```
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("\nThe total number of mutants is " + mutantsList.size());
76                  Mainform.MutantlogArea.append("\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         }
```

```
        public void writeElement(Element element) throws IOException {
            if(this.original1 == element )
            {
                this.anoWriter = mu1.getWriter();
                anoWriter.setWriter(this.writer);
                anoWriter.writeElement(element);
            }
            else if( this.original2 == element){

                this.anoWriter = mu2.getWriter();
                anoWriter.setWriter(this.writer);
                anoWriter.writeElement(element);
            }
            else
            {
                super.writeElement(element);
            }
        }
    }
    package generate;
    import java.io.IOException;
    import java.io.PrintWriter;
    import java.util.ArrayList;
    import java.util.List;
    import com.ustb.bpel.view.Mainform;
    import parser.BPELMutator;
    import parser.mu_List;
    import utils.XMLWriter;
    public class SecondOrder extends BPELMutator {
        private int algorithm_index;
        public List<String[]> secondOrdermuList = new ArrayList<String[]>();
        mu_List mu1;
        mu_List mu2;
        public SecondOrder(int algorithm_index) {
            this.algorithm_index = algorithm_index;
            pre_process();
        }
        public List<String[]> getSecondOrdermuList() {
            return secondOrdermuList;
        }
        public void pre_process() {
            if (algorithm_index == 0) {
                secondOrderGenerate_0();
            } else if (algorithm_index == 1) {
                secondOrderGenerate_1();
            } else {
            }
        }
        private void secondOrderGenerate_0() {
            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;
```

```
import org.dom4j.io.OutputFormat;
import parser.ConcreteVisitor;
import parser.Parser;
import parser.mu_List;
public class BPELMutator extends ConcreteVisitor {
    public int muID = 0;
    public Date now;
    public final String mutantPath = "D:/Mutant";
    public static String className = null;
    public Document document;
    File file;
    protected static List<mu_List> MutantsList = new ArrayList<mu_List>();
    public BPELMutator() {
        this.document = Parser.getDocument();
        File f = new File(mutantPath);
        if (!f.exists()) {
            f.mkdir();
        }
        String timeNow = getTimeNow();
         file = new File(f,timeNow);
        file.mkdir();
}
    public BPELMutator(Document document) {
        this.document = document;
    }
    public BPELMutator(Document document, String Path, String Name) {
        this.document = document;
        className = Name;
    }
    public static List<mu_List> getMutantsList() {
        return MutantsList;
    }
public String getTimeNow(){
        now = new Date();
        SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH：mm");
        return df.format(now);
    }
    public String getClassName() {
        Class cc = this.getClass();
        return exclude(cc.getName(), cc.getPackage().getName());
    }
    public String exclude(String a, String b) {
        return a.substring(b.length() + 1, a.length());
    }
    public String getMuantID() {
        String str = getClassName() + "_" + this.muID;
        return str;
    }
    public String getMuFileName(String op_name,int muID) {
        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;
```

```
import org.dom4j.Document;
import utils.XMLReader;
public class Parser{
    private static String filePath;
    private static Document document;
    private List<String> operatorList;
public Parser(String filePath, List<String> operatorList){
        System.out.println("*******start Parser*********");
        this.filePath = filePath;
        this.document = XMLReader.openXMLFile(this.filePath);
        this.operatorList = operatorList;
        this.parserTree();
    }
    public void parserTree(){
        System.out.println("***parserTree***");
        String operator;
        ConcreteVisitor visitor;
        for (int i = 0; i < operatorList.size(); i++) {
            operator = operatorList.get(i);
            try {
                Class<?> c = Class.forName(operator);
                try {
                    visitor =    (ConcreteVisitor) c.newInstance();
                    document.accept(visitor);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            } catch(ClassNotFoundException e){
                e.printStackTrace();
            }
        }
    }
    public static Document getDocument() {
        return document;
    }
}
package run;
import generate.GenerateFacade;
import java.util.List;
import parser.BPELMutator;
import parser.Parser;
import parser.mu_List;
public class main
{
    private static int algorithm = -1;
    public static void setAlgorithm(int al) {
        algorithm = al;
    }
    public static List<String[]> test(String path,List<String> operatorList,int flag)
    {
```

```
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++;
```

```
                    mu_List mu_List = new mu_List("APA", element, new APA_Writer(
                            element), muID);
                    MutantsList.add(mu_List);
                } else
                    super.visit(element);
        }
    }
    package operator;
    import java.io.PrintWriter;
    import org.dom4j.Element;
    public class APM_Writer extends Remove_writer{
        public APM_Writer(Element original) {
            super(original);
        }
        public APM_Writer(PrintWriter out) {
            super(out);
        }
    }
    package operator;
    import java.util.List;
    import org.dom4j.Element;
    import parser.BPELMutator;
    import parser.mu_List;
    public class APM extends BPELMutator {
        public APM() {
            super();
            System.out.println("***APM");
        }
        public void visit(Element element) {
            if (element.getName().equals("pick")) {
                System.out.println("**************find pick");
                List<Element> onMessageList = element.elements("onMessage");
                if (onMessageList.size() >= 2) {
                    for (int i = 0; i < onMessageList.size(); i++) {
                        Element onMessagElement = onMessageList.get(i);
                        muID++;
                        mu_List mu_List = new mu_List("APM", onMessagElement,
                                new APM_Writer(onMessagElement), muID);
                        MutantsList.add(mu_List);
                    }
                }
            } else
                super.visit(element);
        }
    }
    package operator;
    import java.io.IOException;
    import java.io.PrintWriter;
    import org.dom4j.Attribute;
    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 {
```

```
                        writer.write(">");
                        if (textOnly) {
                                writeElementContent(element);
                        } else {
                            ++indentLevel;
                            writeElementContent(element);
                            --indentLevel;
                            writePrintln();
                            indent();
                        }
                        writer.write("</");
                        writer.write(qualifiedName);
                        writer.write(">");
                    }
                    while (namespaceStack.size() > previouslyDeclaredNamespaces) {
                        namespaceStack.pop();
                    }
                    lastOutputNodeType = Node.ELEMENT_NODE;
                } else {
                    super.writeElement(element);
                }
            }
        }
        package operator;
        import org.dom4j.Element;
        import parser.BPELMutator;
        import parser.mu_List;
        public class ASF extends BPELMutator {
            public ASF() {
                super();
                System.out.println("***ASF");
            }
            public void visit(Element element) {
                if (element.getName().equals("sequence")) {
                    System.out.println("***************find asf sequence");
                    muID++;
                    mu_List mu_List = new mu_List("ASF", element, new ASF_Writer(
                            element), muID);
                    MutantsList.add(mu_List);

                } else
                    super.visit(element);
            }
        package operator;
        import java.io.IOException;
        import java.io.PrintWriter;
        import org.dom4j.Comment;
        import org.dom4j.Element;
        import org.dom4j.Namespace;
        import org.dom4j.Node;
```

```
import utils.XMLWriter;
public class ASI_Writer extends XMLWriter{
    private Element original;
    private Element child1;
    private Element child2;
    public ASI_Writer(Element original, Element child1, Element child2) {
        this.original = original;
        this.child1 = child1;
        this.child2 = child2;
    }
    public ASI_Writer(PrintWriter out) {
        super(out);
    }
    public void setMutant(Element original, Element child1, Element child2) {
        this.original = original;
        this.child1 = child1;
        this.child2 = child2;
    }
    public void writeElement(Element element) throws IOException {
        Element parElement = element.getParent();
        if (parElement == original) {
            if(element == child1){
                element = child2;
            }
            else if(element == child2)
            {
                element = child1;
            }
            int size = element.nodeCount();
            String qualifiedName = element.getQualifiedName();
            writePrintln();
            indent();
            writer.write("<");
            writer.write(qualifiedName);
            int previouslyDeclaredNamespaces = namespaceStack.size();
            Namespace ns = element.getNamespace();
            if (isNamespaceDeclaration(ns)) {
                namespaceStack.push(ns);
                writeNamespace(ns);
            }
            boolean textOnly = true;
            for (int i = 0; i < size; i++) {
                Node node = element.node(i);
                if (node instanceof Namespace) {
                    Namespace additional = (Namespace) node;
                    if (isNamespaceDeclaration(additional)) {
                        namespaceStack.push(additional);
                        writeNamespace(additional);
                    }
                } else if (node instanceof Element) {
```

```
                        textOnly = false;
                    } else if (node instanceof Comment) {
                        textOnly = false;
                    }
                }
                writeAttributes(element);
                lastOutputNodeType = Node.ELEMENT_NODE;
                if (size <= 0) {
                    writeEmptyElementClose(qualifiedName);
                } else {
                    writer.write(">");
                    if (textOnly) {
                        writeElementContent(element);
                    } else {
                        ++indentLevel;
                        writeElementContent(element);
                        --indentLevel;
                        writePrintln();
                        indent();
                    }
                    writer.write("</");
                    writer.write(qualifiedName);
                    writer.write(">");
                }
                while (namespaceStack.size() > previouslyDeclaredNamespaces) {
                    namespaceStack.pop();
                }
                lastOutputNodeType = Node.ELEMENT_NODE;
            } else {
                super.writeElement(element);
            }
        }
    }
    }
    package operator;
    import java.util.List;
    import org.dom4j.Element;
    import parser.BPELMutator;
    import parser.mu_List;
    public class ASI extends BPELMutator {
        public ASI() {
            super();
            System.out.println("***ASI");
        }
        public void visit(Element element) {
            if (element.getName().equals("sequence")) {
                System.out.println("**********find asi sequence");
                List<Element> childList = element.elements();
                for (int i = 0; i < childList.size(); i++) {
                    Element ele1 = childList.get(i);
```

```
                        for (int j = i + 1; j < childList.size(); j++) {
                            Element ele2 = childList.get(j);
                            System.out.println("***exchange two activity:"
                                    + ele1.getName() + "    " + ele2.getName());
                            muID++;
                            mu_List mu_List = new mu_List("ASI", element,
                                    new ASI_Writer(element, ele1, ele2), muID);
                            MutantsList.add(mu_List);
                        }
                    }
                } else
                    super.visit(element);
        }
    }
    package operator;
    import java.io.IOException;
    import java.io.PrintWriter;
    import org.dom4j.Attribute;
    import org.dom4j.Element;
    import utils.XMLWriter;
    public class attReplace_Writer extends XMLWriter {
        private Element original;
        private Attribute attribute;
        public attReplace_Writer(Element original, Attribute attribute) {
            this.original = original;
            this.attribute = attribute;
        }
        public attReplace_Writer(PrintWriter out) {
            super(out);
        }
        public void setMutant(Element original, Attribute attribute) {
            this.original = original;
            this.attribute = attribute;
        }
        public void writeAttributes(Element element) throws IOException {
            if (original == element) {
                Element copy = element.createCopy();
                Attribute attribute = copy.attribute(this.attribute.getName());
                if (attribute.getValue().equals("yes")) {
                    attribute.setValue("no");
                } else if (attribute.getValue().equals("no")) {
                    attribute.setValue("yes");
                } else {
                    attribute.setValue(this.attribute.getValue());
                }
                element = copy;
            }
            super.writeAttributes(element);
        }
    }
```

```
package operator;
import java.io.IOException;
import java.io.PrintWriter;
import org.dom4j.Attribute;
import org.dom4j.Comment;
import org.dom4j.Document;
import org.dom4j.DocumentType;
import org.dom4j.Element;
import org.dom4j.Entity;
import org.dom4j.Namespace;
import org.dom4j.Node;
import org.dom4j.ProcessingInstruction;
import org.dom4j.Text;
import utils.XMLWriter;
public class AWR_Writer extends XMLWriter {
	private Element original;
	private Element conditonElement = null;
	public AWR_Writer(Element original) {
		this.original = original;
	}
	public AWR_Writer(PrintWriter out) {
		super(out);
	}
	public void setMutant(Element original) {
		this.original = original;
	}
	public void writeElement(Element element) throws IOException {
		if (element == original) {
			int size = element.nodeCount();
			String qualifiedName = null;
			if (element.getName().equals("repeatUntil")) {
				qualifiedName = "while";
			} else if (element.getName().equals("while")) {
				qualifiedName = "repeatUntil";
			}
			writePrintln();
			indent();
			writer.write("<");
			writer.write(qualifiedName);
			int previouslyDeclaredNamespaces = namespaceStack.size();
			Namespace ns = element.getNamespace();
			if (isNamespaceDeclaration(ns)) {
				namespaceStack.push(ns);
				writeNamespace(ns);
			}
			boolean textOnly = true;
			for (int i = 0; i < size; i++) {
				Node node = element.node(i);
				if (node instanceof Namespace) {
					Namespace additional = (Namespace) node;
```

```
                        if (isNamespaceDeclaration(additional)) {
                            namespaceStack.push(additional);
                            writeNamespace(additional);
                        }
                    } else if (node instanceof Element) {
                        textOnly = false;
                    } else if (node instanceof Comment) {
                        textOnly = false;
                    }
                }
                writeAttributes(element);
                lastOutputNodeType = Node.ELEMENT_NODE;
                if (size <= 0) {
                    writeEmptyElementClose(qualifiedName);
                } else {
                    writer.write(">");
                    if (textOnly) {
                        writeElementContent(element);
                    } else {
                        ++indentLevel;
                        writeElementContent(element);
                        --indentLevel;
                        writePrintln();
                        indent();
                    }
                    writer.write("</");
                    writer.write(qualifiedName);
                    writer.write(">");
                }
                while (namespaceStack.size() > previouslyDeclaredNamespaces) {
                    namespaceStack.pop();
                }
                lastOutputNodeType = Node.ELEMENT_NODE;
            } else {
                super.writeElement(element);
            }
        }
        protected void writeElementContent(Element element) throws IOException {
            if (element == original) {
                boolean trim = format.isTrimText();
                boolean oldPreserve = preserve;
                if (trim) {
                    preserve = isElementSpacePreserved(element);
                    trim = !preserve;
                }
                for (int i = 0, size = element.nodeCount(); i < size; i++) {
                    Node node = element.node(i);
                    if (node.getNodeType() == Node.ELEMENT_NODE
                            && node.getName().equals("condition")) {
                        if (element.getName().equals("while")) {
```

```
1101                            this.conditonElement = (Element) node;
1102                        } else if (element.getName().equals("repeatUntil")) {
1103                            writeNode(node);
1104                        }
1105                    }
1106                }
1107                if (trim) {
1108                    Text lastTextNode = null;
1109                    StringBuffer buff = null;
1110                    boolean textOnly = true;
1111                    for (int i = 0, size = element.nodeCount(); i < size; i++) {
1112                        Node node = element.node(i);
1113                        if (!node.getName().equals("condition")) {
1114                            if (node instanceof Text) {
1115                                if (lastTextNode == null) {
1116                                    lastTextNode = (Text) node;
1117                                } else {
1118                                    if (buff == null) {
1119                                        buff = new StringBuffer(
1120                                                lastTextNode.getText());
1121                                    }
1122                                    buff.append(((Text) node).getText());
1123                                }
1124                            } else {
1125                                if (!textOnly && format.isPadText()) {
1126                                    char firstChar = 'a';
1127                                    if (buff != null) {
1128                                        firstChar = buff.charAt(0);
1129                                    } else if (lastTextNode != null) {
1130                                        firstChar = lastTextNode.getText()
1131                                                .charAt(0);
1132                                    }
1133                                    if (Character.isWhitespace(firstChar)) {
1134                                        writer.write(PAD_TEXT);
1135                                    }
1136                                }
1137                                if (lastTextNode != null) {
1138                                    if (buff != null) {
1139                                        writeString(buff.toString());
1140                                        buff = null;
1141                                    } else {
1142                                        writeString(lastTextNode.getText());
1143                                    }
1144                                    if (format.isPadText()) {
1145                                        char lastTextChar = 'a';
1146                                        if (buff != null) {
1147                                            lastTextChar = buff.charAt(buff
1148                                                    .length() - 1);
1149                                        } else if (lastTextNode != null) {
1150                                            String txt = lastTextNode.getText();
```

```
                                        lastTextChar = txt
                                                .charAt(txt.length() - 1);
                                    }
                                    if (Character.isWhitespace(lastTextChar)) {
                                        writer.write(PAD_TEXT);
                                    }
                                }
                                lastTextNode = null;
                            }
                            textOnly = false;
                            writeNode(node);
                        }
                    }
                }
                if (element.getName().equals("while")) {
                    writeNode(conditonElement);
                }
                if (lastTextNode != null) {
                    if (!textOnly && format.isPadText()) {
                        char firstChar = 'a';
                        if (buff != null) {
                            firstChar = buff.charAt(0);
                        } else {
                            firstChar = lastTextNode.getText().charAt(0);
                        }
                        if (Character.isWhitespace(firstChar)) {
                            writer.write(PAD_TEXT);
                        }
                    }
                    if (buff != null) {
                        writeString(buff.toString());
                        buff = null;
                    } else {
                        writeString(lastTextNode.getText());
                    }
                    lastTextNode = null;
                }
            } else {
                Node lastTextNode = null;
                for (int i = 0, size = element.nodeCount(); i < size; i++) {
                    Node node = element.node(i);
                    if (!node.getName().equals("condition")) {
                        if (node instanceof Text) {
                            writeNode(node);
                            lastTextNode = node;
                        } else {
                            if ((lastTextNode != null) && format.isPadText()) {

                                String txt = lastTextNode.getText();
                                char lastTextChar = txt
```

```
                                                .charAt(txt.length() - 1);
                                        if (Character.isWhitespace(lastTextChar)) {
                                            writer.write(PAD_TEXT);
                                        }
                                    }
                                    writeNode(node);
                                    lastTextNode = null;
                                }
                            }
                        }
                        if (element.getName().equals("while")) {
                            writeNode(conditonElement);
                        }
                    }
                    preserve = oldPreserve;
                }
                else {
                    super.writeElementContent(element);
                }
            }
        }
        package operator;
        import org.dom4j.Element;
        import parser.BPELMutator;
        import parser.mu_List;
        public class AWR extends BPELMutator {
            public AWR() {
                super();
                System.out.println("***AWR");
            }
            public void visit(Element element) {
                if (element.getName().equals("repeatUntil")
                        || element.getName().equals("while")) {
                    System.out.println("**************find " + element.getName());
                    muID++;
                    mu_List mu_List = new mu_List("AWR", element, new AWR_Writer(
                            element), muID);
                    MutantsList.add(mu_List);
                } else
                    super.visit(element);
            }
        }
        package operator;
        import org.dom4j.Element;
        public class CCO_Writer extends exReplace_Writer{
            public CCO_Writer(Element original, String originalOp, String replaceOp) {
                super(original, originalOp, replaceOp);
            }
        }
        package operator;
```

```
import java.util.Arrays;
import java.util.List;
import org.dom4j.Element;
import parser.BPELMutator;
import parser.mu_List;
public class CCO extends BPELMutator {
	List<String> mutateList = Arrays.asList("true()", "false()");
	public CCO() {
		super();
		System.out.println("***CCO");
	}
	public void visit(Element element) {
		if (element.getName().equals("condition")
				|| element.getName().equals("transitionCondition")
				|| element.getName().equals("joinCondition")) {
			System.out.println("**************find " + element.getName());
			String expression = element.getText();
			if (expression.contains("and") || expression.contains("or")) {
				String s[] = expression.split(" *(and|or) *");
			for (int j = 0; j < mutateList.size(); j++) {
					String replaceEx = mutateList.get(j);
					String tempEx = expression;
					for (int i = 0; i < s.length; i++) {
						tempEx = tempEx.replace(s[i], replaceEx);
					}
					muID++;
					mu_List mu_List = new mu_List("CCO", element,
							new CCO_Writer(element, expression, tempEx), muID);
					MutantsList.add(mu_List);
				}
			} else {
				for (int i = 0; i < mutateList.size(); i++) {
					String replaceEx = mutateList.get(i);
					muID++;
					mu_List mu_List = new mu_List("CCO", element,
							new CCO_Writer(element, expression, replaceEx),
							muID);
					MutantsList.add(mu_List);
				}
			}
		} else
			super.visit(element);
	}
}
package operator;
import org.dom4j.Element;
public class CDC_Writer extends exReplace_Writer{
	public CDC_Writer(Element original, String originalOp, String replaceOp) {
		super(original, originalOp, replaceOp);
	}
```

```
}
package operator;
import java.util.Arrays;
import java.util.List;
import org.dom4j.Element;
import parser.BPELMutator;
import parser.mu_List;
public class CDC extends BPELMutator {
    List<String> mutateList = Arrays.asList("true()", "false()");
    public CDC() {
        super();
        System.out.println("***CDC");
    }
    public void visit(Element element) {
        if (element.getName().equals("condition")
                || element.getName().equals("transitionCondition")
                || element.getName().equals("joinCondition")) {
            System.out.println("**************find " + element.getName());
            String expression = element.getText();
            for (int i = 0; i < mutateList.size(); i++) {
                String replaceEx = mutateList.get(i);
                muID++;
                mu_List mu_List = new mu_List("CDC", element, new CDC_Writer(
                        element, expression, replaceEx), muID);
                MutantsList.add(mu_List);
            }
            if (expression.contains("and") || expression.contains("or")) {
                String s[] = expression.split(" *(and|or) *");
                for (int j = 0; j < mutateList.size(); j++) {
                    String replaceEx = mutateList.get(j);
                    String tempEx = expression;
                    for (int i = 0; i < s.length; i++) {
                        tempEx = tempEx.replace(s[i], replaceEx);
                    }
                    muID++;
                    mu_List mu_List = new mu_List("CDC", element,
                            new CDC_Writer(element, expression, tempEx), muID);
                    MutantsList.add(mu_List);
                }
            }
        } else
        super.visit(element);
    }
}
package operator;
import org.dom4j.Element;
public class CDE_Writer extends exReplace_Writer{
    public CDE_Writer(Element original, String originalOp, String replaceOp) {
        super(original, originalOp, replaceOp);
    }
```

```
}
package operator;
import java.util.Arrays;
import java.util.List;
import org.dom4j.Element;
import parser.BPELMutator;
import parser.mu_List;
public class CDE extends BPELMutator{
	List<String> mutateList = Arrays.asList("true()", "false()");
	public CDE() {
		super();
		System.out.println("***CDE");
	}
	public void visit(Element element) {
		if (element.getName().equals("condition")
				|| element.getName().equals("transitionCondition")
				|| element.getName().equals("joinCondition")) {
			System.out.println("**************find " + element.getName());
			String expression = element.getText();
			for(int i = 0;i<mutateList.size();i++){
			String replaceEx = mutateList.get(i);
			muID++;
			mu_List mu_List = new mu_List("CDE", element, new CDE_Writer(
					element, expression, replaceEx), muID);
			MutantsList.add(mu_List);
			}
		} else
			super.visit(element);
	}
}
package operator;
import java.io.IOException;
import org.dom4j.Element;
import utils.XMLWriter;
public class CFA_Writer extends XMLWriter {
	private Element original;
	public CFA_Writer(Element original) {
		this.original = original;
	}
	public void writeElement(Element element) throws IOException {
		if (element == original) {
			String qualifiedName = "exit";
			writePrintln();
			indent();
			writer.write("<");
			writer.write(qualifiedName);
			writer.write(" ");
			writer.write("/");
			writer.write(">");
		} else {
```

```
                super.writeElement(element);
            }
        }
    }
    package operator;
    import org.dom4j.Element;
    import parser.BPELMutator;
    import parser.mu_List;
    public class CFA extends BPELMutator{
        public CFA() {
            super();
            System.out.println("***CFA");
        }
        public void visit(Element element) {
            if (element.getName().equals("assign")
                    || element.getName().equals("invoke")
                    || element.getName().equals("reply")
                    || element.getName().equals("throw")
                    || element.getName().equals("wait")
                    || element.getName().equals("exit")
                    || element.getName().equals("rethrow")
                    || element.getName().equals("scope")
                    || element.getName().equals("flow")
                    || element.getName().equals("switch")
                    || element.getName().equals("if")
                    || element.getName().equals("while")
                    || element.getName().equals("repeatuntil")
                    || element.getName().equals("foreach")
                    || element.getName().equals("pick")) {
                System.out.println("***************find " + element.getName());
                muID++;
                mu_List mu_List = new mu_List("CFA", element, new CFA_Writer(
                        element), muID);
                MutantsList.add(mu_List);
            } else if (element.getName().equals("sequence")) {
                Element farElement = element.getParent();
                if (!farElement.getName().equals("process")) {
                    muID++;
                    mu_List mu_List = new mu_List("CFA", element, new CFA_Writer(
                            element), muID);
                    MutantsList.add(mu_List);
                }
            } else
                super.visit(element);
        }
    }
    package alan.program.Configuration;
    public class Workspace {
        public static final String TOMCAT_HOME = " E:\\tool\\apache-tomcat\\";
        public static final String WORKSPACE = TOMCAT_HOME
```

```
                + "webapps/mt4ws/workspace/";
    private String userName;
    private String path;
    public String getPath() {
        return WORKSPACE + userName + "/";
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
}
package alan.program.Configuration;
import java.io.File;
import java.util.ArrayList;
import java.util.List;
import org.dom4j.Attribute;
import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.Element;
import org.dom4j.io.SAXReader;
import org.jaxen.function.LastFunction;
import alan.program.Configuration.*;
public class WSDLParser {
    public static Configuration configuration;
    public static String RunConfiguration(String bpelfilepath) {
        String path=bpelfilepath.substring(0,bpelfilepath.lastIndexOf("\\"));
        List data = new ArrayList();
        data = getData(path, data);
        return null;
    }
    private static List getData(String path, List data) {
        File f = new File(path);
        if (f.isDirectory()) {
            File[] fs = f.listFiles();
            for (int i = 0; i < fs.length; i++) {
                data = getData(fs[i].getPath(), data);
            }
        } else if (f.getName().endsWith("config.xml")) {
            data.add(f.getName());
            parseXml(f);
        }
        return data;
    }
    private static Configuration parseXml(File f) {
        SAXReader reader = new SAXReader();
          configuration = new Configuration();
        Document document;
try {
            document = reader.read(f);
            Element root = document.getRootElement();
            Element clientElement = root.element("client");
```