

```
1  package generate;
2  import java.io.IOException;
3  import java.io.PrintWriter;
4  import java.util.ArrayList;
5  import java.util.List;
6  import parser.BPELMutator;
7  import parser.mu_List;
8  import utils.XMLWriter;
9  public class FirstOrder extends BPELMutator{
10     private List<String[]> firstOrdermuList = new ArrayList<String[]>();
11     public FirstOrder()
12     {
13         this.firstOrderGenerate();
14     }
15     public List<String[]> getFirstOrdermuList() {
16         return firstOrdermuList;
17     }
18     private void firstOrderGenerate() {
19         if(document == null)
20         {
21             System.out.println("document is null");
22             return;
23         }
24         String f_name = null;
25         for(int i = 0;i<MutantsList.size();i++)
26         {
27             mu_List mu = MutantsList.get(i);
28             f_name = getMuFileName(mu.getOperator(),mu.getMuID());
29             String f_path = getAbsoluteMuFilePath(f_name);
30             firstOrdermuList.add(new String[]{f_name.substring(0,f_name.length()-5),
31 f_path});
32             PrintWriter out;
33             try {
34                 out = getPrintWriter(f_name);
35                 XMLWriter writer = mu.getWriter();
36                 writer.setWriter(out);
37                 writer.write(document);
38                 writer.flush();
39                 writer.close();
40             } catch (IOException e) {
41                 System.err.println("Fail to create file " + f_name);
42             }
43         }
44         MutantsList.clear();
45     }
46 }
47 package generate;
48 import java.util.List;
49 import com.ustb.bpel.view.Mainform;
50 public class GenerateFacade {
```

```
51     int algorithmIndex;
52     List<String[]> mutantsList;
53     public GenerateFacade(){ }
54     public GenerateFacade(int algorithmIndex)
55     {
56         this.algorithmIndex = algorithmIndex;
57     }
58     public void firstGenerate(){
59         FirstOrder first = new FirstOrder();
60         mutantsList = first.getFirstOrdermuList();
61         mutantsListPrint();
62     }
63     public void secondGenerate(){
64         SecondOrder second = new SecondOrder(algorithmIndex);
65         mutantsList = second.getSecondOrdermuList();
66         mutantsListPrint();
67     }
68     private void mutantsListPrint() {
69         for(int i = 0;i<mutantsList.size();i++)
70         {
71             System.out.println(mutantsList.get(i)[0]);
72             Mainform.MutantlogArea.append("Generatethe mutant :"+
73 + "*****"+mutantsList.get(i)[0]+"\\n");
74         }
75         System.out.println("\\n\\nThe total number of mutants is " + mutantsList.size());
76         Mainform.MutantlogArea.append("\\n\\nThe total number of mutants is " +
77 mutantsList.size()+"\\n");
78     }
79     public List<String[]> getMutantsList() {
80         return mutantsList;
81     }
82 }
83 package generate;
84 import java.io.IOException;
85 import org.dom4j.Element;
86 import org.dom4j.Node;
87 import parser.mu_List;
88 import utils.XMLWriter;
89 public class SecondOrder_Writer extends XMLWriter{
90     Node original1;
91     Node original2;
92     mu_List mu1;
93     mu_List mu2;
94     XMLWriter anoWriter;
95     public SecondOrder_Writer(mu_List mu1, mu_List mu2){
96         this.mu1 = mu1;
97         this.mu2 = mu2;
98         this.original1 = mu1.getNode();
99         this.original2 = mu2.getNode();
100     }
```

```
101     public void writeElement(Element element) throws IOException {
102         if(this.original1 == element )
103         {
104             this.anoWriter = mu1.getWriter();
105             anoWriter.setWriter(this.writer);
106             anoWriter.writeElement(element);
107         }
108         else if( this.original2 == element){
109
110             this.anoWriter = mu2.getWriter();
111             anoWriter.setWriter(this.writer);
112             anoWriter.writeElement(element);
113         }
114         else
115         {
116             super.writeElement(element);
117         }
118     }
119 }
120 package generate;
121 import java.io.IOException;
122 import java.io.PrintWriter;
123 import java.util.ArrayList;
124 import java.util.List;
125 import com.ustb.bpel.view.Mainform;
126 import parser.BPELMutator;
127 import parser.mu_List;
128 import utils.XMLWriter;
129 public class SecondOrder extends BPELMutator {
130     private int algorithm_index;
131     public List<String[]> secondOrdermuList = new ArrayList<String[]>();
132     mu_List mu1;
133     mu_List mu2;
134     public SecondOrder(int algorithm_index) {
135         this.algorithm_index = algorithm_index;
136         pre_process();
137     }
138     public List<String[]> getSecondOrdermuList() {
139         return secondOrdermuList;
140     }
141     public void pre_process() {
142         if (algorithm_index == 0) {
143             secondOrderGenerate_0();
144         } else if (algorithm_index == 1) {
145             secondOrderGenerate_1();
146         } else {
147         }
148     }
149     private void secondOrderGenerate_0() {
150         for (int i = 0; i < MutantsList.size() / 2; i++) {
```

```
151         int j = MutantsList.size() - i - 1;
152         this.mu1 = MutantsList.get(i);
153         this.mu2 = MutantsList.get(j);
154         OutputToFile();
155     }
156     if (MutantsList.size() % 2 != 0) {
157         this.mu1 = MutantsList.get(MutantsList.size() / 2);
158         this.mu2 = MutantsList.get((MutantsList.size() + 1) / 2);
159         OutputToFile();
160     }
161     MutantsList.clear();
162 }
163 private void secondOrderGenerate_1() {
164 }
165 private void OutputToFile() {
166     if (document == null) {
167         System.out.println("document is null");
168         return;
169     }
170     String f_name = null;
171     f_name = getMuFileName(mu1.getOperator(), mu1.getMuID(),
172         mu2.getOperator(), mu2.getMuID());
173     String f_path = getAbsoluteMuFilePath(f_name);
174     secondOrdermuList.add(new String[] {
175         f_name.substring(0, f_name.length() - 5), f_path });
176     PrintWriter out;
177     try {
178         out = getPrintWriter(f_name);
179         SecondOrder_Writer writer = new SecondOrder_Writer(this.mu1,
180             this.mu2);
181         writer.setWriter(out);
182         writer.write(document);
183         writer.flush();
184         writer.close();
185     } catch (IOException e) {
186         System.err.println("Fail to create file " + f_name);
187     }
188 }
189 }
190 package parser;
191 import java.io.File;
192 import java.io.FileWriter;
193 import java.io.IOException;
194 import java.io.PrintWriter;
195 import java.io.Writer;
196 import java.text.SimpleDateFormat;
197 import java.util.ArrayList;
198 import java.util.Date;
199 import java.util.List;
200 import org.dom4j.Document;
```

```
201 import org.dom4j.io.OutputFormat;
202 import parser.ConcreteVisitor;
203 import parser.Parser;
204 import parser.mu_List;
205 public class BPELMutator extends ConcreteVisitor {
206     public int muID = 0;
207     public Date now;
208     public final String mutantPath = "D:/Mutant";
209     public static String className = null;
210     public Document document;
211     File file;
212     protected static List<mu_List> MutantsList = new ArrayList<mu_List>();
213     public BPELMutator() {
214         this.document = Parser.getDocument();
215         File f = new File(mutantPath);
216         if (!f.exists()) {
217             f.mkdir();
218         }
219         String timeNow = getTimeNow();
220         file = new File(f,timeNow);
221         file.mkdir();
222     }
223     public BPELMutator(Document document) {
224         this.document = document;
225     }
226     public BPELMutator(Document document, String Path, String Name) {
227         this.document = document;
228         className = Name;
229     }
230     public static List<mu_List> getMutantsList() {
231         return MutantsList;
232     }
233     public String getTimeNow(){
234         now = new Date();
235         SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH: mm");
236         return df.format(now);
237     }
238     public String getClassName() {
239         Class cc = this.getClass();
240         return exclude(cc.getName(), cc.getPackage().getName());
241     }
242     public String exclude(String a, String b) {
243         return a.substring(b.length() + 1, a.length());
244     }
245     public String getMuantID() {
246         String str = getClassName() + "_" + this.muID;
247         return str;
248     }
249     public String getMuFileName(String op_name,int muID) {
250         String mu_filename = op_name + "_" + muID + ".bpel";
```

```
251         return mu_filename;
252     }
253     public String getAbsoluteMuFilePath(String f_name){
254         String muFilePath = file.getAbsolutePath() + "\\\" + f_name;
255         return muFilePath;
256     }
257     public String getMuFileName(String op_name1,int muID1,String op_name2,int muID2) {
258         String mu_filename = op_name1 + muID1 + "_" + op_name2 + muID2 + ".bpel";
259         return mu_filename;
260     }
261     public PrintWriter getPrintWriter(String f_name) throws IOException {
262         File outfile = new File(file, f_name);
263         FileWriter fout = new FileWriter(outfile);
264         PrintWriter out = new PrintWriter(fout);
265         return out;
266     }
267 }
268 package parser;
269 import org.dom4j.Node;
270 import org.dom4j.Visitor;
271 public interface BPELVisitor extends Visitor{
272     public void visit(Node node);
273     public void visit(String s);
274 }
275 package parser;
276 import org.dom4j.Attribute;
277 import org.dom4j.CDATA;
278 import org.dom4j.Comment;
279 import org.dom4j.Document;
280 import org.dom4j.DocumentType;
281 import org.dom4j.Element;
282 import org.dom4j.Entity;
283 import org.dom4j.Namespace;
284 import org.dom4j.Node;
285 import org.dom4j.ProcessingInstruction;
286 import org.dom4j.Text;
287 public class ConcreteVisitor implements BPELVisitor{
288     public void visit(Document document) {
289     }
290     public void visit(DocumentType documentType) {
291     }
292     public void visit(Element node) {
293     }
294     public void visit(Attribute node) {
295     }
296     public void visit(CDATA node) {
297     }
298     public void visit(Comment node) {
299     }
300     public void visit(Entity node) {
```

```
301     }
302     public void visit(Namespace namespace) {
303     }
304     public void visit(ProcessingInstruction node) {
305     }
306     public void visit(Text node) {
307     }
308     public void visit(Node node){
309     }
310     public void visit(String s){
311     }
312 }
313 package parser;
314 import utils.XMLWriter;
315 import org.dom4j.Node;
316 public class mu_List {
317     private String operator;
318     private Node node;
319     private XMLWriter writer;
320     private int muID;
321     public mu_List(String operator, Node node, XMLWriter writer, int muID){
322         this.operator = operator;
323         this.node = node;
324         this.writer = writer;
325         this.muID = muID;
326     }
327     public String getOperator() {
328         return operator;
329     }
330     public void setOperator(String operator) {
331         this.operator = operator;
332     }
333     public Node getNode() {
334         return node;
335     }
336     public void setNode(Node node) {
337         this.node = node;
338     }
339     public int getMuID() {
340         return muID;
341     }
342     public void setMuID(int muID) {
343         this.muID = muID;
344     }
345     public XMLWriter getWriter() {
346         return writer;
347     }
348 }
349 package parser;
350 import java.util.List;
```

```
351 import org.dom4j.Document;
352 import utils.XMLReader;
353 public class Parser{
354     private static String filePath;
355     private static Document document;
356     private List<String> operatorList;
357 public Parser(String filePath, List<String> operatorList){
358     System.out.println("*****start Parser*****");
359     this.filePath = filePath;
360     this.document = XMLReader.openXMLFile(this.filePath);
361     this.operatorList = operatorList;
362     this.parserTree();
363 }
364 public void parserTree(){
365     System.out.println("***parserTree***");
366     String operator;
367     ConcreteVisitor visitor;
368     for (int i = 0; i < operatorList.size(); i++) {
369         operator = operatorList.get(i);
370         try {
371             Class<?> c = Class.forName(operator);
372             try {
373                 visitor = (ConcreteVisitor) c.newInstance();
374                 document.accept(visitor);
375             } catch (Exception e) {
376                 e.printStackTrace();
377             }
378         } catch(ClassNotFoundException e){
379             e.printStackTrace();
380         }
381     }
382 }
383 public static Document getDocument() {
384     return document;
385 }
386 }
387 package run;
388 import generate.GenerateFacade;
389 import java.util.List;
390 import parser.BPELMutator;
391 import parser.Parser;
392 import parser.mu_List;
393 public class main
394 {
395     private static int algorithm = -1;
396     public static void setAlgorithm(int al) {
397         algorithm = al;
398     }
399     public static List<String[]> test(String path,List<String> operatorList,int flag)
400     {
```



```
401         new Parser(path,operatorList);
402         GenerateFacade generateFacade = null;
403         if(flag == 0){
404             generateFacade = new GenerateFacade();
405             generateFacade.firstGenerate();
406         }
407         else if(flag == 1)
408         {
409             System.out.println(algorithm);
410             generateFacade = new GenerateFacade(algorithm);
411             generateFacade.secondGenerate();
412         }
413         List<String[]> mutantsList = generateFacade.getMutantsList();
414         return mutantsList;
415     }
416 }
417 package utils;
418 import java.io.File;
419 import org.dom4j.Document;
420 import org.dom4j.io.SAXReader;
421 public class XMLReader {
422     public static Document openXMLFile(String filePath) {
423         Document document = null;
424         SAXReader reader = new SAXReader();
425         try {
426             File file = new File(filePath);
427             document = reader.read(file);
428         } catch (Exception e) {
429             e.printStackTrace();
430         }
431         return document;
432     }
433 }
434 package operator;
435 import java.io.PrintWriter;
436 import org.dom4j.Attribute;
437 import org.dom4j.Element;
438 public class ACI_Writer extends attReplace_Writer {
439     public ACI_Writer(Element original, Attribute attribute) {
440         super(original, attribute);
441     }
442     public ACI_Writer(PrintWriter out) {
443         super(out);
444     }
445 }
446 package operator;
447 import parser.BPELMutator;
448 import parser.mu_List;
449 public class ACI extends BPELMutator {
450     private Element element;
```

```
451     private Attribute previous;
452     private static int recordNo;
453     private int flag;
454     public ACI() {
455         super();
456         System.out.println("****ACI");
457         this.recordNo = 1;
458         this.flag = 0;
459     }
460     public void visit(Attribute attribute) {
461         if (attribute.getName().equals("createInstance")
462             && attribute.getValue().equals("yes")) {
463             System.out.println("*****find createInstance");
464             if (recordNo < 2) {
465                 recordNo++;
466                 this.previous = attribute;
467                 flag = 1;
468                 return;
469             }
470             if (flag == 1) {
471                 muID++;
472                 this.element = previous.getParent();
473                 mu_List mu_List = new mu_List("ACI", previous, new ACI_Writer(
474                     this.element, previous), muID);
475                 MutantsList.add(mu_List);
476                 flag = 0;
477             }
478             muID++;
479             this.element = attribute.getParent();
480             mu_List mu_List = new mu_List("ACI", attribute, new ACI_Writer(
481                 this.element, attribute), muID);
482             MutantsList.add(mu_List);
483         } else
484             super.visit(attribute);
485     }
486 }
487 package operator;
488 import java.io.PrintWriter;
489 import org.dom4j.Element;
490 public class AEL_Writer extends Remove_writer {
491     public AEL_Writer(Element original) {
492         super(original);
493     }
494 }
495 package operator;
496 import org.dom4j.Element;
497 import parser.BPELMutator;
498 import parser.mu_List;
499 public class AEL extends BPELMutator {
500     public AEL() {
```

```

501         super();
502         System.out.println("***AEL");
503     }
504     public void visit(Element element) {
505         if (element.getName().equals("assign")
506             || element.getName().equals("invoke")
507             || element.getName().equals("reply")
508             || element.getName().equals("throw")
509             || element.getName().equals("wait")
510             || element.getName().equals("exit")
511             || element.getName().equals("rethrow")
512             || element.getName().equals("scope")
513             || element.getName().equals("flow")
514             || element.getName().equals("switch")
515             || element.getName().equals("if")
516             || element.getName().equals("while")
517             || element.getName().equals("repeatuntil")
518             || element.getName().equals("foreach")
519             || element.getName().equals("pick")) {
520             System.out.println("*****find " + element.getName());
521             muID++;
522             mu_List mu_List = new mu_List("AEL", element, new AEL_Writer(
523                 element), muID);
524             MutantsList.add(mu_List);
525         } else if (element.getName().equals("sequence")) {
526             Element farElement = element.getParent();
527             if (!farElement.getName().equals("process")) {
528                 muID++;
529                 mu_List mu_List = new mu_List("AEL", element, new AEL_Writer(
530                     element), muID);
531                 MutantsList.add(mu_List);
532             }
533         } else
534             super.visit(element);
535     }
536 }
537 package operator;
538 import java.io.PrintWriter;
539 import org.dom4j.Attribute;
540 import org.dom4j.Element;
541 import utils.XMLWriter;
542 public class AFP_Writer extends attReplace_Writer{
543     public AFP_Writer(Element original, Attribute attribute) {
544         super(original, attribute);
545     }
546     public AFP_Writer(PrintWriter out) {
547         super(out);
548     }
549 }
550 package operator;

```

```
551 import org.dom4j.Attribute;
552 import org.dom4j.Element;
553 import parser.BPELMutator;
554 import parser.mu_List;
555 public class AFP extends BPELMutator {
556     private Element att_element;
557     public AFP() {
558         super();
559         System.out.println("***AFP");
560     }
561     public void visit(Attribute attribute) {
562         if (attribute.getName().equals("parallel")
563             && attribute.getValue().equals("no")) {
564             this.att_element = attribute.getParent();
565             if (this.att_element.getName().equals("forEach")) {
566                 System.out.println("*****find parallel");
567                 muID++;
568                 mu_List mu_List = new mu_List("AFP", attribute, new AFP_Writer(
569                     this.att_element, attribute), muID);
570                 MutantsList.add(mu_List);
571             }
572         } else
573             super.visit(attribute);
574     }
575 }
576 package operator;
577 import java.io.PrintWriter;
578 import org.dom4j.Element;
579 public class AIE_Writer extends Remove_writer{
580     public AIE_Writer(Element original) {
581         super(original);
582     }
583     public AIE_Writer(PrintWriter out){
584         super(out);
585     }
586 }
587 package operator;
588 import org.dom4j.Element;
589 import parser.BPELMutator;
590 import parser.mu_List;
591 public class AIE extends BPELMutator {
592     public AIE() {
593         super();
594         System.out.println("***AIE");
595     }
596     public void visit(Element element) {
597         if (element.getName().equals("elseif")
598             || element.getName().equals("else")) {
599             System.out.println("*****find if");
600             muID++;
```

```

601         mu_List mu_List = new mu_List("AIE", element, new AIE_Writer(
602             element), muID);
603         MutantsList.add(mu_List);
604     } else
605         super.visit(element);
606     }
607 }
608 package operator;
609 import java.io.PrintWriter;
610 import org.dom4j.Attribute;
611 import org.dom4j.Element;
612 public class AIS_Writer extends attReplace_Writer{
613     public AIS_Writer(Element original, Attribute attribute) {
614         super(original, attribute);
615     }
616     public AIS_Writer(PrintWriter out) {
617         super(out);
618     }
619 }
620 package operator;
621 import org.dom4j.Attribute;
622 import org.dom4j.Element;
623 import parser.BPELMutator;
624 import parser.mu_List;
625 public class AIS extends BPELMutator {
626     private Element att_element;
627     public AIS() {
628         super();
629         System.out.println("***AIS");
630     }
631     public void visit(Attribute attribute) {
632         if (attribute.getName().equals("isolated")
633             && attribute.getValue().equals("yes")) {
634             this.att_element = attribute.getParent();
635             if (this.att_element.getName().equals("scope")) {
636                 System.out.println("*****find isolated");
637                 muID++;
638                 mu_List mu_List = new mu_List("AIS", attribute, new AIS_Writer(
639                     this.att_element, attribute), muID);
640                 MutantsList.add(mu_List);
641             }
642         } else
643             super.visit(attribute);
644     }
645 }
646 package operator;
647 import java.io.PrintWriter;
648 import org.dom4j.Element;
649 public class AJC_Writer extends Remove_writer{
650     public AJC_Writer(Element original) {

```

```
651         super(original);
652     }
653     public AJC_Writer(PrintWriter out) {
654         super(out);
655     }
656 }
657 package operator;
658 import org.dom4j.Element;
659 import parser.BPELMutator;
660 import parser.mu_List;
661 public class AJC extends BPELMutator {
662     public AJC() {
663         super();
664         System.out.println("***AJC");
665     }
666     public void visit(Element element) {
667         if (element.getName().equals("joinCondition")) {
668             System.out.println("*****find joinCondition");
669             muID++;
670             mu_List mu_List = new mu_List("AJC", element, new AJC_Writer(
671                 element), muID);
672             MutantsList.add(mu_List);
673         } else
674             super.visit(element);
675     }
676 }
677 package operator;
678 import java.io.PrintWriter;
679 import org.dom4j.Element;
680 public class APA_Writer extends Remove_writer{
681     public APA_Writer(Element original) {
682         super(original);
683     }
684     public APA_Writer(PrintWriter out) {
685         super(out);
686     }
687 }
688 package operator;
689 import org.dom4j.Element;
690 import parser.BPELMutator;
691 import parser.mu_List;
692 public class APA extends BPELMutator {
693     public APA() {
694         super();
695         System.out.println("***APA");
696     }
697     public void visit(Element element) {
698         if (element.getName().equals("onAlarm")) {
699             System.out.println("*****find onAlarm");
700             muID++;
```

```

701         mu_List mu_List = new mu_List("APA", element, new APA_Writer(
702             element), muID);
703         MutantsList.add(mu_List);
704     } else
705         super.visit(element);
706     }
707 }
708 package operator;
709 import java.io.PrintWriter;
710 import org.dom4j.Element;
711 public class APM_Writer extends Remove_writer{
712     public APM_Writer(Element original) {
713         super(original);
714     }
715     public APM_Writer(PrintWriter out) {
716         super(out);
717     }
718 }
719 package operator;
720 import java.util.List;
721 import org.dom4j.Element;
722 import parser.BPELMutator;
723 import parser.mu_List;
724 public class APM extends BPELMutator {
725     public APM() {
726         super();
727         System.out.println("***APM");
728     }
729     public void visit(Element element) {
730         if (element.getName().equals("pick")) {
731             System.out.println("*****find pick");
732             List<Element> onMessageList = element.elements("onMessage");
733             if (onMessageList.size() >= 2) {
734                 for (int i = 0; i < onMessageList.size(); i++) {
735                     Element onMessageElement = onMessageList.get(i);
736                     muID++;
737                     mu_List mu_List = new mu_List("APM", onMessageElement,
738                         new APM_Writer(onMessageElement), muID);
739                     MutantsList.add(mu_List);
740                 }
741             }
742         } else
743             super.visit(element);
744     }
745 }
746 package operator;
747 import java.io.IOException;
748 import java.io.PrintWriter;
749 import org.dom4j.Attribute;
750 import org.dom4j.Comment;

```

```
751 import org.dom4j.Element;
752 import org.dom4j.Namespace;
753 import org.dom4j.Node;
754 import parser.mu_List;
755 import utils.XMLWriter;
756 public class ASF_Writer extends XMLWriter {
757     private Element original;
758     public ASF_Writer(Element original) {
759         this.original = original;
760     }
761     public ASF_Writer(PrintWriter out) {
762         super(out);
763     }
764     public void setMutant(Element original) {
765         this.original = original;
766     }
767     public void writeElement(Element element) throws IOException {
768         if (element == original) {
769             int size = element.nodeCount();
770             String qualifiedName = "flow";
771             writePrintln();
772             indent();
773             writer.write("<");
774             writer.write(qualifiedName);
775             int previouslyDeclaredNamespaces = namespaceStack.size();
776             Namespace ns = element.getNamespace();
777             if (isNamespaceDeclaration(ns)) {
778                 namespaceStack.push(ns);
779                 writeNamespace(ns);
780             }
781             boolean textOnly = true;
782             for (int i = 0; i < size; i++) {
783                 Node node = element.node(i);
784                 if (node instanceof Namespace) {
785                     Namespace additional = (Namespace) node;
786                     if (isNamespaceDeclaration(additional)) {
787                         namespaceStack.push(additional);
788                         writeNamespace(additional);
789                     }
790                 } else if (node instanceof Element) {
791                     textOnly = false;
792                 } else if (node instanceof Comment) {
793                     textOnly = false;
794                 }
795             }
796             writeAttributes(element);
797             lastOutputNodeType = Node.ELEMENT_NODE;
798             if (size <= 0) {
799                 writeEmptyElementClose(qualifiedName);
800             } else {
```



```
801         writer.write(">");
802         if (textOnly) {
803             writeElementContent(element);
804         } else {
805             ++indentLevel;
806             writeElementContent(element);
807             --indentLevel;
808             writePrintln();
809             indent();
810         }
811         writer.write("</");
812         writer.write(qualifiedName);
813         writer.write(">");
814     }
815     while (namespaceStack.size() > previouslyDeclaredNamespaces) {
816         namespaceStack.pop();
817     }
818     lastOutputNodeType = Node.ELEMENT_NODE;
819 } else {
820     super.writeElement(element);
821 }
822 }
823 }
824 package operator;
825 import org.dom4j.Element;
826 import parser.BPELMutator;
827 import parser.mu_List;
828 public class ASF extends BPELMutator {
829     public ASF() {
830         super();
831         System.out.println("****ASF");
832     }
833     public void visit(Element element) {
834         if (element.getName().equals("sequence")) {
835             System.out.println("*****find asf sequence");
836             muID++;
837             mu_List mu_List = new mu_List("ASF", element, new ASF_Writer(
838                 element), muID);
839             MutantsList.add(mu_List);
840         }
841     } else
842         super.visit(element);
843 }
844 package operator;
845 import java.io.IOException;
846 import java.io.PrintWriter;
847 import org.dom4j.Comment;
848 import org.dom4j.Element;
849 import org.dom4j.Namespace;
850 import org.dom4j.Node;
```

```
851 import utils.XMLWriter;
852 public class ASI_Writer extends XMLWriter{
853     private Element original;
854     private Element child1;
855     private Element child2;
856     public ASI_Writer(Element original, Element child1, Element child2) {
857         this.original = original;
858         this.child1 = child1;
859         this.child2 = child2;
860     }
861     public ASI_Writer(PrintWriter out) {
862         super(out);
863     }
864     public void setMutant(Element original, Element child1, Element child2) {
865         this.original = original;
866         this.child1 = child1;
867         this.child2 = child2;
868     }
869     public void writeElement(Element element) throws IOException {
870         Element parElement = element.getParent();
871         if (parElement == original) {
872             if(element == child1){
873                 element = child2;
874             }
875             else if(element == child2)
876             {
877                 element = child1;
878             }
879             int size = element.nodeCount();
880             String qualifiedName = element.getQualifiedName();
881             writePrintln();
882             indent();
883             writer.write("<");
884             writer.write(qualifiedName);
885             int previouslyDeclaredNamespaces = namespaceStack.size();
886             Namespace ns = element.getNamespace();
887             if (isNamespaceDeclaration(ns)) {
888                 namespaceStack.push(ns);
889                 writeNamespace(ns);
890             }
891             boolean textOnly = true;
892             for (int i = 0; i < size; i++) {
893                 Node node = element.node(i);
894                 if (node instanceof Namespace) {
895                     Namespace additional = (Namespace) node;
896                     if (isNamespaceDeclaration(additional)) {
897                         namespaceStack.push(additional);
898                         writeNamespace(additional);
899                     }
900                 } else if (node instanceof Element) {
```

```

901         textOnly = false;
902     } else if (node instanceof Comment) {
903         textOnly = false;
904     }
905 }
906 writeAttributes(element);
907 lastOutputNodeType = Node.ELEMENT_NODE;
908 if (size <= 0) {
909     writeEmptyElementClose(qualifiedName);
910 } else {
911     writer.write(">");
912     if (textOnly) {
913         writeElementContent(element);
914     } else {
915         ++indentLevel;
916         writeElementContent(element);
917         --indentLevel;
918         writePrintln();
919         indent();
920     }
921     writer.write("</");
922     writer.write(qualifiedName);
923     writer.write(">");
924 }
925 while (namespaceStack.size() > previouslyDeclaredNamespaces) {
926     namespaceStack.pop();
927 }
928 lastOutputNodeType = Node.ELEMENT_NODE;
929 } else {
930     super.writeElement(element);
931 }
932 }
933 }
934 }
935 package operator;
936 import java.util.List;
937 import org.dom4j.Element;
938 import parser.BPELMutator;
939 import parser.mu_List;
940 public class ASI extends BPELMutator {
941     public ASI() {
942         super();
943         System.out.println("***ASI");
944     }
945     public void visit(Element element) {
946         if (element.getName().equals("sequence")) {
947             System.out.println("*****find asi sequence");
948             List<Element> childList = element.elements();
949             for (int i = 0; i < childList.size(); i++) {
950                 Element ele1 = childList.get(i);

```

```

951         for (int j = i + 1; j < childList.size(); j++) {
952             Element ele2 = childList.get(j);
953             System.out.println("***exchange two activity:"
954                 + ele1.getName() + " " + ele2.getName());
955             muID++;
956             mu_List mu_List = new mu_List("ASI", element,
957                 new ASI_Writer(element, ele1, ele2), muID);
958             MutantsList.add(mu_List);
959         }
960     }
961 } else
962     super.visit(element);
963 }
964 }
965 package operator;
966 import java.io.IOException;
967 import java.io.PrintWriter;
968 import org.dom4j.Attribute;
969 import org.dom4j.Element;
970 import utils.XMLWriter;
971 public class attReplace_Writer extends XMLWriter {
972     private Element original;
973     private Attribute attribute;
974     public attReplace_Writer(Element original, Attribute attribute) {
975         this.original = original;
976         this.attribute = attribute;
977     }
978     public attReplace_Writer(PrintWriter out) {
979         super(out);
980     }
981     public void setMutant(Element original, Attribute attribute) {
982         this.original = original;
983         this.attribute = attribute;
984     }
985     public void writeAttributes(Element element) throws IOException {
986         if (original == element) {
987             Element copy = element.createCopy();
988             Attribute attribute = copy.attribute(this.attribute.getName());
989             if (attribute.getValue().equals("yes")) {
990                 attribute.setValue("no");
991             } else if (attribute.getValue().equals("no")) {
992                 attribute.setValue("yes");
993             } else {
994                 attribute.setValue(this.attribute.getValue());
995             }
996             element = copy;
997         }
998         super.writeAttributes(element);
999     }
1000 }

```

```
1001 package operator;
1002 import java.io.IOException;
1003 import java.io.PrintWriter;
1004 import org.dom4j.Attribute;
1005 import org.dom4j.Comment;
1006 import org.dom4j.Document;
1007 import org.dom4j.DocumentType;
1008 import org.dom4j.Element;
1009 import org.dom4j.Entity;
1010 import org.dom4j.Namespace;
1011 import org.dom4j.Node;
1012 import org.dom4j.ProcessingInstruction;
1013 import org.dom4j.Text;
1014 import utils.XMLWriter;
1015 public class AWR_Writer extends XMLWriter {
1016     private Element original;
1017     private Element conditonElement = null;
1018     public AWR_Writer(Element original) {
1019         this.original = original;
1020     }
1021     public AWR_Writer(PrintWriter out) {
1022         super(out);
1023     }
1024     public void setMutant(Element original) {
1025         this.original = original;
1026     }
1027     public void writeElement(Element element) throws IOException {
1028         if (element == original) {
1029             int size = element.nodeCount();
1030             String qualifiedName = null;
1031             if (element.getName().equals("repeatUntil")) {
1032                 qualifiedName = "while";
1033             } else if (element.getName().equals("while")) {
1034                 qualifiedName = "repeatUntil";
1035             }
1036             writePrintln();
1037             indent();
1038             writer.write("<");
1039             writer.write(qualifiedName);
1040             int previouslyDeclaredNamespaces = namespaceStack.size();
1041             Namespace ns = element.getNamespace();
1042             if (isNamespaceDeclaration(ns)) {
1043                 namespaceStack.push(ns);
1044                 writeNamespace(ns);
1045             }
1046             boolean textOnly = true;
1047             for (int i = 0; i < size; i++) {
1048                 Node node = element.node(i);
1049                 if (node instanceof Namespace) {
1050                     Namespace additional = (Namespace) node;
```

```

1051         if (isNamespaceDeclaration(additional)) {
1052             namespaceStack.push(additional);
1053             writeNamespace(additional);
1054         }
1055     } else if (node instanceof Element) {
1056         textOnly = false;
1057     } else if (node instanceof Comment) {
1058         textOnly = false;
1059     }
1060 }
1061 writeAttributes(element);
1062 lastOutputNodeType = Node.ELEMENT_NODE;
1063 if (size <= 0) {
1064     writeEmptyElementClose(qualifiedName);
1065 } else {
1066     writer.write(">");
1067     if (textOnly) {
1068         writeElementContent(element);
1069     } else {
1070         ++indentLevel;
1071         writeElementContent(element);
1072         --indentLevel;
1073         writePrintln();
1074         indent();
1075     }
1076     writer.write("</");
1077     writer.write(qualifiedName);
1078     writer.write(">");
1079 }
1080 while (namespaceStack.size() > previouslyDeclaredNamespaces) {
1081     namespaceStack.pop();
1082 }
1083 lastOutputNodeType = Node.ELEMENT_NODE;
1084 } else {
1085     super.writeElement(element);
1086 }
1087 }
1088 protected void writeElementContent(Element element) throws IOException {
1089     if (element == original) {
1090         boolean trim = format.isTrimText();
1091         boolean oldPreserve = preserve;
1092         if (trim) {
1093             preserve = isElementSpacePreserved(element);
1094             trim = !preserve;
1095         }
1096         for (int i = 0, size = element.nodeCount(); i < size; i++) {
1097             Node node = element.node(i);
1098             if (node.getNodeType() == Node.ELEMENT_NODE
1099                 && node.getName().equals("condition")) {
1100                 if (element.getName().equals("while")) {

```

[illegible]

```

1151         lastTextChar = txt
1152             .charAt(txt.length() - 1);
1153     }
1154     if (Character.isWhitespace(lastTextChar)) {
1155         writer.write(PAD_TEXT);
1156     }
1157 }
1158 lastTextNode = null;
1159 }
1160 textOnly = false;
1161 writeNode(node);
1162 }
1163 }
1164 }
1165 if (element.getName().equals("while")) {
1166     writeNode(conditonElement);
1167 }
1168 if (lastTextNode != null) {
1169     if (!textOnly && format.isPadText()) {
1170         char firstChar = 'a';
1171         if (buff != null) {
1172             firstChar = buff.charAt(0);
1173         } else {
1174             firstChar = lastTextNode.getText().charAt(0);
1175         }
1176         if (Character.isWhitespace(firstChar)) {
1177             writer.write(PAD_TEXT);
1178         }
1179     }
1180     if (buff != null) {
1181         writeString(buff.toString());
1182         buff = null;
1183     } else {
1184         writeString(lastTextNode.getText());
1185     }
1186     lastTextNode = null;
1187 }
1188 } else {
1189     Node lastTextNode = null;
1190     for (int i = 0, size = element.nodeCount(); i < size; i++) {
1191         Node node = element.node(i);
1192         if (!node.getName().equals("condition")) {
1193             if (node instanceof Text) {
1194                 writeNode(node);
1195                 lastTextNode = node;
1196             } else {
1197                 if ((lastTextNode != null) && format.isPadText()) {
1198
1199                     String txt = lastTextNode.getText();
1200                     char lastTextChar = txt

```



```

1201         .charAt(txt.length() - 1);
1202         if (Character.isWhitespace(lastTextChar)) {
1203             writer.write(PAD_TEXT);
1204         }
1205     }
1206     writeNode(node);
1207     lastTextNode = null;
1208 }
1209 }
1210 }
1211     if (element.getName().equals("while")) {
1212         writeNode(conditonElement);
1213     }
1214 }
1215     preserve = oldPreserve;
1216 }
1217 else {
1218     super.writeElementContent(element);
1219 }
1220 }
1221 }
1222 package operator;
1223 import org.dom4j.Element;
1224 import parser.BPELMutator;
1225 import parser.mu_List;
1226 public class AWR extends BPELMutator {
1227     public AWR() {
1228         super();
1229         System.out.println("***AWR");
1230     }
1231     public void visit(Element element) {
1232         if (element.getName().equals("repeatUntil")
1233             || element.getName().equals("while")) {
1234             System.out.println("*****find " + element.getName());
1235             muID++;
1236             mu_List mu_List = new mu_List("AWR", element, new AWR_Writer(
1237                 element), muID);
1238             MutantsList.add(mu_List);
1239         } else
1240             super.visit(element);
1241     }
1242 }
1243 package operator;
1244 import org.dom4j.Element;
1245 public class CCO_Writer extends exReplace_Writer{
1246     public CCO_Writer(Element original, String originalOp, String replaceOp) {
1247         super(original, originalOp, replaceOp);
1248     }
1249 }
1250 package operator;

```

```

1251 import java.util.Arrays;
1252 import java.util.List;
1253 import org.dom4j.Element;
1254 import parser.BPELMutator;
1255 import parser.mu_List;
1256 public class CCO extends BPELMutator {
1257     List<String> mutateList = Arrays.asList("true()", "false()");
1258     public CCO() {
1259         super();
1260         System.out.println("****CCO");
1261     }
1262     public void visit(Element element) {
1263         if (element.getName().equals("condition"))
1264             || element.getName().equals("transitionCondition")
1265             || element.getName().equals("joinCondition")) {
1266             System.out.println("*****find " + element.getName());
1267             String expression = element.getText();
1268             if (expression.contains("and") || expression.contains("or")) {
1269                 String s[] = expression.split(" *(and|or) *");
1270                 for (int j = 0; j < mutateList.size(); j++) {
1271                     String replaceEx = mutateList.get(j);
1272                     String tempEx = expression;
1273                     for (int i = 0; i < s.length; i++) {
1274                         tempEx = tempEx.replace(s[i], replaceEx);
1275                     }
1276                     muID++;
1277                     mu_List mu_List = new mu_List("CCO", element,
1278                         new CCO_Writer(element, expression, tempEx), muID);
1279                     MutantsList.add(mu_List);
1280                 }
1281             } else {
1282                 for (int i = 0; i < mutateList.size(); i++) {
1283                     String replaceEx = mutateList.get(i);
1284                     muID++;
1285                     mu_List mu_List = new mu_List("CCO", element,
1286                         new CCO_Writer(element, expression, replaceEx),
1287                         muID);
1288                     MutantsList.add(mu_List);
1289                 }
1290             }
1291         } else
1292             super.visit(element);
1293     }
1294 }
1295 package operator;
1296 import org.dom4j.Element;
1297 public class CDC_Writer extends exReplace_Writer{
1298     public CDC_Writer(Element original, String originalOp, String replaceOp) {
1299         super(original, originalOp, replaceOp);
1300     }

```

```

1301 }
1302 package operator;
1303 import java.util.Arrays;
1304 import java.util.List;
1305 import org.dom4j.Element;
1306 import parser.BPELMutator;
1307 import parser.mu_List;
1308 public class CDC extends BPELMutator {
1309     List<String> mutateList = Arrays.asList("true()", "false()");
1310     public CDC() {
1311         super();
1312         System.out.println("***CDC");
1313     }
1314     public void visit(Element element) {
1315         if (element.getName().equals("condition")
1316             || element.getName().equals("transitionCondition")
1317             || element.getName().equals("joinCondition")) {
1318             System.out.println("*****find " + element.getName());
1319             String expression = element.getText();
1320             for (int i = 0; i < mutateList.size(); i++) {
1321                 String replaceEx = mutateList.get(i);
1322                 muID++;
1323                 mu_List mu_List = new mu_List("CDC", element, new CDC_Writer(
1324                     element, expression, replaceEx), muID);
1325                 MutantsList.add(mu_List);
1326             }
1327             if (expression.contains("and") || expression.contains("or")) {
1328                 String s[] = expression.split(" *(and|or) *");
1329                 for (int j = 0; j < mutateList.size(); j++) {
1330                     String replaceEx = mutateList.get(j);
1331                     String tempEx = expression;
1332                     for (int i = 0; i < s.length; i++) {
1333                         tempEx = tempEx.replace(s[i], replaceEx);
1334                     }
1335                     muID++;
1336                     mu_List mu_List = new mu_List("CDC", element,
1337                         new CDC_Writer(element, expression, tempEx), muID);
1338                     MutantsList.add(mu_List);
1339                 }
1340             }
1341             } else
1342             super.visit(element);
1343     }
1344 }
1345 package operator;
1346 import org.dom4j.Element;
1347 public class CDE_Writer extends exReplace_Writer{
1348     public CDE_Writer(Element original, String originalOp, String replaceOp) {
1349         super(original, originalOp, replaceOp);
1350     }

```

```
1351 }
1352 package operator;
1353 import java.util.Arrays;
1354 import java.util.List;
1355 import org.dom4j.Element;
1356 import parser.BPELMutator;
1357 import parser.mu_List;
1358 public class CDE extends BPELMutator{
1359     List<String> mutateList = Arrays.asList("true()", "false()");
1360     public CDE() {
1361         super();
1362         System.out.println("***CDE");
1363     }
1364     public void visit(Element element) {
1365         if (element.getName().equals("condition")
1366             || element.getName().equals("transitionCondition")
1367             || element.getName().equals("joinCondition")) {
1368             System.out.println("*****find " + element.getName());
1369             String expression = element.getText();
1370             for(int i = 0;i<mutateList.size();i++){
1371                 String replaceEx = mutateList.get(i);
1372                 muID++;
1373                 mu_List mu_List = new mu_List("CDE", element, new CDE_Writer(
1374                     element, expression, replaceEx), muID);
1375                 MutantsList.add(mu_List);
1376             }
1377         } else
1378             super.visit(element);
1379     }
1380 }
1381 package operator;
1382 import java.io.IOException;
1383 import org.dom4j.Element;
1384 import utils.XMLWriter;
1385 public class CFA_Writer extends XMLWriter {
1386     private Element original;
1387     public CFA_Writer(Element original) {
1388         this.original = original;
1389     }
1390     public void writeElement(Element element) throws IOException {
1391         if (element == original) {
1392             String qualifiedName = "exit";
1393             writePrintln();
1394             indent();
1395             writer.write("<");
1396             writer.write(qualifiedName);
1397             writer.write(" ");
1398             writer.write("/");
1399             writer.write(">");
1400         } else {
```

```
1401         super.writeElement(element);
1402     }
1403 }
1404 }
1405 package operator;
1406 import org.dom4j.Element;
1407 import parser.BPELMutator;
1408 import parser.mu_List;
1409 public class CFA extends BPELMutator{
1410     public CFA() {
1411         super();
1412         System.out.println("***CFA");
1413     }
1414     public void visit(Element element) {
1415         if (element.getName().equals("assign")
1416             || element.getName().equals("invoke")
1417             || element.getName().equals("reply")
1418             || element.getName().equals("throw")
1419             || element.getName().equals("wait")
1420             || element.getName().equals("exit")
1421             || element.getName().equals("rethrow")
1422             || element.getName().equals("scope")
1423             || element.getName().equals("flow")
1424             || element.getName().equals("switch")
1425             || element.getName().equals("if")
1426             || element.getName().equals("while")
1427             || element.getName().equals("repeatuntil")
1428             || element.getName().equals("foreach")
1429             || element.getName().equals("pick")) {
1430             System.out.println("*****find " + element.getName());
1431             muID++;
1432             mu_List mu_List = new mu_List("CFA", element, new CFA_Writer(
1433                 element), muID);
1434             MutantsList.add(mu_List);
1435         } else if (element.getName().equals("sequence")) {
1436             Element farElement = element.getParent();
1437             if (!farElement.getName().equals("process")) {
1438                 muID++;
1439                 mu_List mu_List = new mu_List("CFA", element, new CFA_Writer(
1440                     element), muID);
1441                 MutantsList.add(mu_List);
1442             }
1443         } else
1444             super.visit(element);
1445     }
1446 }
1447 package alan.program.Configuration;
1448 public class Workspace {
1449     public static final String TOMCAT_HOME = " E:\\tool\\apache-tomcat\\";
1450     public static final String WORKSPACE = TOMCAT_HOME
```

```
1451         + "webapps/mt4ws/workspace/";
1452     private String userName;
1453     private String path;
1454     public String getPath() {
1455         return WORKSPACE + userName + "/";
1456     }
1457     public void setUserName(String userName) {
1458         this.userName = userName;
1459     }
1460 }
1461 package alan.program.Configuration;
1462 import java.io.File;
1463 import java.util.ArrayList;
1464 import java.util.List;
1465 import org.dom4j.Attribute;
1466 import org.dom4j.Document;
1467 import org.dom4j.DocumentException;
1468 import org.dom4j.Element;
1469 import org.dom4j.io.SAXReader;
1470 import org.jaxen.function.LastFunction;
1471 import alan.program.Configuration.*;
1472 public class WSDLParser {
1473     public static Configuration configuration;
1474     public static String RunConfiguration(String bpelfilepath) {
1475         String path=bpelfilepath.substring(0,bpelfilepath.lastIndexOf("\\"));
1476         List data = new ArrayList();
1477         data = getData(path, data);
1478         return null;
1479     }
1480     private static List getData(String path, List data) {
1481         File f = new File(path);
1482         if (f.isDirectory()) {
1483             File[] fs = f.listFiles();
1484             for (int i = 0; i < fs.length; i++) {
1485                 data = getData(fs[i].getPath(), data);
1486             }
1487         } else if (f.getName().endsWith("config.xml")) {
1488             data.add(f.getName());
1489             parseXml(f);
1490         }
1491         return data;
1492     }
1493     private static Configuration parseXml(File f) {
1494         SAXReader reader = new SAXReader();
1495         configuration = new Configuration();
1496         Document document;
1497     try {
1498         document = reader.read(f);
1499         Element root = document.getRootElement();
1500         Element clientElement = root.element("client");
```