

```
14301 package com.ustb.bpel.bean;
14302 import java.awt.Color;
14303 import java.awt.Component;
14304 import java.io.FileNotFoundException;
14305 import java.io.IOException;
14306 import org.apache.log4j.AppenderSkeleton;
14307 import org.apache.log4j.Logger;
14308 import java.util.Date;
14309 import java.util.HashMap;
14310 import java.util.Iterator;
14311 import javax.management.ObjectName;
14312 import javax.swing.DefaultComboBoxModel;
14313 import javax.swing.DefaultListModel;
14314 import javax.swing.JLabel;
14315 import com.ustb.bpel.constants.Constants;
14316 import com.ustb.bpel.log.TomcatLog;
14317 import com.ustb.bpel.xml.*;
14318 import com.ustb.bpel.xml.impl.BpelFileImpl;
14319 import com.ustb.bpel.xml.impl.ConfigFileImpl;
14320 public class Facade {
14321     LogActionMonitor logActionMonitor;
14322     public void setLogActionMonitor(LogActionMonitor logActionMonitor) {
14323         this.logActionMonitor = logActionMonitor;
14324     }
14325     public String getLastBPELFilePath(){
14326         Configuration config=(Configuration)Constants.beanFactory.getBean("configfileImpl");
14327         return config.getLastBPELFilePath();
14328     }
14329     public String getLastBuildFilePath(){
14330         Configuration
14331         config=(Configuration)Constants.beanFactory.getBean("configfileImpl");
14332         return config.getLastBuildFilePath();
14333     }
14334     public void setLastBPELFilePath(String path){
14335         Configuration config=(Configuration)Constants.beanFactory.getBean("configfileImpl");
14336         config.setLastBPELFilePath(path);
14337     }
14338     public void setLastBuildFilePath(String path){
14339         Configuration config=(Configuration)Constants.beanFactory.getBean("configfileImpl");
14340         config.setLastBuildFilePath(path);
14341     }
14342     public String getServerLogString() {
14343         return Constants.serverLogString;
14344     }
14345     public void startGetLogThread() throws InterruptedException {
14346         TomcatLog tomcatLog=new TomcatLog();
14347         try {
14348             tomcatLog.startTomcatlog();
14349         }
14350         catch (IOException e)
```

```
14351     {
14352         e.printStackTrace();
14353     }
14354 }
14355 public String getTOMCAT_HOME(){
14356     Configuration config=(Configuration)Constants.beanFactory.getBean("configfileImpl");
14357     return config.getTOMCAT_HOME();
14358 }
14359 public void setTOMCAT_HOME(String path){
14360     Configuration config=(Configuration)Constants.beanFactory.getBean("configfileImpl");
14361     config.setTOMCAT_HOME(path);
14362 }
14363 public void processLog(String logString, DefaultListModel model) {
14364     String actionString = logActionMonitor.processLog(logString);
14365     if (actionString.equals("deployed")||actionString.equals("undeployed")) {
14366         System.out.println("model changed: deployed& undeployed");
14367     }
14368 }
14369 public void setEngineStateListener(String logString,JLabel label){
14370     if(logString.contains("ActiveBP EL In-Memory Configuration Started")){
14371         label.setText("completed");
14372         label.setForeground(Color.BLUE);
14373     }
14374 }
14375 public void setDeploymentListener(String logsString, JLabel label, JLabel label2){
14376     if(logsString.contains("deployment")){
14377         label.setForeground(Color.BLUE);
14378         label.setText("deploy success");
14379     }else if(logsString.contains("undeployment")){
14380         label2.setForeground(Color.BLUE);
14381         label2.setText("undeploy success");
14382     }
14383 }
14384 public void addAppender(AppenderSkeleton appender){
14385     Logger.getRootLogger().addAppender(appender);
14386 }
14387 }
14388 package alan.testcase.executor;
14389 import java.io.BufferedReader;
14390 import java.io.BufferedWriter;
14391 import java.io.File;
14392 import java.io.FileReader;
14393 import java.io.FileWriter;
14394 import java.io.IOException;
14395 import java.text.DateFormat;
14396 import java.util.ArrayList;
14397 import java.util.Date;
14398 import java.util.HashMap;
14399 import java.util.List;
14400 import java.util.Map;
```

```
14401 import javax.xml.namespace.QName;
14402 import javax.xml.rpc.ParameterMode;
14403 import org.apache.axis.client.Call;
14404 import org.apache.axis.client.Service;
14405 public class RunCase {
14406     private static Map<String, QName> typeMap = new HashMap<String, QName>();
14407     static {
14408         typeMap.put("string", org.apache.axis.Constants.XSD_STRING);
14409         typeMap.put("int", org.apache.axis.Constants.XSD_INT);
14410         typeMap.put("double", org.apache.axis.Constants.XSD_DOUBLE);
14411         typeMap.put("float", org.apache.axis.Constants.XSD_FLOAT);
14412     }
14413     List<String> paramNameList = new ArrayList<String>();
14414     List<String> paramTypeList = new ArrayList<String>();
14415     String returnType = "";
14416     public List<ExecutionResult> execute(String endpoint, String operationName,
14417         String testcaseFilePath, String MutantName) {
14418         List<ExecutionResult> executionResults = new ArrayList<ExecutionResult>();
14419         BufferedReader reader = null;
14420         BufferedWriter writer = null;
14421         try {
14422             String resultFilePath = testcaseFilePath.substring(0,
14423                 testcaseFilePath.lastIndexOf(".") + "_" + MutantName + "_result.txt");
14424             reader = new BufferedReader(new FileReader(new File(
14425                 testcaseFilePath)));
14426             writer = new BufferedWriter(
14427                 new FileWriter(new File(resultFilePath)));
14428             String line = null;
14429             line = reader.readLine();
14430             String[] inputs = line.split("#");
14431             int inputParamNumber = Integer.parseInt(inputs[0]);
14432             System.out.println(line);
14433             for (int i = 0; i < inputParamNumber; i++) {
14434                 String[] inputParam = inputs[i + 1].split(",");
14435                 String paramName = inputParam[0];
14436                 String paramType = inputParam[1];
14437                 paramNameList.add(paramName);
14438                 paramTypeList.add(paramType);
14439             }
14440             returnType = inputs[inputs.length - 1];
14441             String[] args = new String[inputParamNumber];
14442             int no = 1;
14443             while ((line = reader.readLine()) != null) {
14444                 ExecutionResult er = new ExecutionResult();
14445                 String[] params = line.split("#");
14446                 for (int i = 0; i < inputParamNumber; i++) {
14447                     args[i] = params[i];
14448                 }
14449                 String expectedResult = params[inputParamNumber];
14450                 String CaseInput = line;
```

```
14451         long beginTime = System.currentTimeMillis();
14452         String executedResult = executeTestCase(args, endpoint,
14453             operationName);
14454         long endTime = System.currentTimeMillis();
14455         long interval = endTime - beginTime;
14456         String status = expectedResult.equals(executedResult) ? "SUCCESS"
14457             : "FAILURE";
14458         double n = 0, m = 0;
14459         m = status.equals("SUCCESS") ? m++ : n++;
14460         er.setNo(no);
14461         er.setBeginTime(getDateTime(beginTime));
14462         er.setEndTime(getDateTime(endTime));
14463         er.setInput(CaseInput);
14464         er.setInterval(interval);
14465         er.setExpectedResult(expectedResult);
14466         er.setExecutedResult(executedResult);
14467         er.setStatus(status);
14468         executionResults.add(er);
14469         writer.write(no + "#" + getDateTime(beginTime) + "#"
14470             + getDateTime(endTime) + "#" + interval + "#"
14471             + expectedResult + "#" + executedResult + "#" + status);
14472         writer.write("\r\n");
14473         writer.flush();
14474         no++;
14475     }
14476 } catch (Exception ex) {
14477     ex.printStackTrace();
14478 } finally {
14479     try {
14480         if (reader != null) {
14481             reader.close();
14482         }
14483         if (writer != null) {
14484             writer.close();
14485         }
14486     } catch (IOException ex) {
14487         ex.printStackTrace();
14488     }
14489 }
14490 return executionResults;
14491 }
14492 public String executeTestCase(String[] params, String endpoint,
14493     String operationName) throws Exception {
14494     Call call = createCall(endpoint, operationName);
14495     String result = null;
14496     try {
14497         result = (String) call.invoke(params);
14498     } catch (Exception e) {
14499         result = "unexpected exception seen: " + e.toString();
14500     }
```

```

14501         if (result == null) {
14502             result = "<null>";
14503         }
14504         return result;
14505     }
14506     protected Call createCall(String endpoint, String operationName)
14507         throws Exception {
14508         Service service = new Service();
14509         Call call = (Call) service.createCall();
14510         call.setTargetEndpointAddress(endpoint);
14511         call.setOperationName(operationName);
14512         for (int i = 0; i < paramNameList.size(); i++) {
14513             String paramName = paramNameList.get(i);
14514             String paramType = paramTypeList.get(i);
14515             call.addParameter(paramName, typeMap.get(paramType),
14516                 ParameterMode.IN);
14517         }
14518         call.setReturnType(typeMap.get(returnType));
14519         return call;
14520     }
14521     private String getDateTimes(long date) {
14522         Date now = new Date(date);
14523         DateFormat df = DateFormat.getDateTimeInstance();
14524         return df.format(now);
14525     }
14526 }
14527 package alan.testcase.executor;
14528 import java.io.BufferedReader;
14529 import java.io.IOException;
14530 import java.util.ArrayList;
14531 import java.util.List;
14532 import java.io.File;
14533 import java.io.FileReader;
14534 import org.apache.log4j.LogManager;
14535 import org.apache.log4j.Logger;
14536 import org.apache.log4j.PropertyConfigurator;
14537 import alan.testcase.executor.*;
14538 public class Executor {
14539     private static Logger logger = LogManager.getLogger(Executor.class.getName());
14540     public static List<ExecutionMatrix> execute(String originOutputPath,String
14541         testcaseFilePath){
14542         List<ExecutionMatrix> executionResults = new ArrayList<ExecutionMatrix>();
14543         BufferedReader reader0 = null;
14544         BufferedReader reader1 = null;
14545         ArrayList<String> station=new ArrayList<String>() ;
14546         double m=0,n=0;
14547         try{
14548             reader0 = new BufferedReader(new FileReader(new File(
14549                 originOutputPath)));
14550             reader1= new BufferedReader(new FileReader(new File(

```

```
14551         testcaseFilePath));
14552     String line = null;
14553     String Status;
14554     ExecutionMatrix er = new ExecutionMatrix();
14555     while ((line = reader0.readLine()) != null) {
14556         ++n;
14557         String mutantPath=reader1.readLine();
14558         if(!(line.equals(mutantPath))){
14559             ++m;
14560             Status ="F";
14561         }else{
14562             Status ="T";
14563         }
14564         station.add(Status);
14565     }
14566     String finalStatus=m==0?"SURVIVE":"KILLED";
14567     String
14568     MutantName=testcaseFilePath.substring(testcaseFilePath.lastIndexOf("\\")+1,testcaseFilePat
14569     h.lastIndexOf("."));
14570     double FDR=m/n;
14571     er.setMutantName(MutantName);
14572     er.setFDR(FDR);
14573     er.setStatus(station);
14574     er.setFinalStatus(finalStatus);
14575     executionResults.add(er);
14576     logger.info(er.getMutantName()+"#"+er.getStatus()+"#"+er.getFDR()+"#"+er.getFinalStatus());
14577 }
14578 catch (Exception ex) {
14579     ex.printStackTrace();
14580 }
14581 try {
14582     reader0.close();
14583     reader1.close();
14584 } catch (IOException e) {
14585     e.printStackTrace();
14586 }
14587 return executionResults;
14588 }
14589 }
14590 }
14591 package alan.testcase.executor;
14592 public class ExecutionResult {
14593     private int no;
14594     private String beginTime;
14595     private String endTime;
14596     private String CaseInput;
14597     private long interval;
14598     private String expectedResult;
14599     private String executedResult;
14600     private String status;
```

```
14601     private String MutantName;
14602     private double FDR;
14603     public int getNo() {
14604         return no;
14605     }
14606     public void setNo(int no) {
14607         this.no = no;
14608     }
14609     public String getBeginTime() {
14610         return beginTime;
14611     }
14612     public void setBeginTime(String beginTime) {
14613         this.beginTime = beginTime;
14614     }
14615     public String getEndTime() {
14616         return endTime;
14617     }
14618     public void setEndTime(String endTime) {
14619         this.endTime = endTime;
14620     }
14621     public String getInput() {
14622         return CaseInput;
14623     }
14624     public void setInput(String CaseInput) {
14625         this.CaseInput = CaseInput;
14626     }
14627     public long getInterval() {
14628         return interval;
14629     }
14630     public void setInterval(long interval) {
14631         this.interval = interval;
14632     }
14633     public String getExpectedResult() {
14634         return expectedResult;
14635     }
14636     public void setExpectedResult(String expectedResult) {
14637         this.expectedResult = expectedResult;
14638     }
14639     public String getExecutedResult() {
14640         return executedResult;
14641     }
14642     public void setExecutedResult(String executedResult) {
14643         this.executedResult = executedResult;
14644     }
14645     public String getStatus() {
14646         return status;
14647     }
14648     public void setStatus(String status) {
14649         this.status = status;
14650     }
```

```
14651     public String getMutantName() {
14652         return MutantName;
14653     }
14654     public void setMutantName(String MutantName) {
14655         this.MutantName = MutantName;
14656     }
14657     public double getFDR() {
14658         return FDR;
14659     }
14660     public void setFDR(double FDR) {
14661         this.FDR = FDR;
14662     }
14663 }
14664 package alan.testcase.executor;
14665 import java.util.ArrayList;
14666 public class ExecutionMatrix {
14667     private ArrayList<String> status;
14668     private String MutantName;
14669     private long interval;
14670     private double FDR;
14671     private String finalStatus ;
14672     public String getMutantName() {
14673         return MutantName;
14674     }
14675     public void setMutantName(String mutantName) {
14676         MutantName = mutantName;
14677     }
14678     public long getInterval() {
14679         return interval;
14680     }
14681     public void setInterval(long interval) {
14682         this.interval = interval;
14683     }
14684     public ArrayList<String> getStatus() {
14685         return status;
14686     }
14687     public void setStatus(ArrayList<String> status) {
14688         this.status = status;
14689     }
14690     public double getFDR() {
14691         return FDR;
14692     }
14693     public void setFDR(double fDR) {
14694         FDR = fDR;
14695     }
14696     public String getFinalStatus() {
14697         return finalStatus;
14698     }
14699     public void setFinalStatus(String finalStatus) {
14700         this.finalStatus = finalStatus;
```



```

14701     }
14702 }
14703 package com.ustb.bpel.view;
14704 import java.awt.Dimension;
14705 import java.awt.Font;
14706 import java.awt.Toolkit;
14707 import javax.swing.JFrame;
14708 import javax.swing.JPanel;
14709 import javax.swing.GroupLayout.Alignment;
14710 import javax.swing.GroupLayout;
14711 import javax.swing.LayoutStyle.ComponentPlacement;
14712 import javax.swing.border.LineBorder;
14713 import java.awt.Color;
14714 import java.awt.SystemColor;
14715 public class AboutFrame extends javax.swing.JFrame {
14716     public AboutFrame() {
14717         initComponents();
14718     }
14719     public void start() {
14720         this.setVisible(true);
14721         this.setSize(800, 750);
14722         toCenter();
14723     }
14724     public void toCenter() {
14725         Toolkit kit = Toolkit.getDefaultToolkit();
14726         Dimension screenSize = kit.getScreenSize();
14727         int screenWidth = screenSize.width/2;
14728         int screenHeight = screenSize.height/2;
14729         int height = this.getHeight();
14730         int width = this.getWidth();
14731         setLocation(screenWidth-width/2, screenHeight-height/2);
14732     }
14733     private void initComponents() {
14734         setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
14735         setTitle("Help");
14736         String text = "<html><body> "
14737             + "* ISV: Replace a variable identifier by another of the same type and same
14738 scope. <br>"
14739             + "* CFA: Replace an activity by an exit activity. <br>"
14740             + "* CDE: Applies the decision coverage criteria to a WS-BPEL composition.
14741 <br>"
14742             + "* CDC: Applies the decision/condition coverage criteria to a WS-BPEL
14743 composition.<br>"
14744             + "* CCO: Applies the condition coverage criteria to a WS-BPEL composition.
14745 <br>"
14746             + "* EAA: Replaces an arithmetic operator (+,-,*,div,mod) by another of the
14747 same type. <br>"
14748             + "* EEU: Removes the unary minus operator from any expression. <br>"
14749             + "* ERR: Replaces a relational operator (<, >, <=, >=, =, !=) by another of
14750 the same type. <br>"

```

```

14751         + "* ELL: Replaces a logical operator (and, or) by one of the same type. <br>"
14752         + "* ECC: Replaces a path operator (/, //) by another of the same type.<br>"
14753         + "* ECN: Incrementing or decrementing values in one unit, adding or
14754 removing one digit.<br>"
14755         + "* EMD: Replaces a duration expression by 0 or by half of the value.<br>"
14756         + "* EMF: Replaces a deadline expression by 0 or by half of the value.<br>"
14757         + "* EIU: Insert the XPath unary minus operator in arithmetic expressions.
14758 <br>"
14759         + "* EIN: Insert the XPath negation function (not) in logic expressions. <br>"
14760         + "* EAP: Replace a subexpression by its positive absolute value. <br>"
14761         + "* EAN: Replace a subexpression by its negative absolute value. <br>"
14762         + "* ACI: Changes the value of the createInstance attribute from 'yes' to
14763 'no'.<br>"
14764         + "* AFP: Modifying the value of the parallel attribute from 'no' to 'yes'.<br>"
14765         + "* ASF: Replaces a sequence by a flow activity.<br>"
14766         + "* AIS: Changes the isolated attribute of a scope to 'no'. <br>"
14767         + "* AEL: Remove an activity.<br>"
14768         + "* AIE: Removes an elseif element or the else element of an if activity.
14769 <br>"
14770         + "* AWR: Replaces a while activity by a repeatUntil one and vice versa.
14771 <br>"
14772         + "* AJC: Removes the joinCondition element. <br>"
14773         + "* ASI: Exchanges two activities within a sequence activity. <br>"
14774         + "* APM: Removes an onMessage element of a pick activity if there is more
14775 than one. <br>"
14776         + "* APA: Removes the onAlarm element of a pick activity or an event
14777 handler.<br>"
14778         + "* XMF: Removes one catch or the catchall construct of a fault handler.
14779 <br>"
14780         + "* XMC: Removes a compensation handler definition. <br>"
14781         + "* XMT: Removes a termination handler definition. <br>"
14782         + "* XTF: Replaces the fault name by another one in the same scope. <br>"
14783         + "* XER: Removes a rethrow activity. <br>"
14784         + "* XEE: Removes an onEvent element. <br>"
14785         + "<body></html>";
14786         JPanel panel = new JPanel();
14787         panel.setBorder(new LineBorder(SystemColor.activeCaption, 2, true));
14788         JLabel1 = new javax.swing.JLabel();
14789         JLabel1.setFont(new java.awt.Font("Consolas", 0, 22));
14790         JLabel1.setText("Mutation operators introduction:");
14791         javax.swing.GroupLayout layout = new javax.swing.GroupLayout(
14792 getContentPane());
14793         layout.setHorizontalGroup(
14794             layout.createParallelGroup(Alignment.LEADING)
14795                 .addGroup(layout.createSequentialGroup()
14796                     .addContainerGap()
14797                     .addGroup(layout.createParallelGroup(Alignment.LEADING)
14798                         .addComponent(panel, GroupLayout.PREFERRED_SIZE, 706,
14799 GroupLayout.PREFERRED_SIZE)
14800                         .addComponent(JLabel1))

```

```
14801         .addContainerGap(GroupLayout.DEFAULT_SIZE,
14802         Short.MAX_VALUE))
14803     );
14804     layout.setVerticalGroup(
14805         layout.createParallelGroup(Alignment.LEADING)
14806         .addGroup(layout.createSequentialGroup()
14807         .addContainerGap()
14808         .addComponent(jLabel1)
14809         .addPreferredGap(ComponentPlacement.RELATED,
14810         GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
14811         .addComponent(panel,      GroupLayout.PREFERRED_SIZE,      631,
14812         GroupLayout.PREFERRED_SIZE)
14813         .addContainerGap())
14814     );
14815     panel.setLayout(null);
14816     jLabel2 = new javax.swing.JLabel();
14817     jLabel2.setBounds(10, 0, 688, 621);
14818     panel.add(jLabel2);
14819     jLabel2.setFont(new Font("Consolas", 0, 14));
14820     jLabel2.setText(text);
14821     getContentPane().setLayout(layout);
14822     pack();
14823     } private javax.swing.JLabel jLabel1;
14824     private javax.swing.JLabel jLabel2;
14825 }
14826 package com.ustb.bpel.view;
14827 import javax.swing.JLabel;
14828 import javax.swing.JPanel;
14829 import javax.swing.border.BevelBorder;
14830 import javax.swing.filechooser.FileNameExtensionFilter;
14831 import javax.swing.table.DefaultTableModel;
14832 import javax.swing.DefaultListModel;
14833 import javax.swing.JFileChooser;
14834 import javax.swing.JTable;
14835 import javax.swing.JTextField;
14836 import javax.swing.JButton;
14837 import javax.swing.JList;
14838 import java.awt.event.ActionListener;
14839 import java.awt.event.ActionEvent;
14840 import java.io.File;
14841 import java.util.List;
14842 import javax.swing.JScrollPane;
14843 import javax.swing.border.LineBorder;
14844 import java.awt.Color;
14845 import javax.swing.border.TitledBorder;
14846 import java.awt.Font;
14847 import javax.swing.ImageIcon;
14848 import alan.testcase.executor.ExecutionMatrix;
14849 import alan.testcase.executor.ExecutionResult;
14850 import alan.testcase.executor.Executor;
```

```
14851 import javax.swing.JComboBox;
14852 import org.apache.log4j.LogManager;
14853 import org.apache.log4j.Logger;
14854 import javax.swing.JSeparator;
14855 import javax.swing.UIManager;
14856 public class ComparePanel extends JPanel
14857 {
14858     private JTextField textField;
14859     private JList fileList = null;
14860     private DefaultListModel listModel = null;
14861     private JTable resultTable = null;
14862     private DefaultTableModel model = null;
14863     private JTable table;
14864     double ms=0,mq=0;
14865     private JTextField textField_1;
14866     private JTextField textField_2;
14867     private JTextField textField_3;
14868     public ComparePanel()
14869     {
14870         initLayout();
14871     }
14872     public void initLayout(){
14873         setLayout(null);
14874         listModel = new DefaultListModel();
14875         model = new DefaultTableModel();
14876         model.addColumn("No");
14877         model.addColumn("Mutant Name");
14878         model.addColumn("FDR");
14879         model.addColumn("Status");
14880         model.addColumn("FinalStatus");
14881         JPanel panel_1 = new JPanel();
14882         panel_1.setBorder(new
14883             TitledBorder(UIManager.getBorder("TitledBorder.border"), "Analysis Result",
14884             TitledBorder.CENTER, TitledBorder.TOP, null, null));
14885         panel_1.setBounds(22, 10, 854, 170);
14886         add(panel_1);
14887         panel_1.setLayout(null);
14888         JLabel lblNewLabel = new JLabel("Original Output:\r\n");
14889         lblNewLabel.setFont(new Font("Dialog", Font.PLAIN, 12));
14890         lblNewLabel.setBounds(10, 26, 123, 16);
14891         panel_1.add(lblNewLabel);
14892         textField = new JTextField();
14893         textField.setFont(new Font("Dialog", Font.PLAIN, 12));
14894         textField.setBounds(129, 19, 442, 28);
14895         panel_1.add(textField);
14896         textField.setColumns(10);
14897         JButton btnBrowse = new JButton("Browse");
14898         btnBrowse.setFont(new Font("Dialog", Font.PLAIN, 12));
14899         btnBrowse.setBounds(650, 12, 168, 29);
14900         btnBrowse.addActionListener(new ActionListener() {
```

```
14901     public void actionPerformed(ActionEvent arg0) {
14902         JFileChooser fileChooser = new JFileChooser();
14903         String startPath = System.getProperty("user.dir");
14904         fileChooser.setCurrentDirectory(new File(startPath));
14905         fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
14906         FileNameExtensionFilter filter = new FileNameExtensionFilter("txt Files", "txt");
14907         fileChooser.setFileFilter(filter);
14908         int index = fileChooser.showDialog(null, "Open File");
14909         if (index == JFileChooser.APPROVE_OPTION)
14910         {
14911             textField.setText(fileChooser.getSelectedFile().getAbsolutePath());
14912         }
14913     }
14914 });
14915 panel_1.add(btnBrowse);
14916 JButton btnSelectMutantOutput = new JButton("Select Mutant Output");
14917 btnSelectMutantOutput.setFont(new Font("Dialog", Font.PLAIN, 12));
14918 btnSelectMutantOutput.setBounds(650, 51, 168, 29);
14919 panel_1.add(btnSelectMutantOutput);
14920 JButton btnRun = new JButton("Run");
14921 btnRun.setFont(new Font("Dialog", Font.PLAIN, 12));
14922 btnRun.setBounds(650, 90, 168, 29);
14923 panel_1.add(btnRun);
14924 JButton btnReset = new JButton("Reset");
14925 btnReset.setFont(new Font("Dialog", Font.PLAIN, 12));
14926 btnReset.setBounds(650, 129, 168, 29);
14927 panel_1.add(btnReset);
14928 JScrollPane scrollPane_1 = new JScrollPane();
14929 scrollPane_1.setBounds(129, 57, 442, 101);
14930 panel_1.add(scrollPane_1);
14931 JList fileList_1 = new JList(listModel);
14932 fileList_1.setFont(new Font("Dialog", Font.PLAIN, 12));
14933 scrollPane_1.setViewportView(fileList_1);
14934 fileList_1.setBorder(null);
14935 JLabel lblMutantOutput = new JLabel("Mutant Output:\r\n");
14936 lblMutantOutput.setFont(new Font("Dialog", Font.PLAIN, 12));
14937 lblMutantOutput.setBounds(10, 105, 123, 16);
14938 panel_1.add(lblMutantOutput);
14939 JPanel panel = new JPanel();
14940 panel.setBorder(new TitledBorder(UIManager.getBorder("TitledBorder.border"), "Result",
14941 TitledBorder.CENTER, TitledBorder.TOP, null, null));
14942 panel.setBounds(561, 190, 315, 220);
14943 add(panel);
14944 panel.setLayout(null);
14945 JLabel lblNewLabel_1 = new JLabel("Number of Killed Mutant:\r\n");
14946 lblNewLabel_1.setFont(new Font("Dialog", Font.PLAIN, 12));
14947 lblNewLabel_1.setBounds(34, 54, 184, 15);
14948 panel.add(lblNewLabel_1);
14949 textField_1 = new JTextField();
14950 textField_1.setEditable(false);
```

```
14951     textField_1.setBounds(185, 51, 66, 21);
14952     panel.add(textField_1);
14953     textField_1.setColumns(10);
14954     textField_2 = new JTextField();
14955     textField_2.setEditable(false);
14956     textField_2.setColumns(10);
14957     textField_2.setBounds(185, 103, 66, 21);
14958     panel.add(textField_2);
14959     JLabel lblTotalMutantNumber = new JLabel("Total Number of Mutant:\r\n");
14960     lblTotalMutantNumber.setFont(new Font("Dialog", Font.PLAIN, 12));
14961     lblTotalMutantNumber.setBounds(34, 106, 184, 15);
14962     panel.add(lblTotalMutantNumber);
14963     textField_3 = new JTextField();
14964     textField_3.setEditable(false);
14965     textField_3.setColumns(10);
14966     textField_3.setBounds(185, 159, 66, 21);
14967     panel.add(textField_3);
14968     JLabel lblMutantScore = new JLabel("Mutant Score:\r\n");
14969     lblMutantScore.setFont(new Font("Dialog", Font.PLAIN, 12));
14970     lblMutantScore.setBounds(34, 162, 184, 15);
14971     panel.add(lblMutantScore);
14972     JPanel panel_2 = new JPanel();
14973     panel_2.setBorder(new TitledBorder(null, "Output", TitledBorder.CENTER,
14974     TitledBorder.TOP, null, null));
14975     panel_2.setBounds(22, 190, 529, 220);
14976     add(panel_2);
14977     panel_2.setLayout(null);
14978     JTable resultTable_1 = new JTable(model);
14979     resultTable_1.setColumnSelectionAllowed(true);
14980     resultTable_1.setBounds(346, 253, 204, 190);
14981     resultTable_1.setShowHorizontalLines (true);
14982     resultTable_1.setShowVerticalLines (true);
14983     add(resultTable_1);
14984     JScrollPane scrollPane_2 = new JScrollPane(resultTable_1);
14985     scrollPane_2.setBounds(10, 20, 509, 190);
14986     panel_2.add(scrollPane_2);
14987     btnReset.addActionListener(new ActionListener() {
14988         public void actionPerformed(ActionEvent e) {
14989             textField.setText(null);
14990             model.setRowCount(0);
14991             listModel.removeAllElements();
14992             textField_3.setText(null);
14993             textField_1.setText(null);
14994             textField_2.setText(null);
14995         }
14996     });
14997     btnRun.addActionListener(new ActionListener() {
14998         public void actionPerformed(ActionEvent e) {
14999             String originOutputPath = textField.getText();
15000             for(int i = 0; i < listModel.getSize(); i++) {
```

```

15001         File file = (File)listModel.get(i);
15002         String testcaseFilePath = file.getAbsolutePath();
15003         System.out.println(testcaseFilePath);
15004         compareTestCase(originOutputPath,testcaseFilePath,i+1);
15005     }
15006 }
15007 });
15008 btnSelectMutantOutput.addActionListener(new ActionListener() {
15009     public void actionPerformed(ActionEvent arg0) {
15010         JFileChooser fileChooser = new JFileChooser();
15011         fileChooser.setMultiSelectionEnabled(true);
15012         String startPath = System.getProperty("user.dir");
15013         fileChooser.setCurrentDirectory(new File(startPath));
15014         fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
15015         FileNameExtensionFilter filter = new FileNameExtensionFilter("txt Files", "txt");
15016         fileChooser.setFileFilter(filter);
15017         int index = fileChooser.showDialog(null, "Open File");
15018         if (index == JFileChooser.APPROVE_OPTION) {
15019             File[] files = fileChooser.getSelectedFiles();
15020             for(int i = 0; i < files.length; i++) {
15021                 listModel.addElement(files[i]);
15022             }
15023         }
15024     }
15025 });
15026 }
15027 private void compareTestCase(String originOutputPath,String testcaseFilePath, int i) {
15028     List<ExecutionMatrix> executionResults = Executor.execute(originOutputPath,
15029     testcaseFilePath);
15030     Object[] rowData = new Object[5];
15031     for(int j = 0; j < executionResults.size(); j++) {
15032         ExecutionMatrix er = executionResults.get(j);
15033         rowData[0] = i;
15034         rowData[1] = er.getMutantName();
15035         rowData[2] = er.getFDR();
15036         rowData[3] = er.getStatus();
15037         rowData[4] = er.getFinalStatus();
15038         System.out.println(i+"#" +er.getMutantName()+"#" +er.getFDR()+"#" +er.getStatus()+"#"
15039 +er.getFinalStatus());
15040         ms=rowData[4]=="KILLED"?ms+1:ms+0;
15041         mq++;
15042         java.text.DecimalFormat df =new java.text.DecimalFormat("0.00");
15043         String st=df.format(ms/mq);
15044         textField_3.setText(" "+st);
15045         textField_1.setText(" "+ms);
15046         textField_2.setText(" "+mq);
15047         model.insertRow(model.getRowCount(), rowData);
15048     }
15049 }
15050 }

```

```
15051 package com.ustb.bpel.view;
15052 import javax.swing.JFileChooser;
15053 import javax.swing.JPanel;
15054 import javax.swing.JLabel;
15055 import javax.swing.JTextArea;
15056 import javax.swing.JButton;
15057 import javax.swing.filechooser.FileNameExtensionFilter;
15058 import javax.swing.table.DefaultTableModel;
15059 import java.awt.event.ActionListener;
15060 import java.awt.event.ActionEvent;
15061 import java.io.BufferedReader;
15062 import java.io.File;
15063 import java.io.IOException;
15064 import java.util.List;
15065 import javax.swing.JTextField;
15066 import java.awt.Color;
15067 import javax.swing.border.TitledBorder;
15068 import java.awt.Font;
15069 import javax.swing.SwingConstants;
15070 import javax.swing.JSeparator;
15071 import javax.swing.DefaultListModel;
15072 import javax.swing.JScrollPane;
15073 import javax.swing.JList;
15074 import javax.swing.JTable;
15075 import alan.program.Configuration.WSDLParser;
15076 import alan.testcase.executor.*;
15077 import javax.swing.border.LineBorder;
15078 import java.awt.SystemColor;
15079 import javax.swing.border.BevelBorder;
15080 import javax.swing.UIManager;
15081 public class RunPanel extends JPanel {
15082     String bpelFilePath;
15083     String bpelfile;
15084     private JTextField textField;
15085     JTextArea textArea_1;
15086     private DefaultListModel listModel = null;
15087     private DefaultListModel listModel2 = null;
15088     private DefaultTableModel model = null;
15089     String endpoint;
15090     String operationName;
15091     private static Process process;
15092     private static BufferedReader br;
15093     public RunPanel() {
15094         setBorder(new BevelBorder(BevelBorder.LOWERED, null, null, null, null));
15095         initLayout();
15096         initControl();
15097     }
15098     private void initControl() {
15099     }
15100     private void initLayout() {
```



```
15101         setLayout(null);
15102         listModel = new DefaultListModel();
15103         listModel2 = new DefaultListModel();
15104         model = new DefaultTableModel();
15105         model.addColumn("No");
15106         model.addColumn("Begin Time");
15107         model.addColumn("End Time");
15108         model.addColumn("Input");
15109         model.addColumn("Expected Result");
15110         model.addColumn("Excuted Result");
15111         model.addColumn("Status");
15112         JSeparator separator = new JSeparator();
15113         separator.setBounds(0, 0, 2, 418);
15114         add(separator);
15115         separator.setOrientation(SwingConstants.VERTICAL);
15116         separator.setForeground(Color.WHITE);
15117         JPanel panel_1 = new JPanel();
15118         panel_1.setBorder(new TitledBorder(null, "Run Test Case", TitledBorder.CENTER,
15119 TitledBorder.TOP, null, null));
15120         panel_1.setBounds(12, 6, 869, 412);
15121         add(panel_1);
15122         panel_1.setLayout(null);
15123         JButton btnNewButton_1 = new JButton("Browse");
15124         btnNewButton_1.setBounds(646, 23, 145, 23);
15125         panel_1.add(btnNewButton_1);
15126         btnNewButton_1.setFont(new Font("Dialog", Font.PLAIN, 12));
15127         JLabel lblBpelFile = new JLabel("BPEL Build File:");
15128         lblBpelFile.setBounds(31, 25, 138, 19);
15129         panel_1.add(lblBpelFile);
15130         lblBpelFile.setFont(new Font("Dialog", Font.PLAIN, 12));
15131         textField = new JTextField();
15132         textField.setBounds(179, 25, 429, 23);
15133         panel_1.add(textField);
15134         textField.setColumns(10);
15135         JScrollPane scrollPane_3 = new JScrollPane();
15136         scrollPane_3.setBounds(179, 79, 429, 121);
15137         panel_1.add(scrollPane_3);
15138         JList list1 = new JList(listModel2);
15139         scrollPane_3.setViewportView(list1);
15140         JButton btnSelectMutant = new JButton("Select Mutant Files");
15141         btnSelectMutant.setBounds(646, 78, 145, 29);
15142         panel_1.add(btnSelectMutant);
15143         btnSelectMutant.addActionListener(new selectMutantAction());
15144         btnSelectMutant.setFont(new Font("Dialog", Font.PLAIN, 12));
15145         JLabel lblBpelMutantFiles = new JLabel("BPEL Mutant Files:");
15146         lblBpelMutantFiles.setBounds(31, 63, 138, 19);
15147         panel_1.add(lblBpelMutantFiles);
15148         lblBpelMutantFiles.setFont(new Font("Dialog", Font.PLAIN, 12));
15149         JScrollPane scrollPane_1 = new JScrollPane();
15150         scrollPane_1.setBounds(179, 224, 429, 131);
```

```

15151         panel_1.add(scrollPane_1);
15152         JList list = new JList(listModel);
15153         scrollPane_1.setViewportView(list);
15154         JButton btnSelectTestcase = new JButton("Select Test Cases");
15155         btnSelectTestcase.setBounds(649, 221, 142, 29);
15156         panel_1.add(btnSelectTestcase);
15157         btnSelectTestcase.setFont(new Font("Dialog", Font.PLAIN, 12));
15158         JLabel lblTestcaseFiles = new JLabel("Test Case Files:");
15159         lblTestcaseFiles.setFont(new Font("Dialog", Font.PLAIN, 12));
15160         lblTestcaseFiles.setBounds(27, 219, 111, 19);
15161         panel_1.add(lblTestcaseFiles);
15162         JButton btnDeployProcess = new JButton("Run Test Cases");
15163         btnDeployProcess.setFont(new Font("Dialog",
15164 Font.PLAIN, 12));
15165         btnDeployProcess.setBounds(646, 281, 145, 23);
15166         panel_1.add(btnDeployProcess);
15167         JButton btnClear = new JButton("Clear\r\n");
15168         btnClear.setFont(new Font("Dialog", Font.PLAIN, 12));
15169         btnClear.setBounds(646, 332, 145, 23);
15170         btnClear.addActionListener(new ActionListener() {
15171             public void actionPerformed(ActionEvent e) {
15172                 listModel2.removeAllElements();
15173                 listModel.removeAllElements();
15174                 textField.setText(null);
15175             }
15176         });
15177         panel_1.add(btnClear);
15178         btnNewButton_1.addActionListener(new
15179 selectOriginalBPELAction());
15180         btnDeployProcess.addActionListener(new
15181 DeployAction());
15182         btnSelectTestcase.addActionListener(new SelectTestcaseAction());
15183     }
15184     private class selectOriginalBPELAction implements ActionListener {
15185         public void actionPerformed(ActionEvent e) {
15186             JFileChooser jChooser = new JFileChooser();
15187             String startPath = System.getProperty("user.dir");
15188             jChooser.setCurrentDirectory(new File(startPath));
15189             jChooser.setDialogTitle("please select buile.xml file");
15190             int index = jChooser.showDialog(null, "Open File");
15191             if (index == JFileChooser.APPROVE_OPTION) {
15192                 bpelFilePath = jChooser.getSelectedFile().getAbsolutePath();
15193                 textField.setText(bpelFilePath);
15194                 System.out.println(bpelFilePath);
15195             }
15196             WSDLParser.RunConfiguration(bpelFilePath);
15197             operationName = WSDLParser.configuration.getOperationName();
15198             endpoint = WSDLParser.configuration.getEndPoint();
15199         }
15200     }

```

```

15201 private class SelectTestcaseAction implements ActionListener {
15202     public void actionPerformed(ActionEvent e) {
15203         JFileChooser fileChooser = new JFileChooser();
15204         fileChooser.setMultiSelectionEnabled(true);
15205         String startPath = System.getProperty("user.dir");
15206         fileChooser.setCurrentDirectory(new File(startPath));
15207         fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
15208         FileNameExtensionFilter filter = new FileNameExtensionFilter(
15209             "txt Files", "txt");
15210         fileChooser.setFileFilter(filter);
15211         int index = fileChooser.showDialog(null, "Open File");
15212         if (index == JFileChooser.APPROVE_OPTION) {
15213             File[] files = fileChooser.getSelectedFiles();
15214             for (int i = 0; i < files.length; i++) {
15215                 listModel.addElement(files[i]);
15216             }
15217         }
15218     }
15219 }
15220 private class selectMutantAction implements ActionListener {
15221     @SuppressWarnings("unchecked")
15222     public void actionPerformed(ActionEvent e) {
15223         JFileChooser fileChooser = new JFileChooser();
15224         fileChooser.setMultiSelectionEnabled(true);
15225         String startPath = System.getProperty("user.dir");
15226         fileChooser.setCurrentDirectory(new File(startPath));
15227         fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
15228         FileNameExtensionFilter filter = new FileNameExtensionFilter(
15229             "bpel Files", "bpel");
15230         fileChooser.setFileFilter(filter);
15231         int index = fileChooser.showDialog(null, "Open File");
15232         if (index == JFileChooser.APPROVE_OPTION) {
15233             File[] files = fileChooser.getSelectedFiles();
15234             for (int i = 0; i < files.length; i++) {
15235                 listModel2.addElement(files[i]);
15236             }
15237         }
15238     }
15239 }
15240 private class DeployAction implements ActionListener {
15241     public void actionPerformed(ActionEvent e) {
15242         for (int i=0;i<listModel2.getSize();i++){
15243             String fileString =(String) listModel2.getElementAt(i).toString();
15244             System.out.println(fileString);
15245             try {
15246                 FileCopy.copyFile(fileString, Mainform.BPELpath.getText());
15247                 deployThread pThread=new deployThread(bpelFilePath);
15248                 pThread.start();
15249                 pThread.join();
15250                 for (int j = 0; j < listModel.getSize(); j++) {

```

```
15251         File file = (File) listModel.get(j);
15252         String testcaseFilePath = file.getAbsolutePath();
15253         String
15254         MutantName=fileString.substring(fileString.lastIndexOf("\\")+1,fileString.lastIndexOf(".")) ;
15255         System.out.println(MutantName);
15256         runThread rThread=new runThread(endpoint, operationName,
15257         testcaseFilePath,MutantName);
15258         rThread.start();
15259     }
15260     } catch (IOException | InterruptedException e1) {
15261         e1.printStackTrace();
15262     }
15263 }
15264 }
15265 }
15266 }
15267 package com.ustb.bpel.view;
15268 import javax.swing.JPanel;
15269 import javax.swing.BorderFactory;
15270 import javax.swing.JFileChooser;
15271 import javax.swing.JTextField;
15272 import javax.swing.JButton;
15273 import javax.swing.JComboBox;
15274 import javax.swing.JLabel;
15275 import javax.swing.DefaultComboBoxModel;
15276 import java.awt.BorderLayout;
15277 import java.awt.Dimension;
15278 import java.awt.GridBagConstraints;
15279 import java.awt.GridBagLayout;
15280 import java.awt.Font;
15281 import java.util.ArrayList;
15282 import java.util.Iterator;
15283 import java.util.List;
15284 import org.dom4j.Document;
15285 import org.dom4j.Element;
15286 import alan.testcase.generator.Message;
15287 import alan.testcase.generator.Param;
15288 import alan.testcase.generator.Parameter;
15289 import alan.testcase.generator.TestCaseGenerator;
15290 import alan.testcase.generator.TestCaseInfo;
15291 import java.awt.event.ActionListener;
15292 import java.awt.event.ActionEvent;
15293 import java.io.File;
15294 import javax.swing.border.LineBorder;
15295 import java.awt.SystemColor;
15296 import java.awt.Color;
15297 import javax.swing.border.TitledBorder;
15298 import javax.swing.UIManager;
15299 public class InputDataGenerator extends JPanel {
15300     private JTextField filePathTextField;
```

```
15301 private JButton openButton;
15302 private JComboBox messageComboBox = null;
15303 private DefaultComboBoxModel defaultComboBoxModel = null;
15304 JLabel[] nameLabels = null;
15305 JTextField[] textFields1 = null;
15306 JTextField[] textFields2 = null;
15307 JLabel[] typeLabels = null;
15308 JComboBox[] signalComboBox1 = null;
15309 JComboBox[] signalComboBox2 = null;
15310 private JLabel testCaseNumberLabel = null;
15311 private JTextField testCaseNumberTextField = null;
15312 private JTextField savePathTextField = null;
15313 private JLabel expectedPathLabel = null;
15314 private JTextField expectedPathTextField = null;
15315 private JLabel expectedResultLabel = null;
15316 private JTextField expectedResultTextField = null;
15317 private JComboBox expectedResultTypeComboBox = null;
15318 private JButton browseButton = null;
15319 private JButton generateButton = null;
15320 private JButton clearButton = null;
15321 private List<Message> messages = new ArrayList<Message>();
15322 String[] signal = new String[] { "<", "<=", "!=", "=" };
15323 String[] signalStr = new String[] { "!=", "=" };
15324 String[] options = new String[] { "string", "char", "int", "double",
15325     "float" };
15326 public InputDataGenerator() {
15327     setBorder(new LineBorder(SystemColor.controlDkShadow, 2));
15328     this.setLayout(null);
15329     initControl();
15330     initLayout();
15331 }
15332 private void initControl() {
15333     defaultComboBoxModel = new DefaultComboBoxModel();
15334     messageComboBox = new JComboBox(defaultComboBoxModel);
15335     messageComboBox.setFont(new Font("Dialog", Font.PLAIN, 12));
15336     messageComboBox.setBounds(525, 20, 183, 21);
15337     expectedPathLabel = new JLabel("Expected Path: ");
15338     expectedPathTextField = new JTextField(18);
15339     savePathTextField = new JTextField(20);
15340     savePathTextField.setFont(new Font("Dialog", Font.PLAIN, 12));
15341     savePathTextField.setBounds(10, 23, 206, 21);
15342     savePathTextField.setText(getOutputPath());
15343     browseButton = new JButton("Browse");
15344     browseButton.setFont(new Font("Dialog", Font.PLAIN, 12));
15345     browseButton.addActionListener(new ActionListener() {
15346         public void actionPerformed(ActionEvent e) {
15347             JFileChooser fileChooser = new JFileChooser();
15348             String startPath = System.getProperty("user.dir");
15349             fileChooser.setCurrentDirectory(new File(startPath));
15350             fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
```

```
15351         int index = fileChooser.showDialog(null, "Open File");
15352         if (index == JFileChooser.APPROVE_OPTION) {
15353             savePathTextField.setText(fileChooser.getSelectedFile()
15354                                     .getAbsolutePath());
15355         }
15356     }
15357 });
15358 browseButton.setBounds(226, 22, 105, 23);
15359 clearButton = new JButton("Clear");
15360 clearButton.setFont(new Font("Dialog", Font.PLAIN, 12));
15361 clearButton.addActionListener(new ActionListener() {
15362     public void actionPerformed(ActionEvent e) {
15363         testCaseNumberTextField.setText(null);
15364         filePathTextField.setText(null);
15365         expectedResultTextField.setText(null);
15366     }
15367 });
15368 clearButton.setBounds(189, 19, 112, 23);
15369 openButton = new JButton("Browse\r\n");
15370 openButton.setFont(new Font("Dialog", Font.PLAIN, 12));
15371 openButton.addActionListener(new ActionListener() {
15372     public void actionPerformed(ActionEvent e) {
15373         JFileChooser fileChooser = new JFileChooser();
15374         String startPath = System.getProperty("user.dir");
15375         fileChooser.setCurrentDirectory(new File(startPath));
15376         fileChooser.setSelectionMode(JFileChooser.FILES_ONLY);
15377         int index = fileChooser.showDialog(null, "Open File");
15378         if (index == JFileChooser.APPROVE_OPTION) {
15379             filePathTextField.setText(fileChooser.getSelectedFile()
15380                                     .getAbsolutePath());
15381         }
15382         ParseWSDL();
15383     }
15384 });
15385 openButton.setBounds(343, 20, 141, 23);
15386 filePathTextField = new JTextField(30);
15387 filePathTextField.setBounds(166, 20, 150, 21);
15388 generateButton = new JButton("Generate");
15389 generateButton.setFont(new Font("Dialog", Font.PLAIN, 12));
15390 generateButton.setBounds(46, 19, 96, 23);
15391 messageComboBox.addActionListener(new SelectAction());
15392 generateButton.addActionListener(new GenerateAction());
15393 }
15394 private void initLayout() {
15395     this.setLayout(null);
15396     setLayout(null);
15397     this.add(getOpenPanel());
15398     this.add(getInputParameterPanel());
15399     {
15400     }
```

```
15401     }
15402     private JPanel getOpenPanel() {
15403         JPanel panel = new JPanel();
15404         panel.setBounds(10, 10, 727, 56);
15405         panel.setBorder(new TitledBorder(UIManager.getBorder("TitledBorder.border"),
15406 "Specify File", TitledBorder.CENTER, TitledBorder.TOP, null, null));
15407         panel.setLayout(null);
15408         panel.add(filePathTextField);
15409         panel.add(openButton);
15410         panel.add(messageComboBox);
15411         {
15412             JLabel lblNewLabel_1 = new JLabel("BPEL's WSDL");
15413             lblNewLabel_1.setFont(new Font("Dialog", Font.PLAIN, 12));
15414             lblNewLabel_1.setBounds(24, 23, 132, 15);
15415             panel.add(lblNewLabel_1);
15416         }
15417         return panel;
15418     }
15419     private JPanel getExpectedOutputPanel() {
15420         JPanel panel = new JPanel();
15421         panel.setBounds(10, 22, 341, 52);
15422         panel.setBorder(BorderFactory.createTitledBorder("Expected Output"));
15423         panel.setLayout(null);
15424         testCaseNumberTextField = new JTextField(18);
15425         testCaseNumberTextField.setBounds(165, 24, 48, 18);
15426         panel.add(testCaseNumberTextField);
15427         {
15428             JLabel lblNewLabel = new JLabel("Number of Test Case: ");
15429             lblNewLabel.setFont(new Font("Dialog", Font.PLAIN, 12));
15430             lblNewLabel.setBounds(10, 27, 145, 15);
15431             panel.add(lblNewLabel);
15432         }
15433     return panel;
15434     }
15435     private JPanel getBrowsePanel() {
15436         JPanel panel = new JPanel();
15437         panel.setBounds(10, 84, 341, 61);
15438         panel.setBorder(BorderFactory.createTitledBorder("Save Path"));
15439         panel.setLayout(null);
15440         panel.add(savePathTextField);
15441         panel.add(browseButton);
15442         return panel;
15443     }
15444     private JPanel getRightPanel() {
15445         JPanel panel = new JPanel();
15446         panel.setBounds(332, 28, 361, 257);
15447         panel.setLayout(null);
15448         panel.add(getBrowsePanel());
15449         panel.add(getButtonPanel());
15450         panel.add(getExpectedOutputPanel());
```

```
15451         return panel;}
15452     private void ParseWSDL() {
15453         String variable = "";
15454         String type = "";
15455         String midPath = "";
15456         String path = "";
15457         int index;
15458         String filePath = filePathTextField.getText();
15459         Mainform.textArea_1.append("Open the WSDL file:"+"---"+filePath+ "\r\n");
15460         Document document = alan.converter.core.XMLHelper.openXMLFile(filePath);
15461         Element root = document.getRootElement();
15462         for (Iterator iter = root.elementIterator(); iter.hasNext();) {
15463             Element element = (Element) iter.next();
15464             if (element.getName().equals("message")) {
15465                 Message m = new Message();
15466                 m.setName(element.attributeValue("name"));
15467                 for (Iterator ii = element.elementIterator(); ii.hasNext();) {
15468                     Element ele = (Element) ii.next();
15469                     Parameter p = new Parameter();
15470                     p.setName(ele.attributeValue("name"));
15471                     p.setType(getTypeFromXMLSchem(ele.attributeValue("type")));
15472                     m.getParameters().add(p);
15473                 }
15474                 defaultComboBoxModel.insertElementAt(m.getName(), 0);
15475                 messages.add(m);
15476             }
15477         }
15478         paint();
15479     }
15480     private JPanel getLeftPanel() {
15481         JPanel panel = new JPanel();
15482         panel.setBounds(22, 28, 300, 257);
15483         panel.setPreferredSize(new Dimension(300, 160));
15484         panel.setBorder(BorderFactory.createTitledBorder("Input Constraint"));
15485         panel.setLayout(new GridBagLayout());
15486         GridBagConstraints c = new GridBagConstraints();
15487         c.gridwidth = 1;
15488         c.gridheight = 1;
15489         c.gridx = 0;
15490         c.gridy = 0;
15491         if (nameLabels != null && textFields1 != null && textFields2 != null) {
15492             for (int i = 0; i < nameLabels.length; i++) {
15493                 panel.add(textFields1[i], c);
15494                 c.gridx++;
15495                 panel.add(signalComboBox1[i], c);
15496                 c.gridx++;
15497                 panel.add(nameLabels[i], c);
15498                 c.gridx++;
15499                 panel.add(signalComboBox2[i], c);
15500                 c.gridx++;
```



```
15501         panel.add(textFields2[i], c);
15502         c.gridx++;
15503         panel.add(typeLabels[i], c);
15504         c.gridx = 0;
15505         c.gridy++;
15506     }
15507 }
15508 return panel;
15509 }
15510 private JPanel getInputParameterPanel() {
15511     JPanel panel = new JPanel();
15512     panel.setBorder(new TitledBorder(null, "Output", TitledBorder.CENTER,
15513 TitledBorder.TOP, null, null));
15514     panel.setBounds(10, 76, 727, 295);
15515     panel.setLayout(null);
15516     panel.add(getLeftPanel());
15517     panel.add(getRightPanel());
15518     return panel;
15519 }
15520 private void paint() {
15521     this.removeAll();
15522     this.add(getOpenPanel(), BorderLayout.NORTH);
15523     this.add(getInputParameterPanel(), BorderLayout.SOUTH);
15524 }
15525 private String getTypeFromXMLSchem(String type) {
15526     String ret = "";
15527     if (type.equals("xsd:string")) {
15528         ret = "string";
15529     } else if (type.equals("xsd:int")) {
15530         ret = "int";
15531     } else if (type.equals("xsd:double")) {
15532         ret = "double";
15533     } else if (type.equals("xsd:float")) {
15534         ret = "float";
15535     } else if (type.equals("xsd:char")) {
15536         ret = "char";
15537     }
15538     return ret;
15539 }
15540 private class SelectAction implements ActionListener {
15541     public void actionPerformed(ActionEvent e) {
15542         String message = (String) messageComboBox.getSelectedItem();
15543         int index = getIndexOfMessages(message);
15544         int length = messages.get(index).getParameters().size();
15545         nameLabels = new JLabel[length];
15546         textFields1 = new JTextField[length];
15547         signalComboBox1 = new JComboBox[length];
15548         signalComboBox2 = new JComboBox[length];
15549         textFields2 = new JTextField[length];
15550         typeLabels = new JLabel[length];
```

```

15551         for (int i = 0; i < length; i++) {
15552             Parameter p = (Parameter) messages.get(index).getParameters()
15553                 .get(i);
15554             if (p.getType().equals("string")) {
15555                 textFields1[i] = new JTextField(3);
15556                 textFields1[i].setEditable(false);
15557                 signalComboBox1[i] = new JComboBox(signalStr);
15558                 signalComboBox1[i].setEnabled(false);
15559                 nameLabels[i] = new JLabel(p.getName());
15560                 signalComboBox2[i] = new JComboBox(signalStr);
15561                 textFields2[i] = new JTextField(3);
15562                 typeLabels[i] = new JLabel(p.getType());
15563             } else {
15564                 textFields1[i] = new JTextField(3);
15565                 signalComboBox1[i] = new JComboBox(signal);
15566                 nameLabels[i] = new JLabel(p.getName());
15567                 signalComboBox2[i] = new JComboBox(signal);
15568                 textFields2[i] = new JTextField(3);
15569                 typeLabels[i] = new JLabel(p.getType());
15570             }
15571         }
15572         paint();
15573     }
15574 }
15575 private String getOutputPath() {
15576     return System.getProperty("user.dir") + "\\testcase\\" + "testcases.txt";
15577 }
15578 private int getIndexofMessages(String message) {
15579     int ret = -1;
15580     for (int i = 0; i < messages.size(); i++) {
15581         if (message.equals(messages.get(i).getName())) {
15582             ret = i; break;
15583         }
15584     }
15585     return ret;
15586 }
15587 private class GenerateAction implements ActionListener {
15588     public void actionPerformed(ActionEvent e) {
15589         TestCaseInfo tci = new TestCaseInfo();
15590         int number = nameLabels.length;
15591         for (int i = 0; i < number; i++) {
15592             String literal = "";
15593             String type = typeLabels[i].getText();
15594             if (type.equals("string")) {
15595                 literal += nameLabels[i].getText();
15596                 literal += signalComboBox2[i].getSelectedItem().toString();
15597                 literal += "\"" + textFields2[i].getText() + "\"";
15598             } else {
15599                 literal += textFields1[i].getText();
15600                 literal += signalComboBox1[i].getSelectedItem().toString();

```

```

15601         literal += nameLabels[i].getText();
15602         literal += signalComboBox2[i].getSelectedItem().toString();
15603         literal += textFields2[i].getText();
15604     }
15605     Param p = new Param();
15606     p.setLiteral(literal);
15607     p.setType(type);
15608     Mainform.textArea_1.append("The parameter:"+"---"+p.toString()+"\r\n");
15609     tci.getParams().add(p);
15610 }
15611 Param p = new Param();
15612 p.setLiteral("null");
15613 p.setType("string");
15614 tci.setExpectedResult(p);
15615 tci.setExpectedPath(expectedPathTextField.getText());
15616 tci.setTestCaseNumber(Integer.parseInt(testCaseNumberTextField
15617     .getText()));
15618 TestCaseGenerator.getInstance().generateRandomTestCase(tci,
15619     savePathTextField.getText());
15620 }
15621 }
15622 }
15623 package alan.testcase.executor;
15624 import java.io.File;
15625 import java.io.FileInputStream;
15626 import java.io.FileOutputStream;
15627 import java.io.IOException;
15628 public class FileCopy {
15629     public static void copyFile(String file1,String file2)throws IOException{
15630
15631         FileInputStream fis=new FileInputStream(file1);
15632         FileOutputStream fos=new FileOutputStream(file2);
15633         int temp;
15634         while((temp=fis.read())!=-1){
15635             fos.write(temp);
15636         }
15637         fis.close();
15638         fos.close();
15639         System.out.println("从"+file1+"到"+file2);
15640     }
15641 package alan.testcase.executor;
15642 import java.io.IOException;
15643 public class deployThread extends Thread{
15644     String bpelFilePath;
15645     public deployThread(String bpelFilePath){
15646         this.bpelFilePath=bpelFilePath;
15647     }
15648     public void run(){
15649         Runtime deployrt = Runtime.getRuntime();
15650         Process deployProcess;

```

```

15651         try {
15652             deployProcess = deployrt.exec("cmd /c start ant -f " + bpelFilePath + "\n" + "deploy");
15653             try {sleep(60000);
15654             } catch (InterruptedException e)
15655             {
15656                 e.printStackTrace();
15657             }
15658             try {
15659                 deployProcess.waitFor();
15660             } catch (InterruptedException e1) {
15661                 e1.printStackTrace();
15662             }
15663             int i = deployProcess.exitValue();
15664             if (i == 0) {
15665                 System.out.println("执行完成.");
15666             } else {
15667                 System.out.println("执行失败.");
15668             }
15669             deployProcess.destroy();
15670             deployProcess = null;
15671             } catch (IOException e2) {
15672                 e2.printStackTrace();
15673             }
15674             killProcess();
15675         }
15676         public void killProcess(){
15677             Runtime rt = Runtime.getRuntime();
15678             Process p = null;
15679             try {
15680                 rt.exec("cmd.exe /C start wmic process where name='cmd.exe' call terminate");
15681             } catch (IOException e)
15682             {
15683                 e.printStackTrace();
15684             }
15685         }
15686     }
15687 package alan.testcase.executor;
15688 import java.util.List;
15689 public class runThread extends Thread{
15690     String operationName;
15691     String endpoint;
15692     String testcaseFilePath;
15693     String MutantName;
15694     public runThread(String endpoint, String operationName,
15695                     String testcaseFilePath, String MutantName)
15696     {
15697         this.endpoint=endpoint;
15698         this.operationName=operationName;
15699         this.testcaseFilePath=testcaseFilePath;
15700         this.MutantName=MutantName;

```

```

15701     }
15702     public void run()
15703     {
15704         runBPELTestClient(endpoint, operationName, testcaseFilePath, MutantName);
15705         try {
15706             sleep(3000);
15707         } catch (InterruptedException e) {
15708             e.printStackTrace();
15709         }
15710     }
15711     private void runBPELTestClient(String endpoint, String operationName,
15712         String testcaseFilePath, String MutantName)
15713     {
15714         RunCase executor = new RunCase();
15715         List<ExecutionResult> executionResults = executor.execute(endpoint,
15716             operationName, testcaseFilePath, MutantName);
15717         Object[] rowData = new Object[7];
15718         for (int j = 0; j < executionResults.size(); j++) {
15719             ExecutionResult er = executionResults.get(j);
15720             rowData[0] = er.getNo();
15721             rowData[1] = er.getBeginTime();
15722             rowData[2] = er.getEndTime();
15723             rowData[3] = er.getInput();
15724             rowData[4] = er.getExpectedResult();
15725             rowData[5] = er.getExecutedResult();
15726             rowData[6] = er.getStatus();
15727         }
15728     }
15729 }
15730 package operator;
15731 import java.util.Arrays;
15732 import java.util.List; import org.dom4j.Element;
15733 import parser.BPELMutator;
15734 import parser.mu_List;
15735 public class EAA extends BPELMutator {
15736     List<String> arithList = Arrays.asList("+", "-", "*", "div", "mod");
15737     public EAA() {
15738         super();
15739     }
15740     public void visit(Element element) {
15741         if (element.getName().equals("condition"))
15742             || element.getName().equals("transitionCondition")
15743             || element.getName().equals("joinCondition")
15744             || element.getName().equals("until")
15745             || element.getName().equals("for")
15746             || element.getName().equals("repeatEvery")
15747             || element.getName().equals("startCounterValue")
15748             || element.getName().equals("finalCounterValue")
15749             || element.getName().equals("branches")
15750             || element.getName().equals("from")

```

```

15751         || element.getName().equals("to")) {
15752             String expression = element.getText();
15753             int index = -1;
15754             if (expression.contains("+"))
15755                 index = 0;
15756             else if (expression.contains("-"))
15757                 index = 1;
15758             else if (expression.contains("*"))
15759                 index = 2;
15760             else if (expression.contains("/"))
15761                 index = 3;
15762             else if (expression.contains("%"))
15763                 index = 4;
15764             if (index >= 0) {
15765                 for (int i = 0; i < arithList.size(); i++) {
15766                     if (i != index) {
15767                         muID++;
15768                         mu_List mu_List = new mu_List("EAA", element,
15769                             new EAA_Writer(element, arithList.get(index),
15770                                 arithList.get(i)), muID);
15771                         MutantsList.add(mu_List);
15772                     }
15773                 }
15774             }
15775             } else
15776                 super.visit(element);
15777         }
15778     }
15779     package operator;
15780     import org.dom4j.Element;
15781     import parser.BPELMutator;
15782     import parser.mu_List;
15783     public class XMC extends BPELMutator {
15784         public XMC()
15785         {
15786             super();
15787         }
15788         public void visit(Element element) {
15789             if (element.getName().equals("compensationHandler")) {
15790                 System.out.println("find " + element.getName());
15791                 muID++;
15792                 mu_List mu_List = new mu_List("XMC", element, new XMC_Writer(
15793                     element), muID);
15794                 MutantsList.add(mu_List);
15795             }
15796             else{
15797                 super.visit(element);
15798             }
15799         }
15800     }

```