrbOpen);
		buttonGroup.add(jrbAll);
		jrbAll.setSelected(true);

		JPanel p1 = new JPanel(new FlowLayout());
		p1.add(new JLabel(IconDatabase.iconModel));
		p1.add(jrbAll);
		pChoices.add(p1);

		p1 = new JPanel(new FlowLayout());
		p1.add(new JLabel(IconDatabase.iconNewFile));
		p1.add(jrbNew);
		pChoices.add(p1);

		p1 = new JPanel(new FlowLayout());
		p1.add(new JLabel(IconDatabase.iconOpenFile));
		p1.add(jrbOpen);
		pChoices.add(p1);

		listOpen.setPreferredSize(new Dimension(350, 115));
		listOpen
				.setBorder(BorderFactory
				.createTitledBorder(Messages.getString("AssistantF
rameTriades.8"))); //$NON-NLS-1$
		listOpen.addMouseListener(new MouseListener() {
			@Override
			public void mouseClicked(MouseEvent arg0) {
				jrbOpen.setSelected(true);
				if (arg0.getClickCount() == 2) {
					onButtonOk();
```

```
126:                                }
127:                        }
128:
129:                        @Override
130:                        public void mouseEntered(MouseEvent arg0) {
131:                        }
132:
133:                        @Override
134:                        public void mouseExited(MouseEvent arg0) {
135:                        }
136:
137:                        @Override
138:                        public void mousePressed(MouseEvent arg0) {
139:                        }
140:
141:                        @Override
142:                        public void mouseReleased(MouseEvent arg0) {
143:                        }
144:                });
145:                center.setLayout(new BoxLayout(center, BoxLayout.Y_AXIS));
146:                center.add(pChoices);
147:                center.add(new JScrollPane(listOpen));
148:
149:                // Zone boutons
150:                jbOk.addActionListener(new ActionListener() {
151:                        @Override
152:                        public void actionPerformed(ActionEvent e) {
153:                                onButtonOk();
154:                        }
155:                });
156:                jbCancel.addActionListener(new ActionListener() {
157:                        @Override
158:                        public void actionPerformed(ActionEvent e) {
159:                                setVisible(false);
160:                        }
161:                });
162:                p2Buttons.add(jbOk);
163:                p2Buttons.add(jbCancel);
164:                pButtons.add(Box.createHorizontalStrut(100));
165:                pButtons.add(p2Buttons);
166:                pButtons.add(Box.createHorizontalStrut(100));
167:                south.add(new JSeparator(), BorderLayout.NORTH);
168:                south.add(pButtons, BorderLayout.CENTER);
169:
170:                // Panel global
171:                panel.add(north, BorderLayout.NORTH);
172:                panel.add(center, BorderLayout.CENTER);
173:                panel.add(south, BorderLayout.SOUTH);
174:                return panel;
175:        }
176:
177:        protected void onButtonOk() {
178:                if (jrbAll.isSelected()) {
179:                        setVisible(false);
180:                        ((MainFrameTriades) mf).displayObject(mf.getDataPack()
181:                                        .getExportModule().getFullSession());
182:                        mf.getDataPack()
183:                                        .setCurrentSession(
184:                                                        mf.getDataPack().getExport
Module()
185:                                                                        .getFullSe
ssion());
186:                } else if (jrbNew.isSelected()) {
187:                        setVisible(false);
188:                        ((MainFrameTriades) mf)
189:                                        .displayObject(new ActorListView(mf));
190:                } else if (jrbOpen.isSelected()) {
191:                        setVisible(false);
192:                        Session selectedSession = (Session)listOpen.getSelectedVal
ue();
193:                        mf.getDataPack().getExportModule().upSession(
194:                                        selectedSession);
195:                        ((MainFrameTriades) mf).displayObject(selectedSession);
196:                        mf.getDataPack().setCurrentSession(selectedSession);
197:                }
198:        }
199:
200:        public void refreshAndShow() {
201:                DefaultComboBoxModel listModel = new DefaultComboBoxModel(mf.getDa
taPack().getExportModule().getSessionList().toArray());
202:                listOpen.setModel(listModel);
203:                if (listModel.getSize() == 0) {
204:                        listOpen.setEnabled(false);
205:                        jrbOpen.setEnabled(false);
206:                } else {
207:                        listOpen.setEnabled(true);
208:                        jrbOpen.setEnabled(true);
209:
210:                }
211:                jrbAll.setSelected(true);
212:
213:                validate();
214:                setVisible(true);
215:        }
216: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.awt.BorderLayout;
    4: import java.awt.GridLayout;
    5: import java.util.Vector;
    6:
    7: import javax.swing.JPanel;
    8:
    9: import models.Brick;
   10: import models.BrickEdge;
   11: import models.BrickVertex;
   12: import models.BrickView;
   13: import translation.Messages;
   14: import client.NavigationTree;
   15: import client.SchemaTree;
   16: import client.Session;
   17: import client.export.ExportPopUp;
   18: import client.export.ExportTree;
   19: import client.export.ExportView;
   20: import client.export.ExportingMousePlugin;
   21: import dataPack.AutoSaveCreator;
   22: import dataPack.DataPack;
   23: import dataPack.JTreeActors;
   24:
   25: public class MainFrameTriades extends MainFrame {
   26:
   27:         private static final long serialVersionUID = -1230397799638872518L;
   28:
   29:         protected ActorListView actorListView;
   30:
   31:         protected AssistantFrameTriades myAssistantFrameTriades;
   32:         protected Vector<BrickView> schemaViewList;
   33:
   34:         public MainFrameTriades() {
   35:                 setTitle(Messages.getString("MainFrameTriades.0")); //$NON-NLS-1$
   36:                 schemaViewList = new Vector<BrickView>();
   37:         }
   38:
   39:         @Override
   40:         protected JPanel myPanel(Object object, JPanel mainPanel) {
   41:                 return myPanel(object, false, mainPanel);
   42:         }
   43:
   44:         protected JPanel myPanel(Object object, boolean enableLayout, JPanel mainP
anel) {
   45:                 JTreeActors jta = null;
   46:                 PopUpView popUp = null;
   47:                 pSchema = new JPanel(new BorderLayout());
   48:
   49:                 if (object.getClass() == ActorListView.class) {
   50:                         jta = new JTreeActors(datapack);
   51:                         mainJta = jta;
   52:                         actorListView = (ActorListView) object;
   53:                         pSchema.add(new TitleBar(IconDatabase.iconSchema,
   54:                         Messages.getString("MainFrameTriades.1")), BorderLayout.NO
RTH); //$NON-NLS-1$
   55:                         pSchema.add(actorListView, BorderLayout.CENTER);
   56:
   57:                         // } else if (object.getClass() == SchemaSelectionView.cla
ss) {
   58:                         // SchemaSelectionView view = (SchemaSelectionView) object
;
   59:                         // if (view != null) {
   60:                         // jta = new JTreeActors(view.getSchema());
   61:                         //
   62:                         // pSchema.add(new TitleBar(IconDatabase.iconSchema,
   63:                         // "SchÃ©mas gÃ©nÃ©rÃ©s"), BorderLayout.NORTH);
   64:                         // pSchema.add(view, BorderLayout.CENTER);
   65:                         // }
   66:                 } else if (object.getClass() == Brick.class) {
   67:                         Brick brick= (Brick) object;
   68:
   69:                         if (brick != null) {
   70:                                 BrickView<BrickVertex, BrickEdge> view = new Brick
View<BrickVertex, BrickEdge>(brick, new RelationChooserPopUp(), datapack, null);
   71:                                 jta = new SchemaTree(brick, view.getMousePlugin())
;
   72:                                 view.setTreeListener(jta.getListener());
   73:                                 // jta = new JTreeActors(schema);
   74:                                 popUp = view.getPopUp();
   75:                                 pSchema.add(new TitleBar(IconDatabase.iconSchema,
Messages.getString("MainFrameTriades.2")), //$NON-NLS-1$
   76:                                                 BorderLayout.NORTH);
   77:                                 pSchema.add(view, BorderLayout.CENTER);
   78:                         }
   79:
   80:                 } else if (object.getClass() == ExportView.class) {
   81:                         ExportView view = (ExportView) object;
   82:                         if (view != null) {
   83:                                 jta = new ExportTree(view.getExportModel(), (Expor
tPopUp)view.getPopUp(), view.getMousePlugin());
   84:                                 view.setTreeListener(jta.getListener());
   85:                                 popUp = view.getPopUp();
   86:                                 ((ExportingMousePlugin)view.getMousePlugin()).setE
xportView(view);
   87:                                 pSchema.add(new TitleBar(IconDatabase.iconSchema,
Messages.getString("MainFrameTriades.3")), //$NON-NLS-1$
   88:                                                 BorderLayout.NORTH);
   89:                                 pSchema.add(view, BorderLayout.CENTER);

   90:                         }
   91:                 }
   92:
   93:                 return drawPanel(jta, popUp, pSchema, mainPanel);
   94:         }
   95:
   96:         @Override
   97:         public void setDataPack(DataPack dtp) {
   98:                 datapack = dtp;
   99:
  100:                 if(autoSaveCreator == null) {
  101:                         autoSaveCreator = new AutoSaveCreator(datapack);
  102:                         AutoSaveCreator.extentionAutoSave = "";
  103:                 }
  104:
  105:                 if (dtp != null && dtp.getCompanyInfo().getName() != null) {
  106:                         String name = dtp.getCompanyInfo().getName().trim();
  107:                         setTitle(Messages.getString("MainFrameTriades.0") + (name.
compareTo("") == 0 ? "" : (" : " + dtp.getCompanyInfo().getName()))); //$NON-NLS-1$
  108:                 } else
  109:                         setTitle(Messages.getString("MainFrameTriades.0")); //$NON
-NLS-1$
  110:
  111:                 tabbedPane.removeAll();
  112:
  113:                 if (datapack == null) {
  114:                         return;
  115:                 }
  116:
  117:                 dtp.init();
  118:
  119:                 popUpHelp = new PopUpHelpView();
  120:                 popUpHelp.setView(PopUpHelpView.showSelectionActorsHelp());
  121:                 AssistantFrameTriades aft = new AssistantFrameTriades(this);
```

```
  122:                 aft.setVisible(true);
  123:
  124:         }
  125:
  126:         @Override
  127:         public void initialState(MainFrame mf) {
  128:
  129:         }
  130:
  131:
  132:         @Override
  133:         public BrickView getActiveModelView() {
  134:                 // TODO ComplÃ©ter cette fonction inconnue... :D bonne chance
  135:                 return null;
  136:         }
  137:
  138:         public void displayObject(Session session) {
  139:                 JPanel drawingPanel = new JPanel(new GridLayout());
  140:
  141:                 JPanel panel = drawPanel(new NavigationTree(session, drawingPanel)
,
  142:                                 null, drawingPanel, new JPanel());
  143:
  144:                 clearTabsAndShow(Messages.getString("MainFrameTriades.4"), panel);
 //$NON-NLS-1$
  145:         }
  146:
  147:         public void displayObject(ActorListView actorListView) {
  148:                 clearTabsAndShow(Messages.getString("MainFrameTriades.5"), myPanel
(actorListView)); //$NON-NLS-1$
  149:         }
  150:
  151: }
```

```java
package graphicalUserInterface;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.JTextField;

import main.ActionTime;
import main.Statut;
import main.StatutObjet;
import models.Brick;
import relations.EnsembleRelation;
import relations.RelationBrowser;
import translation.Messages;
import dataPack.Activite;
import dataPack.CompanyInfoView;
import dataPack.DataPack;
import dataPack.JeuActeur;
import dataPack.Moyen;

public class DataPackView extends JPanel {

	private static final long serialVersionUID = -4909765521512012778L;

	protected DataPack dtp;
	JLabel nameLabel;

	public DataPackView(DataPack _dtp, MainFrame mf) {
		dtp = _dtp;
		build(mf);
	}

	private void build(final MainFrame mf) {
		final DataPackView access = this;

		this.setLayout(new BorderLayout());
		JPanel north = new JPanel(new BorderLayout());
		JPanel center = new JPanel(new BorderLayout());
		JPanel south = new JPanel(new BorderLayout());
		JPanel pTitle = new JPanel();
		JPanel pButtons = new JPanel();
		JPanel pInfo = new JPanel();
		JPanel pAddActorSetView = new JPanel();
		JPanel pAddActor = new JPanel();
		JPanel pAddGroup = new JPanel();
		JPanel pAddActivity = new JPanel();
		JPanel pAddMoyen = new JPanel();
		JPanel pAddActivityMoyen = new JPanel();
		JPanel pEditRelation = new JPanel();
		JPanel pAddBrick = new JPanel();
		JPanel pAddActionTime = new JPanel();
		JPanel pAddStep = new JPanel();
		JPanel megaAdd = new JPanel();
		JPanel superAdd = new JPanel();
		JPanel pLists = new JPanel(new BorderLayout());
		JPanel pInformation = new JPanel(new GridLayout(0, 2));
		final JList listBricks = new JList(dtp.getBrickList());
		JScrollPane jspBricks = new JScrollPane(listBricks);
		JScrollPane jspCenter = new JScrollPane(center);
		final JComboBox genericMoyens = new JComboBox(main.StatutObjet.values());
		final JComboBox genericActors = new JComboBox(main.Statut.values());
		final JComboBox activities = new JComboBox(dtp.getActivities()
				.getActivities());
		final JComboBox moyens = new JComboBox(dtp.getMoyenListe());
		final JComboBox iconActivities = new JComboBox();
		final JComboBox actionTimes = new JComboBox(dtp.getActionTimeList());
		final JComboBox steps = new JComboBox(dtp.getSteps());
		final JComboBox actorSets = new JComboBox(dtp
				.getJeuxActeur());
		final JTextField newActorName = new JTextField(
				Messages.getString("DataPackView.0")); //$NON-NLS-1$
		final JTextField newGroupName = new JTextField(
				Messages.getString("DataPackView.1")); //$NON-NLS-1$
		final JTextField newActivityName = new JTextField(
				Messages.getString("DataPackView.2")); //$NON-NLS-1$
		final JTextField newMoyenName = new JTextField(
				Messages.getString("DataPackView.3")); //$NON-NLS-1$
		final JTextField newActionTimeName = new JTextField(
				Messages.getString("DataPackView.4")); //$NON-NLS-1$
		final JTextField newStepName = new JTextField(
				Messages.getString("DataPackView.5")); //$NON-NLS-1$

		JButton infoButton = new JButton("Modifier");
		JButton save = new JButton(Messages.getString("DataPackView.6")); //$NON-NLS-1$
		JButton addActorSetView = new JButton(Messages.getString("DataPackView.7")); //$NON-NLS-1$
		JButton addActor = new JButton(Messages.getString("DataPackView.8")); //$NON-NLS-1$
		JButton addGroup = new JButton(Messages.getString("DataPackView.9")); //$NON-NLS-1$
		JButton addActivity = new JButton(Messages.getString("DataPackView.10")); //$NON-NLS-1$
		JButton addMoyen = new JButton(Messages.getString("DataPackView.11")); //$NON-NLS-1$
		JButton addActionTime = new JButton(Messages.getString("DataPackView.12")); //$NON-NLS-1$
		JButton addStep = new JButton(Messages.getString("DataPackView.13")); //$NON-NLS-1$

		// Barre de titre
		pTitle.setBackground(Color.WHITE);
		pTitle.setLayout(new BoxLayout(pTitle, BoxLayout.X_AXIS));
		pTitle.add(Box.createHorizontalStrut(10));
```

```
118:                 pTitle.add(new JLabel(Messages.getString("DataPackView.14") + dtp.
toString())); //$NON-NLS-1$
119:                 pTitle.add(Box.createGlue());
120:                 pTitle.add(new JLabel(IconDatabase.iconTitleDataPack));
121:                 north.add(new JSeparator(), BorderLayout.NORTH);
122:                 north.add(pTitle, BorderLayout.CENTER);
123:                 north.add(new JSeparator(), BorderLayout.SOUTH);
124:
125:                 //Informations sur l'entreprise
126:                 pInfo.setLayout(new BoxLayout(pInfo, BoxLayout.X_AXIS));
127:                 pInfo.setBorder(BorderFactory.createTitledBorder("Entreprise"));
128:                 infoButton.addActionListener(new ActionListener() {
129:                         @Override
130:                         public void actionPerformed(ActionEvent e) {
131:                                 new CompanyInfoView(mf.datapack.getCompanyInfo(),
access);
132:                         }
133:                 });
134:
135:                 nameLabel = new JLabel(getCompanyNameLabel());
136:                 pInfo.add(nameLabel);
137:                 pInfo.add(Box.createVerticalGlue());
138:                 pInfo.add(Box.createHorizontalGlue());
139:                 pInfo.add(infoButton);
140:
141:                 // Ajouter acteur
142:                 pAddActor.setLayout(new BoxLayout(pAddActor, BoxLayout.X_AXIS));
143:                 pAddActor.add(new JLabel(IconDatabase.iconAddActor));
144:                 pAddActor.add(Box.createHorizontalStrut(10));
145:                 newActorName.setPreferredSize(new Dimension(200, 25));
146:                 newActorName.setMaximumSize(new Dimension(200, 25));
147:                 newActorName
148:                 .addFocusListener(generateMouseListener(Messages.getString("DataPa
ckView.15"))); //$NON-NLS-1$
149:                 pAddActor.add(newActorName);
150:                 pAddActor.add(Box.createHorizontalStrut(10));
151:                 pAddActor.add(new JLabel(Messages.getString("DataPackView.16")));
//$NON-NLS-1$
152:                 pAddActor.add(Box.createHorizontalStrut(10));
153:                 genericActors.setPreferredSize(new Dimension(200, 25));
154:                 genericActors.setMaximumSize(new Dimension(200, 25));
155:                 pAddActor.add(genericActors);
156:                 pAddActor.add(Box.createHorizontalGlue());
157:                 addActor.addActionListener(new ActionListener() {
158:
159:                         @Override
160:                         public void actionPerformed(ActionEvent arg0) {
161:                                 if ((newActorName.getText().length() > 0)
162:                                                 && (!newActorName.getText().equals
(
163:                                                                 Messages.getString
("DataPackView.17")))) { //$NON-NLS-1$
164:                                         if (dtp.addActor(
165:                                                         ((Statut) genericActors.ge
tSelectedItem()).id,
166:                                                         newActorName.getText()))
167:                                         {
168:                                                 newActorName.setText(Messages.getS
tring("DataPackView.18")); //$NON-NLS-1$
169:                                                 newActorName.requestFocus();
170:                                         }
171:                                 } else
172:                                         DialogHandlerFrame.showErrorDialog(mf,
173:                                                         Messages.getString("DataPa
ckView.19")); //$NON-NLS-1$
174:                         }
175:
176:                 });
177:                 pAddActor.add(addActor);
178:
179:                 // Ajouter groupe
180:                 pAddGroup.setLayout(new BoxLayout(pAddGroup, BoxLayout.X_AXIS));
181:                 pAddGroup.add(new JLabel(IconDatabase.iconAddGroup));
182:                 pAddGroup.add(Box.createHorizontalStrut(10));
183:                 newGroupName.setPreferredSize(new Dimension(200, 25));
184:                 newGroupName.setMaximumSize(new Dimension(200, 25));
185:                 newGroupName
186:                 .addFocusListener(generateMouseListener(Messages.getString("DataPa
ckView.20"))); //$NON-NLS-1$
187:                 pAddGroup.add(newGroupName);
188:                 pAddGroup.add(Box.createHorizontalGlue());
189:                 addGroup.addActionListener(new ActionListener() {
190:
191:                         @Override
192:                         public void actionPerformed(ActionEvent arg0) {
193:                                 if ((newGroupName.getText().length() > 0)
194:                                                 && (!newGroupName.getText().equals
(
195:                                                                 Messages.getString
("DataPackView.21")))) { //$NON-NLS-1$
196:                                         if (dtp.addGroup(newGroupName.getText()))
{
197:                                                 newGroupName.setText(Messages.getS
tring("DataPackView.22")); //$NON-NLS-1$
198:                                                 newGroupName.requestFocus();
199:                                         }
200:                                 } else
201:                                         DialogHandlerFrame.showErrorDialog(mf,
202:                                                         Messages.getString("DataPa
ckView.23")); //$NON-NLS-1$
203:                         }
204:
205:                 });
206:                 pAddGroup.add(addGroup);
207:                 // Ajout du tout au panel supÃ©rieur
208:                 superAdd.setLayout(new BoxLayout(superAdd, BoxLayout.Y_AXIS));
209:                 superAdd.setBorder(BorderFactory
210:                                 .createTitledBorder(Messages.getString("DataPackVi
ew.24"))); //$NON-NLS-1$
211:                 superAdd.add(pAddGroup);
212:                 superAdd.add(Box.createVerticalStrut(10));
213:                 superAdd.add(pAddActor);
214:
215:                 // Ajouter activitÃ© ou moyen
216:                 pAddActivity.setLayout(new BoxLayout(pAddActivity, BoxLayout.X_AXI
S));
217:                 pAddActivity.add(new JLabel(IconDatabase.iconAddActivity));
218:                 pAddActivity.add(Box.createHorizontalStrut(10));
219:                 newActivityName.setPreferredSize(new Dimension(200, 25));
220:                 newActivityName.setMaximumSize(new Dimension(200, 25));
221:                 newActivityName
222:                 .addFocusListener(generateMouseListener(Messages.getString("DataPa
ckView.25"))); //$NON-NLS-1$
223:                 pAddActivity.add(newActivityName);
224:                 pAddActivity.add(Box.createHorizontalStrut(10));
225:                 pAddActivity.add(new JLabel(Messages.getString("DataPackView.26"))
); //$NON-NLS-1$
226:                 pAddActivity.add(Box.createHorizontalStrut(12));
227:                 iconActivities.setPreferredSize(new Dimension(70, 25));
228:                 iconActivities.setMaximumSize(new Dimension(70, 25));
229:                 for (int i = 0; i < IconDatabase.vectorIconActivities.size(); i +=
 2)
230:                         iconActivities.addItem(IconDatabase.vectorIconActivities
231:                                         .elementAt(i));
```

```
  232:                 pAddActivity.add(iconActivities);
  233:                 pAddActivity.add(Box.createHorizontalGlue());
  234:                 addActivity.addActionListener(new ActionListener() {
  235:                         @Override
  236:                         public void actionPerformed(ActionEvent arg0) {
  237:                                 if ((newActivityName.getText().length() > 0)
  238:                                                 && (!newActivityName.getText().equ
als(
  239:                                                                 Messages.getString
("DataPackView.27")))) { //$NON-NLS-1$
  240:                                         Activite newActivity = new Activite(newAct
ivityName
  241:                                                         .getText(), iconActivities
.getSelectedIndex() * 2);
  242:                                         activities.addItem(newActivity);
  243:                                         activities.setSelectedItem(newActivity);
  244:                                         activities.validate();
  245:                                         newActivityName.setText(Messages.getString
("DataPackView.28")); //$NON-NLS-1$
  246:                                         newActivityName.requestFocus();
  247:                                 } else
  248:                                         DialogHandlerFrame.showErrorDialog(mf,
  249:                                                         Messages.getString("DataPa
ckView.29")); //$NON-NLS-1$
  250:                         }
  251:                 });
  252:                 pAddActivity.add(addActivity);
  253:                 //
  254:                 pAddMoyen.setLayout(new BoxLayout(pAddMoyen, BoxLayout.X_AXIS));
  255:                 pAddMoyen.add(new JLabel(IconDatabase.iconAddMoyen));
  256:                 pAddMoyen.add(Box.createHorizontalStrut(10));
  257:                 newMoyenName.setPreferredSize(new Dimension(200, 25));
  258:                 newMoyenName.setMaximumSize(new Dimension(200, 25));
  259:                 newMoyenName
  260:                 .addFocusListener(generateMouseListener(Messages.getString("DataPa
ckView.30"))); //$NON-NLS-1$
  261:                 pAddMoyen.add(newMoyenName);
  262:                 pAddMoyen.add(Box.createHorizontalStrut(10));
  263:                 pAddMoyen.add(new JLabel(Messages.getString("DataPackView.31")));
//$NON-NLS-1$
  264:                 pAddMoyen.add(Box.createHorizontalStrut(15));
  265:                 genericMoyens.setPreferredSize(new Dimension(200, 25));
  266:                 genericMoyens.setMaximumSize(new Dimension(200, 25));
  267:                 pAddMoyen.add(genericMoyens);
  268:                 pAddMoyen.add(Box.createHorizontalGlue());
  269:                 addMoyen.addActionListener(new ActionListener() {
  270:                         @Override
  271:                         public void actionPerformed(ActionEvent arg0) {
  272:                                 if ((newMoyenName.getText().length() > 0)
  273:                                                 && (!newMoyenName.getText().equals
(
  274:                                                                 Messages.getString
("DataPackView.32")))) { //$NON-NLS-1$
  275:                                         Moyen newMoyen = dtp.addMoyen(newMoyenName
.getText(),
  276:                                                         ((StatutObjet) genericMoye
ns.getSelectedItem()).id);
  277:
  278:                                         if (newMoyen != null) {
  279:                                                 moyens.addItem(newMoyen);
  280:                                                 moyens.validate();
  281:                                         } else {
  282:                                                 JOptionPane.showMessageDialog(null
,
  283:                                                                 Messages.getString
("DataPackView.33"), Messages.getString("DataPackView.34"), //$NON-NLS-1$ //$NON-NLS-2$
  284:                                                                 JOptionPane.WARNIN
G_MESSAGE);
  285:                                         }
  286:                                         newMoyenName.setText(Messages.getString("D
ataPackView.35")); //$NON-NLS-1$
  287:                                         newMoyenName.requestFocus();
  288:                                 } else
  289:                                         DialogHandlerFrame.showErrorDialog(mf,
  290:                                                         Messages.getString("DataPa
ckView.36")); //$NON-NLS-1$
  291:                                 }
  292:                 });
  293:                 pAddMoyen.add(addMoyen);
  294:                 //
  295:                 pAddActivityMoyen.setLayout(new BoxLayout(pAddActivityMoyen,
  296:                                 BoxLayout.Y_AXIS));
  297:                 pAddActivityMoyen.setBorder(BorderFactory
  298:                                 .createTitledBorder(Messages.getString("DataPackVi
ew.37"))); //$NON-NLS-1$
  299:                 pAddActivityMoyen.add(pAddActivity);
  300:                 // pAddActivityMoyen.add(Box.createVerticalStrut(10));
  301:                 pAddActivityMoyen.add(pAddMoyen);
  302:
  303:                 // Editer le jeu de relation
  304:                 pEditRelation.setLayout(new BoxLayout(pEditRelation, BoxLayout.X_A
XIS));
  305:                 pEditRelation.setBorder(BorderFactory
  306:                                 .createTitledBorder(Messages.getString("DataPackVi
ew.38"))); //$NON-NLS-1$
  307:                 JButton editRelation = new JButton(Messages.getString("DataPackVie
w.39")); //$NON-NLS-1$
  308:                 editRelation.addActionListener(new ActionListener() {
  309:
  310:                         @Override
  311:                         public void actionPerformed(ActionEvent arg0) {
  312:                                 for (int i = 0; i < mf.tabbedPane.getTabCount(); i
++)
  313:                                 {
  314:                                         if (mf.tabbedPane.getTitleAt(i).equals(Mes
sages.getString("DataPackView.40"))) //$NON-NLS-1$
  315:                                         {
  316:                                                 mf.tabbedPane.setSelectedIndex(i);
  317:                                                 return;
  318:                                         }
  319:
  320:                                 }
  321:                                 if (mf.datapack.getRelations() == null)
  322:                                 {
  323:                                         mf.datapack.setRelations(new EnsembleRelat
ion());
  324:                                         System.out.println(Messages.getString("Dat
aPackView.41")); //$NON-NLS-1$
  325:                                 }
  326:                                 mf.tabbedPane.addTab(Messages.getString("DataPackV
iew.42"), new RelationBrowser( //$NON-NLS-1$
  327:                                                 mf.datapack.getRelations()));
  328:                                 mf.tabbedPane.setSelectedIndex(mf.tabbedPane.getTa
bCount() - 1);
  329:                         }
  330:
  331:                 });
  332:                 pEditRelation.add(Box.createGlue());
  333:                 pEditRelation.add(editRelation);
  334:
  335:                 // Ajouter un temps d'action
  336:                 pAddActionTime
  337:                 .setLayout(new BoxLayout(pAddActionTime, BoxLayout.X_AXIS));
  338:                 pAddActionTime.setBorder(BorderFactory
```

```
339:                                .createTitledBorder(Messages.getString("DataPackVi
ew.43"))); //$NON-NLS-1$
340:                pAddActionTime.add(new JLabel(IconDatabase.iconAddActionTime));
341:                pAddActionTime.add(Box.createHorizontalStrut(10));
342:                newActionTimeName.setPreferredSize(new Dimension(200, 25));
343:                newActionTimeName.setMaximumSize(new Dimension(200, 25));
344:                newActionTimeName
345:                .addFocusListener(generateMouseListener(Messages.getString("DataPa
ckView.44"))); //$NON-NLS-1$
346:                pAddActionTime.add(newActionTimeName);
347:                pAddActionTime.add(Box.createHorizontalGlue());
348:                addActionTime.addActionListener(new ActionListener() {
349:                        @Override
350:                        public void actionPerformed(ActionEvent arg0) {
351:                                if ((newActionTimeName.getText().length() > 0)
352:                                                && (!newActionTimeName.getText().e
quals(
353:                                                                Messages.getString
("DataPackView.45")))) { //$NON-NLS-1$
354:                                        dtp.addActionTime(newActionTimeName.getTex
t());
355:                                        actionTimes.validate();
356:                                        actionTimes
357:                                        .setSelectedIndex(actionTimes.getItemCount
() - 1);
358:                                        newActionTimeName
359:                                        .setText(Messages.getString("DataPackView.
46")); //$NON-NLS-1$
360:                                        newActionTimeName.requestFocus();
361:                                } else
362:                                        DialogHandlerFrame.showErrorDialog(mf,
363:                                                        Messages.getString("DataPa
ckView.47")); //$NON-NLS-1$
364:                        }
365:                });
366:                pAddActionTime.add(addActionTime);
367:
368:                // Ajouter une Ã©tape
369:                pAddStep.setLayout(new BoxLayout(pAddStep, BoxLayout.X_AXIS));
370:                pAddStep.setBorder(BorderFactory
371:                                .createTitledBorder(Messages.getString("DataPackVi
ew.48"))); //$NON-NLS-1$
372:                pAddStep.add(new JLabel(IconDatabase.iconAddStep));
373:                pAddStep.add(Box.createHorizontalStrut(10));
374:                newStepName.setPreferredSize(new Dimension(200, 25));
375:                newStepName.setMaximumSize(new Dimension(200, 25));
376:                newStepName
377:                .addFocusListener(generateMouseListener(Messages.getString("DataPa
ckView.49"))); //$NON-NLS-1$
378:                pAddStep.add(newStepName);
379:                pAddStep.add(Box.createHorizontalGlue());
380:                addStep.addActionListener(new ActionListener() {
381:                        @Override
382:                        public void actionPerformed(ActionEvent arg0) {
383:                                if ((newStepName.getText().length() > 0)
384:                                                && (!newStepName.getText().equals(
385:                                                                Messages.getString
("DataPackView.50")))) { //$NON-NLS-1$
386:                                        dtp.addStep(newStepName.getText());
387:                                        steps.validate();
388:                                        steps.setSelectedIndex(steps.getItemCount(
) - 1);
389:                                        newStepName.setText(Messages.getString("Da
taPackView.51")); //$NON-NLS-1$
390:                                        newStepName.requestFocus();
391:                                } else
392:                                        DialogHandlerFrame.showErrorDialog(mf,
393:                                                        Messages.getString("DataPa
ckView.52")); //$NON-NLS-1$
394:                        }
395:                });
396:                pAddStep.add(addStep);
397:
398:                // Ajouter jeu d'acteurs
399:                pAddActorSetView.setLayout(new BoxLayout(pAddActorSetView,
400:                                BoxLayout.Y_AXIS));
401:                pAddActorSetView.setBorder(BorderFactory
402:                                .createTitledBorder(Messages.getString("DataPackVi
ew.53"))); //$NON-NLS-1$
403:                //
404:                JPanel panelUp = new JPanel();
405:                panelUp.setLayout(new BoxLayout(panelUp, BoxLayout.X_AXIS));
406:                final JTextField newActorSetName = new JTextField(
407:                                Messages.getString("DataPackView.54")); //$NON-NLS
-1$
408:                newActorSetName.setPreferredSize(new Dimension(200, 25));
409:                newActorSetName.setMaximumSize(new Dimension(200, 25));
410:                newActorSetName
411:                .addFocusListener(generateMouseListener(Messages.getString("DataPa
ckView.55"))); //$NON-NLS-1$
412:                addActorSetView.addActionListener(new ActionListener() {
413:                        @Override
414:                        public void actionPerformed(ActionEvent arg0) {
415:                                String name = newActorSetName.getText();
416:                                if ((name.length() > 0)
417:                                                && (!name.equals(Messages.getStrin
g("DataPackView.56")))) { //$NON-NLS-1$
418:                                        if (dtp.getJeuActeurByName(name) == null)
{
419:                                                JeuActeur ja = new JeuActeur(name)
;
420:                                                dtp.addJeuActeur(ja);
421:                                                actorSets.validate();
422:                                                actorSets
423:                                                .setSelectedIndex(actorSets.getIte
mCount() - 1);
424:                                                int index = mf.tabbedPane.getTabCo
unt();
425:                                                mf.tabbedPane.add(ja.toString(), n
ew ActorSetView(ja));
426:                                                mf.tabbedPane.setSelectedIndex(ind
ex);
427:                                                newActorSetName
428:                                                .setText(Messages.getString("DataP
ackView.58")); //$NON-NLS-1$
429:
430:                                        } else
431:                                                DialogHandlerFrame
432:                                                .showErrorDialog(mf,
433:                                                                Messages.getString
("DataPackView.59")); //$NON-NLS-1$
434:                                } else
435:                                        DialogHandlerFrame.showErrorDialog(mf,
436:                                                        Messages.getString("DataPa
ckView.60")); //$NON-NLS-1$
437:                        }
438:                });
439:                panelUp.add(newActorSetName);
440:                panelUp.add(Box.createHorizontalGlue());
441:                panelUp.add(addActorSetView);
442:                //
443:                JPanel panelDown = new JPanel();
444:                panelDown.setLayout(new BoxLayout(panelDown, BoxLayout.X_AXIS));
445:                actorSets.setPreferredSize(new Dimension(225, 25));
```

```
446:                actorSets.setMaximumSize(new Dimension(225, 25));
447:                JButton modify = new JButton(Messages.getString("DataPackView.61")
); //$NON-NLS-1$
448:                modify.addActionListener(new ActionListener() {
449:                        @Override
450:                        public void actionPerformed(ActionEvent e) {
451:                                JeuActeur ja = dtp.getJeuActeurByName(actorSets
452:                                                .getSelectedItem().toString());
453:                                if (actorSets.getSelectedItem() != null
454:                                                && actorSets.getSelectedItem().toS
tring().length() > 0
455:                                                && ja != null) {
456:                                        int index = mf.tabbedPane.getTabCount();
457:                                        if (!mf.showTabByName(ja.toString()))
458:                                        {
459:                                                mf.tabbedPane.add(ja.toString(),
460:                                                                new ActorSetView(j
a));
461:                                                mf.tabbedPane.setSelectedIndex(ind
ex);
462:                                        }
463:                                } else
464:                                        DialogHandlerFrame
465:                                        .showErrorDialog(
466:                                                        mf,
467:                                                        Messages.getString("DataPa
ckView.62")); //$NON-NLS-1$
468:                        }
469:                });
470:                panelDown.add(actorSets);
471:                panelDown.add(Box.createHorizontalGlue());
472:                panelDown.add(modify);
473:                //
474:                pAddActorSetView.add(panelUp);
475:                pAddActorSetView.add(panelDown);
476:
477:                // Super barre : ajouter acteur/gpe + ajouter activitÃ©/moyen + aj
outer
478:                // brique + ajouter tps action
479:                megaAdd.setLayout(new BoxLayout(megaAdd, BoxLayout.Y_AXIS));
480:                megaAdd.add(pInfo);
481:                megaAdd.add(superAdd);
482:                megaAdd.add(pEditRelation);
483:                megaAdd.add(pAddActivityMoyen);
484:                megaAdd.add(pAddBrick);
485:                megaAdd.add(pAddActionTime);
486:                megaAdd.add(pAddStep);
487:                megaAdd.add(pAddActorSetView);
488:
489:                // JLists
490:                MouseListener listenerList = new MouseListener() {
491:                        @Override
492:                        public void mouseClicked(MouseEvent arg0) {
493:                                if (arg0.getClickCount() == 2) {
494:                                        JList source = (JList) arg0.getSource();
495:                                        Object object = source.getSelectedValue();
496:                                        if (object != null) {
497:                                                mf.addTab(object);
498:                                        }
499:                                }
500:                        }
501:
502:                        @Override
503:                        public void mouseEntered(MouseEvent arg0) {
504:                        }
505:
506:                        @Override
507:                        public void mouseExited(MouseEvent arg0) {
508:                        }
509:
510:                        @Override
511:                        public void mousePressed(MouseEvent arg0) {
512:                        }
513:
514:                        @Override
515:                        public void mouseReleased(MouseEvent arg0) {
516:                        }
517:                };
518:                listBricks.addMouseListener(listenerList);
519:                listBricks
520:                .setToolTipText(Messages.getString("DataPackView.63")); //$NON-NLS
-1$
521:
522:                FocusListener focusListenerList = new FocusListener() {
523:                        @Override
524:                        public void focusGained(FocusEvent e) {
525:                                listBricks.setListData(dtp.getBrickList());
526:                        }
527:
528:                        @Override
529:                        public void focusLost(FocusEvent e) {
530:                        }
531:                };
532:                mf.tabbedPane.addFocusListener(focusListenerList);
533:                listBricks.addFocusListener(focusListenerList);
534:
535:                ComponentListener compListListener = new ComponentListener() {
536:                        @Override
537:                        public void componentHidden(ComponentEvent e) {
538:                        }
539:
540:                        @Override
541:                        public void componentMoved(ComponentEvent e) {
542:                        }
543:
544:                        @Override
545:                        public void componentResized(ComponentEvent e) {
546:                        }
547:
548:                        @Override
549:                        public void componentShown(ComponentEvent e) {
550:                                System.out.println("Appel componentShown JLIST");
//$NON-NLS-1$
551:                                if (e.getSource() == listBricks) {
552:                                        if (listBricks.getModel().getSize() < dtp.
getBrickList()
553:                                                        .size()) {
554:                                                DefaultListModel listModel = (Defa
ultListModel) listBricks
555:                                                .getModel();
556:                                                listModel.removeAllElements();
557:                                                for (Brick brick : dtp.getBrickLis
t()) {
558:                                                        listModel.addElement(brick
);
559:                                                }
560:                                        }
561:                                }
562:
563:                        }
564:                };
565:                listBricks.addComponentListener(compListListener);
566:                jspBricks.setBorder(BorderFactory
567:                                .createTitledBorder(Messages.getString("DataPackVi
```

```
ew.65"))); //$NON-NLS-1$
  568:                  pLists.add(jspBricks, BorderLayout.WEST);
  569:
  570:                  JPanel sideListPanel = new JPanel(new BorderLayout());
  571:                  JPanel listButtonPanel = new JPanel();
  572:                  listButtonPanel.setLayout(new BoxLayout(listButtonPanel,BoxLayout.
Y_AXIS));
  573:                  final BrickParameterPopUp brickParameterPanel = new BrickParameter
PopUp((Brick)listBricks.getSelectedValue());
  574:                  sideListPanel.add(brickParameterPanel, BorderLayout.CENTER);
  575:                  JButton openBrick = new JButton(IconDatabase.iconOpen);
  576:                  openBrick.setToolTipText(Messages.getString("DataPackView.66")); /
/$NON-NLS-1$
  577:                  openBrick.addActionListener(new ActionListener() {
  578:
  579:                          @Override
  580:                          public void actionPerformed(ActionEvent e) {
  581:                                  if (listBricks.getSelectedValue() != null)
  582:                                          mf.addTab(listBricks.getSelectedValue());
  583:                          }
  584:                  });
  585:
  586:                  JButton editBrick = new JButton(IconDatabase.iconRename);
  587:                  editBrick.setToolTipText(Messages.getString("DataPackView.67")); /
/$NON-NLS-1$
  588:                  editBrick.addActionListener(new ActionListener() {
  589:
  590:                          @Override
  591:                          public void actionPerformed(ActionEvent e) {
  592:                                  if (listBricks.getSelectedValue() != null)
  593:                                          brickParameterPanel.showMeTheBrick((Brick)
listBricks.getSelectedValue());
  594:                          }
  595:                  });
  596:
  597:                  JButton copyBrick = new JButton(IconDatabase.iconNew);
  598:                  copyBrick.setToolTipText("Copier la brique");
  599:                  copyBrick.addActionListener(new ActionListener() {
  600:
  601:                          @Override
  602:                          public void actionPerformed(ActionEvent e) {
  603:                                  if (listBricks.getSelectedValue() != null)
  604:                                  {
  605:                                          Brick source = (Brick)listBricks.getSelect
edValue();
  606:                                          Brick copy = new Brick(source);
  607:                                          copy.setName("Copie de "+source.getName())
;
  608:                                          brickParameterPanel.showMeTheBrick(copy);
  609:                                          dtp.getBrickList().add(copy);
  610:                                  }
  611:
  612:                          }
  613:                  });
  614:
  615:                  JButton deleteBrick = new JButton(IconDatabase.iconDelete);
  616:                  deleteBrick.setToolTipText(Messages.getString("DataPackView.68"));
 //$NON-NLS-1$
  617:                  deleteBrick.addActionListener(new ActionListener() {
  618:
  619:                          @Override
  620:                          public void actionPerformed(ActionEvent e) {
  621:
  622:                                  Program.myMainFrame.closeTab(((Brick)listBricks.ge
tSelectedValue()).toString());
  623:
  624:                                  if (DialogHandlerFrame.showYesNoDialog(Messages.ge
tString("DataPackView.69")+ listBricks.getSelectedValue()+ " ?") == JOptionPane.YES_OPTIO
N) //$NON-NLS-1$ //$NON-NLS-2$
  625:                                  {
  626:                                          dtp.removeBrick((Brick)listBricks.getSelec
tedValue());
  627:                                  }
  628:
  629:
  630:                          }
  631:                  });
  632:
  633:                  listButtonPanel.add(Box.createGlue());
  634:                  listButtonPanel.add(openBrick);
  635:                  listButtonPanel.add(editBrick);
  636:                  listButtonPanel.add(copyBrick);
  637:                  listButtonPanel.add(deleteBrick);
  638:                  listButtonPanel.add(Box.createGlue());
  639:                  sideListPanel.add(listButtonPanel,BorderLayout.EAST);
  640:                  pLists.add(sideListPanel, BorderLayout.EAST);
  641:
  642:                  // ComboBoxes : activitÃ©s, groupes et bricktypes
  643:
  644:                  JPanel activitiesCB = new JPanel();
  645:                  activitiesCB.setLayout(new BoxLayout(activitiesCB, BoxLayout.X_AXI
S));
  646:                  JButton delete = new JButton(IconDatabase.iconDelete);
  647:                  JButton rename = new JButton(IconDatabase.iconRename);
  648:                  rename.setToolTipText(Messages.getString("DataPackView.71")); //$N
ON-NLS-1$
  649:                  delete.setToolTipText(Messages.getString("DataPackView.72")); //$N
ON-NLS-1$
  650:
  651:                  rename.addActionListener(new ActionListener() {
  652:
  653:                          @Override
  654:                          public void actionPerformed(ActionEvent e) {
  655:                                  if (activities.getSelectedItem() == null)
  656:                                          return;
  657:
  658:                                  String newName = JOptionPane.showInputDialog(Messa
ges.getString("DataPackView.73") + activities.getSelectedItem() + Messages.getString("Dat
aPackView.74"), activities.getSelectedItem()); //$NON-NLS-1$ //$NON-NLS-2$
  659:                                  newName.trim();
  660:                                  if (newName.length()>0 && !newName.equals(activiti
es.getSelectedItem().toString()))
  661:                                  {
  662:                                          if (dtp.renameActivity(activities.getSelec
tedItem().toString(), newName))
  663:                                                  activities.repaint();
  664:                                          else
  665:                                                  DialogHandlerFrame.showErrorDialog
(Messages.getString("DataPackView.75")); //$NON-NLS-1$
  666:                                  }
  667:
  668:
  669:                          }
  670:                  });
  671:
  672:
  673:                  delete.addActionListener(new ActionListener() {
  674:
  675:                          @Override
  676:                          public void actionPerformed(ActionEvent e) {
  677:                                  Activite deletedActivity = (Activite) activities
  678:                                  .getSelectedItem();
  679:                                  if (deletedActivity != null)
  680:                                  {
```

```
681:                                        if (dtp.removeActivity(deletedActivity)) {
682:                                                if(activities.getItemCount() > 0)
{
683:                                                        activities.setSelectedInde
x(0);
684:                                                } else {
685:                                                        activities.setSelectedItem
(null);
686:                                                }
687:                                                activities.validate();
688:                                        }
689:                                }
690:                        }
691:                });
692:                activitiesCB.add(activities);
693:                activitiesCB.add(rename);
694:                activitiesCB.add(delete);
695:
696:                JPanel stepsCB = new JPanel();
697:                stepsCB.setLayout(new BoxLayout(stepsCB,BoxLayout.X_AXIS));
698:
699:                rename = new JButton(IconDatabase.iconRename);
700:
701:                rename.addActionListener(new ActionListener() {
702:
703:                        @Override
704:                        public void actionPerformed(ActionEvent e) {
705:
706:                                if (steps.getSelectedItem() == null)
707:                                        return;
708:
709:                                String newName = JOptionPane.showInputDialog(Messa
ges.getString("DataPackView.76")+ steps.getSelectedItem() +Messages.getString("DataPackVi
ew.77"), steps.getSelectedItem()); //$NON-NLS-1$ //$NON-NLS-2$
710:                                newName.trim();
711:                                if (newName.length()>0 && !newName.equals(steps.ge
tSelectedItem().toString()))
712:                                {
713:                                        if (dtp.renameStep(steps.getSelectedItem()
.toString(), newName))
714:                                                steps.repaint();
715:                                        else
716:                                                DialogHandlerFrame.showErrorDialog
(Messages.getString("DataPackView.78")); //$NON-NLS-1$
717:                                }
718:                        }
719:                });
720:
721:                delete = new JButton(IconDatabase.iconDelete);
722:                delete.addActionListener(new ActionListener() {
723:
724:                        @Override
725:                        public void actionPerformed(ActionEvent e) {
726:
727:                                String deletedStep = (String) steps
728:                                .getSelectedItem();
729:                                if (deletedStep != null)
730:                                {
731:                                        if (dtp.removeStep(deletedStep)) {
732:                                                if(steps.getItemCount() > 0) {
733:                                                        steps.setSelectedIndex(0);
734:                                                } else {
735:                                                        steps.setSelectedItem(null
);
736:                                                }
737:                                                steps.validate();
738:                                        }
739:                                }
740:
741:                        }
742:                });
743:
744:                rename.setToolTipText(Messages.getString("DataPackView.79")); //$N
ON-NLS-1$
745:                delete.setToolTipText(Messages.getString("DataPackView.80")); //$N
ON-NLS-1$
746:                stepsCB.add(steps);
747:                stepsCB.add(rename);
748:                stepsCB.add(delete);
749:
750:                JPanel actionTimesCB = new JPanel();
751:                actionTimesCB.setLayout(new BoxLayout(actionTimesCB, BoxLayout.X_A
XIS));
752:
753:                rename = new JButton(IconDatabase.iconRename);
754:
755:                rename.addActionListener(new ActionListener() {
756:
757:                        @Override
758:                        public void actionPerformed(ActionEvent e) {
759:                                if (actionTimes.getSelectedItem() == null)
760:                                        return;
761:
762:                                String newName = JOptionPane.showInputDialog(Messa
ges.getString("DataPackView.81") + actionTimes.getSelectedItem() + Messages.getString("Da
taPackView.82"), actionTimes.getSelectedItem()); //$NON-NLS-1$ //$NON-NLS-2$
763:                                newName.trim();
764:                                if (newName.length()>0 && !newName.equals(actionTi
mes.getSelectedItem().toString()))
765:                                {
766:                                        if (dtp.renameActionTime(actionTimes.getSe
lectedItem().toString(), newName))
767:                                                actionTimes.repaint();
768:                                        else
769:                                                DialogHandlerFrame.showErrorDialog
(Messages.getString("DataPackView.83")); //$NON-NLS-1$
770:                                }
771:                        }
772:                });
773:
774:                delete = new JButton(IconDatabase.iconDelete);
775:                delete.addActionListener(new ActionListener() {
776:
777:                        @Override
778:                        public void actionPerformed(ActionEvent arg0) {
779:
780:                                ActionTime deletedActionTime = (ActionTime) action
Times
781:                                .getSelectedItem();
782:                                if (deletedActionTime != null) {
783:                                        if (dtp.removeActionTime(deletedActionTime
)) {
784:                                                {
785:                                                        if(actionTimes.getItemCoun
t() > 0) {
786:                                                                actionTimes.setSel
ectedIndex(0);
787:                                                        } else {
788:                                                                actionTimes.setSel
ectedItem(null);
789:                                                        }
790:                                                        actionTimes.validate();
791:                                                }
792:                                        }
```

```
793:                                    }
794:
795:                            }
796:                    });
797:
798:                    rename.setToolTipText(Messages.getString("DataPackView.84")); //$N
ON-NLS-1$
799:                    delete.setToolTipText(Messages.getString("DataPackView.85")); //$N
ON-NLS-1$
800:                    actionTimesCB.add(actionTimes);
801:                    actionTimesCB.add(rename);
802:                    actionTimesCB.add(delete);
803:
804:                    JPanel moyensCB = new JPanel();
805:                    moyensCB.setLayout(new BoxLayout(moyensCB, BoxLayout.X_AXIS));
806:
807:                    rename = new JButton(IconDatabase.iconRename);
808:
809:                    rename.addActionListener(new ActionListener() {
810:
811:                            @Override
812:                            public void actionPerformed(ActionEvent e) {
813:                                    DialogHandlerFrame.showInformationDialog(mf, Messa
ges.getString("DataPackView.86"), Messages.getString("DataPackView.87"), null); //$NON-NL
S-1$ //$NON-NLS-2$
814:                            }
815:                    });
816:
817:                    delete = new JButton(IconDatabase.iconDelete);
818:                    delete.addActionListener(new ActionListener() {
819:
820:                            @Override
821:                            public void actionPerformed(ActionEvent arg0) {
822:
823:                                    Moyen deletedMoyen = (Moyen) moyens.getSelectedIte
m();
824:                                    if (deletedMoyen != null) {
825:                                            if (dtp.removeMoyen(deletedMoyen)) {
826:                                                    moyens.removeItem(deletedMoyen);
827:                                                    if(moyens.getItemCount() > 0) {
828:                                                            moyens.setSelectedIndex(0)
;
829:                                                    } else {
830:                                                            moyens.setSelectedItem(nul
l);
831:                                                    }
832:                                                    moyens.validate();
833:                                            }
834:                                    }
835:
836:                            }
837:                    });
838:
839:                    rename.setToolTipText(Messages.getString("DataPackView.88")); //$N
ON-NLS-1$
840:                    delete.setToolTipText(Messages.getString("DataPackView.89")); //$N
ON-NLS-1$
841:                    moyensCB.add(moyens);
842:                    moyensCB.add(rename);
843:                    moyensCB.add(delete);
844:
845:                    pInformation
846:                    .setBorder(BorderFactory.createTitledBorder(Messages.getString("Da
taPackView.90"))); //$NON-NLS-1$
847:                    pInformation.add(new JLabel(Messages.getString("DataPackView.91"))
); //$NON-NLS-1$
848:                    pInformation.add(new JLabel(Messages.getString("DataPackView.92"))
); //$NON-NLS-1$
849:                    pInformation.add(activitiesCB);
850:                    pInformation.add(stepsCB);
851:                    pInformation.add(new JLabel(Messages.getString("DataPackView.93"))
); //$NON-NLS-1$
852:                    pInformation.add(new JLabel(Messages.getString("DataPackView.94"))
); //$NON-NLS-1$
853:                    pInformation.add(actionTimesCB);
854:                    pInformation.add(moyensCB);
855:
856:                    // Espace central
857:                    center.add(megaAdd, BorderLayout.NORTH);
858:                    center.add(pInformation, BorderLayout.CENTER);
859:                    center.add(pLists, BorderLayout.SOUTH);
860:                    jspCenter.setBorder(null);
861:
862:                    // Barre de boutons
863:                    pButtons.setLayout(new BoxLayout(pButtons, BoxLayout.X_AXIS));
864:                    pButtons.add(Box.createGlue());
865:                    save.addActionListener(Program.myMainFrame.controller.actionSave);
866:                    pButtons.add(save);
867:                    pButtons.add(Box.createGlue());
868:                    south.add(new JSeparator(), BorderLayout.NORTH);
869:                    south.add(pButtons, BorderLayout.CENTER);
870:
871:                    // Panel total
872:                    add(north, BorderLayout.NORTH);
873:                    jspCenter.getVerticalScrollBar().setUnitIncrement(50);
874:                    add(jspCenter, BorderLayout.CENTER);
875:                    add(south, BorderLayout.SOUTH);
876:            }
877:
878:            private String getCompanyNameLabel() {
879:                    String companyName = dtp.getCompanyInfo().getName();
880:                    if(companyName == null) {
881:                            companyName = "Nom de l’entreprise non renseignÃ©";
882:                    } else {
883:                            companyName = "Nom de l’entreprise : " + companyName;
884:                    }
885:
886:                    return companyName;
887:            }
888:
889:            public static FocusListener generateMouseListener(
890:                            final String initialString) {
891:                    return new FocusListener() {
892:                            @Override
893:                            public void focusGained(FocusEvent e) {
894:                                    JTextField txtField = (JTextField) e.getSource();
895:                                    if (txtField.getText().equals(initialString)) {
896:                                            txtField.setText(""); //$NON-NLS-1$
897:                                    }
898:                            }
899:
900:                            @Override
901:                            public void focusLost(FocusEvent e) {
902:                                    JTextField txtField = (JTextField) e.getSource();
903:                                    if (txtField.getText().equals("")) { //$NON-NLS-1$
904:                                            txtField.setText(initialString);
905:                                    }
906:                            }
907:                    };
908:            }
909:
910:            public void updateView() {
911:                    nameLabel.setText(getCompanyNameLabel());
912:            }
```

```
913: }
```

```
    1: package graphicalUserInterface;
    2:
    3: import java.io.File;
    4: import java.io.FileNotFoundException;
    5: import java.io.FileOutputStream;
    6: import java.io.PrintStream;
    7: import java.util.Date;
    8:
    9: public class Logger {
   10:         final private PrintStream err = System.err;
   11:         final private PrintStream out = System.out;
   12:
   13:         final private String fileLogName = System.getProperty("user.home") + File.
separatorChar + "Triade" + File.separatorChar + "log"; //$NON-NLS-1$
   14:         final private String fileLogTriadeName = "triadeLog"; //$NON-NLS-1$
   15:         final private String fileLogCreatorName = "creatorLog"; //$NON-NLS-1$
   16:
   17:         private String fileLogePath;
   18:
   19:         static private Logger singleton = null;
   20:
   21:         public void setLoggerTriade(Boolean value) {
   22:                 if (value) {
   23:                         setLogger(fileLogName + File.separatorChar + fileLogTriade
Name); //$NON-NLS-1$
   24:                 } else {
   25:                         setStandardLogger();
   26:                 }
   27:         }
   28:
   29:         public void setLoggerCreator(Boolean value) {
   30:                 if (value) {
   31:                         setLogger(fileLogName + File.separatorChar + fileLogCreato
rName); //$NON-NLS-1$
   32:                 } else {
   33:                         setStandardLogger();
   34:                 }
   35:         }
   36:
   37:         private void setLogger(String directory) {
   38:                 File logPath = new File(directory);
   39:                 if ((logPath.exists() && logPath.isDirectory()) == false) {
   40:                         logPath.mkdirs();
   41:                 }
   42:
   43:                 String date = new Date().toString();
   44:                 date = date.replace(':', '-');
   45:
   46:                 FileOutputStream fos = null;
   47:
   48:                 try {
   49:                         fileLogePath = directory + File.separatorChar + "log_erreu
r-" + date; //$NON-NLS-1$
   50:                         fos = new FileOutputStream(fileLogePath, true);
   51:                 } catch (FileNotFoundException e) {
   52:                         e.printStackTrace();
   53:                 }
   54:
   55:                 PrintStream pS = new PrintStream(fos, true);
   56:
   57:                 System.setErr(pS);
   58:                 System.setOut(pS);
   59:         }
   60:
   61:         public void setStandardLogger() {
   62:                 System.setErr(err);
   63:                 System.setOut(out);
   64:         }
   65:
   66:         public String getFileLogPath() {
   67:                 return fileLogePath;
   68:         }
   69:
   70:         static public Logger getInstance() {
   71:                 if(singleton == null) {
   72:                         singleton = new Logger();
   73:                 }
   74:
   75:                 return singleton;
   76:         }
   77: }
```

```
    1: package translation;
    2:
    3: import java.util.MissingResourceException;
    4: import java.util.ResourceBundle;
    5:
    6: public class Messages {
    7:         private static final String BUNDLE_NAME = "translation.messages"; //$NON-N
LS-1$
    8:
    9:         private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle
   10:                         .getBundle(BUNDLE_NAME);
   11:
   12:         private Messages() {
   13:         }
   14:
   15:         public static String getString(String key) {
   16:                 try {
   17:                         return RESOURCE_BUNDLE.getString(key);
   18:                 } catch (MissingResourceException e) {
   19:                         return '!' + key + '!';
   20:                 }
   21:         }
   22: }
```

```
   1: package Mailing;
   2:
   3: import graphicalUserInterface.Program;
   4:
   5: import java.awt.Dimension;
   6: import java.awt.GridBagLayout;
   7: import java.awt.event.ActionEvent;
   8: import java.awt.event.ActionListener;
   9:
  10: import javax.swing.Box;
  11: import javax.swing.BoxLayout;
  12: import javax.swing.JButton;
  13: import javax.swing.JComboBox;
  14: import javax.swing.JFrame;
  15: import javax.swing.JLabel;
  16: import javax.swing.JPanel;
  17: import javax.swing.JScrollPane;
  18: import javax.swing.JTextArea;
  19: import javax.swing.JTextField;
  20:
  21: import translation.Messages;
  22:
  23: public class MailView extends JFrame implements ActionListener {
  24:         private static final long serialVersionUID = 9022335411694658914L;
  25:
  26:         private final JTextField subject;
  27:         private final JTextArea text;
  28:         private final JTextField mail;
  29:         private final JComboBox formType;
  30:         private final MailForm mailForm;
  31:
  32:         private final JButton cancelButton;
  33:         private final JButton okButton;
  34:
  35:         public MailView() {
  36:                 this(Messages.getString("MailView.0"), null); //$NON-NLS-1$
  37:         }
  38:
  39:         public MailView(String title) {
  40:                 this(title, null);
  41:         }
  42:
  43:         public MailView(String viewTitle, MailForm mailForm) {
  44:                 super(viewTitle);
  45:
  46:                 this.mailForm = mailForm;
  47:
  48:                 setLayout(new BoxLayout(this.getContentPane(), BoxLayout.Y_AXIS));
  49:                 //setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
  50:
  51:                 subject = new JTextField(mailForm.getInitSubject()); //$NON-NLS-1$
  52:                 subject.setPreferredSize(new Dimension(200, 25));
  53:                 text = new JTextArea(mailForm.getInitText()); //$NON-NLS-1$
  54:                 text.setPreferredSize(new Dimension(200, 400));
  55:                 mail = new JTextField(Program.myMainFrame.getDataPack().getCompany
Info().getEmail());
  56:                 mail.setPreferredSize(new Dimension(200, 25));
  57:                 formType = new JComboBox(MailFormFactory.getFormTypes());
  58:                 formType.setPreferredSize(new Dimension(200, 25));
  59:
  60:                 JScrollPane textScrollPane = new JScrollPane(text);
  61:
  62:                 JLabel comboBoxLabel = new JLabel(Messages.getString("MailView.4")
); //$NON-NLS-1$
  63:                 comboBoxLabel.setPreferredSize(new Dimension(200, 25));
  64:
  65:                 JLabel subjectLabel = new JLabel(Messages.getString("MailView.5"))
; //$NON-NLS-1$
  66:                 subjectLabel.setPreferredSize(new Dimension(200, 25));
  67:                 JLabel textLabel = new JLabel(Messages.getString("MailView.6")); /
/$NON-NLS-1$
  68:                 textLabel.setPreferredSize(new Dimension(200, 25));
  69:                 JLabel mailLabel = new JLabel(Messages.getString("MailView.7")); /
/$NON-NLS-1$
  70:                 mailLabel.setPreferredSize(new Dimension(200, 25));
  71:                 JLabel title = new JLabel(Messages.getString("MailView.8")); //$NO
N-NLS-1$
  72:
  73:                 cancelButton = new JButton(Messages.getString("MailView.9")); //$N
ON-NLS-1$
  74:                 okButton = new JButton(Messages.getString("MailView.10")); //$NON-
NLS-1$
  75:
  76:                 okButton.addActionListener(this);
  77:                 cancelButton.addActionListener(this);
  78:
  79:                 JPanel comboBoxPanel = new JPanel(new GridBagLayout());
  80:
  81:                 JPanel subjectPanel = new JPanel();
  82:                 subjectPanel.setLayout(new BoxLayout(subjectPanel, BoxLayout.X_AXI
S));
  83:
  84:                 JPanel textPanel = new JPanel();
  85:                 textPanel.setLayout(new BoxLayout(textPanel, BoxLayout.X_AXIS));
  86:
  87:                 JPanel mailPanel = new JPanel();
  88:                 mailPanel.setLayout(new BoxLayout(mailPanel, BoxLayout.X_AXIS));
  89:
  90:                 JPanel buttonsPanel = new JPanel();
  91:                 buttonsPanel.setLayout(new BoxLayout(buttonsPanel, BoxLayout.X_AXI
S));
  92:
  93:                 JPanel titlePanel = new JPanel(new GridBagLayout());
  94:
  95:                 comboBoxPanel.add(comboBoxLabel);
  96:                 comboBoxPanel.add(Box.createVerticalGlue());
  97:                 comboBoxPanel.add(formType);
  98:
  99:                 titlePanel.add(title);
 100:
 101:                 mailPanel.add(mailLabel);
 102:                 mailPanel.add(Box.createVerticalGlue());
 103:                 mailPanel.add(mail);
 104:
 105:                 subjectPanel.add(subjectLabel);
 106:                 subjectPanel.add(Box.createVerticalGlue());
 107:                 subjectPanel.add(subject);
 108:
 109:                 textPanel.add(textLabel);
 110:                 textPanel.add(Box.createVerticalGlue());
 111:                 textPanel.add(textScrollPane);
 112:
 113:                 buttonsPanel.add(okButton);
 114:                 buttonsPanel.add(cancelButton);
 115:
 116:                 if(mailForm == null) {
 117:                         add(comboBoxPanel);
 118:                 }
 119:
 120:                 add(titlePanel);
 121:                 add(mailPanel);
 122:                 add(subjectPanel);
 123:                 add(textPanel);
 124:                 add(buttonsPanel);
```

```
  125:
  126:                 setSize(new Dimension(800, 600));
  127:                 setVisible(true);
  128:
  129:                 validate();
  130:                 repaint();
  131:         }
  132:
  133:         @Override
  134:         public void actionPerformed(ActionEvent arg0) {
  135:
  136:
  137:                 if(arg0.getSource() == okButton) {
  138:                         mailForm.setContent(subject.getText(), text.getText(), mai
l.getText());
  139:                         if(mailForm.sendMail()) {
  140:                                 setVisible(false);
  141:                         }
  142:
  143:                         System.out.println("ok"); //$NON-NLS-1$
  144:                 } else if(arg0.getSource() == cancelButton) {
  145:                         setVisible(false);
  146:                         System.out.println("cancel"); //$NON-NLS-1$
  147:                 }
  148:         }
  149:         /*
  150:         public static void main(String[] args) {
  151:                 JFrame frame = new JFrame();
  152: //              frame.setLayout(new GridLayout());
  153:                 frame.setSize(800,600);
  154:                 frame.setVisible(true);
  155:
  156:                 frame.add(new MailView());
  157:
  158:                 frame.validate();
  159:         }*/
  160: }
```

```
    1: package Mailing;
    2:
    3: import graphicalUserInterface.Program;
    4: import translation.Messages;
    5:
    6: public class MailAddRelationForm extends MailForm {
    7:
    8:         public MailAddRelationForm() {
    9:                 super(MailSender.robertMail);
   10:         }
   11:
   12:         @Override
   13:         protected String getSubject() {
   14:                 return  subject; //$NON-NLS-1$
   15:         }
   16:
   17:         @Override
   18:         protected String getText() {
   19:                 return  text; //$NON-NLS-1$
   20:         }
   21:
   22:         @Override
   23:         protected String getInitText() {
   24:                 return Messages.getString("MailAddRelationForm.addRelationSubject"
);
   25:         }
   26:
   27:         @Override
   28:         protected String getInitSubject() {
   29:                 return Program.myMainFrame.getDataPack().getCompanyInfo().getName(
) + " " + Messages.getString("MailAddRelationForm.addRelationIntroduction");
   30:         }
   31: }
```

```
    1: package Mailing;
    2:
    3: import graphicalUserInterface.DialogHandlerFrame;
    4: import graphicalUserInterface.Program;
    5: import translation.Messages;
    6:
    7:
    8: abstract public class MailForm {
    9:         protected String mail;
   10:         protected String subject;
   11:         protected String text;
   12:         protected final String to;
   13:
   14:         abstract protected String getInitText();
   15:         abstract protected String getInitSubject();
   16:
   17:         abstract protected String getSubject();
   18:         abstract protected String getText();
   19:
   20:         public MailForm(String to) {
   21:                 this.to = to;
   22:         }
   23:
   24:         protected String getDescription() {
   25:                 return ""; //$NON-NLS-1$
   26:         }
   27:
   28:         public void setContent(String subject, String text, String mail) {
   29:                 this.mail = mail;
   30:                 this.subject = subject;
   31:                 this.text = text;
   32:         }
   33:
   34:         public boolean sendMail() {
   35:                 String text = getText();
   36:                 String subject = getSubject();
   37:
   38:                 boolean result;
   39:
   40:                 if(to.compareTo(MailSender.supportMail) == 0) {
   41:                         result = MailSender.sendMailToSupportTeam(mail, subject, t
ext);
   42:                 } else {
   43:                         result = MailSender.getSingleton().sendMailTo(mail, to, su
bject, text, null);
   44:                 }
   45:
   46:                 if(result) {
   47:                         DialogHandlerFrame.showInformationDialog(Program.myMainFra
me, Messages.getString("MailForm.1"), Messages.getString("MailForm.2"), null); //$NON-NLS
-1$ //$NON-NLS-2$
   48:                 } else {
   49:                         DialogHandlerFrame.showErrorDialog(Messages.getString("Mai
lForm.3")); //$NON-NLS-1$
   50:                 }
   51:
   52:                 return result;
   53:         }
   54: }
```

```
    1: package Mailing;
    2:
    3:
    4: public class MailErrorForm extends MailForm {
    5:
    6:         private final Throwable exception;
    7:
    8:         public MailErrorForm(Throwable e) {
    9:                 super(MailSender.supportMail);
   10:                 exception = e;
   11:         }
   12:
   13:         @Override
   14:         protected String getInitText() {
   15:                 String text = "Informations complÃ©mentaire (DÃ©taillez l'action q
ui a provoquÃ© l'erreur) : \n\n\nRapport d'erreur : \n";
   16:                 text += exception.toString() + "<-\n";
   17:                 for(int i = 0 ; i < exception.getStackTrace().length ; i++) {
   18:                         text += exception.getStackTrace()[i].toString() + "\n";
   19:                 }
   20:                 return  text;
   21:         }
   22:
   23:         @Override
   24:         protected String getInitSubject() {
   25:                 return "Rapport d'erreur";
   26:         }
   27:
   28:         @Override
   29:         protected String getSubject() {
   30:                 return subject;
   31:         }
   32:
   33:         @Override
   34:         protected String getText() {
   35:                 return text;
   36:         }
   37:
   38: }
```

```
    1: package Mailing;
    2:
    3: import java.util.Vector;
    4:
    5: import translation.Messages;
    6:
    7: public class MailFormFactory {
    8:         private static final String relationForm = Messages.getString("MailFormFac
tory.addRelationSubject"); //$NON-NLS-1$
    9:         private static final String informationForm = Messages.getString("MailForm
Factory.informationRequestSubject"); //$NON-NLS-1$
   10:
   11:         public static MailForm getInstance(Object name) {
   12:                 MailForm mailForm = null;
   13:
   14:                 if(name == relationForm) {
   15:                         mailForm = new MailAddRelationForm();
   16:                 } else {
   17:                         throw new IllegalArgumentException(name + " is not a MailF
orm type !"); //$NON-NLS-1$
   18:                 }
   19:
   20:                 return mailForm;
   21:         }
   22:
   23:         public static Object[] getFormTypes() {
   24:                 Vector<String> types = new Vector<String>();
   25:
   26:                 types.add(relationForm);
   27:                 types.add(informationForm);
   28:
   29:                 return types.toArray();
   30:         }
   31: }
```

```java
package Mailing;

import graphicalUserInterface.Logger;
import graphicalUserInterface.Program;

import java.io.File;
import java.util.Date;
import java.util.Properties;
import java.util.Vector;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.mail.Message;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;

import translation.Messages;
import dataPack.DataPack;

public class MailSender {
	private final Properties props;
	private final Session session;
	private final Message message;

		public static String robertMail = "dutertre@enseirb-matmeca.fr, nadal@labri.fr"; //$NON-NLS-1$
		public static String supportMail = "dutertre@enseirb-matmeca.fr, nadal@labri.fr"; //$NON-NLS-1$
	//	public static String robertMail = "dutertre@enseirb-matmeca.fr"; //$NON-NLS-1$
	//	public static String supportMail = "dutertre@enseirb-matmeca.fr"; //$NON-NLS-1$


		private static String defaultUsername = "popey.laurent@gmail.com"; //$NON-NLS-1$
	private static String defaultPassword = "Poplaurent"; //$NON-NLS-1$

	private static final MailSender defaultMailSender = new MailSender(
			defaultUsername, defaultPassword);

	public MailSender(final String username, final String password) {
		// smtp properties
		props = new Properties();
		props.put("mail.smtp.host", "smtp.gmail.com"); //$NON-NLS-1$ //$NON-NLS-2$
		props.put("mail.smtp.socketFactory.port", "465"); //$NON-NLS-1$ //$NON-NLS-2$
		props.put("mail.smtp.socketFactory.class", //$NON-NLS-1$
		"javax.net.ssl.SSLSocketFactory"); //$NON-NLS-1$
		props.put("mail.smtp.auth", "true"); //$NON-NLS-1$ //$NON-NLS-2$
		props.put("mail.smtp.port", "465"); //$NON-NLS-1$ //$NON-NLS-2$

		// authentification
		session = Session.getDefaultInstance(props,
				new javax.mail.Authenticator() {
			@Override
			protected PasswordAuthentication getPasswordAuthentication() {

				return new PasswordAuthentication(username, password);
			}
		});

		message = new MimeMessage(session);
	}

	public boolean sendMailTo(String from, String to, String subject,
			String text, Vector<String> files) {
		try {
			// set message
			message.setRecipients(Message.RecipientType.TO,
					InternetAddress.parse(to));
			message.setSubject(subject);

			MimeMultipart body = new MimeMultipart();

			MimeBodyPart textBody = new MimeBodyPart();
			textBody.setText(text);

			body.addBodyPart(textBody);

			if (files != null) {
				for (String filename : files) {
					if(filename != null) {
						File file = new File(filename);
						if(file.exists() == false) {
							System.out.println("fichier " + filename + " non trouvÃ©, impossible de l'ajouter au mail");
							continue;
						}

						System.out.println("Add file to mail : " + filename + "(" + filename.substring(filename.lastIndexOf('/')+1) + ")"); //$NON-NLS-1$
						MimeBodyPart dataBody = new MimeBodyPart();

						FileDataSource fileDataSource = new FileDataSource(filename) {
							@Override
							public String getContentType() {
								return "application/octet-stream"; //$NON-NLS-1$
							}
						};
						dataBody.setDataHandler(new DataHandler(fileDataSource));
						dataBody.setFileName(filename.substring(filename.lastIndexOf('/')+1));

						body.addBodyPart(dataBody);
					}
				}
			}

			message.setSentDate(new Date());
			message.setContent(body, "application/octet-stream"); //$NON-NLS-1$
			message.setReplyTo(InternetAddress.parse(from));

			// send email
			Transport.send(message);

			return true;
		} catch (Exception e) {
			e.printStackTrace();
			return false;
		}
```

```
  116:         }
  117:
  118:         private static String createSubject(String subject) {
  119:                 return Messages.getString("MailSender.subjectPrefix") + subject; //
/$NON-NLS-1$
  120:         }
  121:
  122:         static public boolean sendMailToRobert(String from, String subject,
  123:                         String text) {
  124:                 System.out.println("Send mail to robert"); //$NON-NLS-1$
  125:                 // return true;
  126:                 return defaultMailSender.sendMailTo(from, robertMail,
  127:                                 createSubject(subject) + Messages.getString("MailS
ender.subjectSuffix"), text, null); //$NON-NLS-1$
  128:         }
  129:
  130:         static public boolean sendMailToSupportTeam(String from, String subject,
  131:                         String text) {
  132:
  133:                 System.out.println("Send mail to support team"); //$NON-NLS-1$
  134:
  135:                 Vector<String> files = new Vector<String>();
  136:
  137:                 File file;
  138:                 int cmp = 0;
  139:
  140:                 DataPack datapack = Program.myMainFrame.datapack;
  141:
  142:                 if(datapack.getFilePath() != null) {
  143:                         do {
  144:                                 file = new File(datapack.getFilePath() + ".save" +
 (cmp++) + ".err");
  145:                         } while (file.exists());
  146:                 } else {
  147:                         file = new File("." + File.separatorChar + "datapackError.
err");
  148:                         if(file.exists()) {
  149:                                 file.delete();
  150:                         }
  151:                 }
  152:
  153:                 if(Program.saveObject(datapack, file.getPath(), true)) {
  154:                         files.add(file.getPath());
  155:                 } else {
  156:                         System.out.println("Impossible d'envoyer le datapack lors
d'une erreur");
  157:                 }
  158:
  159:                 files.add(Logger.getInstance().getFileLogPath());
  160:
  161:                 boolean result = defaultMailSender.sendMailTo(from, supportMail,
  162:                                 createSubject(subject), text, files);
  163:
  164:                 if(file.exists()) {
  165:                         file.delete();
  166:                 }
  167:
  168:                 return result;
  169:         }
  170:
  171:         static public MailSender getSingleton() {
  172:                 return defaultMailSender;
  173:         }
  174: }
```

```
    1: package Mailing;
    2:
    3: import java.util.MissingResourceException;
    4: import java.util.ResourceBundle;
    5:
    6: public class Messages {
    7:         private static final String BUNDLE_NAME = "Mailing.messages"; //$NON-NLS-1
$
    8:
    9:         private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle
   10:                         .getBundle(BUNDLE_NAME);
   11:
   12:         private Messages() {
   13:         }
   14:
   15:         public static String getString(String key) {
   16:                 try {
   17:                         return RESOURCE_BUNDLE.getString(key);
   18:                 } catch (MissingResourceException e) {
   19:                         return '!' + key + '!';
   20:                 }
   21:         }
   22: }
```

```
    1: package tools;
    2:
    3: import graphicalUserInterface.Program;
    4:
    5: import java.beans.XMLDecoder;
    6: import java.beans.XMLEncoder;
    7: import java.io.BufferedInputStream;
    8: import java.io.BufferedOutputStream;
    9: import java.io.File;
   10: import java.io.FileInputStream;
   11: import java.io.FileOutputStream;
   12:
   13:
   14: abstract public class Config {
   15:         static private String configFilePath = Program.isTriades() ? "configTriade
s" : "configCreator";
   16:
   17:         static private Config singleton = null;
   18:
   19:         abstract protected void valideData();
   20:
   21:         public boolean save() {
   22:                 try {
   23:                         XMLEncoder encoder = new XMLEncoder(new BufferedOutputStre
am(new FileOutputStream(configFilePath)));
   24:                         encoder.writeObject(this);
   25:                         encoder.close();
   26:                 } catch (Exception e) {
   27:                         e.printStackTrace();
   28:                         return false;
   29:                 }
   30:
   31:                 return true;
   32:         }
   33:
   34:         static protected Config buildInstance(Class<? extends Config> configClass)
 {
   35:                 if(singleton == null) {
   36:                         File file = new File(configFilePath);
   37:                         if(file.exists() == false) {
   38:                                 try {
   39:                                         singleton = configClass.newInstance();
   40:                                 } catch (Exception e) {
   41:                                         e.printStackTrace();
   42:                                         return null;
   43:                                 }
   44:                         } else {
   45:                                 try {
   46:                                         XMLDecoder decoder = new XMLDecoder(new Bu
fferedInputStream(new FileInputStream(configFilePath)));
   47:                                         singleton = (Config)decoder.readObject();
   48:                                 } catch (Exception e) {
   49:                                         e.printStackTrace();
   50:
   51:                                         try {
   52:                                                 singleton = configClass.newInstanc
e();
   53:                                         } catch (Exception e1) {
   54:                                                 e1.printStackTrace();
   55:                                                 return null;
   56:                                         }
   57:                                 }
   58:                         }
   59:
   60:                         singleton.valideData();
   61:                 }
   62:
   63:                 return singleton;
   64:         }
   65: }
```

```
 1: package tools;
 2:
 3: import java.io.Serializable;
 4: import java.util.Vector;
 5:
 6: public class LastObjectUsed<T> implements Serializable {
 7:         private static final long serialVersionUID = -2027869057319634120L;
 8:
 9:         private int maxSize;
10:         private Vector<T> lastObjects;
11:
12:         public LastObjectUsed() {
13:                 this(10);
14:         }
15:
16:         public LastObjectUsed(int size) {
17:                 maxSize = size;
18:                 lastObjects = new Vector<T>();
19:         }
20:
21:         public int getMaxSize() {
22:                 return maxSize;
23:         }
24:
25:         public void setMaxSize(int maxSize) {
26:                 this.maxSize = maxSize;
27:         }
28:
29:         public Vector<T> getLastObjects() {
30:                 return lastObjects;
31:         }
32:
33:         public void setLastObjects(Vector<T> lastObject) {
34:                 this.lastObjects = lastObject;
35:         }
36:
37:         public void addLastObject(T object) {
38:                 if(object == null) {
39:                         return;
40:                 }
41:
42:                 if(lastObjects.contains(object)) {
43:                         lastObjects.remove(object);
44:                 }
45:
46:                 while(lastObjects.size() >= maxSize) {
47:                         lastObjects.remove(lastObjects.lastElement());
48:                 }
49:
50:                 lastObjects.add(0, object);
51:         }
52: }
```

```
 1: package tools;
 2:
 3: import java.io.File;
 4:
 5: public class ConfigCreator extends Config {
 6:
 7:         private LastObjectUsed<String> lastDatapack;
 8:
 9:         public ConfigCreator() {
10:                 lastDatapack = new LastObjectUsed<String>();
11:         }
12:
13:         public LastObjectUsed<String> getLastDatapack() {
14:                 return lastDatapack;
15:         }
16:
17:         public void setLastDatapack(LastObjectUsed<String> lastDatapack) {
18:                 this.lastDatapack = lastDatapack;
19:         }
20:
21:         @Override
22:         protected void valideData() {
23:                 for(int i = 0 ; i < lastDatapack.getLastObjects().size();) {
24:                         boolean removePath = false;
25:                         String path = lastDatapack.getLastObjects().elementAt(i);
26:
27:                         if(path != null) {
28:                                 File file = new File(path);
29:                                 if(file.exists() == false) {
30:                                         removePath = true;
31:                                 }
32:                         } else {
33:                                 removePath = true;
34:                         }
35:
36:                         if(removePath) {
37:                                 lastDatapack.getLastObjects().remove(i);
38:                         } else {
39:                                 i++;
40:                         }
41:                 }
42:         }
43:
44:         static public ConfigCreator getInstance() {
45:                 return (ConfigCreator)Config.buildInstance(ConfigCreator.class);
46:         }
47: }
```

```
    1: package tools;
    2:
    3: import graphicalUserInterface.IconDatabase;
    4:
    5: import java.io.BufferedInputStream;
    6: import java.io.BufferedOutputStream;
    7: import java.io.ByteArrayOutputStream;
    8: import java.io.File;
    9: import java.io.FileInputStream;
   10: import java.io.FileOutputStream;
   11: import java.io.IOException;
   12: import java.io.InputStream;
   13: import java.io.ObjectInputStream;
   14: import java.io.ObjectOutputStream;
   15: import java.io.OutputStream;
   16:
   17: import javax.crypto.Cipher;
   18: import javax.crypto.SecretKey;
   19: import javax.crypto.spec.SecretKeySpec;
   20:
   21: public class Encrypting {
   22:         private final SecretKey secretKey;
   23:         static protected int keySize = 16;
   24:         static protected String algorithm = "AES";
   25:
   26:         private static Encrypting singleton = null;
   27:
   28:         private Encrypting() {
   29:                 byte[] keyBuffer = getKeyBuffer();
   30:                 int offset = 0;
   31:
   32:                 secretKey = new SecretKeySpec(keyBuffer, offset, keySize, algorith
m);
   33:         }
   34:
   35:         private byte[] getKeyBuffer() {
   36:                 try {
   37:                         int offset = 0;
   38:
   39:                         InputStream in = getClass().getResourceAsStream(IconDataba
se.defaultIconsDirectory);
   40:
   41:                         byte[] buffer = new byte[keySize];
   42:
   43:                         int tmpSize = 1024;
   44:                         byte[] tmp = new byte[tmpSize];
   45:                         int count = 0;
   46:                         while(count < offset) {
   47:                                 count += in.read(tmp, 0, Math.min(tmpSize, offset
- count));
   48:                         }
   49:
   50:                         in.read(buffer, 0, keySize);
   51:                         return buffer;
   52:                 } catch (Exception e) {
   53:                         e.printStackTrace();
   54:                         return null;
   55:                 }
   56:         }
   57:
   58:         public boolean saveEncryptedObject(Object object, String path) {
   59:                 try {
   60:                         System.out.println("Save crypted : " + path + " (" + objec
t + ")");
   61:                         Cipher cipher = Cipher.getInstance(algorithm);
   62:                         cipher.init(Cipher.ENCRYPT_MODE, secretKey);
   63:
   64:                         //convert object to byte[]
   65:                         ByteArrayOutputStream bStream = new ByteArrayOutputStream(
);
   66:                         ObjectOutputStream oStream = new ObjectOutputStream(bStrea
m);
   67:                         oStream.writeObject(object);
   68:                         byte[] clearObject = bStream.toByteArray();
   69:
   70:                         byte[] cryptedObject = cipher.doFinal(clearObject);
   71:                         File tmpFile = File.createTempFile("temp", "tmp");
   72:
   73:                         OutputStream oos = new BufferedOutputStream(new FileOutput
Stream(tmpFile));
   74:                         oos.write(cryptedObject);
   75:                         oos.close();
   76:
   77:                         return renameFile(tmpFile, new File(path));
   78:                 } catch (Exception e) {
   79:                         e.printStackTrace();
   80:                 }
   81:
   82:                 System.out.println("Impossible de sauver " + object + " dans le fi
chier " + path);
   83:                 return false;
   84:         }
   85:
   86:         public static boolean renameFile(File source, File dest){
   87:                 if(source.exists() == false) {
   88:                         return false;
   89:                 }
   90:
   91:                 if(dest.exists()) {
   92:                         if(dest.delete() == false) {
   93:                                 System.out.println("Impossible de supprimer " + de
st);
   94:                                 return false;
   95:                         }
   96:                 }
   97:
   98:                 if(source.renameTo(dest) == false) {
   99:                         System.out.println("Impossible de renomer le ficher " + so
urce);
  100:                         if(copyFile(source, dest) == false) {
  101:                                 return false;
  102:                         }
  103:                 }
  104:
  105:                 return true;
  106:         }
  107:
  108:                 private static boolean copyFile(File source, File dest) {
  109:                 try{
  110:                         // Declaration et ouverture des flux
  111:                         java.io.FileInputStream sourceFile = new java.io.FileInput
Stream(source);
  112:
  113:                         try{
  114:                                 java.io.FileOutputStream destinationFile = null;
  115:
  116:                                 try{
  117:                                         destinationFile = new FileOutputStream(des
t);
  118:
  119:                                         // Lecture par segment de 0.5Mo
  120:                                         byte buffer[] = new byte[512 * 1024];
  121:                                         int nbLecture;
  122:
```

```
  123:                                    while ((nbLecture = sourceFile.read(buffer
)) != -1){
  124:                                            destinationFile.write(buffer, 0, n
bLecture);
  125:                                    }
  126:                            } finally {
  127:                                    destinationFile.close();
  128:                            }
  129:                    } finally {
  130:                            sourceFile.close();
  131:                    }
  132:                } catch (IOException e){
  133:                        e.printStackTrace();
  134:                        System.out.println("Impossible de copier le fichier " + so
urce + " vers " + dest);
  135:                        return false; // Erreur
  136:                }
  137:
  138:                return true; // RÃ©sultat OK
  139:        }
  140:
  141:        public Object loadEncryptedObject(String path) {
  142:                System.out.println("load crypted : " + path);
  143:                try {
  144:                        Cipher cipher = Cipher.getInstance(algorithm);
  145:                        cipher.init(Cipher.DECRYPT_MODE, secretKey);
  146:
  147:                        File file = new File(path);
  148:                        InputStream in = new BufferedInputStream(new FileInputStre
am(file));
  149:                        byte[] encryptedObject = new byte[(int)file.length()];
  150:                        in.read(encryptedObject);
  151:                        in.close();
  152:
  153:                        byte[] clearObject = cipher.doFinal(encryptedObject);
  154:
  155:                        File tmpFile = File.createTempFile("dtp", "tmp");
  156:                        OutputStream out = new BufferedOutputStream(new FileOutput
Stream(tmpFile));
  157:                        out.write(clearObject);
  158:                        out.close();
  159:
  160:                        ObjectInputStream input = new ObjectInputStream(new Buffer
edInputStream(new FileInputStream(tmpFile)));
  161:                        Object result = input.readObject();
  162:                        input.close();
  163:                        tmpFile.delete();
  164:
  165:                        return result;
  166:                } catch (Exception e) {
  167:                        System.err.println(e.getLocalizedMessage()+" Caused by : "
+e.getCause());
  168:
  169:                }
  170:
  171:                return null;
  172:        }
  173:
  174:        static public Encrypting getInstance() {
  175:                if(singleton == null) {
  176:                        singleton = new Encrypting();
  177:                }
  178:
  179:                return singleton;
  180:        }
  181: }
```

```
    1: package tools;
    2:
    3: import java.io.File;
    4:
    5: import client.export.ExportImageData;
    6:
    7: public class ConfigTriades extends Config {
    8:
    9:         private LastObjectUsed<ExportImageData> lastImages;
   10:
   11:         public ConfigTriades() {
   12:                 lastImages = new LastObjectUsed<ExportImageData>();
   13:         }
   14:
   15:         public LastObjectUsed<ExportImageData> getLastImages() {
   16:                 return lastImages;
   17:         }
   18:
   19:         public void setLastImages(LastObjectUsed<ExportImageData> lastImages) {
   20:                 this.lastImages = lastImages;
   21:         }
   22:
   23:         @Override
   24:         protected void valideData() {
   25:                 for (int i = 0 ; i < lastImages.getLastObjects().size();) {
   26:                         ExportImageData data = lastImages.getLastObjects().element
At(i);
   27:                         boolean removeData = false;
   28:                         if(data != null) {
   29:                                 if(data.getImageIndex() < 0) {
   30:                                         String path = data.getImagePath();
   31:                                         if(path != null) {
   32:                                                 File file = new File(path);
   33:                                                 if(file.exists() == false) {
   34:                                                         removeData = true;
   35:                                                 }
   36:                                         } else {
   37:                                                 removeData = true;
   38:                                         }
   39:                                 }
   40:                         } else {
   41:                                 removeData = true;
   42:                         }
   43:
   44:                         if(removeData) {
   45:                                 lastImages.getLastObjects().remove(i);
   46:                         } else {
   47:                                 i++;
   48:                         }
   49:                 }
   50:         }
   51:
   52:         static public ConfigTriades getInstance() {
   53:                 return (ConfigTriades)Config.buildInstance(ConfigTriades.class);
   54:         }
   55:
   56: }
```

```
    1: package resolution;
    2:
    3: import java.util.Collection;
    4: import java.util.Iterator;
    5: import java.util.Set;
    6: import java.util.TreeSet;
    7: import java.util.Vector;
    8:
    9: import models.Brick;
   10: import models.BrickVertex;
   11: import models.Model;
   12: import models.ModelVertex;
   13: import models.Schema;
   14:
   15: import dataPack.DataPack;
   16: import dataPack.ListeActeurSelectionne;
   17:
   18: public class Solver {
   19:
   20:         protected DataPack datapack;
   21:         protected Vector<Brick> possibleBricks;
   22:         protected Vector<Model> possibleModels;
   23:         protected Set<Integer> avaibleContent;
   24:         protected ListeActeurSelectionne list;
   25:
   26:         public Solver(DataPack _datapack,ListeActeurSelectionne _list)
   27:         {
   28:                 datapack = _datapack;
   29:                 list = _list;
   30:                 possibleBricks = new Vector<Brick>();
   31:                 possibleModels = new Vector<Model>();
   32:                 avaibleContent = new TreeSet<Integer>();
   33:                 avaibleContent.addAll(list.getIdSet());
   34:
   35:         }
   36:
   37:         public Vector<Schema> generateSchema(String _name)
   38:         {
   39:                 selectPossibleBricks();
   40:                 selectPossibleModels();
   41:
   42:
   43:                 Vector<Schema> bestSchema = null;
   44:                 for (int i = 0; i< possibleModels.size();i++)
   45:                 {
   46:                         SchemaGenerator schemGen = new SchemaGenerator(possibleMod
els.elementAt(i),_name,datapack,this);
   47:                         Vector<Schema> result = schemGen.generateSchema();
   48:                         if (result != null )
   49:                         {
   50:                                 if (bestSchema != null)
   51:                                 {
   52:                                         if (result.firstElement().getFitness() ==
bestSchema.firstElement().getFitness() )
   53:                                         {
   54:                                                 bestSchema.addAll(result);
   55:                                         }
   56:                                         else if (result.firstElement().getFitness(
) > bestSchema.firstElement().getFitness())
   57:                                         {
   58:                                                 bestSchema = result;
   59:                                         }
   60:                                 }
   61:                                 else
   62:                                 {
   63:                                         bestSchema = result;
   64:                                 }
   65:                         }
   66:                 }
   67:                 return bestSchema;
   68:         }
   69:
   70:         public void selectPossibleBricks()
   71:         {
   72:                 boolean ok;
   73:                 for (Brick brick:datapack.getBrickList())
   74:                 {
   75:                         ok = true;
   76:                         Iterator<BrickVertex> iterator = brick.getVertices().itera
tor();
   77:                         while (ok && iterator.hasNext())
   78:                         {
   79:                                 BrickVertex vertex = iterator.next();
   80:
   81:                                 if (vertex.getPossiblesContents().iterator().next(
).intValue()>=0 && !isUnionNotEmpty(avaibleContent,vertex.getPossiblesContents()))
   82:                                         ok = false;
   83:                         }
   84:                         if (ok)
   85:                         {
   86:
   87:                                 possibleBricks.add(brick);
   88:                                 avaibleContent.add(brick.getType().getBrickTypeId(
));
   89:                         }
   90:
   91:
   92:
   93:                 }
   94:         }
   95:
   96:         public void selectPossibleModels()
   97:         {
   98:                 boolean ok;
   99:                 for (Model model:datapack.getModelList())
  100:                 {
  101:                         ok = true;
  102:                         Iterator<ModelVertex> iterator = model.getVertices().itera
tor();
  103:                         while (ok && iterator.hasNext())
  104:                         {
  105:                                 ModelVertex vertex = iterator.next();
  106:                                 if (!avaibleContent.contains(vertex.getContentId()
))
  107:                                 {
  108:                                         ok = false;
  109:                                 }
  110:                         }
  111:                         if (ok)
  112:                         {
  113:                                 possibleModels.add(model);
  114:                         }
  115:                 }
  116:         }
  117:
  118:         public boolean isUnionNotEmpty(Collection<?> a, Collection<?> b)
  119:         {
  120:                 for (Object objA:a)
  121:                 {
  122:                         for (Object objB:b)
  123:                         {
  124:                                 if (objA.equals(objB))
  125:                                         return true;
  126:
```

```
127:                         }
128:                 }
129:
130:
131:                 return false;
132:         }
133:
134:         public ListeActeurSelectionne getListeActeurSelectionne() {
135:                 return list;
136:         }
137:
138: }
```

```
 1: package resolution;
 2:
 3: import java.util.Map;
 4: import java.util.TreeMap;
 5: import java.util.Vector;
 6:
 7: public class VirtualRelationInformation {
 8:
 9:         protected final Integer brickType;
10:         protected Vector<Arete> infos;
11:
12:
13:
14:
15:         public VirtualRelationInformation(Integer typeId)
16:         {
17:                 brickType = typeId;
18:                 infos = new Vector<Arete>();
19:         }
20:
21:         public void addInformation(Integer source, Integer destination)
22:         {
23:                 infos.add(new Arete(source,destination));
24:         }
25:
26:         public Integer getSource(Integer destination) {
27:                 for (Arete arete:infos)
28:                 {
29:                         if (arete.dest.equals(destination))
30:                                 return arete.source;
31:                 }
32:                 return null;
33:         }
34:
35:         public Integer getDestination(Integer source){
36:
37:                 for (Arete arete:infos)
38:                 {
39:                         if (arete.source.equals(source))
40:                                 return arete.dest;
41:                 }
42:                 return null;
43:
44:         }
45:
46: }
```

```
    1: package resolution;
    2:
    3: public class Arete implements Cloneable, Comparable<Arete>{
    4:         public Integer source;
    5:         public Integer dest;
    6:
    7:         public Arete(Integer src, Integer dst)
    8:         {
    9:                 source = src;
   10:                 dest = dst;
   11:         }
   12:
   13:         //@Override
   14:         public boolean equals(Object obj)
   15:         {
   16:                 if (obj instanceof Arete)
   17:                         return source.equals(((Arete)obj).source) && dest.equals((
(Arete)obj).dest);
   18:
   19:                 System.err.println("Impossible de comparer " + obj.getClass() + "a
 une Arete");
   20:                 return false;
   21:         }
   22:
   23:         @Override
   24:         public int compareTo(Arete o) {
   25:
   26:                 if (source.equals(source.equals(o.source)))
   27:                         return (dest.compareTo(o.dest));
   28:                 else
   29:                         return (source.compareTo(o.source));
   30:
   31:         }
   32: }
```

```
    1: package resolution;
    2:
    3: import java.util.Vector;
    4:
    5: import main.Statut;
    6: import models.Model;
    7:
    8: import dataPack.DataPack;
    9:
   10: public class VirtualRelationSolver {
   11:
   12:         protected int nbTypeBrick;
   13:         protected Vector<VirtualRelationInformation> entrante;
   14:         protected Vector<VirtualRelationInformation> sortante;
   15:         protected DataPack datapack;
   16:         protected Model model;
   17:
   18:
   19:         public VirtualRelationSolver(int nb, DataPack _datapack, Model _model)
   20:         {
   21:                 datapack = _datapack;
   22:                 nbTypeBrick = nb;
   23:                 entrante = new Vector<VirtualRelationInformation>(nb);
   24:                 sortante = new Vector<VirtualRelationInformation>(nb);
   25:                 model = _model;
   26:         }
   27:
   28:
   29:         public VirtualRelationSolver(VirtualRelationSolver source) {
   30:                 datapack = source.datapack;
   31:                 model = source.model;
   32:                 nbTypeBrick = source.nbTypeBrick;
   33:                 entrante = new Vector<VirtualRelationInformation>(source.entrante)
;
   34:                 sortante = new Vector<VirtualRelationInformation>(source.sortante)
;
   35:         }
   36:
   37:
   38:         public void addVirtualEdge(Integer typeBrick, Integer orig, Integer dest,b
oolean entrant)
   39:         {
   40:                 if (entrant)
   41:                 {
   42:
   43:                         for (VirtualRelationInformation infoEdge: entrante)
   44:                         {
   45:                                 if (infoEdge.brickType.equals(typeBrick))
   46:                                 {
   47:                                         infoEdge.addInformation(orig,dest);
   48:                                         return;
   49:                                 }
   50:                         }
   51:                 }
   52:                 else
   53:                 {
   54:                         for (VirtualRelationInformation infoEdge: sortante)
   55:                         {
   56:                                 if (infoEdge.brickType.equals(typeBrick))
   57:                                 {
   58:                                         infoEdge.addInformation(orig,dest);
   59:                                         return;
   60:                                 }
   61:                         }
   62:                 }
   63:
   64:                 VirtualRelationInformation temp = new VirtualRelationInformation(t
ypeBrick);
   65:                 temp.addInformation(orig, dest);
   66:                 if (entrant)
   67:                         entrante.add(temp);
   68:                 else
   69:                         sortante.add(temp);
   70:         }
   71:
   72:
   73:         public Integer solveVirtualSource(Integer brickType, Integer destination)
{
   74:                 //Ici le brickType est contenu dans source
   75:                 //destination contient soit un acteur (positif) soit une autre bri
que.
   76:                 if (destination.intValue()>= 0)
   77:                 {
   78:                         if (Statut.values()[datapack.getActors().getActeur(destina
tion).getIdStatut().intValue()].rang >= 0)
   79:                         {
   80:                                 //On cherche une relation sortante allant vers un
manager
   81:                                 return getSource(brickType,new Integer(-1));
   82:                         }
   83:                         else
   84:                         {
   85:                                 //On cherche une relation entrante allant vers un
partenaire
   86:                                 return getSource(brickType,new Integer(-2));
   87:                         }
   88:                 }
   89:                 else
   90:                         return getSource(brickType,destination);
   91:
   92:
   93:         }
   94:
   95:         public Integer solveVirtualDestination(Integer brickType, Integer source)
{
   96:                 //Ici le brickType est contenu dans source
   97:                 //destination contient soit un acteur (positif) soit une autre bri
que.
   98:                 if (source.intValue()>= 0)
   99:                 {
  100:                         if (Statut.values()[datapack.getActors().getActeur(source)
.getIdStatut().intValue()].rang >= 0)
  101:                         {
  102:                                 //On cherche une relation sortante allant vers un
manager
  103:                                 return getDestination(brickType,new Integer(-1));
  104:                         }
  105:                         else
  106:                         {
  107:                                 //On cherche une relation entrante allant vers un
partenaire
  108:                                 return getDestination(brickType,new Integer(-2));
  109:                         }
  110:                 }
  111:                 else
  112:                         return getDestination(brickType,source);
  113:
  114:
  115:         }
  116:
  117:         public Integer getSource(Integer brickType, Integer destination)
  118:         {
  119:                 for (VirtualRelationInformation info: sortante )
  120:                 {
```

```
121:                         if (info.brickType.equals(brickType))
122:                         {
123:                                 return info.getSource(destination);
124:                         }
125:                 }
126:
127:                 return null;
128:         }
129:
130:
131:         public Integer getDestination(Integer brickType, Integer source) {
132:                 for (VirtualRelationInformation info: entrante )
133:                 {
134:                         if (info.brickType.equals(brickType))
135:                         {
136:                                 return info.getDestination(source);
137:                         }
138:                 }
139:                 return null;
140:         }
141: }
```

```
   1: package resolution;
   2:
   3: import java.awt.geom.Point2D;
   4: import java.util.BitSet;
   5: import java.util.Hashtable;
   6: import java.util.Set;
   7: import java.util.TreeSet;
   8: import java.util.Vector;
   9:
  10: import models.Brick;
  11: import models.BrickVertex;
  12: import models.Model;
  13: import models.BrickEdge;
  14: import models.ModelVertex;
  15: import models.Schema;
  16: import models.SchemaVertex;
  17: import dataPack.DataPack;
  18: import dataPack.ActeurSelectionne;
  19: import graphicalUserInterface.DialogHandlerFrame;
  20: import main.ActionTimeListe;
  21:
  22: public class SchemaGenerator implements Cloneable {
  23:
  24:
  25:         protected Set<Integer> unfoldedContent;
  26:         protected Set<Integer> avaibleActors;
  27:         protected Brick currentBrick;
  28:         protected Point2D currentBrickLocation;
  29:         protected int currentBrickVertexId;
  30:         protected Hashtable<Integer,SchemaVertex> brickToVertexAssociation;
  31:         protected VirtualRelationSolver virtualRelationSolver;
  32:         protected Schema builtSchema;
  33:         protected final Model model;
  34:         protected final Solver solver;
  35:
  36:         public SchemaGenerator(Model workingModel, String schemaName, DataPack dat
apack, Solver _solver)
  37:         {
  38:                 model = workingModel;
  39:                 solver = _solver;
  40:                 //hypothesisList = new Stack<Hypothesis>();
  41:                 unfoldedContent = new TreeSet<Integer>();
  42:                 avaibleActors = new TreeSet<Integer>(solver.avaibleContent);
  43:
  44:                 virtualRelationSolver = new VirtualRelationSolver(5,datapack,worki
ngModel);
  45:                 builtSchema = new Schema(datapack,schemaName);
  46:                 currentBrickLocation = null;
  47:                 brickToVertexAssociation = null;
  48:                 currentBrickVertexId = 0;
  49:                 currentBrick = null;
  50:                 for (ModelVertex vertex:model.getVertices())
  51:                 {
  52:                         unfoldedContent.add(vertex.getContentId());
  53:                 }
  54:
  55:
  56:         }
  57:
  58:         public SchemaGenerator(SchemaGenerator source)
  59:         {
  60:                 model = source.model;
  61:                 builtSchema = new Schema(source.builtSchema);
  62:                 currentBrick = source.currentBrick;
  63:                 currentBrickVertexId = source.currentBrickVertexId;
  64:                 currentBrickLocation = source.currentBrickLocation;
  65:                 if (source.brickToVertexAssociation != null)
  66:                         brickToVertexAssociation = new Hashtable<Integer,SchemaVer
tex>(source.brickToVertexAssociation);
  67:                 virtualRelationSolver = new VirtualRelationSolver(source.virtualRe
lationSolver);
  68:                 unfoldedContent = new TreeSet<Integer>(source.unfoldedContent);
  69:                 solver = source.solver;
  70:                 avaibleActors = new TreeSet<Integer>(source.avaibleActors);
  71:
  72:         }
  73:
  74:         protected SchemaVertex addActor(Integer idActor)
  75:         {
  76:                 if (idActor.intValue()>=0)
  77:                 {
  78:                         if (avaibleActors.contains(idActor))
  79:                         {
  80:                                 for (ActeurSelectionne selectedActor:solver.list.g
etActorsSelection())
  81:                                 {
  82:                                         if (selectedActor.getIdActeur().equals(idA
ctor))
  83:                                         {
  84:                                                 if (builtSchema.getVertexByActor(s
electedActor)!= null)
  85:                                                 {
  86:                                                         return null;
  87:                                                 }
  88:                                                 else
  89:                                                 {
  90:                                                         avaibleActors.remove(idAct
or);
  91:                                                         return builtSchema.addSche
maVertex(idActor, null);
  92:
  93:                                                 }
  94:                                         }
  95:                                 }
  96:                         }
  97:                 }
  98:                 else
  99:                 {
 100:                         return builtSchema.addSchemaVertex(idActor, null);
 101:                 }
 102:                 System.out.println("Echec lors de l'ajout d'un acteur d'id : "+idA
ctor);
 103:                 return null;
 104:         }
 105:
 106:         public Vector<Schema> generateSchema()
 107:         {
 108:
 109:                 while (unfoldedContent.size() > 0 || currentBrick != null)
 110:                 {
 111:                         if (currentBrick != null)
 112:                         {
 113:                                 if (currentBrickVertexId < currentBrick.getVertice
s().size())
 114:                                 {
 115:                                         BrickVertex vertex = currentBrick.getVerti
ces().elementAt(currentBrickVertexId);
 116:
 117:                                         if (vertex.getPossiblesContents().size() =
= 1)
 118:                                         {
 119:                                                 SchemaVertex newVertex = addActor(
vertex.getPossiblesContents().iterator().next());
 120:                                                 if (newVertex == null)
```

```
  121:                                                    {
  122:                                                            return null;
  123:                                                    }
  124:                                                    else
  125:                                                    {
  126:                                                            newVertex.setLocation(new
Point2D.Double(currentBrickLocation.getX()+currentBrick.getVertices().elementAt(currentBr
ickVertexId).getLocation().getX()/1.4,currentBrickLocation.getY()+currentBrick.getVertice
s().elementAt(currentBrickVertexId).getLocation().getY()/1.4));
  127:                                                            brickToVertexAssociation.p
ut(vertex.getVertexId(), newVertex);
  128:                                                            currentBrickVertexId++;
  129:                                                    }
  130:                                            }
  131:                                            else
  132:                                            {
  133:                                                    currentBrickVertexId ++;
  134:                                                    Set<Integer> actorsSet = vertex.ge
tPossiblesContents();
  135:                                                    actorsSet.retainAll(avaibleActors)
;
  136:                                                    Vector<Schema> bestSchema = null;
  137:                                                    for (Integer id : actorsSet)
  138:                                                    {
  139:                                                            SchemaGenerator schemGen =
 new SchemaGenerator(this);
  140:                                                            SchemaVertex newVertex = s
chemGen.addActor(id);
  141:                                                            if (newVertex != null)
  142:                                                            {
  143:                                                                    newVertex.setLocat
ion(new Point2D.Double(currentBrickLocation.getX()+currentBrick.getVertices().elementAt(c
urrentBrickVertexId).getLocation().getX(),currentBrickLocation.getY()+currentBrick.getVer
tices().elementAt(currentBrickVertexId).getLocation().getY()));
  144:                                                                    schemGen.brickToVe
rtexAssociation.put(vertex.getVertexId(), newVertex);
  145:                                                                    Vector<Schema> res
ult = schemGen.generateSchema();
  146:                                                                    if (result != null
 )
  147:                                                                    {
  148:                                                                            if (bestSc
hema != null)
  149:                                                                            {
  150:                                                                                    if
 (result.firstElement().getFitness() == bestSchema.firstElement().getFitness() )
  151:                                                                                    {

  152:                                                                                            
bestSchema.addAll(result);
  153:                                                                                    }

  154:                                                                                    el
se if (result.firstElement().getFitness() > bestSchema.firstElement().getFitness())
  155:                                                                                    {
  156:                                                                                            
bestSchema = result;
  157:                                                                                    }
  158:                                                                            }
  159:                                                                            else
  160:                                                                            {
  161:                                                                                    be
stSchema = result;
  162:                                                                            }
  163:                                                                    }
  164:                                                            }
  165:                                                    }
  166:                                                    return bestSchema;
  167:
  168:                                            }
  169:                                    }
  170:                                    else
  171:                                    {
  172:                                            //Traitement de fin de brique :D
  173:                                            //Pour chaque arÃªtes de la liste, soit on
 l'ajoute aux arÃªtes virtuelles (rÃ©fÃ©renÃ§ant l'id virtuelle et l'id du SchemaVertex)
  174:                                            SchemaVertex activityVertex = builtSchema.
addSchemaVertex(currentBrick.getActivityId(), null);
  175:                                            activityVertex.setLocation(currentBrickLoc
ation);
  176:
  177:                                            for (BrickEdge edge:currentBrick.getEdges(
))
  178:                                            {
  179:
  180:                                                    if (edge.getSource().intValue() <
0 && edge.getSource().intValue() > -100)
  181:                                                    {
  182:                                                            if (edge.getDestination().
intValue() >= 0)
  183:                                                                    virtualRelationSol
ver.addVirtualEdge(currentBrick.getType().getBrickTypeId(), edge.getSource(), brickToVert
exAssociation.get(edge.getDestination()).getVertexId(),true);
  184:                                                            else
  185:                                                                    virtualRelationSol
ver.addVirtualEdge(currentBrick.getType().getBrickTypeId(), edge.getSource(), edge.getDes
tination(),true);
  186:
  187:                                                    }
  188:                                                    else if (edge.getDestination().int
Value() < 0 && edge.getDestination().intValue() > -100)
  189:                                                    {
  190:                                                            if (edge.getSource().intVa
lue() >= 0)
  191:                                                                    virtualRelationSol
ver.addVirtualEdge(currentBrick.getType().getBrickTypeId(), brickToVertexAssociation.get(
edge.getSource()).getVertexId(),edge.getDestination(),false);
  192:                                                            else
  193:                                                                    virtualRelationSol
ver.addVirtualEdge(currentBrick.getType().getBrickTypeId(), edge.getSource(),edge.getDest
ination(),false);
  194:
  195:                                                    }
  196:                                                    else
  197:                                                    {
  198:                                                            Integer source;
  199:                                                            Integer destination;
  200:                                                            if (edge.getSource().intVa
lue() <= -50)
  201:                                                                    source = activityV
ertex.getVertexId();
  202:                                                            else
  203:                                                                    source = brickToVe
rtexAssociation.get(edge.getSource()).getVertexId();
  204:
  205:                                                            if (edge.getDestination().
intValue() <= -50)
  206:                                                                    destination = acti
vityVertex.getVertexId();
  207:                                                            else
  208:                                                                    destination = bric
kToVertexAssociation.get(edge.getDestination()).getVertexId();
  209:
  210:                                                            builtSchema.addModelEdge(s
```

```
ource,destination , edge.getCompleteRelation(),null);
  211:                                                  }
  212:                                          }
  213:                                          brickToVertexAssociation = null;
  214:                                          currentBrick = null;
  215:                                          currentBrickLocation = null;
  216:                                          currentBrickVertexId = 0;
  217:
  218:                                  }
  219:                          }
  220:                          else
  221:                          {
  222:                                  Integer nextContent = unfoldedContent.iterator().n
ext();
  223:                                  unfoldedContent.remove(nextContent);
  224:                                  if (nextContent.intValue() < 0)
  225:                                  {
  226:                                          //On rÃ©cupÃ¨re un brickType
  227:                                          Vector<Brick> possiblesBricks = new Vector
<Brick>();
  228:                                          for (Brick brick:solver.possibleBricks)
  229:                                          {
  230:                                                  if (brick.getType().getBrickTypeId
().equals(nextContent))
  231:                                                  {
  232:                                                          possiblesBricks.add(brick)
;
  233:                                                  }
  234:                                          }
  235:                                          if (possiblesBricks.isEmpty())
  236:                                          {
  237:                                                  return null;
  238:                                          }
  239:                                          else if (possiblesBricks.size() == 1)
  240:                                          {
  241:                                                  currentBrick = possiblesBricks.fir
stElement();
  242:                                                  currentBrickLocation = model.getVe
rtexByContent(nextContent).getLocation();
  243:                                                  brickToVertexAssociation = new Has
htable<Integer,SchemaVertex>(currentBrick.getVertices().size());
  244:                                          }
  245:                                          else
  246:                                          {
  247:                                                  Vector<Schema> bestSchema = null;
  248:                                                  for (Brick brick:possiblesBricks)
  249:                                                  {

  250:                                                          currentBrickLocation = mod
el.getVertexByContent(nextContent).getLocation();
  251:                                                          brickToVertexAssociation =
 new Hashtable<Integer,SchemaVertex>();
  252:                                                          SchemaGenerator schemGen =
 new SchemaGenerator(this);
  253:                                                          schemGen.currentBrick = br
ick;
  254:                                                          Vector<Schema> result = sc
hemGen.generateSchema();
  255:                                                          if (result != null )
  256:                                                          {
  257:                                                                  if (bestSchema !=
null)
  258:                                                                  {
  259:                                                                          if (result
.firstElement().getFitness() == bestSchema.firstElement().getFitness() )
  260:                                                                          {
  261:                                                                                  be
stSchema.addAll(result);
  262:                                                                          }
  263:                                                                          else if (r
esult.firstElement().getFitness() > bestSchema.firstElement().getFitness())
  264:                                                                          {
  265:                                                                                  be
stSchema = result;
  266:                                                                          }
  267:                                                                  }
  268:                                                                  else
  269:                                                                  {
  270:                                                                          bestSchema
 = result;
  271:                                                                  }
  272:
  273:                                                          }
  274:
  275:                                                  }
  276:                                                  return bestSchema;
  277:                                          }
  278:
  279:                                  }
  280:                                  else
  281:                                  {
  282:                                          SchemaVertex result = addActor(nextContent
);
  283:                                          if (result != null)
  284:                                                  result.setLocation(new Point2D.Dou
ble(model.getVertexByContent(nextContent).getLocation().getX(),model.getVertexByContent(n
extContent).getLocation().getY()));
  285:                                          else
  286:                                                  return null;
  287:                                  }
  288:
  289:                          }
  290:                  }
  291:                  //RÃ©solution des relation du modÃ¨le.
  292:                  for (BrickEdge edge:model.getEdges())
  293:                  {
  294:
  295:                          ModelVertex src = model.getModelVertex(edge.getSource());
  296:                          ModelVertex dst = model.getModelVertex(edge.getDestination
());
  297:                          Integer source;
  298:                          Integer destination;
  299:                          if (src.getContentId().intValue()< 0)
  300:                                  source = virtualRelationSolver.solveVirtualSource(
src.getContentId(),dst.getContentId());
  301:                          else
  302:                          {
  303:                                  ActeurSelectionne actor = solver.list.getActorSele
ctionById(src.getContentId());
  304:                                  source = builtSchema.getVertexByActor(actor).getVe
rtexId();
  305:                          }
  306:                          if (dst.getContentId().intValue()< 0)
  307:                                  destination = virtualRelationSolver.solveVirtualDe
stination(dst.getContentId(),src.getContentId());
  308:                          else
  309:                          {
  310:                                  ActeurSelectionne actor = solver.list.getActorSele
ctionById(dst.getContentId());
  311:                                  destination = builtSchema.getVertexByActor(actor).
getVertexId();
  312:                          }
  313:
  314:                          if (destination != null && source != null )
```

```
  315:                         {                                                          375:
  316:                                 builtSchema.addModelEdge(source, destination,edge.     376: }
getCompleteRelation(), null);
  317:                         }
  318:                 }
  319:                 evaluateFitness();
  320:                 Vector<Schema> result = new Vector<Schema>();
  321:                 result.add(builtSchema);
  322:                 return result;
  323:         }
  324:
  325:
  326:
  327:         public SchemaGenerator clone() throws CloneNotSupportedException
  328:         {
  329:                 return (SchemaGenerator)super.clone();
  330:
  331:         }
  332:
  333:         private BitSet getActiveTime(SchemaVertex vertex)
  334:         {
  335:                 BitSet result = new BitSet(model.getDataPack().getActionTimeList()
.size());
  336:                 result.clear();
  337:                 for (BrickEdge edge:builtSchema.getEdges())
  338:                 {
  339:                         if (edge.getSource().equals(vertex.getVertexId()) || edge.
getDestination().equals(vertex.getVertexId()))
  340:                         {
  341:                                 result.or(edge.getCompleteRelation().getActiveTime
());
  342:                         }
  343:                 }
  344:
  345:                 return result;
  346:         }
  347:
  348:         public void evaluateFitness()
  349:         {
  350:
  351:
  352:                 int fitness = 0;
  353:                 for (SchemaVertex vertex : builtSchema.getVertices())
  354:                 {
  355:                         if (vertex.getContent().intValue()>= 0)
  356:                         {
  357:                                 BitSet activeTime = getActiveTime(vertex);
  358:                                 BitSet actorActiveTime = solver.list.getActorSelec
tionById(vertex.getContent()).getActiveTime();
  359:                                 for (int i = 0; i< model.getDataPack().getActionTi
meList().size();i++)
  360:                                 {
  361:                                         if (activeTime.get(i) == actorActiveTime.g
et(i))
  362:                                                 fitness ++;
  363:                                 }
  364:                         }
  365:                 }
  366:
  367:                 for (ActeurSelectionne actor: solver.list.getActorsSelection())
  368:                 {
  369:                         if (builtSchema.getVertexByActor(actor) == null)
  370:                                 fitness -= 5;
  371:                 }
  372:                 }
  373:
  374:
```

```
    1: package relations;
    2:
    3: import java.io.Serializable;
    4:
    5: import client.stringTranslator.StringTranslator;
    6:
    7: /**
    8:  * DÃ©crit l'ensemble des acteurs pouvant intervenir dans une relation.
    9:  * @author babcool
   10:  *
   11:  */
   12: public class RelationDescription implements Serializable {
   13:
   14:         /**
   15:          *
   16:          */
   17:         private static final long serialVersionUID = 8819964772447156207L;
   18:         protected boolean content[][];
   19:         protected String name;
   20:         protected RelationPossibility possibility;
   21:         protected boolean onlyRealRelation;
   22:
   23:         protected RelationDescription(String name) {
   24:                 this.name = name;
   25:                 // Taille du tableau : 18 statuts diffÃ©rents + 4 moyens + 1 objet
 = 23
   26:
   27:                 content = new boolean[23][23];
   28:                 possibility = new RelationPossibility();
   29:         }
   30:
   31:
   32:
   33:
   34:         public boolean allow(Integer a, Integer b, boolean realRelation) {
   35:                 return (realRelation || !this.onlyRealRelation)
   36:                                 && content[a.intValue()][b.intValue()];
   37:
   38:         }
   39:
   40:         public void set(int a, int b, boolean value) {
   41:                 content[a][b] = value;
   42: //              System.out.println("Nouvelle valeur pour " + a + " et " + b + " ve
rs " + value);
   43:         }
   44:
   45:         public void setRect(int minA, int maxA, int minB, int maxB, boolean value)
 {
   46:                 for(int a = minA ; a <= maxA ; a++) {
   47:                         for(int b = minB ; b <= maxB ; b++) {
   48:                                 set(a, b, value);
   49:                         }
   50:                 }
   51:         }
   52:
   53:
   54:         public void setTriangleInf(int a, int minB, int maxB, boolean value) {
   55:                 for(int i = 0 ; i < maxB - minB ; i++) {
   56:                         for(int j = 0 ; j < maxB - minB - i ; j++) {
   57:                                 set(a+i, minB+j, value);
   58:                         }
   59:                 }
   60:         }
   61:
   62:         public void setTriangleSup(int a, int minB, int maxB, boolean value) {
   63:                 for(int i = 0 ; i < maxB - minB ; i++) {
   64:                         for(int j = 0 ; j < i ; j++) {
   65:                                 set(a+j, minB+i, value);
   66:                         }
   67:                 }
   68:         }
   69:
   70:         public RelationPossibility getPossibility() {
   71:                 return possibility;
   72:         }
   73:
   74:         public void setRealRelation(boolean realRelation) {
   75:                 onlyRealRelation = realRelation;
   76:         }
   77:
   78:         public boolean isRealRelation() {
   79:                 return onlyRealRelation;
   80:         }
   81:
   82:         public String getNoTranslatedString() {
   83:                 return name;
   84:         }
   85:
   86:         @Override
   87:         public String toString() {
   88:                 return StringTranslator.getTranslatedString(this, StringTranslator
.StringType.groupType);
   89:         }
   90:
   91:         public void setName(String text) {
   92:                 if (name != "") //$NON-NLS-1$
   93:                         name = text;
   94:         }
   95:
   96: }
```

```java
package relations;

import graphicalUserInterface.DialogHandlerFrame;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

import translation.Messages;

public class RelationPanel extends JPanel {

	/**
	 *
	 */
	private static final long serialVersionUID = 9071668279819902914L;

	protected RelationDescription relation;

	protected JList objectives;
	protected DefaultListModel objectivesModel;
	protected JList meanings;
	protected DefaultListModel meaningsModel;

	protected ArrayList<Mean> currentMeanings;
	protected JButton addObj;
	protected JButton addMean;
	protected JButton deleteObj;
	protected JButton deleteMean;

	public RelationPanel() {
		build();
	}

	protected void build() {
		setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
		JPanel objPanel = new JPanel(new BorderLayout());
		objectivesModel = new DefaultListModel();
		objectives = new JList(objectivesModel);
		objectives.setEnabled(false);

		objectives.addListSelectionListener(new ListSelectionListener() {

			@Override
			public void valueChanged(ListSelectionEvent arg0) {
				refreshMeanings();

			}
		});

		JPanel addObjPanel = new JPanel();
		addObjPanel.setLayout(new BorderLayout());
		final JTextField objName = new JTextField();
		addObjPanel.add(objName, BorderLayout.CENTER);
		addObj = new JButton(Messages.getString("RelationPanel.0")); //$NO
N-NLS-1$
		addObj.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				if (!objName.getText().trim().equals(""))
				{
					addObjective(objName.getText());
					objName.setText(""); //$NON-NLS-1$
					objName.requestFocus();
				}
			}
		});


		deleteObj = new JButton(Messages.getString("RelationPanel.2")); //
$NON-NLS-1$
		deleteObj.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent arg0) {
				if (objectives.getSelectedIndex() == -1) {
					DialogHandlerFrame
					.showErrorDialog(Messages.getString("Relat
ionPanel.4")); //$NON-NLS-1$
					return;
				}

				if (JOptionPane.YES_OPTION == DialogHandlerFrame
						.showYesNoDialog(Messages.getStrin
g("RelationPanel.3"))) //$NON-NLS-1$
				{
					String obj = objectives.getSelectedValue()
.toString();
					relation.getPossibility().possibility.remo
ve(objectives
							.getSelectedValue());
					objectivesModel.removeElement(obj);
					objectives.setSelectedIndex(0);
					refreshMeanings();
				}

			}
		});

		addObjPanel.add(deleteObj, BorderLayout.SOUTH);
		addObjPanel.add(addObj, BorderLayout.EAST);
		addObjPanel.setBorder(BorderFactory
				.createTitledBorder(Messages.getString("RelationPa
nel.5"))); //$NON-NLS-1$
		objPanel.add(addObjPanel, BorderLayout.SOUTH);
		objPanel.setBorder(BorderFactory.createTitledBorder(Messages.getSt
ring("RelationPanel.6"))); //$NON-NLS-1$
		objPanel.add(new JScrollPane(objectives), BorderLayout.CENTER);

		JPanel meanPanel = new JPanel(new BorderLayout());
		meaningsModel = new DefaultListModel();
		meanings = new JList(meaningsModel);
		meanings.setEnabled(false);

		JPanel addMeanPanel = new JPanel();
		addMeanPanel.setLayout(new BorderLayout());
		final JTextField meanName = new JTextField();
		addMeanPanel.add(meanName, BorderLayout.CENTER);
		addMean = new JButton(Messages.getString("RelationPanel.7")); //$N
```

```
ON-NLS-1$
  127:                 addMean.addActionListener(new ActionListener() {
  128:
  129:                         @Override
  130:                         public void actionPerformed(ActionEvent arg0) {
  131:                                 if (!meanName.getText().trim().equals(""))
  132:                                 {
  133:                                         Mean newMean = new Mean(meanName.getText()
);
  134:                                         currentMeanings.add(newMean);
  135:                                         meaningsModel.add(meaningsModel.size(),new
Mean);
  136:                                         meanName.setText(""); //$NON-NLS-1$
  137:                                         meanName.requestFocus();
  138:                                         refreshMeanings();
  139:                                 }
  140:                         }
  141:                 });
  142:                 addMean.setEnabled(false);
  143:
  144:                 deleteMean = new JButton(Messages.getString("RelationPanel.9")); /
/$NON-NLS-1$
  145:                 deleteMean.addActionListener(new ActionListener() {
  146:
  147:                         @Override
  148:                         public void actionPerformed(ActionEvent arg0) {
  149:                                 if (meanings.getSelectedValue() != null)
  150:                                 {
  151:                                         String mean = meanings.getSelectedValue().
toString();
  152:                                         currentMeanings.remove(mean);
  153:                                         meaningsModel.removeElement(mean);
  154:                                         meanings.setSelectedIndex(0);
  155:                                         refreshMeanings();
  156:                                 }
  157:                         }
  158:                 });
  159:                 addMeanPanel.add(deleteMean, BorderLayout.SOUTH);
  160:                 addMeanPanel.add(addMean, BorderLayout.EAST);
  161:                 addMeanPanel.setBorder(BorderFactory
  162:                                 .createTitledBorder(Messages.getString("RelationPa
nel.10"))); //$NON-NLS-1$
  163:                 meanPanel.add(addMeanPanel, BorderLayout.SOUTH);
  164:                 meanPanel.setBorder(BorderFactory.createTitledBorder(Messages.getS
tring("RelationPanel.11"))); //$NON-NLS-1$
  165:                 meanPanel.add(new JScrollPane(meanings), BorderLayout.CENTER);
  166:                 add(Box.createHorizontalGlue());
  167:                 add(objPanel);
  168:                 add(meanPanel);
  169:                 add(Box.createHorizontalGlue());
  170:
  171:                 loadRelation(null);
  172:         }
  173:
  174:         public void loadRelation(RelationDescription newRelation) {
  175:                 relation = newRelation;
  176:                 objectivesModel.removeAllElements();
  177:
  178:                 if(newRelation != null) {
  179:                         for (Goal s : relation.getPossibility().possibility.keySet
()) {
  180:                                 objectivesModel.addElement(s);
  181:                         }
  182:                         if (objectivesModel.size() > 0) {
  183:                                 objectives.setSelectedIndex(0);
  184:                                 refreshMeanings();
  185:                         } else {
  186:                                 addMean.setEnabled(false);
  187:                                 meanings.setEnabled(false);
  188:                                 deleteMean.setEnabled(false);
  189:                         }
  190:                         addObj.setEnabled(true);
  191:                         objectives.setEnabled(true);
  192:                         deleteObj.setEnabled(true);
  193:                 } else {
  194:                         meaningsModel.removeAllElements();
  195:
  196:                         objectives.setEnabled(false);
  197:                         addObj.setEnabled(false);
  198:                         addMean.setEnabled(false);
  199:                         meanings.setEnabled(false);
  200:                         deleteMean.setEnabled(false);
  201:                         deleteObj.setEnabled(false);
  202:                 }
  203:                 validate();
  204:                 repaint();
  205:         }
  206:
  207:         protected void addObjective(String newObjective) {
  208:                 if (relation == null) {
  209:                         System.err.println("Relation null lors d'un ajout"); //$NO
N-NLS-1$
  210:                         return;
  211:                 }
  212: Goal newGoal = new Goal(newObjective);
  213:                 if (relation.getPossibility().possibility.containsKey(newGoal)) {
  214:                         DialogHandlerFrame
  215:                         .showErrorDialog(Messages.getString("RelationPanel.13"));
//$NON-NLS-1$
  216:
  217:                         return;
  218:                 }
  219:
  220:                 if (newObjective != "") { //$NON-NLS-1$
  221:                         currentMeanings = new ArrayList<Mean>();
  222:                         relation.getPossibility().possibility.put(newGoal,
  223:                                         currentMeanings);
  224:                         objectivesModel.addElement(newGoal);
  225:                         objectives.setSelectedValue(newGoal, true);
  226:                         refreshMeanings();
  227:
  228:                 } else {
  229:                         DialogHandlerFrame
  230:                         .showErrorDialog(Messages.getString("RelationPanel.15"));
//$NON-NLS-1$
  231:                 }
  232:         }
  233:
  234:         protected void refreshMeanings() {
  235:                 if(relation != null) {
  236:                         if (objectives.getSelectedValue() != null)
  237:                                 currentMeanings = relation.getPossibility().getMap
().get(objectives.getSelectedValue());
  238:                         else
  239:                                 currentMeanings = null;
  240:                         meaningsModel.removeAllElements();
  241:                         if (currentMeanings != null)
  242:                                 for (Mean s : currentMeanings) {
  243:                                         meaningsModel.addElement(s);
  244:                                 }
  245:                         addMean.setEnabled(true);
  246:                         deleteMean.setEnabled(true);
  247:                         meanings.setEnabled(true);
  248:                 }
```

```
249:                 validate();
250:         }
251: }
```

```
    1: package relations;
    2:
    3: import java.io.Serializable;
    4:
    5: import client.stringTranslator.StringTranslator;
    6:
    7: public class Goal implements Comparable<Goal>, Serializable{
    8:
    9:         /**
   10:          *
   11:          */
   12:         private static final long serialVersionUID = 2333173474899739781L;
   13:         public String name;
   14:
   15:         public Goal(String name)
   16:         {
   17:                 this.name = name;
   18:         }
   19:
   20:         public void setName(String newName)
   21:         {
   22:                 name = newName;
   23:         }
   24:
   25:         public boolean equals(Goal other)
   26:         {
   27:                 System.out.println("Goat equals (" + other.name + "-" + name + ")
: " + other.name.equals(name));
   28:                 return other.name.equals(name);
   29:         }
   30:
   31:         public String getNoTranslatedString() {
   32:                 return name;
   33:         }
   34:
   35:         @Override
   36:         public boolean equals(Object other)
   37:         {
   38:                 if (other instanceof Goal)
   39:                         return equals((Goal)other);
   40:                 return false;
   41:         }
   42:
   43:         @Override
   44:         public String toString()
   45:         {
   46:                 return StringTranslator.getTranslatedString(this, StringTranslator
.StringType.objectifType);
   47:         }
   48:
   49:         @Override
   50:         public int compareTo(Goal o) {
   51:                 return name.compareTo(o.name);
   52:         }
   53: }
```

```
    1: package relations;
    2:
    3: import java.io.IOException;
    4: import java.io.ObjectInputStream;
    5: import java.io.ObjectOutputStream;
    6: import java.io.Serializable;
    7: import java.util.ArrayList;
    8: import java.util.HashMap;
    9: import java.util.Hashtable;
   10: import java.util.TreeMap;
   11:
   12: import translation.Messages;
   13:
   14: public class RelationPossibility implements Serializable {
   15:
   16:         /**
   17:          *
   18:          */
   19:         private static final long serialVersionUID = -4953398953739613952L;
   20:         /**
   21:          * Toutes les possibilitÃ©s sont contenues dans cette map Les objectifs
   22:          * correspondent aux clÃ©es et les moyens sont stockÃ©s dans l'arrayList
   23:          * correspondant
   24:          */
   25:
   26:         public static final int RELATIONREELLE = 1;
   27:         public static final int RELATIONSTRUCTURELLE = 0;
   28:
   29:         public static Mean UNDEFINED_MEAN = new Mean(Messages.getString("RelationP
ossibility.0")); //$NON-NLS-1$
   30:         public static Mean NORELATION_MEAN = new Mean(Messages.getString("Relation
Possibility.1")); //$NON-NLS-1$
   31:         public static Goal UNDEFINED_GOAL= new Goal(Messages.getString("RelationPo
ssibility.0")); //$NON-NLS-1$
   32:         public static Goal NORELATION_GOAL = new Goal(Messages.getString("Relation
Possibility.1")); //$NON-NLS-1$
   33:         public static String NORELATION_STRING = Messages.getString("RelationPossi
bility.1"); //$NON-NLS-1$
   34:         public static String UNDEFINED_STRING =  Messages.getString("RelationPossi
bility.0"); //$NON-NLS-1$
   35: //      HashMap<String, ArrayList<String>> possibility;
   36:         TreeMap<Goal, ArrayList<Mean> > possibility;
   37:
   38:         public RelationPossibility() {
   39: //              possibility = new HashMap<String, ArrayList<String>>();
   40:                 possibility = new TreeMap<Goal, ArrayList<Mean>>();
   41:         }
   42:
   43:         public void addAll(RelationPossibility newPossibility) {
   44:
   45:                 for (Goal objectif : newPossibility.possibility.keySet()) {
   46:                         ArrayList<Mean> newMeanings = newPossibility.possibility.g
et(objectif);
   47:                         ArrayList<Mean> meanings = possibility.get(objectif);
   48:                         if(possibility.get(objectif) != null) {
   49:                                 for (Mean meaning : newMeanings) {
   50:                                         if(meanings.contains(meaning) == false) {
   51:                                                 meanings.add(meaning);
   52:                                         }
   53:                                 }
   54:                         } else {
   55:                                 possibility.put(objectif, newMeanings);
   56:                         }
   57:                 }
   58:         }
   59:
   60: //      public HashMap<String, ArrayList<String>> getMap() {
   61: //              return possibility;
   62: //      }
   63:
   64:         public TreeMap<Goal, ArrayList<Mean> > getMap() {
   65:                 return possibility;
   66:         }
   67:
   68:         public void modifyKey(Goal oldKey, Goal newKey) {
   69:                 if (newKey.equals(oldKey))
   70:                         return;
   71:
   72:                 if (possibility.containsKey(newKey)) {
   73:                         throw new IllegalArgumentException(
   74:                                         Messages.getString("RelationPossibility.2"
)); //$NON-NLS-1$
   75:                 }
   76:
   77:                 possibility.put(newKey, possibility.remove(oldKey));
   78:         }
   79:
   80:
   81:         private void writeObject(ObjectOutputStream out) throws IOException
   82:         {
   83:                 out.defaultWriteObject();
   84:         }
   85:
   86:         private void readObject(ObjectInputStream in) throws IOException, ClassNot
FoundException
   87:         {
   88:                 ObjectInputStream.GetField fields = in.readFields();
   89:                 Object possibility = fields.get("possibility", new TreeMap<Goal, A
rrayList<Mean> >());
   90:
   91:                 if (possibility instanceof TreeMap<?,?>)
   92:                 {
   93:                         this.possibility = (TreeMap<Goal, ArrayList<Mean>>)possibi
lity;
   94:                 }
   95:                 else if (possibility instanceof HashMap<?,?>)
   96:                 {
   97:                         HashMap<String, ArrayList<String> > oldPossibility = (Hash
Map<String, ArrayList<String>>) possibility;
   98:                         this.possibility = new TreeMap<Goal, ArrayList<Mean>>();
   99:                         for (String o : oldPossibility.keySet())
  100:                         {
  101:                                 ArrayList<Mean> aM = new ArrayList<Mean>();
  102:                                 for (String m : oldPossibility.get(o))
  103:                                 {
  104:                                         aM.add(new Mean(m));
  105:                                 }
  106:                                 this.possibility.put(new Goal(o), aM);
  107:                         }
  108:
  109:                 }
  110:         }
  111:
  112: }
```

```
    1: package relations;
    2:
    3: import java.io.Serializable;
    4:
    5: import client.stringTranslator.StringTranslator;
    6:
    7: public class Mean implements Comparable<Mean>, Serializable{
    8:
    9:         /**
   10:          *
   11:          */
   12:         private static final long serialVersionUID = 5819830618061598223L;
   13:         public String name;
   14:
   15:         public Mean(String name)
   16:         {
   17:                 this.name = name;
   18:         }
   19:
   20:
   21:         public void setName(String newName)
   22:         {
   23:                 name = newName;
   24:         }
   25:
   26:         public boolean equals(Mean other)
   27:         {
   28:                 return other.name.equals(name);
   29:         }
   30:
   31:         @Override
   32:         public String toString()
   33:         {
   34:                 return StringTranslator.getTranslatedString(this, StringTranslator
.StringType.moyenType);
   35:         }
   36:
   37:         public String getNoTranslatedString() {
   38:                 return name;
   39:         }
   40:
   41:         @Override
   42:         public int compareTo(Mean o) {
   43:
   44:                 return name.compareTo(o.name);
   45:         }
   46: }
```

```
    1: package relations;
    2:
    3: import graphicalUserInterface.DialogHandlerFrame;
    4:
    5: import java.util.ArrayList;
    6: import java.util.HashSet;
    7:
    8: import main.Liens;
    9: import main.ObjectifActeurMoyen;
   10: import main.ObjectifActeurObjet;
   11: import main.ObjectifFormesDecisionnelles;
   12: import main.ObjectifLienHierarchique;
   13: import main.ObjectifLienPartenaire;
   14: import main.ObjectifMoyenActeur;
   15: import main.ObjectifMoyenMoyen;
   16: import main.ObjectifMoyenObjet;
   17: import main.ObjectifObjetActeur;
   18: import main.ObjectifObjetMoyen;
   19: import main.ObjectifRetourLienHierarchique;
   20: import translation.Messages;
   21: import dataPack.Acteur;
   22: import dataPack.Activite;
   23: import dataPack.Content;
   24: import dataPack.Moyen;
   25: import dataPack.SavableObject;
   26:
   27: public class EnsembleRelation implements SavableObject {
   28:
   29:
   30:
   31:         /**
   32:          *
   33:          */
   34:         private static final long serialVersionUID = -2385124293655813004L;
   35:         HashSet<RelationDescription> table;
   36:
   37:
   38:         public EnsembleRelation() {
   39:                 table = new HashSet<RelationDescription>();
   40:         }
   41:
   42:         public void addRelationDescription(RelationDescription newRelDesc) {
   43:                 table.add(newRelDesc);
   44:
   45:         }
   46:
   47:         public HashSet<RelationDescription> getRelationDescriptions() {
   48:                 return table;
   49:         }
   50:
   51:         public RelationPossibility getRelationPossibility(Content first, Content s
econd,
   52:                         boolean realRelation) {
   53:                 Integer a = null;
   54:                 Integer b = null;
   55:                 if (first instanceof Acteur)
   56:                 {
   57:                         a = ((Acteur)first).getStatut().id;
   58:                 }
   59:                 else if (first instanceof Moyen)
   60:                 {
   61:                         a = new Integer(18 + ((Moyen) first).getIdGenerique().intV
alue());
   62:                 }
   63:                 else if (first instanceof Activite)
   64:                 {
   65:                         a = new Integer(22);
   66:                 }
   67:
   68:                 if (second instanceof Acteur) {
   69:                         b = ((Acteur) second).getStatut().id;
   70:                 } else if (second instanceof Moyen) {
   71:                         b = new Integer(18 + ((Moyen) second).getIdGenerique().int
Value());
   72:                 } else if (second instanceof Activite) {
   73:                         b = new Integer(22);
   74:                 }
   75:
   76:                 if (a == null || b == null) {
   77:                         DialogHandlerFrame
   78:                                         .showErrorDialog(Messages.getString("Ensem
bleRelation.0")); //$NON-NLS-1$
   79:                         return null;
   80:                 }
   81:
   82:
   83:                 RelationPossibility result = new RelationPossibility();
   84:                 for (RelationDescription relation : table) {
   85:                         if (relation.allow(a, b, realRelation)) {
   86:                                 result.addAll(relation.getPossibility());
   87:                         }
   88:                 }
   89:
   90:                 return result;
   91:         }
   92:
   93:         public void addEnsembleRelation(EnsembleRelation newSet) {
   94:                 table.addAll(newSet.table);
   95:         }
   96:
   97:         @Override
   98:         public String getFilePath() {
   99:                 return null;
  100:         }
  101:
  102:         @Override
  103:         public void setFilePath(String filePath) {
  104:
  105:         }
  106:
  107:         static public EnsembleRelation getDefaultRelations() {
  108:                 EnsembleRelation ensembleRelation = new EnsembleRelation();
  109:
  110:                 //Acteurs 0-7
  111:                 //Partenaires 8-17
  112:                 //Objectifs 18-21
  113:                 //ActivitÃ© 22
  114:
  115:
  116:                 //Objectif Acteur Objectif
  117:                 RelationDescription relationDescription = new RelationDescription(
Messages.getString("EnsembleRelation.1")); //$NON-NLS-1$
  118:                 //ajoute les relations possible
  119:                 relationDescription.setRect(0, 17, 18, 21, true);
  120:                 //ajoute les relations Ã  choisir
  121:                 for(ObjectifActeurMoyen objectif : ObjectifActeurMoyen.values()) {
  122:                         relationDescription.getPossibility().getMap().put(new Goal
(objectif.toString()), Liens.getStrings(objectif));
  123:                 }
  124:                 ensembleRelation.addRelationDescription(relationDescription);
  125:
  126:                 //Objectif Acteur ActivitÃ©
  127:                 relationDescription = new RelationDescription(Messages.getString("
EnsembleRelation.2")); //$NON-NLS-1$
```

```
  128:                //ajoute les relations possible
  129:                relationDescription.setRect(0, 17, 22, 22, true);
  130:                //ajoute les relations Ã  choisir
  131:                for(ObjectifActeurObjet objectif : ObjectifActeurObjet.values()) {
  132:                        relationDescription.getPossibility().getMap().put(new Goal
(objectif.toString()), Liens.getStrings(objectif));
  133:                }
  134:                ensembleRelation.addRelationDescription(relationDescription);
  135:
  136:                //Objectif formesDecisionnelles (relation rÃ©els acteur)
  137:                relationDescription = new RelationDescription(Messages.getString("
EnsembleRelation.3")); //$NON-NLS-1$
  138:                //ajoute les relations possible
  139:                relationDescription.setRealRelation(true);
  140:                relationDescription.setRect(0, 17, 0, 17, true);
  141:                //ajoute les relations Ã  choisir
  142:                for(ObjectifFormesDecisionnelles objectif : ObjectifFormesDecision
nelles.values()) {
  143:                        relationDescription.getPossibility().getMap().put(new Goal
(objectif.toString()), Liens.getStrings(objectif));
  144:                }
  145:                ensembleRelation.addRelationDescription(relationDescription);
  146:
  147:                //Objectif lien hierarchique
  148:                relationDescription = new RelationDescription(Messages.getString("
EnsembleRelation.4")); //$NON-NLS-1$
  149:                //ajoute les relations possible
  150:                relationDescription.setTriangleSup(0, 0, 8, true);
  151:                //ajoute les relations Ã  choisir
  152:                for(ObjectifLienHierarchique objectif : ObjectifLienHierarchique.v
alues()) {
  153:                        relationDescription.getPossibility().getMap().put(new Goal
(objectif.toString()), Liens.getStrings(objectif));
  154:                }
  155:                ensembleRelation.addRelationDescription(relationDescription);
  156:
  157:                //Objectif lien partenaire
  158:                relationDescription = new RelationDescription(Messages.getString("
EnsembleRelation.5")); //$NON-NLS-1$
  159:                //ajoute les relations possible
  160:                relationDescription.setRect(0, 17, 0, 17, true);
  161:                //ajoute les relations Ã  choisir
  162:                for(ObjectifLienPartenaire objectif : ObjectifLienPartenaire.value
s()) {
  163:                        relationDescription.getPossibility().getMap().put(new Goal
(objectif.toString()), Liens.getStrings(objectif));
  164:                }
  165:                //ObjectifFormesDecisionnelles formesDecisionnelles = ObjectifForm
esDecisionnelles.values()[1];
  166:                //relationDescription.getPossibility().getMap().put(formesDecision
nelles.toString(), Liens.getStrings(formesDecisionnelles));
  167:
  168:                ensembleRelation.addRelationDescription(relationDescription);
  169:
  170:                //Objectif objectif acteur
  171:                relationDescription = new RelationDescription(Messages.getString("
EnsembleRelation.6")); //$NON-NLS-1$
  172:                //ajoute les relations possible
  173:                relationDescription.setRect(18, 21, 0, 17, true);
  174:                //ajoute les relations Ã  choisir
  175:                for(ObjectifMoyenActeur objectif : ObjectifMoyenActeur.values()) {
  176:                        relationDescription.getPossibility().getMap().put(new Goal
(objectif.toString()), Liens.getStrings(objectif));
  177:                }
  178:                ensembleRelation.addRelationDescription(relationDescription);
  179:
  180:                //Objectif objectif objectif
  181:                relationDescription = new RelationDescription(Messages.getString("
EnsembleRelation.7")); //$NON-NLS-1$
  182:                //ajoute les relations possible
  183:                relationDescription.setRect(18, 21, 18, 21, true);
  184:                //ajoute les relations Ã  choisir
  185:                for(ObjectifMoyenMoyen objectif : ObjectifMoyenMoyen.values()) {
  186:                        relationDescription.getPossibility().getMap().put(new Goal
(objectif.toString()), Liens.getStrings(objectif));
  187:                }
  188:                ensembleRelation.addRelationDescription(relationDescription);
  189:
  190:                //Objectif objectif activite
  191:                relationDescription = new RelationDescription(Messages.getString("
EnsembleRelation.8")); //$NON-NLS-1$
  192:                //ajoute les relations possible
  193:                relationDescription.setRect(18, 21, 22, 22, true);
  194:                //ajoute les relations Ã  choisir
  195:                for(ObjectifMoyenObjet objectif : ObjectifMoyenObjet.values()) {
  196:                        relationDescription.getPossibility().getMap().put(new Goal
(objectif.toString()), Liens.getStrings(objectif));
  197:                }
  198:                ensembleRelation.addRelationDescription(relationDescription);
  199:
  200:                //Objectif activite acteur
  201:                relationDescription = new RelationDescription(Messages.getString("
EnsembleRelation.9")); //$NON-NLS-1$
  202:                //ajoute les relations possible
  203:                relationDescription.setRect(22, 22, 0, 17, true);
  204:                //ajoute les relations Ã  choisir
  205:                for(ObjectifObjetActeur objectif : ObjectifObjetActeur.values()) {
  206:                        relationDescription.getPossibility().getMap().put(new Goal
(objectif.toString()), Liens.getStrings(objectif));
  207:                }
  208:                ensembleRelation.addRelationDescription(relationDescription);
  209:
  210:                //Objectif activite objectif
  211:                relationDescription = new RelationDescription(Messages.getString("
EnsembleRelation.10")); //$NON-NLS-1$
  212:                //ajoute les relations possible
  213:                relationDescription.setRect(22, 22, 18, 21, true);
  214:                //ajoute les relations Ã  choisir
  215:                for(ObjectifObjetMoyen objectif : ObjectifObjetMoyen.values()) {
  216:                        relationDescription.getPossibility().getMap().put(new Goal
(objectif.toString()), Liens.getStrings(objectif));
  217:                }
  218:                ensembleRelation.addRelationDescription(relationDescription);
  219:
  220:                //Objectif hierarchique retour
  221:                relationDescription = new RelationDescription(Messages.getString("
EnsembleRelation.11")); //$NON-NLS-1$
  222:                //ajoute les relations possible
  223:                relationDescription.setTriangleInf(0, 0, 8, true);
  224:                //ajoute les relations Ã  choisir
  225:                for(ObjectifRetourLienHierarchique objectif : ObjectifRetourLienHi
erarchique.values()) {
  226:                        relationDescription.getPossibility().getMap().put(new Goal
(objectif.toString()), Liens.getStrings(objectif));
  227:                }
  228:                ensembleRelation.addRelationDescription(relationDescription);
  229:
  230:                //Non defini
  231:                ArrayList<Mean> undefStrings = new ArrayList<Mean>();
  232:                undefStrings.add(RelationPossibility.UNDEFINED_MEAN);
  233:                ArrayList<Mean> noRelationStrings = new ArrayList<Mean>();
  234:                noRelationStrings.add(RelationPossibility.NORELATION_MEAN);
```

```
  235:                 relationDescription = new RelationDescription(RelationPossibility.
UNDEFINED_STRING+" & "+RelationPossibility.NORELATION_STRING); //$NON-NLS-1$
  236:                 relationDescription.setRect(0, 22, 0, 22, true);
  237:                 relationDescription.getPossibility().getMap().put(RelationPossibil
ity.UNDEFINED_GOAL, undefStrings);
  238:                 relationDescription.getPossibility().getMap().put(RelationPossibil
ity.NORELATION_GOAL, noRelationStrings);
  239:                 ensembleRelation.addRelationDescription(relationDescription);
  240:
  241:
  242:                 return ensembleRelation;
  243:         }
  244:
  245:
  246: }
```

```java
package relations;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.Vector;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.border.BevelBorder;
import javax.swing.event.CaretEvent;
import javax.swing.event.CaretListener;

import main.Statut;
import main.StatutObjet;
import translation.Messages;

public class DescriptionBuilder extends JPanel{

	private enum SelectedMode {SimpleMode, AddMode, OnlyMode, InversMode};

	private static final long serialVersionUID = 2656852414355161636L;

	private final Vector<JCheckBox> descriptions; // ligne les une a la suite des autre (debut des index a 0)
	private final Vector<JLabel> descriptionsName;

	private final RelationDescription relationDescription;

	private SelectedMode selectingMode = SelectedMode.SimpleMode;
	private int hierarchiqueMode = 0;
	private boolean isLineMode = true;
	private boolean isObjectifMode = false;
	private boolean isActivitiesMode = false;
	private boolean isPartnerMode = false;
	private boolean isAllMode = false;

	private final JPanel complexFilter;

	private final Color lineColor = Color.RED;
	private final Color columnColor = Color.YELLOW;

	private final DescriptionBuilder descriptionBuilder;

	private final int descriptionSize = 23;

	public DescriptionBuilder(RelationDescription relationDescription) {
		descriptionBuilder = this;

		descriptions = new Vector<JCheckBox>(descriptionSize*descriptionSize);
		descriptionsName = new Vector<JLabel>(descriptionSize);

		this.relationDescription = relationDescription;
		//TODO On ajoute la relation non-def/non-def par default ou non ? on ce demandait pourquoi il y avait la case 0,0 toujours cochÃ© et bien c'etait pour ca.
		//relationDescription.set(0, 0, true);

		complexFilter = new JPanel();

		buildView();
	}

	private void buildView() {
		this.setVisible(true);
		this.setLayout(new BorderLayout());

		this.add(createNameView(), BorderLayout.WEST);
		this.add(createCheckBoxView(), BorderLayout.CENTER);
		this.add(new JScrollPane(createFilterView()), BorderLayout.NORTH);

		this.validate();
	}

	private JPanel createNameView() {
		JPanel view = new JPanel(new GridLayout(0, 1));

		for (int i = 0 ; i < descriptionSize ; i++) {
			String label;

			if(i < 18) {
				label = Statut.values()[i].toString();
			} else if (i < 22) {
				label = StatutObjet.values()[i - 18].toString();
			} else {
				label = Messages.getString("DescriptionBuilder.0") ; //$NON-NLS-1$
			}

			JLabel textLabel = new JLabel(label);
			descriptionsName.add(textLabel);
			textLabel.setBorder(BorderFactory.createEmptyBorder(0, 2, 0, 2));
			view.add(textLabel);
		}

		return view;
	}

	private JPanel createCheckBoxView() {
		JPanel view = new JPanel(new GridLayout(descriptionSize, descriptionSize));

		int descriptionSizeSqrt = descriptionSize*descriptionSize;
		for (int i = 0 ; i < descriptionSizeSqrt ; i++) {
			final int a = i/descriptionSize;
			final int b = i%descriptionSize;

			JCheckBox checkBox = new JCheckBox();

			boolean borderRight = b == 17 || b == 7 || b == 21;
			boolean borderBottom = a == 17 || a == 7 || a == 21;

			checkBox.setBorder(BorderFactory.createEmptyBorder());
			checkBox.setSelected(relationDescription.allow(a, b, relationDescription.isRealRelation()));
			checkBox.removeMouseListener(checkBox.getMouseListeners()[0]);

			JPanel container = new JPanel();
```

```
  127:                         container.setLayout(new GridBagLayout());
  128:
  129:                         int borderRightSize = b == 7 ? 1 : 2;
  130:                         int borderBottomSize = a == 7 ? 1 : 2;
  131:
  132:                         if(borderBottom) {
  133:                                 if(borderRight)
  134:                                         container.setBorder(BorderFactory.createMa
tteBorder(0, 0, borderBottomSize, borderRightSize, Color.BLACK));
  135:                                 else
  136:                                         container.setBorder(BorderFactory.createCo
mpoundBorder(BorderFactory.createMatteBorder(0, 0, borderBottomSize, 0, Color.BLACK),Bord
erFactory.createEmptyBorder(0, 0, borderBottomSize, 0)));
  137:                         } else if (borderRight) {
  138:                                 container.setBorder(BorderFactory.createCompoundBo
rder(BorderFactory.createMatteBorder(0, 0, 0, borderRightSize, Color.BLACK),BorderFactory
.createEmptyBorder(0, 0, 0, borderRightSize)));
  139:                         }
  140:
  141:                         container.add(checkBox);
  142:
  143:                         checkBox.addItemListener(new ItemListener() {
  144:
  145:                                 @Override
  146:                                 public void itemStateChanged(ItemEvent e) {
  147:                                         relationDescription.set(a, b, e.getStateCh
ange() == ItemEvent.SELECTED);
  148:                                 }
  149:                         });
  150:
  151:                         MouseListener mouseListner = new MouseListener() {
  152:                                 @Override
  153:                                 public void mouseEntered(MouseEvent e) {
  154:                                         if(a == b) {
  155:                                                 descriptionsName.get(a).setBorder(
BorderFactory.createCompoundBorder(BorderFactory.createLineBorder(lineColor, 1), BorderFa
ctory.createLineBorder(columnColor, 1)));
  156:                                         } else {
  157:                                                 descriptionsName.get(a).setBorder(
BorderFactory.createLineBorder(lineColor, 2));
  158:                                                 descriptionsName.get(b).setBorder(
BorderFactory.createLineBorder(columnColor, 2));
  159:                                         }
  160:
  161:                                         if(selectingMode == SelectedMode.SimpleMod
e) {
  162:                                                 try {
  163:                                                         ((JPanel)e.getSource()).se
tBackground(Color.LIGHT_GRAY);
  164:                                                 } catch (Exception e1) {
  165:                                                         ((JCheckBox)e.getSource())
.getParent().setBackground(Color.LIGHT_GRAY);
  166:                                                 }
  167:                                         } else {
  168:                                                 descriptionBuilder.setPaternVisibl
e(a, b, true);
  169:                                         }
  170:                                 }
  171:
  172:                                 @Override
  173:                                 public void mouseExited(MouseEvent e) {
  174:                                         descriptionsName.get(a).setBorder(BorderFa
ctory.createEmptyBorder(0, 2, 0, 2));
  175:                                         descriptionsName.get(b).setBorder(BorderFa
ctory.createEmptyBorder(0, 2, 0, 2));
  176:
  177:                                         if(selectingMode == SelectedMode.SimpleMod
e) {
  178:                                                 try {
  179:                                                         ((JPanel)e.getSource()).se
tBackground(null);
  180:                                                 } catch (Exception e1) {
  181:                                                         ((JCheckBox)e.getSource())
.getParent().setBackground(null);
  182:                                                 }
  183:                                         } else {
  184:                                                 descriptionBuilder.setPaternVisibl
e(a, b, false);
  185:                                         }
  186:                                 }
  187:
  188:                                 @Override
  189:                                 public void mouseReleased(MouseEvent e) {}
  190:                                 @Override
  191:                                 public void mousePressed(MouseEvent e) {}

  192:                                 @Override
  193:                                 public void mouseClicked(MouseEvent e) {
  194:                                         descriptionBuilder.applyPatern(a, b);
  195:                                 }
  196:                         };
  197:
  198:                         container.addMouseListener(mouseListner);
  199:                         checkBox.addMouseListener(mouseListner);
  200:
  201:                         view.add(container);
  202:                         view.setBorder(BorderFactory.createBevelBorder(BevelBorder
.RAISED));
  203:
  204:                         descriptions.add(checkBox);
  205:                 }
  206:
  207:                 return view;
  208:         }
  209:
  210:         private JPanel createFilterView() {
  211:                 JPanel view = new JPanel();
  212:                 view.setLayout(new BoxLayout(view, BoxLayout.X_AXIS));
  213:
  214:                 String legendText = "Une case cochÃ©e indique une relation\nentre
l'acteur encadrÃ© en rouge vers \nl'acteur encadrÃ© en jaune.";
  215:                 JTextArea legend = new JTextArea(legendText);
  216:                 legend.setBorder(BorderFactory.createTitledBorder("LÃ©gend"));
  217:                 legend.setOpaque(false);
  218:                 legend.setEditable(false);
  219:                 view.add(legend);
  220:
  221:                 JPanel nameAndRealPanel = new JPanel(new BorderLayout());
  222:                 final JTextField nameField = new JTextField(relationDescription
  223:                                 .toString(), 25);
  224:                 nameField.addCaretListener(new CaretListener() {
  225:
  226:                         @Override
  227:                         public void caretUpdate(CaretEvent arg0) {
  228:                                 relationDescription.setName(nameField.getText());
  229:                         }
  230:                 });
  231:                 JPanel embeddedNamePanel = new JPanel(new GridBagLayout());
  232:                 embeddedNamePanel.add(nameField);
  233:
  234:                 nameAndRealPanel.add(embeddedNamePanel, BorderLayout.CENTER);
  235:                 JCheckBox realRelations = new JCheckBox(Messages.getString("Descri
ptionBuilder.1")); //$NON-NLS-1$
  236:                 realRelations.setSelected(relationDescription.isRealRelation());
```

```
  237:                 nameAndRealPanel.add(realRelations, BorderLayout.SOUTH);
  238:
  239:                 nameAndRealPanel.setBorder(BorderFactory
  240:                                 .createTitledBorder(Messages.getString("Descriptio
nBuilder.2"))); //$NON-NLS-1$
  241:
  242:                 view.add(nameAndRealPanel);
  243:
  244:                 String[] logics = {Messages.getString("DescriptionBuilder.3"), Mes
sages.getString("DescriptionBuilder.4"), Messages.getString("DescriptionBuilder.5"), Mess
ages.getString("DescriptionBuilder.6")}; //$NON-NLS-1$ //$NON-NLS-2$ //$NON-NLS-3$ //$NON
-NLS-4$
  245:
  246:                 final JComboBox logicFilter = new JComboBox(logics);
  247:                 logicFilter.addItemListener(new ItemListener() {
  248:                         @Override
  249:                         public void itemStateChanged(ItemEvent e) {
  250:                                 descriptionBuilder.selectingMode = SelectedMode.va
lues()[logicFilter.getSelectedIndex()];
  251:                                 setEnableRecursively(complexFilter, descriptionBui
lder.selectingMode != SelectedMode.SimpleMode);
  252:                         }
  253:                 });
  254:
  255:
  256:                 JPanel embeddingLogicPanel = new JPanel(new GridBagLayout());
  257:                 embeddingLogicPanel.add(logicFilter);
  258:                 view.add(embeddingLogicPanel);
  259:
  260:                 complexFilter.setLayout(new BoxLayout(complexFilter, BoxLayout.X_A
XIS));
  261:
  262:                 JCheckBox allBox = new JCheckBox(Messages.getString("DescriptionBu
ilder.7")); //$NON-NLS-1$
  263:                 allBox.addItemListener(new ItemListener() {
  264:                         @Override
  265:                         public void itemStateChanged(ItemEvent e) {
  266:                                 descriptionBuilder.isAllMode = e.getStateChange()
== ItemEvent.SELECTED;
  267:                         }
  268:                 });
  269:                 complexFilter.add(allBox);
  270:
  271:                 JPanel hierarchiqueView = new JPanel();
  272:                 hierarchiqueView.setLayout(new BoxLayout(hierarchiqueView, BoxLayo
ut.Y_AXIS));
  273:
  274:                 JCheckBox hierarchiqueBox = new JCheckBox(Messages.getString("Desc
riptionBuilder.8")); //$NON-NLS-1$
  275:                 hierarchiqueBox.addItemListener(new ItemListener() {
  276:                         @Override
  277:                         public void itemStateChanged(ItemEvent e) {
  278:                                 descriptionBuilder.hierarchiqueMode += e.getStateC
hange() == ItemEvent.SELECTED ? 1 : -1;
  279:                         }
  280:                 });
  281:
  282:                 hierarchiqueView.add(hierarchiqueBox);
  283:
  284:                 JCheckBox supHierarchiqueBox = new JCheckBox(Messages.getString("D
escriptionBuilder.9")); //$NON-NLS-1$
  285:                 supHierarchiqueBox.addItemListener(new ItemListener() {
  286:                         @Override
  287:                         public void itemStateChanged(ItemEvent e) {
  288:                                 descriptionBuilder.hierarchiqueMode += e.getStateC
hange() == ItemEvent.SELECTED ? 2 : -2;
  289:                         }
  290:                 });
  291:                 hierarchiqueView.add(supHierarchiqueBox);
  292:
  293:                 JCheckBox infHierarchiqueBox = new JCheckBox(Messages.getString("D
escriptionBuilder.10")); //$NON-NLS-1$
  294:                 infHierarchiqueBox.addItemListener(new ItemListener() {
  295:                         @Override
  296:                         public void itemStateChanged(ItemEvent e) {
  297:                                 descriptionBuilder.hierarchiqueMode += e.getStateC
hange() == ItemEvent.SELECTED ? 4 : -4;
  298:                         }
  299:                 });
  300:                 hierarchiqueView.add(infHierarchiqueBox);
  301:
  302:                 complexFilter.add(hierarchiqueView);
  303:
  304:                 JCheckBox partnerBox = new JCheckBox(Messages.getString("Descripti
onBuilder.11")); //$NON-NLS-1$
  305:                 partnerBox.addItemListener(new ItemListener() {
  306:                         @Override
  307:                         public void itemStateChanged(ItemEvent e) {
  308:                                 descriptionBuilder.isPartnerMode = e.getStateChang
e() == ItemEvent.SELECTED;
  309:                         }
  310:                 });
  311:                 complexFilter.add(partnerBox);
  312:
  313:                 JCheckBox objectifBox = new JCheckBox(Messages.getString("Descript
ionBuilder.12")); //$NON-NLS-1$
  314:                 objectifBox.addItemListener(new ItemListener() {
  315:                         @Override
  316:                         public void itemStateChanged(ItemEvent e) {
  317:                                 descriptionBuilder.isObjectifMode = e.getStateChan
ge() == ItemEvent.SELECTED;
  318:                         }
  319:                 });
  320:                 complexFilter.add(objectifBox);
  321:
  322:                 JCheckBox activiteBox = new JCheckBox(Messages.getString("Descript
ionBuilder.13")); //$NON-NLS-1$
  323:                 activiteBox.addItemListener(new ItemListener() {
  324:                         @Override
  325:                         public void itemStateChanged(ItemEvent e) {
  326:                                 descriptionBuilder.isActivitiesMode = e.getStateCh
ange() == ItemEvent.SELECTED;
  327:                         }
  328:                 });
  329:                 complexFilter.add(activiteBox);
  330:
  331:                 String[] lineColone = {Messages.getString("DescriptionBuilder.14")
, Messages.getString("DescriptionBuilder.15")}; //$NON-NLS-1$ //$NON-NLS-2$
  332:
  333:                 final JComboBox lineColoneBox = new JComboBox(lineColone);
  334:                 lineColoneBox.addItemListener(new ItemListener() {
  335:                         @Override
  336:                         public void itemStateChanged(ItemEvent e) {
  337:                                 descriptionBuilder.isLineMode = lineColoneBox.getS
electedIndex() == 0;
  338:                         }
  339:                 });
  340:                 JPanel embeddingLineColumnPanel = new JPanel(new GridBagLayout());
  341:                 embeddingLineColumnPanel.add(lineColoneBox);
  342:                 complexFilter.add(embeddingLineColumnPanel);
  343:
  344:                 setEnableRecursively(complexFilter, false);
  345:
```

```
  346:                 view.add(complexFilter);
  347:
  348:                 JPanel filterView = new JPanel(new GridBagLayout());
  349:                 filterView.add(view);
  350:                 return view;
  351:         }
  352:
  353:         public void setValue(int line, int column, boolean value) {
  354:                 descriptions.get(getIndex(line, column)).setSelected(value);
  355:         }
  356:
  357:         public boolean getValue(int line, int column) {
  358:                 return descriptions.get(getIndex(line, column)).isSelected();
  359:         }
  360:
  361:         public String getName(int lineOrColumn) {
  362:                 return descriptionsName.get(lineOrColumn).getText();
  363:         }
  364:
  365:         private int getIndex(int line, int column) {
  366:                 return line + column*descriptionSize;
  367:         }
  368:
  369:         public Vector<JCheckBox> getCheckBoxSelected(int a, int b) {
  370:                 if(selectingMode == SelectedMode.SimpleMode) {
  371:                         return new Vector<JCheckBox>();
  372:                 }
  373:
  374:                 Vector<JCheckBox> selectedBoxes = new Vector<JCheckBox>();
  375:                 Vector<JCheckBox> columLineBoxes = isLineMode ? getLine(a) : getCo
lumn(b);
  376:                 int hierarchiqueLevel = isLineMode ? a : b;
  377:
  378:                 if(isAllMode) {
  379:                         return columLineBoxes;
  380:                 }
  381:
  382:                 for(int i = 0 ; i < descriptionSize ; i++) {
  383:                         JCheckBox currentCheckBox = columLineBoxes.get(i);
  384:
  385:                         if(i<8) {
  386:                                 if((hierarchiqueMode & 1) == 1 || (hierarchiqueMod
e !=0 && hierarchiqueLevel > 7)) {
  387:                                         selectedBoxes.add(currentCheckBox);
  388:                                 } else {
  389:                                         if ((hierarchiqueMode & 2) == 2)
  390:                                                 if(isLineMode ? i<hierarchiqueLeve
l : i<hierarchiqueLevel)
  391:                                                         selectedBoxes.add(currentC
heckBox);
  392:
  393:                                         if ((hierarchiqueMode & 4) == 4)
  394:                                                 if(isLineMode ? i>hierarchiqueLeve
l : i>hierarchiqueLevel)
  395:                                                         selectedBoxes.add(currentC
heckBox);
  396:
  397:                                         if((hierarchiqueMode & 6) == 6 && i == hie
rarchiqueLevel)
  398:                                                 selectedBoxes.add(currentCheckBox)
;
  399:                                 }
  400:                         } else if (i<18) {
  401:                                 if(isPartnerMode) {
  402:                                         selectedBoxes.add(currentCheckBox);
  403:                                 }
  404:                         } else if(i<22) {
  405:                                 if(isObjectifMode) {
  406:                                         selectedBoxes.add(currentCheckBox);
  407:                                 }
  408:                         } else if (isActivitiesMode) {
  409:                                 selectedBoxes.add(currentCheckBox);
  410:                         }
  411:                 }
  412:
  413:                 return selectedBoxes;
  414:         }
  415:
  416:         public void setPaternVisible(int a, int b, boolean visible) {
  417:                 for(JCheckBox checkBox : getCheckBoxSelected(a, b)) {
  418:                         if(visible) {
  419:                                 checkBox.getParent().setBackground(isLineMode ? li
neColor : columnColor);
  420:                         } else {
  421:                                 checkBox.getParent().setBackground(null);
  422:                         }
  423:
  424:                         checkBox.validate();
  425:                 }
  426:         }
  427:
  428:         public void applyPatern(int a, int b) {
  429:                 if(selectingMode == SelectedMode.SimpleMode) {
  430:                         descriptions.get(a*descriptionSize + b).setSelected(!descr
iptions.get(a*descriptionSize + b).isSelected());
  431:                 } else {
  432:                         if(selectingMode == SelectedMode.OnlyMode) {
  433:                                 //dÃ©coche toute la ligne ou colonne
  434:                                 for(JCheckBox checkBox : isLineMode ? descriptionB
uilder.getLine(a) : descriptionBuilder.getColumn(b)) {
  435:                                         checkBox.setSelected(false);
  436:                                 }
  437:                         }
  438:
  439:                         for(JCheckBox checkBox : getCheckBoxSelected(a, b)) {
  440:                                 checkBox.setSelected(selectingMode == SelectedMode
.InversMode ? (!checkBox.isSelected()) : true);
  441:                         }
  442:                 }
  443:         }
  444:
  445:         public Vector<JCheckBox> getLine(int index) {
  446:                 Vector<JCheckBox> line = new Vector<JCheckBox>();
  447:
  448:                 for(int i = 0 ; i < descriptionSize ; i++) {
  449:                         line.add(descriptions.get(i + descriptionSize*index));
  450:                 }
  451:
  452:                 return line;
  453:         }
  454:
  455:         public Vector<JCheckBox> getColumn(int index) {
  456:                 Vector<JCheckBox> column = new Vector<JCheckBox>();
  457:
  458:                 for(int i = 0 ; i < descriptionSize ; i++) {
  459:                         column.add(descriptions.get(i*descriptionSize + index));
  460:                 }
  461:
  462:                 return column;
  463:         }
  464:
  465:         public void setEnableRecursively(JComponent component, boolean enable) {
  466:                 try {
  467:                 for(Component childComponent : component.getComponents()) {
```

```
  468:                         setEnableRecursively((JComponent)childComponent, enable);
  469:                 }
  470:                 } catch (Exception e) {
  471:
  472:                 }
  473:
  474:                 component.setEnabled(enable);
  475:         }
  476:
  477: //      public static void main(String[] args) {
  478: //              JFrame window = new JFrame();
  479: //
  480: //              RelationDescription relationTest = new RelationDescription("Test r
elation description");
  481: //              relationTest.setRealRelation(true);
  482: //              //relationTest.set(10, 10, true);
  483: //
  484: //              window.add(new DescriptionBuilder(relationTest));
  485: //              window.setSize(800, 600);
  486: //              window.setVisible(true);
  487: //              window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  488: //              window.pack();
  489: //      }
  490: }
```

```
  1: package relations;
  2:
  3: import graphicalUserInterface.DataPackView;
  4: import graphicalUserInterface.DialogHandlerFrame;
  5:
  6: import java.awt.BorderLayout;
  7: import java.awt.CardLayout;
  8: import java.awt.event.ActionEvent;
  9: import java.awt.event.ActionListener;
 10:
 11: import javax.swing.Box;
 12: import javax.swing.BoxLayout;
 13: import javax.swing.DefaultListModel;
 14: import javax.swing.JButton;
 15: import javax.swing.JList;
 16: import javax.swing.JOptionPane;
 17: import javax.swing.JPanel;
 18: import javax.swing.JScrollPane;
 19: import javax.swing.JTextField;
 20: import javax.swing.event.ListSelectionEvent;
 21: import javax.swing.event.ListSelectionListener;
 22:
 23: import translation.Messages;
 24:
 25:
 26: public class RelationBrowser extends JPanel {
 27:
 28:         /**
 29:          *
 30:          */
 31:         private static final long serialVersionUID = 5902621915174739377L;
 32:
 33:
 34:         protected RelationPanel relationPanel;
 35:
 36:         protected JPanel mainPanel;
 37:         protected EnsembleRelation relationSet;
 38:         protected JList relationList;
 39:         protected JButton switchButton;
 40:         protected JPanel descriptionPanel;
 41:         protected JPanel cardPanel;
 42:         protected CardLayout cardLayout;
 43:         protected boolean contentMode;
 44:         protected DefaultListModel relationVector;
 45:
 46:
 47:         public RelationBrowser(final EnsembleRelation relationSet) {
 48:
 49:                 mainPanel = new JPanel(new BorderLayout());
 50:
 51:                 this.relationSet = relationSet;
 52:                 relationPanel = new RelationPanel();
 53:                 contentMode = true;
 54:                 setLayout(new BorderLayout());
 55:
 56:                 cardLayout = new CardLayout();
 57:                 cardPanel = new JPanel(cardLayout);
 58:                 mainPanel.add(relationPanel,BorderLayout.CENTER);
 59:
 60:                 JPanel listPanel = new JPanel(new BorderLayout());
 61:                 relationVector = new DefaultListModel();
 62:                 for (RelationDescription relation : relationSet.table) {
 63:                         relationVector.addElement(relation);
 64:                 }
 65:                 relationList = new JList(relationVector);
 66:
 67:                 relationList.addListSelectionListener(new ListSelectionListener()
{
 68:
 69:                         @Override
 70:                         public void valueChanged(ListSelectionEvent arg0) {
 71:                                 openRelationDescription((RelationDescription) rela
tionList
 72:                                                 .getSelectedValue());
 73:                         }
 74:                 });
 75:
 76:                 listPanel.add(new JScrollPane(relationList), BorderLayout.CENTER);
 77:                 switchButton = new JButton(Messages.getString("RelationBrowser.0")
); //$NON-NLS-1$
 78:                 switchButton.addActionListener(new ActionListener() {
 79:
 80:                         @Override
 81:                         public void actionPerformed(ActionEvent e) {
 82:                                 switchShownPanel();
 83:
 84:                         }
 85:
 86:                 });
 87:                 switchButton.setEnabled(false);
 88:                 JPanel addPanel = new JPanel();
 89:                 addPanel.setLayout(new BoxLayout(addPanel,BoxLayout.X_AXIS));
 90:                 final JTextField name = new JTextField(
 91:                                 Messages.getString("RelationBrowser.1")); //$NON-N
LS-1$
 92:                 name
 93:                 .addFocusListener(DataPackView
 94:                                 .generateMouseListener(Messages.getString("Relatio
nBrowser.2"))); //$NON-NLS-1$
 95:                 JButton add = new JButton(Messages.getString("RelationBrowser.3"))
; //$NON-NLS-1$
 96:                 add.addActionListener(new ActionListener() {
 97:
 98:                         @Override
 99:                         public void actionPerformed(ActionEvent e) {
100:                                 RelationDescription newRelation = new RelationDesc
ription(name.getText());
101:                                 relationVector.addElement(newRelation);
102:
103:                                 relationSet.addRelationDescription(newRelation);
104:                                 name.setText(Messages.getString("RelationBrowser.4
")); //$NON-NLS-1$
105:                                 relationList
106:                                 .setSelectedIndex(relationList.getModel().getSize(
) - 1);
107:                                 openRelationDescription((RelationDescription) rela
tionList
108:                                                 .getSelectedValue());
109:
110:                         }
111:                 });
112:                 addPanel.add(name);
113:                 addPanel.add(add);
114:
115:                 JPanel delPanel = new JPanel();
116:                 delPanel.setLayout(new BoxLayout(delPanel, BoxLayout.X_AXIS));
117:
118:                 JButton delRelation = new JButton(Messages.getString("RelationBrow
ser.5")); //$NON-NLS-1$
119:                 delRelation.addActionListener(new ActionListener() {
120:
121:                         @Override
122:                         public void actionPerformed(ActionEvent arg0) {
123:                                 if (DialogHandlerFrame
```

```
  124:                                                .showYesNoDialog(Messages.getStrin
g("RelationBrowser.6")) == JOptionPane.YES_OPTION) { //$NON-NLS-1$
  125:                                        relationSet.table.remove(relationList.getS
electedValue());
  126:                                        relationVector.removeElement(relationList
  127:                                                        .getSelectedValue());
  128:                                        relationPanel.loadRelation(null);
  129:                                }
  130:
  131:                        }
  132:                });
  133:
  134:                delPanel.add(Box.createGlue());
  135:                delPanel.add(delRelation);
  136:
  137:                JPanel buttonPanel = new JPanel(new BorderLayout());
  138:                buttonPanel.add(addPanel, BorderLayout.CENTER);
  139:                buttonPanel.add(delPanel, BorderLayout.SOUTH);
  140:
  141:                listPanel.add(buttonPanel, BorderLayout.SOUTH);
  142:                JPanel compressPanel = new JPanel(new BorderLayout());
  143:                compressPanel.setLayout(new BoxLayout(compressPanel, BoxLayout.X_A
XIS));
  144:                compressPanel.add(Box.createGlue());
  145:                compressPanel.add(listPanel);
  146:                compressPanel.add(Box.createGlue());
  147:
  148:                mainPanel.add(compressPanel, BorderLayout.NORTH);
  149:                cardPanel.add(mainPanel,"main"); //$NON-NLS-1$
  150:                add(switchButton, BorderLayout.SOUTH);
  151:
  152:                if (relationVector.size() > 0)
  153:                {
  154:                        relationList.setSelectedIndex(0);
  155:                        openRelationDescription((RelationDescription) relationList
  156:                                        .getSelectedValue());
  157:                }
  158:                add(cardPanel, BorderLayout.CENTER);
  159:
  160:        }
  161:
  162:        protected void openRelationDescription(RelationDescription newRelation)
  163:        {
  164:                if (newRelation != null) {
  165:                        if (!contentMode)
  166:                                switchShownPanel();
  167:
  168:                        if (descriptionPanel != null)
  169:                                cardPanel.remove(descriptionPanel);
  170:
  171:                        relationPanel.loadRelation(newRelation);
  172:                        descriptionPanel = new DescriptionBuilder(newRelation);
  173:
  174:                        cardPanel.add(descriptionPanel,"description"); //$NON-NLS-
1$
  175:
  176:                        switchButton.setEnabled(true);
  177:                        validate();
  178:                } else {
  179:                        relationPanel.loadRelation(null);
  180:                }
  181:        }
  182:
  183:
  184:
  185:        protected void switchShownPanel() {
  186:                if (contentMode)
  187:                        switchButton.setText(Messages.getString("RelationBrowser.9
"); //$NON-NLS-1$
  188:                else
  189:                        switchButton.setText(Messages.getString("RelationBrowser.1
0")); //$NON-NLS-1$
  190:
  191:                cardLayout.next(cardPanel);
  192:                contentMode = !contentMode;
  193:
  194:                cardPanel.validate();
  195:        }
  196:
  197:        public void refreshList()
  198:        {
  199:                relationVector.clear();
  200:                for (RelationDescription relation : relationSet.table) {
  201:                        relationVector.addElement(relation);
  202:                }
  203:
  204:                relationList.validate();
  205:        }
  206:
  207: }
```