package generate;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.ArrayList;

import java.util.List;

import parser.BPELMutator;

import parser.mu\_List;

import utils.XMLWriter;

public class FirstOrder extends BPELMutator{

private List<String[]> firstOrdermuList = new ArrayList<String[]>();

public FirstOrder()

{

this.firstOrderGenerate();

}

public List<String[]> getFirstOrdermuList() {

return firstOrdermuList;

}

private void firstOrderGenerate() {

if(document == null)

{

System.out.println("document is null");

return;

}

String f\_name = null;

for(int i = 0;i<MutantsList.size();i++)

{

mu\_List mu = MutantsList.get(i);

f\_name = getMuFileName(mu.getOperator(),mu.getMuID());

String f\_path = getAbsoluteMuFilePath(f\_name);

firstOrdermuList.add(new String[]{f\_name.substring(0,f\_name.length()-5), f\_path});

PrintWriter out;

try {

out = getPrintWriter(f\_name);

XMLWriter writer = mu.getWriter();

writer.setWriter(out);

writer.write(document);

writer.flush();

writer.close();

} catch (IOException e) {

System.err.println("Fail to create file " + f\_name);

}

}

MutantsList.clear();

}

}

package generate;

import java.util.List;

import com.ustb.bpel.view.Mainform;

public class GenerateFacade {

int algorithmIndex;

List<String[]> mutantsList;

public GenerateFacade(){}

public GenerateFacade(int algorithmIndex)

{

this.algorithmIndex = algorithmIndex;

}

public void firstGenerate(){

FirstOrder first = new FirstOrder();

mutantsList = first.getFirstOrdermuList();

mutantsListPrint();

}

public void secondGenerate(){

SecondOrder second = new SecondOrder(algorithmIndex);

mutantsList = second.getSecondOrdermuList();

mutantsListPrint();

}

private void mutantsListPrint() {

for(int i = 0;i<mutantsList.size();i++)

{

System.out.println(mutantsList.get(i)[0]);

Mainform.MutantlogArea.append("Generatethe mutant :"

+"\*\*\*\*"+mutantsList.get(i)[0]+"\n");

}

System.out.println("\nThe total number of mutants is " + mutantsList.size());

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

}

public List<String[]> getMutantsList() {

return mutantsList;

}

}

package generate;

import java.io.IOException;

import org.dom4j.Element;

import org.dom4j.Node;

import parser.mu\_List;

import utils.XMLWriter;

public class SecondOrder\_Writer extends XMLWriter{

Node original1;

Node original2;

mu\_List mu1;

mu\_List mu2;

XMLWriter anoWriter;

public SecondOrder\_Writer(mu\_List mu1, mu\_List mu2){

this.mu1 = mu1;

this.mu2 = mu2;

this.original1 = mu1.getNode();

this.original2 = mu2.getNode();

}

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++) {

int j = MutantsList.size() - i - 1;

this.mu1 = MutantsList.get(i);

this.mu2 = MutantsList.get(j);

OutputToFile();

}

if (MutantsList.size() % 2 != 0) {

this.mu1 = MutantsList.get(MutantsList.size() / 2);

this.mu2 = MutantsList.get((MutantsList.size() + 1) / 2);

OutputToFile();

}

MutantsList.clear();

}

private void secondOrderGenerate\_1() {

}

private void OutputToFile() {

if (document == null) {

System.out.println("document is null");

return;

}

String f\_name = null;

f\_name = getMuFileName(mu1.getOperator(), mu1.getMuID(),

mu2.getOperator(), mu2.getMuID());

String f\_path = getAbsoluteMuFilePath(f\_name);

secondOrdermuList.add(new String[] {

f\_name.substring(0, f\_name.length() - 5), f\_path });

PrintWriter out;

try {

out = getPrintWriter(f\_name);

SecondOrder\_Writer writer = new SecondOrder\_Writer(this.mu1,

this.mu2);

writer.setWriter(out);

writer.write(document);

writer.flush();

writer.close();

} catch (IOException e) {

System.err.println("Fail to create file " + f\_name);

}

}

}

package parser;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.io.Writer;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

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";

return mu\_filename;

}

public String getAbsoluteMuFilePath(String f\_name){

String muFilePath = file.getAbsolutePath() + "\\" + f\_name;

return muFilePath;

}

public String getMuFileName(String op\_name1,int muID1,String op\_name2,int muID2) {

String mu\_filename = op\_name1 + muID1 + "\_" + op\_name2 + muID2 + ".bpel";

return mu\_filename;

}

public PrintWriter getPrintWriter(String f\_name) throws IOException {

File outfile = new File(file, f\_name);

FileWriter fout = new FileWriter(outfile);

PrintWriter out = new PrintWriter(fout);

return out;

}

}

package parser;

import org.dom4j.Node;

import org.dom4j.Visitor;

public interface BPELVisitor extends Visitor{

public void visit(Node node);

public void visit(String s);

}

package parser;

import org.dom4j.Attribute;

import org.dom4j.CDATA;

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;

public class ConcreteVisitor implements BPELVisitor{

public void visit(Document document) {

}

public void visit(DocumentType documentType) {

}

public void visit(Element node) {

}

public void visit(Attribute node) {

}

public void visit(CDATA node) {

}

public void visit(Comment node) {

}

public void visit(Entity node) {

}

public void visit(Namespace namespace) {

}

public void visit(ProcessingInstruction node) {

}

public void visit(Text node) {

}

public void visit(Node node){

}

public void visit(String s){

}

}

package parser;

import utils.XMLWriter;

import org.dom4j.Node;

public class mu\_List {

private String operator;

private Node node;

private XMLWriter writer;

private int muID;

public mu\_List(String operator, Node node, XMLWriter writer, int muID){

this.operator = operator;

this.node = node;

this.writer = writer;

this.muID = muID;

}

public String getOperator() {

return operator;

}

public void setOperator(String operator) {

this.operator = operator;

}

public Node getNode() {

return node;

}

public void setNode(Node node) {

this.node = node;

}

public int getMuID() {

return muID;

}

public void setMuID(int muID) {

this.muID = muID;

}

public XMLWriter getWriter() {

return writer;

}

}

package parser;

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)

{

new Parser(path,operatorList);

GenerateFacade generateFacade = null;

if(flag == 0){

generateFacade = new GenerateFacade();

generateFacade.firstGenerate();

}

else if(flag == 1)

{

System.out.println(algorithm);

generateFacade = new GenerateFacade(algorithm);

generateFacade.secondGenerate();

}

List<String[]> mutantsList = generateFacade.getMutantsList();

return mutantsList;

}

}

package utils;

import java.io.File;

import org.dom4j.Document;

import org.dom4j.io.SAXReader;

public class XMLReader {

public static Document openXMLFile(String filePath) {

Document document = null;

SAXReader reader = new SAXReader();

try {

File file = new File(filePath);

document = reader.read(file);

} catch (Exception e) {

e.printStackTrace();

}

return document;

}

}

package operator;

import java.io.PrintWriter;

import org.dom4j.Attribute;

import org.dom4j.Element;

public class ACI\_Writer extends attReplace\_Writer {

public ACI\_Writer(Element original, Attribute attribute) {

super(original, attribute);

}

public ACI\_Writer(PrintWriter out) {

super(out);

}

}

package operator;

import parser.BPELMutator;

import parser.mu\_List;

public class ACI extends BPELMutator {

private Element element;

private Attribute previous;

private static int recordNo;

private int flag;

public ACI() {

super();

System.out.println("\*\*\*ACI");

this.recordNo = 1;

this.flag = 0;

}

public void visit(Attribute attribute) {

if (attribute.getName().equals("createInstance")

&& attribute.getValue().equals("yes")) {

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*find createInstance");

if (recordNo < 2) {

recordNo++;

this.previous = attribute;

flag = 1;

return;

}

if (flag == 1) {

muID++;

this.element = previous.getParent();

mu\_List mu\_List = new mu\_List("ACI", previous, new ACI\_Writer(

this.element, previous), muID);

MutantsList.add(mu\_List);

flag = 0;

}

muID++;

this.element = attribute.getParent();

mu\_List mu\_List = new mu\_List("ACI", attribute, new ACI\_Writer(

this.element, attribute), muID);

MutantsList.add(mu\_List);

} else

super.visit(attribute);

}

}

package operator;

import java.io.PrintWriter;

import org.dom4j.Element;

public class AEL\_Writer extends Remove\_writer {

public AEL\_Writer(Element original) {

super(original);

}

}

package operator;

import org.dom4j.Element;

import parser.BPELMutator;

import parser.mu\_List;

public class AEL extends BPELMutator {

public AEL() {

super();

System.out.println("\*\*\*AEL");

}

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("AEL", element, new AEL\_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("AEL", element, new AEL\_Writer(

element), muID);

MutantsList.add(mu\_List);

}

} else

super.visit(element);

}

}

package operator;

import java.io.PrintWriter;

import org.dom4j.Attribute;

import org.dom4j.Element;

import utils.XMLWriter;

public class AFP\_Writer extends attReplace\_Writer{

public AFP\_Writer(Element original, Attribute attribute) {

super(original, attribute);

}

public AFP\_Writer(PrintWriter out) {

super(out);

}

}

package operator;

import org.dom4j.Attribute;

import org.dom4j.Element;

import parser.BPELMutator;

import parser.mu\_List;

public class AFP extends BPELMutator {

private Element att\_element;

public AFP() {

super();

System.out.println("\*\*\*AFP");

}

public void visit(Attribute attribute) {

if (attribute.getName().equals("parallel")

&& attribute.getValue().equals("no")) {

this.att\_element = attribute.getParent();

if (this.att\_element.getName().equals("forEach")) {

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*find parallel");

muID++;

mu\_List mu\_List = new mu\_List("AFP", attribute, new AFP\_Writer(

this.att\_element, attribute), muID);

MutantsList.add(mu\_List);

}

} else

super.visit(attribute);

}

}

package operator;

import java.io.PrintWriter;

import org.dom4j.Element;

public class AIE\_Writer extends Remove\_writer{

public AIE\_Writer(Element original) {

super(original);

}

public AIE\_Writer(PrintWriter out){

super(out);

}

}

package operator;

import org.dom4j.Element;

import parser.BPELMutator;

import parser.mu\_List;

public class AIE extends BPELMutator {

public AIE() {

super();

System.out.println("\*\*\*AIE");

}

public void visit(Element element) {

if (element.getName().equals("elseif")

|| element.getName().equals("else")) {

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*find if");

muID++;

mu\_List mu\_List = new mu\_List("AIE", element, new AIE\_Writer(

element), muID);

MutantsList.add(mu\_List);

} else

super.visit(element);

}

}

package operator;

import java.io.PrintWriter;

import org.dom4j.Attribute;

import org.dom4j.Element;

public class AIS\_Writer extends attReplace\_Writer{

public AIS\_Writer(Element original, Attribute attribute) {

super(original, attribute);

}

public AIS\_Writer(PrintWriter out) {

super(out);

}

}

package operator;

import org.dom4j.Attribute;

import org.dom4j.Element;

import parser.BPELMutator;

import parser.mu\_List;

public class AIS extends BPELMutator {

private Element att\_element;

public AIS() {

super();

System.out.println("\*\*\*AIS");

}

public void visit(Attribute attribute) {

if (attribute.getName().equals("isolated")

&& attribute.getValue().equals("yes")) {

this.att\_element = attribute.getParent();

if (this.att\_element.getName().equals("scope")) {

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*find isolated");

muID++;

mu\_List mu\_List = new mu\_List("AIS", attribute, new AIS\_Writer(

this.att\_element, attribute), muID);

MutantsList.add(mu\_List);

}

} else

super.visit(attribute);

}

}

package operator;

import java.io.PrintWriter;

import org.dom4j.Element;

public class AJC\_Writer extends Remove\_writer{

public AJC\_Writer(Element original) {

super(original);

}

public AJC\_Writer(PrintWriter out) {

super(out);

}

}

package operator;

import org.dom4j.Element;

import parser.BPELMutator;

import parser.mu\_List;

public class AJC extends BPELMutator {

public AJC() {

super();

System.out.println("\*\*\*AJC");

}

public void visit(Element element) {

if (element.getName().equals("joinCondition")) {

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*find joinCondition");

muID++;

mu\_List mu\_List = new mu\_List("AJC", element, new AJC\_Writer(

element), muID);

MutantsList.add(mu\_List);

} else

super.visit(element);

}

}

package operator;

import java.io.PrintWriter;

import org.dom4j.Element;

public class APA\_Writer extends Remove\_writer{

public APA\_Writer(Element original) {

super(original);

}

public APA\_Writer(PrintWriter out) {

super(out);

}

}

package operator;

import org.dom4j.Element;

import parser.BPELMutator;

import parser.mu\_List;

public class APA extends BPELMutator {

public APA() {

super();

System.out.println("\*\*\*APA");

}

public void visit(Element element) {

if (element.getName().equals("onAlarm")) {

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*find onAlarm"); 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;

import org.dom4j.Element;

import org.dom4j.Namespace;

import org.dom4j.Node;

import parser.mu\_List;

import utils.XMLWriter;

public class ASF\_Writer extends XMLWriter {

private Element original;

public ASF\_Writer(Element original) {

this.original = original;

}

public ASF\_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 = "flow";

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 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")) {

this.conditonElement = (Element) node;

} else if (element.getName().equals("repeatUntil")) {

writeNode(node);

}

}

}

if (trim) {

Text lastTextNode = null;

StringBuffer buff = null;

boolean textOnly = true;

for (int i = 0, size = element.nodeCount(); i < size; i++) {

Node node = element.node(i);

if (!node.getName().equals("condition")) {

if (node instanceof Text) {

if (lastTextNode == null) {

lastTextNode = (Text) node;

} else {

if (buff == null) {

buff = new StringBuffer(

lastTextNode.getText());

}

buff.append(((Text) node).getText());

}

} else {

if (!textOnly && format.isPadText()) {

char firstChar = 'a';

if (buff != null) {

firstChar = buff.charAt(0);

} else if (lastTextNode != null) {

firstChar = lastTextNode.getText()

.charAt(0);

}

if (Character.isWhitespace(firstChar)) {

writer.write(PAD\_TEXT);

}

}

if (lastTextNode != null) {

if (buff != null) {

writeString(buff.toString());

buff = null;

} else {

writeString(lastTextNode.getText());

}

if (format.isPadText()) {

char lastTextChar = 'a';

if (buff != null) {

lastTextChar = buff.charAt(buff

.length() - 1);

} else if (lastTextNode != null) {

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");