package edu.buaa.sei.datamodel;

import edu.buaa.sei.run.Publisher;

public class Dependency {

private Publisher srcPublisher;

private Publisher dstPublisher;

private SendData sendData;

private double time;

private double reliability;

private LostPackage lostPackage;

public double getReliability() {

return reliability;

}

public void setReliability(double reliability) {

this.reliability = reliability;

}

public SendData getSendData() {

return sendData;

}

public void setSendData(SendData sendData) {

this.sendData = sendData;

}

public double getTime() {

return time;

}

public void setTime(double time) {

this.time = time;

}

public LostPackage getLostPackage() {

return lostPackage;

}

public void setLostPackage(LostPackage lostPackage) {

this.lostPackage = lostPackage;

}

public Publisher getSrcPublisher() {

return srcPublisher;

}

public void setSrcPublisher(Publisher srcPublisher) {

this.srcPublisher = srcPublisher;

}

public Publisher getDstPublisher() {

return dstPublisher;

}

public void setDstPublisher(Publisher dstPublisher) {

this.dstPublisher = dstPublisher;

}

}

package edu.buaa.sei.datamodel;

public class LostPackage {

private int firstLostPackage;

private int secondLostPackage;

private String baseDependencyId;

public int getFirstLostPackage() {

return firstLostPackage;

}

public void setFirstLostPackage(int firstLostPackage) {

this.firstLostPackage = firstLostPackage;

}

public int getSecondLostPackage() {

return secondLostPackage;

}

public void setSecondLostPackage(int secondLostPackage) {

this.secondLostPackage = secondLostPackage;

}

public String getBaseDependencyId() {

return baseDependencyId;

}

public void setBaseDependencyId(String baseDependencyId) {

this.baseDependencyId = baseDependencyId;

}

}

package edu.buaa.sei.datamodel;

public class Message {

public String title;

public String name;

public String id;

public double time;

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public double getTime() {

return time;

}

public void setTime(double time) {

this.time = time;

}

}

package edu.buaa.sei.datamodel;

import java.util.ArrayList;

public class Process {

public String name;

public ArrayList<String> threads = new ArrayList<String>();

private double needTime;

private double timeUsed;

public void addThreads(String str) {

threads.add(str);

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public ArrayList<String> getThreads() {

return threads;

}

public void setThreads(ArrayList<String> threads) {

this.threads = threads;

}

public double getNeedTime() {

return needTime;

}

public void setNeedTime(double needTime) {

this.needTime = needTime;

}

public double getTimeUsed() {

return timeUsed;

}

public void setTimeUsed(double timeUsed) {

this.timeUsed = timeUsed;

}

}

package edu.buaa.sei.datamodel;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

import org.xml.sax.SAXException;

import edu.buaa.sei.utils.StringHandle;

public class Receiver {

private ArrayList<Message> messageList = new ArrayList<Message>();

public double getFIFOSendTime() {

System.out.println("calaulating time from message list, which size is "

+ messageList.size());

double time = 0;

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

time += messageList.get(i).getTime();

}

System.out.println("FIFO: time used " + time + "ms.");

return time;

}

public boolean containDumplicate(Message m) {

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

if (messageList.get(i).title.compareTo(m.title) == 0)

return true;

}

return false;

}

private Message findTimeById(String id, NodeList list) {

for (int temp = 0; temp < list.getLength(); temp++) {

Node nNode = (Node) list.item(temp);

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

String timeId = eElement.getAttribute("base\_NamedElement");

if (timeId.equals(id)) {

Message mes = new Message();

for (Node node = nNode.getFirstChild(); node != null; node = node

.getNextSibling()) {

if (node.getNodeType() == Node.ELEMENT\_NODE) {

if (node.getNodeName().equals("execTime")) {

String timeStr = node.getFirstChild()

.getNodeValue();

String timeStr1 = StringHandle

.delUnusedStr(timeStr);

String[] str = timeStr1.split(",");

if (str.length < 2)

continue;

mes.time = Double.valueOf(str[0]);

return mes;

}

}

}

}

}

}

return null;

}

public void getReveiver(String umlPath)

throws ParserConfigurationException, SAXException, IOException {

File fXmlFile = new File(umlPath);

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(fXmlFile);

doc.getDocumentElement().normalize();

NodeList nList = doc.getElementsByTagName("message");

NodeList timeList = doc.getElementsByTagName("GQAM:GaStep");

// System.out.println("Reveiver message count : " + nList.getLength());

messageList.clear();

// scan xml and get all valid message.

for (int temp = 0; temp < nList.getLength(); temp++) {

Node nNode = (Node) nList.item(temp);

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

String type = eElement.getAttribute("xmi:type");

if (type.equals("uml:Message")) {

Message mes = new Message();

String nameStr = eElement.getAttribute("name");

String[] strList = nameStr.split(": ");

if (strList.length < 2)

continue;

mes.title = strList[0];

mes.name = strList[1];

mes.id = eElement.getAttribute("xmi:id");

Message mes\_t = findTimeById(mes.id, timeList);

if (mes\_t == null)

continue;

mes.time = mes\_t.time;

if (containDumplicate(mes))

continue;

// System.out.println("title: " + mes.title + ", name: "

// + mes.name + ", time: " + mes.time);

messageList.add(mes);

}

}

}

// System.out.println("valid count : " + messageList.size());

}

}

package edu.buaa.sei.datamodel;

public class SendData {

private int DataNum;

private int DataSize;//kb

private String dependencyDstId;

public String getDependencyDstId() {

return dependencyDstId;

}

public void setDependencyDstId(String dependencyDstId) {

this.dependencyDstId = dependencyDstId;

}

public int getDataNum() {

return DataNum;

}

public void setDataNum(int dataNum) {

DataNum = dataNum;

}

public int getDataSize() {

return DataSize;

}

public void setDataSize(int dataSize) {

DataSize = dataSize;

}

}

package edu.buaa.sei.datamodel;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

import org.xml.sax.SAXException;

import edu.buaa.sei.utils.StringHandle;

public class Sender {

private ArrayList<Message> messageList = new ArrayList<Message>();

public double getFIFOSendTime() {

System.out.println("calaulating time from message list, which size is "

+ messageList.size());

double time = 0;

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

time += messageList.get(i).getTime();

}

System.out.println("FIFO: time used " + time + "ms.");

return time;

}

public ArrayList<Double> getTimeTableTime(int threads, int timeTableUnit) {

ArrayList<Double> timeList = new ArrayList<Double>();

ArrayList<TimeTable> timeTableList = new ArrayList<TimeTable>();

int threads\_t = threads;

double neededTime = getFIFOSendTime();

for (int i = 0; i < threads; i++) {

TimeTable timeTable = new TimeTable();

timeTable.setTimeUsed(0);

timeTable.setNeededTime(neededTime);

timeTableList.add(timeTable);

}

double timeNow = 0;

double timePerProcess;

while (threads\_t > 0) {

timePerProcess = (double)timeTableUnit/(double)threads\_t;//ms

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

if (timeTableList.get(i).getNeededTime() <= 0) {//this process done.

continue;

} else if (timeTableList.get(i).getNeededTime() > timePerProcess) {

timeTableList.get(i).setNeededTime(timeTableList.get(i).getNeededTime() - timePerProcess);

timeNow += timePerProcess;

} else {

timeNow += timeTableList.get(i).getNeededTime();

timeTableList.get(i).setNeededTime(0);

timeTableList.get(i).setTimeUsed(timeNow);

threads\_t--;

}

}

}//while

for (int i = 0; i < threads; i++) {

timeList.add(timeTableList.get(i).getTimeUsed());

}

return timeList;

}

public boolean containDumplicate(Message m) {

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

if (messageList.get(i).title.compareTo(m.title) == 0)

return true;

}

return false;

}

private Message findTimeById(String id, NodeList list) {

for (int temp = 0; temp < list.getLength(); temp++) {

Node nNode = (Node) list.item(temp);

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

String timeId = eElement.getAttribute("base\_NamedElement");

if (timeId.equals(id)) {

Message mes = new Message();

for (Node node = nNode.getFirstChild(); node != null; node = node

.getNextSibling()) {

if (node.getNodeType() == Node.ELEMENT\_NODE) {

if (node.getNodeName().equals("execTime")) {

String timeStr = node.getFirstChild()

.getNodeValue();

String timeStr1 = StringHandle

.delUnusedStr(timeStr);

String[] str = timeStr1.split(",");

if (str.length < 2)

continue;

mes.time = Double.valueOf(str[0]);

return mes;

}

}

}

}

}

}

return null;

}

public void getSender(String umlPath)

throws ParserConfigurationException, SAXException, IOException {

File fXmlFile = new File(umlPath);

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(fXmlFile);

doc.getDocumentElement().normalize();

NodeList nList = doc.getElementsByTagName("message");

NodeList timeList = doc.getElementsByTagName("GQAM:GaStep");

// System.out.println("Sender message count : " + nList.getLength());

messageList.clear();

// scan xml and get all valid message.

for (int temp = 0; temp < nList.getLength(); temp++) {

Node nNode = (Node) nList.item(temp);

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

String type = eElement.getAttribute("xmi:type");

if (type.equals("uml:Message")) {

Message mes = new Message();

String nameStr = eElement.getAttribute("name");

String[] strList = nameStr.split(": ");

if (strList.length < 2)

continue;

mes.title = strList[0];

mes.name = strList[1];

mes.id = eElement.getAttribute("xmi:id");

Message mes\_t = findTimeById(mes.id, timeList);

if (mes\_t == null)

continue;

mes.time = mes\_t.time;

if (containDumplicate(mes))

continue;

// System.out.println("title: " + mes.title + ", name: "

// + mes.name + ", time: " + mes.time);

messageList.add(mes);

}

}

}

// System.out.println("valid count : " + messageList.size());

}

public ArrayList<Message> getMessageList() {

return messageList;

}

public void setMessageList(ArrayList<Message> messageList) {

this.messageList = messageList;

}

}

package edu.buaa.sei.datamodel;

public class TimeTable {

private double neededTime;

private double timeUsed;

public double getNeededTime() {

return neededTime;

}

public void setNeededTime(double neededTime) {

this.neededTime = neededTime;

}

public double getTimeUsed() {

return timeUsed;

}

public void setTimeUsed(double timeUsed) {

this.timeUsed = timeUsed;

}

}

package edu.buaa.sei.utils;

import java.util.Random;

public class RandomGenerator {

public static double getARandomNum(double min, double max) {

Random random = new Random();

double s = (random.nextDouble()\*max)%(max-min+1) + min;

return s;

}

public static double getARandomNumD(double min, double max) {

Random random = new Random();

double s = random.nextDouble()\*max;

while (s < min)

s = random.nextDouble()\*max;

return s;

}

}

package edu.buaa.sei.utils;

public class StringHandle {

public static String delUnusedStr(String str) {

String rstr = null;

for (int i = 0; i < str.length(); i++) {

if (str.charAt(i) == '{' || str.charAt(i) == '}' || str.charAt(i) == ' ') {

continue;

} else {

if (rstr == null)

rstr = String.*valueOf*(str.charAt(i));

else

rstr += str.charAt(i);

}

}

return rstr;

}

public static int getKbFromStr(String str) {

char metric = str.charAt(str.length()-1);

int multiSize = 0;

if (metric == 'M' || metric == 'm')

multiSize = 1024;

else if (metric == 'G' || metric == 'g')

multiSize = 1024\*1024;

else if (metric == 'K' || metric == 'k')

multiSize = 1;

else

System.*out*.println("no such metric : " + metric);

String num = str.substring(0, str.length()-1);

int rv = Integer.*valueOf*(num);

return rv\*multiSize;

}

}

package edu.buaa.sei.run;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

import org.xml.sax.SAXException;

import edu.buaa.sei.datamodel.Dependency;

import edu.buaa.sei.datamodel.LostPackage;

import edu.buaa.sei.datamodel.Message;

import edu.buaa.sei.datamodel.SendData;

import edu.buaa.sei.datamodel.Sender;

import edu.buaa.sei.utils.RandomGenerator;

import edu.buaa.sei.utils.StringHandle;

public class DDS {

ArrayList<Dependency> dependencyList = new ArrayList<Dependency>();

ArrayList<Publisher> publisherList = new ArrayList<Publisher>();

ArrayList <LostPackage> lostPackageList = new ArrayList <LostPackage>();

ArrayList <SendData> sendDataList = new ArrayList <SendData>();

public Publisher getPublisherByBonId(String sonId) {

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

Publisher pub = publisherList.get(i);

for (int j = 0; j < pub.getSonId().size(); j++) {

String sonIdStr = pub.getSonId().get(j);

if (sonIdStr.equals(sonId)) {

return pub;

}

}

}

return null;

}

public LostPackage getLostPackageByDepId(String depID) {

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

LostPackage lostPackage = lostPackageList.get(i);

if (lostPackage.getBaseDependencyId().equals(depID)) {

return lostPackage;

}

}

return null;

}

public SendData getSendDataByDepDstId(String depID) {

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

SendData sendData = sendDataList.get(i);

if (sendData.getDependencyDstId().equals(depID)) {

return sendData;

}

}

return null;

}

public void getAllDependency(String umlPath)

throws ParserConfigurationException, SAXException, IOException {

File fXmlFile = new File(umlPath);

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(fXmlFile);

doc.getDocumentElement().normalize();

NodeList nList = doc.getElementsByTagName("packagedElement");

getALLPublisher(umlPath);

getAllLostPackage(umlPath);

getAllSendData(umlPath);

dependencyList.clear();

// scan xml and get all valid message.

for (int temp = 0; temp < nList.getLength(); temp++) {

Node nNode = (Node) nList.item(temp);

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

String type = eElement.getAttribute("xmi:type");

if (type.equals("uml:Dependency")) {

Dependency dep = new Dependency();

String srcId = eElement.getAttribute("client");

String dstId = eElement.getAttribute("supplier");

Publisher srcPub = getPublisherByBonId(srcId);

Publisher dstPub = getPublisherByBonId(dstId);

dep.setSrcPublisher(srcPub);

dep.setDstPublisher(dstPub);

String depId = eElement.getAttribute("xmi:id");

dep.setLostPackage(getLostPackageByDepId(depId));

dep.setSendData(getSendDataByDepDstId(srcPub.getPublisherId()));

dependencyList.add(dep);

}

}

}

}

public void printDependencyInfo() {

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

Dependency dep = dependencyList.get(i);

System.out.println(dep.getSrcPublisher().getPublisherName() + " --> " + dep.getDstPublisher().getPublisherName()

+ ", send data size: " + dep.getSendData().getDataSize() + "Kb, num: " + dep.getSendData().getDataNum()

+ ", lost package 1st: " + dep.getLostPackage().getFirstLostPackage()

+ ", 2nd: " + dep.getLostPackage().getSecondLostPackage()

+ ", time: " + dep.getTime() + ", reliability: " + dep.getReliability());

}

}

public void calculateDependency(Sender sender) {

//calculate send time.

//calculate transmite time.

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

Dependency dep = dependencyList.get(i);

}

}

public double recursiveGetTime(Publisher pub) {

double minTime = 99999999;

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

Dependency dep = dependencyList.get(i);

if (dep.getDstPublisher().getPublisherId().equals(pub.getPublisherId())) {

double curTime = dep.getTime() + recursiveGetTime(dep.getSrcPublisher());

if (curTime <= minTime) {

minTime = curTime;

}

}

}

if (minTime == 99999999)

return 0;

return minTime;

}

public double recursiveGetReliability(Publisher pub) {

double reliability = 1;

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

Dependency dep = dependencyList.get(i);

if (dep.getDstPublisher().getPublisherId().equals(pub.getPublisherId())) {

reliability \*= dep.getReliability() \* recursiveGetReliability(dep.getSrcPublisher());

}

}

return reliability;

}

public void calculateTime() {

ArrayList<Publisher> leafPublisher = getLeafPublisher();

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

Publisher pub = leafPublisher.get(i);

double time = recursiveGetTime(pub);

System.out.printf("%s : %.3f ms.\n", pub.getPublisherName(), time);

}

}

public void calculateReliability() {

ArrayList<Publisher> leafPublisher = getLeafPublisher();

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

Publisher pub = leafPublisher.get(i);

double reliability = recursiveGetReliability(pub);

System.out.printf("%s : %.3f%%.\n", pub.getPublisherName(), reliability\*100);

}

}

public void showResults() {

ArrayList<Publisher> leafPublisher = getLeafPublisher();

System.out.printf("\n%7s\t%10s\t%6s\n", "name", "reliability(%s)", "time(ms)");

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

Publisher pub = leafPublisher.get(i);

double reliability = recursiveGetReliability(pub);

double time = recursiveGetTime(pub);

System.out.printf("%7s\t%10.3f\t%6.3f\n", pub.getPublisherName(), reliability\*100, time);

}

}

public ArrayList<Publisher> getRootPublisher() {

ArrayList<Publisher> rl = new ArrayList<Publisher>();

return rl;

}

public boolean isLeafPublisher(Publisher pub) {

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

Dependency dep = dependencyList.get(i);

if (dep.getSrcPublisher().getPublisherId().equals(pub.getPublisherId())) {

return false;

}

}

return true;

}

public ArrayList<Publisher> getLeafPublisher() {

ArrayList<Publisher> rl = new ArrayList<Publisher>();

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

Publisher pub = publisherList.get(i);

if (isLeafPublisher(pub)) {

rl.add(pub);

}

}

return rl;

}

public void getAllLostPackage(String umlPath) throws ParserConfigurationException, SAXException, IOException {

File fXmlFile = new File(umlPath);

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(fXmlFile);

doc.getDocumentElement().normalize();

NodeList nList = doc.getElementsByTagName("Profile:lostpackage");

lostPackageList.clear();

// scan xml and get all valid message.

for (int temp = 0; temp < nList.getLength(); temp++) {

Node nNode = (Node) nList.item(temp);

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

String baseDependencyId = eElement.getAttribute("base\_Dependency");

String firstLostPackageStr = eElement.getAttribute("FirstLostPackge");

String secondLostPackageStr = eElement.getAttribute("SecondLostPackage");

LostPackage lostPackage = new LostPackage();

lostPackage.setBaseDependencyId(baseDependencyId);

if (firstLostPackageStr != null) {

if (firstLostPackageStr.length() != 0)

lostPackage.setFirstLostPackage(Integer.valueOf(firstLostPackageStr));

}

if (secondLostPackageStr != null) {

if (secondLostPackageStr.length() != 0)

lostPackage.setSecondLostPackage(Integer.valueOf(secondLostPackageStr));

}

lostPackageList.add(lostPackage);

}

}

}

public void getAllSendData(String umlPath) throws ParserConfigurationException, SAXException, IOException {

File fXmlFile = new File(umlPath);

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(fXmlFile);

doc.getDocumentElement().normalize();

NodeList nList = doc.getElementsByTagName("SW\_Interaction:MessageComResource");

sendDataList.clear();

// scan xml and get all valid message.

for (int temp = 0; temp < nList.getLength(); temp++) {

Node nNode = (Node) nList.item(temp);

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

String baseDependencyId = eElement.getAttribute("base\_Classifier");

Publisher pub = getPublisherByBonId(baseDependencyId);

SendData sendData = new SendData();

sendData.setDependencyDstId(pub.getPublisherId());

int size = 0;

int count = 0;

for (Node node = nNode.getFirstChild(); node != null; node = node

.getNextSibling()) {

if (node.getNodeType() == Node.ELEMENT\_NODE) {

if (node.getNodeName().equals("capacity")) {

String str = node.getFirstChild().getNodeValue();

String[] strList = str.split("\\\*");

if (strList.length < 2)

continue;

int count\_t = Integer.valueOf(strList[1]);

int sizeKB = StringHandle.getKbFromStr(strList[0]);

size += sizeKB\*count\_t;

count += count\_t;

}

}

}

sendData.setDataSize(size);

sendData.setDataNum(count);

sendDataList.add(sendData);

}

}

}

public void getALLPublisher(String umlPath)

throws ParserConfigurationException, SAXException, IOException {

File fXmlFile = new File(umlPath);

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(fXmlFile);

doc.getDocumentElement().normalize();

NodeList nList = doc.getElementsByTagName("nestedClassifier");

publisherList.clear();

// scan xml and get all valid message.

for (int temp = 0; temp < nList.getLength(); temp++) {

Node nNode = (Node) nList.item(temp);

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

String type = eElement.getAttribute("xmi:type");

if (type.equals("uml:Device")) {

Dependency dep = new Dependency();

String id = eElement.getAttribute("xmi:id");

String name = eElement.getAttribute("name");

Publisher publisher = new Publisher();

publisher.setPublisherId(id);

publisher.setPublisherName(name);

ArrayList<String> sonIdList = new ArrayList<String>();

for (Node node = nNode.getFirstChild(); node != null; node = node

.getNextSibling()) {

if (node.getNodeType() == Node.ELEMENT\_NODE) {

if (node.getNodeName().equals("nestedClassifier")) {

if (((Element)node).getAttribute("xmi:type").equals("uml:Artifact")) {

String sonId = ((Element)node).getAttribute("xmi:id");

sonIdList.add(sonId);

}

}

}

}

publisher.setSonId(sonIdList);

publisherList.add(publisher);

}

}

}

}

public void calculateDependencyTime() {

}

public void calculateDDSTime() {

}

}

package edu.buaa.sei.run;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

import org.xml.sax.SAXException;

import edu.buaa.sei.datamodel.Message;

import edu.buaa.sei.datamodel.Process;

import edu.buaa.sei.utils.StringHandle;

public class NodeSend {

private String schedPolicy;

private ArrayList<Process> pList = new ArrayList<Process>();

public void parseNodeSend(String umlPath)

throws ParserConfigurationException, SAXException, IOException {

File fXmlFile = new File(umlPath);

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(fXmlFile);

doc.getDocumentElement().normalize();

NodeList nList = doc.getElementsByTagName("GRM:Scheduler");

for (int temp = 0; temp < nList.getLength(); temp++) {

Node nNode = (Node) nList.item(temp);

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

//get schedPolicy, FIFO timeTable priority.

String type = eElement.getAttribute("schedPolicy");

schedPolicy = type;

break;

}

}

System.out.println("SchedPolicy:" + schedPolicy);

}

private void getProcessContent(Document doc, String[] processList) {

pList.clear();

NodeList nList = doc.getElementsByTagName("packagedElement");

for (int temp = 0; temp < nList.getLength(); temp++) {

Node nNode = (Node) nList.item(temp);

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

String processStr = eElement.getAttribute("name");

for (int i = 0; i < processList.length; i++) {

String name = processList[i] + ":Task";

if (name.compareTo(processStr) == 0) {

Process process = new Process();

process.name = processList[i];

System.out.println("find " + process.name);

for (Node node = nNode.getFirstChild(); node != null; node = node

.getNextSibling()) {

if (node.getNodeType() == Node.ELEMENT\_NODE) {

if (node.getNodeName().equals("ownedOperation")) {

Element e = (Element) node;

String ownedOperationName = e.getAttribute("name");

process.addThreads(ownedOperationName);

System.out.println("add thread " + ownedOperationName);

}

}

}

pList.add(process);

break;

}

}

}

}

}

public String getSchedPolicy() {

return schedPolicy;

}

public void setSchedPolicy(String schedPolicy) {

this.schedPolicy = schedPolicy;

}

public ArrayList<Process> getpList() {

return pList;

}

public void setpList(ArrayList<Process> pList) {

this.pList = pList;

}

}

package edu.buaa.sei.run;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

import org.xml.sax.SAXException;

import edu.buaa.sei.datamodel.Message;

import edu.buaa.sei.utils.StringHandle;

public class Publisher {

private String publisherName;

private String publisherId;

private ArrayList<String> sonId = new ArrayList<String> ();

public String getPublisherName() {

return publisherName;

}

public void setPublisherName(String publisherName) {

this.publisherName = publisherName;

}

public String getPublisherId() {

return publisherId;

}

public void setPublisherId(String publisherId) {

this.publisherId = publisherId;

}

public ArrayList<String> getSonId() {

return sonId;

}

public void setSonId(ArrayList<String> sonId) {

this.sonId = sonId;

}

}

package edu.buaa.sei.run;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

import org.xml.sax.SAXException;

import edu.buaa.sei.datamodel.Message;

import edu.buaa.sei.utils.RandomGenerator;

import edu.buaa.sei.utils.StringHandle;

public class Transporter {

// FIFO

// FixedPriority

// TimeTableDrivenff

private String schedPolicy;

private double paskageUnitSize = 1.4;

private double totalsize;// kb

private double minKb;

private double maxKb;

public void getTransporter(String umlPath)

throws ParserConfigurationException, SAXException, IOException {

File fXmlFile = new File(umlPath);

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(fXmlFile);

doc.getDocumentElement().normalize();

NodeList nList = doc.getElementsByTagName("GQAM:GaCommStep");

int size\_kb = -1;

String transportId = null;

double maxThroughPut = 0, minThroughput = 0;

for (int temp = 0; temp < nList.getLength(); temp++) {

Node nNode = (Node) nList.item(temp);

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

transportId = eElement.getAttribute("base\_NamedElement");

for (Node node = nNode.getFirstChild(); node != null; node = node

.getNextSibling()) {

if (node.getNodeType() == Node.ELEMENT\_NODE) {

if (node.getNodeName().equals("msgSize")) {

String sizeStr = node.getFirstChild()

.getNodeValue();

size\_kb = StringHandle.getKbFromStr(sizeStr);

// System.out.println("size:" + size\_kb + "kb");

break;

}

}

}

if (size\_kb != -1)

break;

}

}

// get MaxThroughPut and MinThroughput

NodeList throughPutList = doc

.getElementsByTagName("Profile:SendMessage");

for (int temp = 0; temp < throughPutList.getLength(); temp++) {

Node nNode = (Node) throughPutList.item(temp);

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

String id = eElement.getAttribute("base\_Message");

if (id.compareTo(transportId) == 0) {

for (Node node = nNode.getFirstChild(); node != null; node = node

.getNextSibling()) {

if (node.getNodeType() == Node.ELEMENT\_NODE) {

if (node.getNodeName().equals("MaxThroughPut")) {

String maxStr = ((Element) node)

.getAttribute("precision");

// System.out.println("max:" + maxStr);

maxThroughPut = Double.valueOf(maxStr);

}

}

if (node.getNodeType() == Node.ELEMENT\_NODE) {

if (node.getNodeName().equals("MinThroughput")) {

String minStr = ((Element) node)

.getAttribute("precision");

// System.out.println("min:" + minStr);

minThroughput = Double.valueOf(minStr);

}

}

}

}

}

}

totalsize = size\_kb;

minKb = minThroughput;

maxKb = maxThroughPut;

}

public double calculateTransportFIFO() {

double time = 0;

// int packageNum = (int) Math.ceil(totalsize / paskageUnitSize);

double bandWidth = RandomGenerator.getARandomNum(minKb, maxKb);

time = (totalsize / bandWidth) \* 1000;

System.out.println("transport time used " + time + "ms.");

return time;

}

}

package edu.buaa.sei.run;

import java.io.IOException;

import java.util.ArrayList;

import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;

import edu.buaa.sei.datamodel.Message;

import edu.buaa.sei.datamodel.Process;

import edu.buaa.sei.datamodel.Receiver;

import edu.buaa.sei.datamodel.Sender;

public class Wrapper {

public static void main(String[] args) throws IOException, ParserConfigurationException, SAXException {

//setp 1: parse uml.

NodeSend nodeSend = new NodeSend();

nodeSend.parseNodeSend("CaseStudy/NodeSend.uml");

String schedPolicy = nodeSend.getSchedPolicy();

Sender sender = new Sender();

sender.getSender("CaseStudy/send.uml");

Receiver receiver = new Receiver();

receiver.getReveiver("CaseStudy/receive.uml");

Transporter transporter = new Transporter();

transporter.getTransporter("CaseStudy/transport.uml");

//step 2:

DDS dds = new DDS();

dds.getAllDependency("CaseStudy/publisher.uml");

dds.calculateDependency(sender);

dds.showResults();

}

}

package edu.buaa.sei.run;

import java.util.ArrayList;

import java.util.List;

public class BaseModelManager {

List<UMLComponent> umlCompList = new ArrayList<UMLComponent>();

List<UMLDependency> umlDepList = new ArrayList<UMLDependency>();

List<UMLMessage> umlMsgList = new ArrayList<UMLMessage>();

String startCompName;

String endCompName;

private static BaseModelManager instance;

public static BaseModelManager INS = instance();

private static BaseModelManager instance() {

if (instance == null) {

instance = new BaseModelManager();

}

return instance;

}

public void addUMLComponent(UMLComponent comp) {

umlCompList.add(comp);

}

public void addUMLDependency(UMLDependency dep) {

umlDepList.add(dep);

}

public void addUMLMessage(UMLMessage msg) {

umlMsgList.add(msg);

}

public UMLComponent findUMLComponent(String compName) {

for (UMLComponent comp : umlCompList) {

if (comp.getCompName().equals(compName))

return comp;

}

return null;

}

public List<UMLDependency> findUMLDependency(String compName, String type) {

List<UMLDependency> resultDepList = new ArrayList<UMLDependency>();

for (UMLDependency dep : umlDepList) {

if ((type.equals("supplier") && dep.getSupplier().getCompName()

.equals(compName))

|| (type.equals("client") && dep.getClient().getCompName()

.equals(compName)))

resultDepList.add(dep);

}

return resultDepList;

}

public List<UMLDependency> findUMLDependency(UMLComponent supplier,

UMLComponent client) {

List<UMLDependency> resultDepList = new ArrayList<UMLDependency>();

for (UMLDependency dep : umlDepList) {

if (dep.getSupplier().equals(supplier)

&& dep.getClient().equals(client))

resultDepList.add(dep);

}

return resultDepList;

}

public List<UMLMessage> findUMLMessage(String compName, String type) {

List<UMLMessage> resultMsgList = new ArrayList<UMLMessage>();

for (UMLMessage msg : umlMsgList) {

if ((type.equals("sender") && msg.getSender().equals(compName))

|| (type.equals("receiver") && msg.getReceiver().equals(

compName)))

resultMsgList.add(msg);

}

return resultMsgList;

}

public List<UMLComponent> getUmlCompList() {

return umlCompList;

}

public List<UMLDependency> getUmlDepList() {

return umlDepList;

}

public List<UMLMessage> getUmlMsgList() {

return umlMsgList;

}

public String getStartCompName() {

return startCompName;

}

public void setStartCompName(String startCompName) {

this.startCompName = startCompName;

}

public String getEndCompName() {

return endCompName;

}

public void setEndCompName(String endCompName) {

this.endCompName = endCompName;

}

}

package edu.buaa.sei.run;

import java.util.ArrayList;

import java.util.List;

public class BaseModelManager\_V2 {

List<UMLClassIns> umlClaInsList = new ArrayList<UMLClassIns>();

List<UMLMessage\_V2> umlMsgList = new ArrayList<UMLMessage\_V2>();

String startInsName;

String endInsName = "finishNode";

private static BaseModelManager\_V2 instance;

public static BaseModelManager\_V2 INS = instance();

private static BaseModelManager\_V2 instance() {

if (instance == null) {

instance = new BaseModelManager\_V2();

}

return instance;

}

public void addUMLMessage(UMLMessage\_V2 msg) {

umlMsgList.add(msg);

}

public List<UMLMessage\_V2> findUMLMessage(String claInsName, String type) {

List<UMLMessage\_V2> resultMsgList = new ArrayList<UMLMessage\_V2>();

for (UMLMessage\_V2 msg : umlMsgList) {

if ((type.equals("sender") && msg.getSender().getInsName()

.equals(claInsName))

|| (type.equals("receiver") && msg.getReceiver()

.getInsName().equals(claInsName)))

resultMsgList.add(msg);

}

return resultMsgList;

}

public List<UMLMessage\_V2> getUmlMsgList() {

return umlMsgList;

}

public void addUMLClaIns(UMLClassIns umlClaIns) {

this.umlClaInsList.add(umlClaIns);

}

public List<UMLClassIns> findInsByClaName(String umlClaName) {

List<UMLClassIns> resultList = new ArrayList<UMLClassIns>();

for (UMLClassIns umlClaIns : this.umlClaInsList) {

if (umlClaIns.getClassName().equals(umlClaName))

resultList.add(umlClaIns);

}

return resultList;

}

public UMLClassIns findInsByInsName(String insName) {

for (UMLClassIns umlClaIns : this.umlClaInsList) {

if (umlClaIns.getInsName().equals(insName))

return umlClaIns;

}

return null;

}

public String getStartInsName() {

return startInsName;

}

public void setStartInsName(String startInsName) {

this.startInsName = startInsName;

}

public String getEndInsName() {

return endInsName;

}

public void setEndInsName(String endInsName) {

this.endInsName = endInsName;

}

public List<UMLClassIns> getUmlClaInsList() {

return umlClaInsList;

}

public void clearAll() {

this.umlMsgList.clear();

this.umlClaInsList.clear();

this.startInsName = "";

}

}

package edu.buaa.sei.run;

import java.util.ArrayList;

import java.util.List;

public class ClaInsFragment {

String claInsName;

List<TacticInfo> tiList;

public ClaInsFragment(String claInsName, List<TacticInfo> tiList) {

super();

this.claInsName = claInsName;

this.tiList = tiList;

}

public String getClaInsName() {

return claInsName;

}

public List<TacticInfo> getTiList() {

return tiList;

}

}

package edu.buaa.sei.run;

public class Constants {

public final static int MCOST\_TYPE\_AC[] = { 1, 10 };

public final static int MCOST\_TYPE\_AP[] = { 1, 10 };

public final static int MCOST\_TYPE\_RP[] = { 1, 10 };

public final static int MCOST\_TYPE\_MA[] = { 1, 10 };

public final static int MCOST\_TYPE\_MR[] = { 1, 10 };

public final static String T\_HEARTBEAT = "Heartbeat";

public final static String T\_PING\_ECHO = "Ping/Echo";

public final static String T\_TIMESTAMP = "TimeStamp";

public final static String T\_SANITY\_CHECKING = "SanityChecking";

public final static String T\_CONDITION\_MONITORING = "ConditionMonitoring";

public final static String T\_EXCEPTION\_DETECTION = "ExceptionDetection";

public final static String T\_SELF\_TEST = "SelfTest";

public final static String T\_ACTIVE\_REDUNDANCY = "ActiveRedundancy";

public final static String T\_PASSIVE\_REDUNDANCY = "PassiveRedundancy";

public final static String T\_VOTING = "Voting";

public final static String T\_STATE\_RESYNCHRONIZATION = "StateResynchronization";

public final static String T\_CHECKPOINT\_ROLLBACK = "Checkpoint/Rollback";

public final static String T\_REMOVAL\_FROM\_SERVICE = "RemovalFromService";

public final static String T\_TRANSACTIONS = "Transactions";

public final static String T\_EXCEPTION\_PREVENTION = "ExceptionPrevention";

public final static String T\_INCREASE\_CONPETENCE\_SET = "IncreaseConpetenceSet";

public final static String T\_EXCEPTION\_HANDLING = "ExceptionHandling";

public final static String T\_SOFTWARE\_UPGRADE = "SoftwareUpgrade";

public final static String T\_RETRY = "Retry";

public final static String T\_IGONRE\_FAULTY\_BEHAVIOR = "IgonreFaultyBehavior";

public final static String T\_DEGRADATION = "Degradation";

public final static String T\_RECONFIGURATION = "Reconfiguration";

public final static int MAX\_NUM\_BACKUP = 3;

public final static String FAULT\_DETECTION\_TACTIC[] = { T\_HEARTBEAT,

T\_PING\_ECHO, T\_TIMESTAMP, T\_SANITY\_CHECKING,

T\_CONDITION\_MONITORING, T\_EXCEPTION\_DETECTION, T\_SELF\_TEST };

public final static String REDUNDANCY\_TACTIC[] = { T\_ACTIVE\_REDUNDANCY,

T\_PASSIVE\_REDUNDANCY, T\_VOTING };

public final static String FAULT\_RECOVERY\_TACTIC[] = {

T\_STATE\_RESYNCHRONIZATION, T\_CHECKPOINT\_ROLLBACK };

public final static String FAULT\_PREVENTION\_TACTIC[] = {

T\_REMOVAL\_FROM\_SERVICE, T\_TRANSACTIONS, T\_EXCEPTION\_PREVENTION,

T\_INCREASE\_CONPETENCE\_SET };

public final static String FAULT\_REPAIR\_TACTIC[] = { T\_EXCEPTION\_HANDLING,

T\_SOFTWARE\_UPGRADE, T\_RETRY, T\_IGONRE\_FAULTY\_BEHAVIOR,

T\_DEGRADATION, T\_RECONFIGURATION };

}

package edu.buaa.sei.run;

public class SearchResult {

double fitness;

double bestSR;

double bestCost;

int[] v;

public SearchResult(double fitness, double bestSR, double bestCost, int[] v) {

super();

this.fitness = fitness;

this.bestSR = bestSR;

this.bestCost = bestCost;

this.v = v;

}

public String getFitness() {

return "" + fitness;

}

public String getBestSRAndCost() {

return bestSR + "," + bestCost;

}

public String getV() {

String temp = "" + this.v[0];

for (int i = 1; i < this.v.length; i++) {

temp = temp + "," + this.v[i];

}

return temp;

}

}

package edu.buaa.sei.run;

public class TacticInfo {

int tacticType;

int minRedNum, maxRedNum;

public TacticInfo(int tacticType, int minRedNum, int maxRedNum) {

super();

this.tacticType = tacticType;

this.minRedNum = minRedNum;

this.maxRedNum = maxRedNum;

}

public int getTacticType() {

return tacticType;

}

public int getMinRedNum() {

return minRedNum;

}

public int getMaxRedNum() {

return maxRedNum;

}

}

package edu.buaa.sei.run;

import java.util.ArrayList;

import java.util.List;

public class UMLClassIns {

String insName;

String className;

double tacticSafety = 1.0;

double reliability;

double failureRate;

double cost;

double meanFR;

double lowFR;

double highFR;

List<UMLClassIns> assInsList = new ArrayList<UMLClassIns>();

public UMLClassIns(String insName, String className) {

super();

this.insName = insName;

this.className = className;

}

public void addassIns(UMLClassIns assIns) {

this.assInsList.add(assIns);

}

public String getInsName() {

return insName;

}

public double getCost() {

return cost;

}

public void setCost(double cost) {

this.cost = cost;

}

public double getMeanFR() {

return meanFR;

}

public void setMeanFR(double meanFR) {

this.meanFR = meanFR;

}

public double getLowFR() {

return lowFR;

}

public void setLowFR(double lowFR) {

this.lowFR = lowFR;

}

public double getHighFR() {

return highFR;

}

public void setHighFR(double highFR) {

this.highFR = highFR;

}

public double getFailureRate() {

return failureRate;

}

public void setFailureRate(double failureRate) {

this.failureRate = failureRate;

}

public String getClassName() {

return className;

}

public List<UMLClassIns> getAssInsList() {

return assInsList;

}

public double getReliability() {

return reliability;

}

public void setReliability(double reliability) {

this.reliability = reliability;

}

public double getTacticSafety() {

return tacticSafety;

}

public void setTacticSafety(double tacticSafety) {

this.tacticSafety = tacticSafety;

}

}

package edu.buaa.sei.run;

public class UMLDependency {

String depName;

UMLComponent supplier;

UMLComponent client;

public UMLDependency(String depName, UMLComponent supplier,

UMLComponent client) {

super();

this.depName = depName;

this.supplier = supplier;

this.client = client;

}

public UMLComponent getSupplier() {

return supplier;

}

public void setSupplier(UMLComponent supplier) {

this.supplier = supplier;

}

public UMLComponent getClient() {

return client;

}

public void setClient(UMLComponent client) {

this.client = client;

}

public String getDepName() {

return depName;

}

public void setDepName(String depName) {

this.depName = depName;

}

}

package edu.buaa.sei.run;

public class UMLMessage {

String msgName;

UMLComponent sender;

UMLComponent receiver;

public UMLMessage(String msgName, UMLComponent sender, UMLComponent receiver) {

super();

this.msgName = msgName;

this.sender = sender;

this.receiver = receiver;

}

public String getMsgName() {

return msgName;

}

public UMLComponent getSender() {

return sender;

}

public UMLComponent getReceiver() {

return receiver;

}

}

package edu.buaa.sei.run;

public class UMLMessage\_V2 {

String msgName;

UMLClassIns sender;

UMLClassIns receiver

public UMLMessage\_V2(String msgName, UMLClassIns sender, UMLClassIns receiver) {

super();

this.msgName = msgName;

this.sender = sender;

this.receiver = receiver;

}

public String getMsgName() {

return msgName;

}

public UMLClassIns getSender() {

return sender;

}

public UMLClassIns getReceiver() {

return receiver;

}

}

package edu.buaa.sei.run;

import java.util.ArrayList;

import java.util.Comparator;

import java.util.Iterator;

import java.util.List;

import java.util.PriorityQueue;

import edu.buaa.sei.clock.SimulationClock;

import edu.buaa.sei.event.IEvent;

import edu.buaa.sei.resource.ResourceContainer;

/\*\*

\* Class controls simulation.

\* @author sei

\*

\*/

public class SimulationController implements ISimulationController {

// containers in this simulation

private List<ResourceContainer> containers;

// event list

private PriorityQueue<IEvent> eventList;

// single instance in this simulation engine.

public static SimulationController instance = new SimulationController();

public SimulationClock clock;

// creation of this class is forbidden.

private SimulationController() {

eventList = new PriorityQueue<IEvent>(1, new Comparator<IEvent>() {

public int compare(IEvent first, IEvent second) {

if (first.scheduledAtTime() - second.scheduledAtTime() < 0) return -1;

if (first.scheduledAtTime() - second.scheduledAtTime() == 0) return 0;

return 1;

}

});

eventList.clear();

clock = new SimulationClock();

containers = new ArrayList<ResourceContainer>();

}

// get current time;

@Override

public int currentTime() {

return clock.getCurrentTime();

}

// add container

@Override

public boolean addContainer(ResourceContainer container) {

if (containers.contains(container)) return false;

containers.add(container);

return true;

}

// add event to event list.

@Override

public void addEvent(IEvent event) {

eventList.add(event);

}

// start simulation

@Override

public void start() {

// start all the container.

Iterator<ResourceContainer> iter = containers.iterator();

while (iter.hasNext()) {

iter.next().start();

}

// Iterate all the event to process.

while(!eventList.isEmpty()) {

IEvent event = eventList.remove();

clock.setCurrentTime(event.scheduledAtTime());

event.eventRoutine();

}

}

// stop simulation

@Override

public void stop() {

// left empty.

}

// pause simulation

@Override

public void pause() {

// left empty.

}

// resume simulation

@Override

public void resume() {

// left empty.

}

}

package edu.buaa.sei.run;

public class SimulationClock {

private int currentTime;

public int getCurrentTime() {

return currentTime;

}

public void setCurrentTime(int currentTime) {

this.currentTime = currentTime;

}

public void updateCurrentTime(int plus) {

this.currentTime += plus;

}

}

package edu.buaa.sei.run;

import java.util.UUID;

/\*\*

\* A simulated event, which can be scheduled to occur at a specified point

\* time in the simulated future. When reaching the specified simulated time instant,

\* the eventRoutine method is being executed. In this way, this class supports the

\* so-called event-scheduling simulation modeling.

\* @author sei

\*

\*/

public interface IEvent {

/\*\*

\* Schedules this event to occur in delay simulated time units.

\* @param process

\* @param delay the period of simulated time to wait before this event is executed.

\*/

public void schedule();

/\*\*

\* cancel this event

\*/

public void cancelEvent();

/\*\*

\* return the time this event is about to occur.

\* @return

\*/

public int scheduledAtTime();

/\*\*

\* Executes the simulation logic associated with this event.

\* Notice, that this method is not intended to be called by clients. Instead,

\* the event scheduler of the respective simulation library invokes this method

\* as soon as the simulation is reached at which the event has bean scheduled.

\*

\* @param process the process associated with this event.

\*/

public void eventRoutine();

/\*\*

\* Unique identifier of this event.

\*/

public UUID getId();

}

package edu.buaa.sei.run;

import java.util.UUID;

import edu.buaa.sei.controller.SimulationController;

import edu.buaa.sei.util.util;

public abstract class AbstractEvent implements IEvent {

// Unique identifier of this event.

private UUID id;

// time this event is about to finished.

private int time;

public AbstractEvent(int time) {

this.id = util.generateId();

this.time = time;

}

// Get identifier of this event.

@Override

public UUID getId() {

return id;

}

/\*\*

\* Test the equal of two events.

\*/

@Override

public boolean equals(Object obj) {

if (obj instanceof IEvent)

return ((IEvent) obj).getId() == getId();

return false;

}

/\*\*

\* Put this event to event handler.

\*/

@Override

public void schedule() {

SimulationController.instance.addEvent(this);

}

/\*\*

\* Time this event is about to vanish.

\*/

@Override

public int scheduledAtTime() {

return time;

}

@Override

public void cancelEvent() {

// left empty.

}

}

package edu.buaa.sei.run;

public class ProcessorFinishedEvent extends AbstractEvent {

// processor produces this event.

private ActiveResourceProcessor processor;

// process execute this event.

private ISchedulableProcess process;

/\*\*

\* Default constructor of this class.

\* @param processor

\* @param process

\*/

public ProcessorFinishedEvent(ActiveResourceProcessor processor,

ISchedulableProcess process, int time) {

super(time);

this.process = process;

this.processor = processor;

}

@Override

public void eventRoutine() {

// When process finished. Proceed with the next instruction.

processor.dispatchProcess(process);

processor.release(process);

}

}

package edu.buaa.sei.run;

public class StorageFinishedEvent extends AbstractEvent {

// storage resource associated with this event.

private ActiveResourceStorage resource;

// Process associated with this event. left for extension.

private ISchedulableProcess process;

/\*\*

\* Default constructor for this class.

\* @param resource

\* @param process

\* @param time

\*/

public StorageFinishedEvent(ActiveResourceStorage resource,

ISchedulableProcess process, int time) {

super(time);

this.resource = resource;

this.process = process;

}

/\*\*

\* Forward the execution to next request.

\*/

@Override

public void eventRoutine() {

process.getProcessor().process(process);

resource.release(process);

}

}

package edu.buaa.sei.run;

import java.util.List;

/\*\*

\* Load balancer for new created process only. Process is executed

\* in the same processor until finished. Currently, the load balancer

\* does not consume time.

\* @author sei

\*

\*/

public interface ILoadBalancer {

/\*\*

\* Add a processor to this load balancer. If this processor has been

\* added before, return false. Otherwise return true.

\* @param processor

\* @return

\*/

public boolean addProcessor(ActiveResourceProcessor processor);

/\*\*

\*

\* @param process

\* @return

\*/

public void addProcess(ISchedulableProcess process);

}

package edu.buaa.sei.run;

import java.util.Iterator;

import java.util.List;

/\*\*

\* Load balancer for new created process only. Process is executed

\* in the same processor until finished. In each computation node,

\* there could be many processors but only one storage resource.

\* There may be many communication resource, but it's not allocated

\* by balancer, but allocated by developer.

\* @author sei

\*

\*/

public class LoadBalancer implements ILoadBalancer {

List<ActiveResourceProcessor> processors;

public LoadBalancer(List<ActiveResourceProcessor> processors) {

this.processors = processors;

}

@Override

public boolean addProcessor(ActiveResourceProcessor processor) {

if (processors.contains(processor))

return false;

processors.add(processor);

return true;

}

@Override

public void addProcess(ISchedulableProcess process) {

Iterator<ActiveResourceProcessor> iter = processors.iterator();

int length = Integer.MAX\_VALUE;

ActiveResourceProcessor processor = null, tmp = null;

while (iter.hasNext()) {

tmp = iter.next();

if (length > tmp.getWaitingLength()) {

processor = tmp;

length = tmp.getWaitingLength();

}

}

process.setProcessor(processor);

processor.process(process);

}

}

package edu.buaa.sei.run;

public class AbstractResource {

// resource instances number.

private int capacity;

// name of this resource, many resource may have the same name

// to show the type.

private String name = "";

// id of this resource, unique.

private String id = "";

public AbstractResource(String name, String id, int capacity) {

this.name = name;

this.id = id;

this.capacity = capacity;

}

public int getCapacity() {

return capacity;

}

public void setCapacity(int capacity) {

this.capacity = capacity;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

@Override

public boolean equals(Object obj) {

if (obj != null && obj instanceof AbstractResource)

return id == ((AbstractResource) obj).id;

return false;

}

}

package edu.buaa.sei.run;

/\*\*

\* Interface for controlling the simulation.

\* @author sei

\*

\*/

public interface ISimulationController {

boolean addContainer(ResourceContainer container);

void addEvent(IEvent event);

int currentTime();

void pause();

void resume();

void start();

void stop();

}

package edu.buaa.sei.run;

public class DispatchStartEvent extends AbstractEvent {

private ActiveResourceProcessor processor;

private ISchedulableProcess process;

public DispatchStartEvent(ActiveResourceProcessor processor,

ISchedulableProcess process, int time) {

super(time);

this.process = process;

this.processor = processor;

}

@Override

public void eventRoutine() {

processor.dispatchProcess(process);

}

}

package edu.buaa.sei.run;

public class LinkingFinishedEvent extends AbstractEvent {

private ActiveResourceLinking resource;

private ISchedulableProcess process;

public LinkingFinishedEvent(ActiveResourceLinking resource,

ISchedulableProcess process, int time) {

super(time);

this.resource = resource;

this.process = process;

}

@Override

public void eventRoutine() {

process.getProcessor().process(process);

resource.release(process);

}

}

package edu.buaa.sei.run;

public class LogicalAcquireFinishedEvent extends AbstractEvent {

@SuppressWarnings("unused")

private PassiveResource resource;

private ISchedulableProcess process;

public LogicalAcquireFinishedEvent(PassiveResource resource,

ISchedulableProcess process, int time) {

super(time);

this.resource = resource;

this.process = process;

}

@Override

public void eventRoutine() {

process.getProcessor().process(process);

}

}

package edu.buaa.sei.run;

public class LogicalReleaseFinishedEvent extends AbstractEvent {

private IPassiveResource resource;

private ISchedulableProcess process;

public LogicalReleaseFinishedEvent(IPassiveResource resource,

ISchedulableProcess process, int time) {

super(time);

this.resource = resource;

this.process = process;

}

@Override

public void eventRoutine() {

resource.release(process, process.getLogicDemand().getSize());

}

}

package edu.buaa.sei.run;

public class PassiveResourceTimeoutEvent extends AbstractEvent {

// The process requiring the resource.

private ISchedulableProcess process;

// The resource to be required.

private IPassiveResource resource;

// create this event with the corresponding resource and process.

public PassiveResourceTimeoutEvent(ISchedulableProcess process,

IPassiveResource resource, int time) {

super(time);

this.resource = resource;

this.process = process;

}

@Override

public void eventRoutine() {

process.getProcessor().process(process);

// remove notify of this process.

resource.cancelWaiting(process);

}

}

package edu.buaa.sei.run;

/\*\*

\* Abstract instruction implements basic methods.

\* @author sei

\*

\*/

public abstract class AbstractInstruction implements Instruction {

// required process time.

private int processorDemand;

// required storage size.

private int storageDemand;

// required communication resource.

private CommuEntity commuDemand;

// required logical resource.

private LogicalEntity logicalDemand;

// name of this instruction.

private String name;

// Is asynchronous message or not

private boolean isAsync;

// resource type.

private RESOURCE\_TYPE resourceType;

@Override

public boolean isAsync() {

return isAsync;

}

/\*\*

\* Get name of this instruction.

\*/

@Override

public String getName() {

return name;

}

@Override

public void setName(String name) {

this.name = name;

}

@Override

public int getProcessorDemand() {

return processorDemand;

}

@Override

public int getStorageDemand() {

return storageDemand;

}

@Override

public CommuEntity getCommuDemand() {

return commuDemand;

}

@Override

public LogicalEntity getLogicDemand() {

return logicalDemand;

}

@Override

public void setProcessorDemand(int processorDemand) {

this.processorDemand = processorDemand;

}

@Override

public void setStorageDemand(int storageDemand) {

this.storageDemand = storageDemand;

}

@Override

public void setLogicalDemand(LogicalEntity logicalDemand) {

this.logicalDemand = logicalDemand;

}

@Override

public void setCommuDemand(CommuEntity commuDemand) {

this.commuDemand = commuDemand;

}

@Override

public RESOURCE\_TYPE getResourceType() {

return resourceType;

}

@Override

public void setResourceType(RESOURCE\_TYPE resourceType) {

this.resourceType = resourceType;

}

}

package edu.buaa.sei.run;

public class AsyncInstruction extends AbstractInstruction {

@Override

public boolean isAtomic() {

// TODO Auto-generated method stub

return false;

}

@Override

public void loadNextInstruction() {

// TODO Auto-generated method stub

}

@Override

public boolean hasNext() {

// TODO Auto-generated method stub

return false;

}

@Override

public Queue<Instruction> getNestedInstructions() {

// TODO Auto-generated method stub

return null;

}

@Override

public void setNestedInstructions(Queue<Instruction> instructions) {

// TODO Auto-generated method stub

}

@Override

public boolean reset() {

// TODO Auto-generated method stub

return false;

}

}

package edu.buaa.sei.run;

public class BranchInstruction extends AbstractInstruction {

private Queue<Instruction> branches = null;

private Queue<Double> probability = null;

Instruction chosenBranch = null;

public Queue<Instruction> getBranches() {

return branches;

}

public void setBranches(Queue<Instruction> branches) {

this.branches = branches;

}

public Queue<Double> getProbability() {

return probability;

}

public void setProbability(Queue<Double> probability) {

this.probability = probability;

}

public BranchInstruction(Queue<Instruction> branches, Queue<Double> probability) {

this.branches = branches;

this.probability = probability;

}

@Override

public boolean isAtomic() {

return false;

}

@Override

public void loadNextInstruction() {

if (chosenBranch == null) {

// chose one temporally.

chosenBranch = branches.peek();

}

if (chosenBranch.isAtomic()) {

} else {

}

}

@Override

public boolean hasNext() {

// TODO Auto-generated method stub

return false;

}

@Override

public Queue<Instruction> getNestedInstructions() {

// TODO Auto-generated method stub

return null;

}

@Override

public void setNestedInstructions(Queue<Instruction> instructions) {

// TODO Auto-generated method stub

}

@Override

public boolean reset() {

// TODO Auto-generated method stub

return false;

}

}

package edu.buaa.sei.run;

/\*\*

\* Entity stores communication resource requires.

\* @author sei

\*

\*/

public class CommuEntity {

private ResourceContainer from;

private ResourceContainer to;

private int size;

public CommuEntity(ResourceContainer from, ResourceContainer to, int size) {

this.from = from;

this.to = to;

this.size = size;

}

public ResourceContainer getFrom() {

return from;

}

public void setFrom(ResourceContainer from) {

this.from = from;

}

public ResourceContainer getTo() {

return to;

}

public void setTo(ResourceContainer to) {

this.to = to;

}

public int getSize() {

return size;

}

public void setSize(int size) {

this.size = size;

}

}

package edu.buaa.sei.run;

import java.util.ArrayDeque;

import java.util.Iterator;

import java.util.Queue;

/\*\*

\* Class manages list of instructions.

\* @author sei

\*/

public class EmbeddedInstruction extends AbstractInstruction {

// list of instructions stored in this class.

private Queue<Instruction> instructions = null;

// Current instruction.

private Instruction current = null;

// Iterator for the internal queue.

private Iterator<Instruction> iter = null;

/\*\*

\* Constructor for this class.

\*/

public EmbeddedInstruction() {

instructions = new ArrayDeque<Instruction>();

iter = instructions.iterator();

}

public EmbeddedInstruction(Queue<Instruction> instructions) {

this.instructions = instructions;

iter = instructions.iterator();

}

/\*\*

\* Test if this is atomic instruction.

\*/

@Override

public boolean isAtomic() {

return false;

}

private void initResourceDemand() {

this.setProcessorDemand(0);

this.setStorageDemand(0);

this.setCommuDemand(null);

this.setLogicalDemand(null);

this.setResourceType(RESOURCE\_TYPE.NONE);

}

private void initResourceDemand(Instruction instr) {

// Initialize resource type first.

this.setResourceType(instr.getResourceType());

// Set resource demand accordingly.

this.setProcessorDemand(instr.getProcessorDemand());

switch(instr.getResourceType()) {

case PROCESSOR:

// does nothing.

break;

case STORAGE:

this.setStorageDemand(instr.getStorageDemand());

break;

case COMMU:

this.setCommuDemand(instr.getCommuDemand());

break;

case LOGIC:

this.setLogicalDemand(instr.getLogicDemand());

break;

default:

throw new UnsupportedOperationException("Unsupported case occured.");

}

}

/\*\*

\* Test if this instruction have more.

\*/

public boolean hasNext() {

if (current != null) return true;

if (iter.hasNext()) return true;

return false;

}

/\*\*

\* This instruction must be called before get resource demand.

\*/

@Override

public void loadNextInstruction() {

// Initialize resource demand first.

initResourceDemand();

// If there's no more instruction, just return.

if (null == current) {

if (iter.hasNext()) current = iter.next();

else return;

}

// If current instruction is not atomic, and there's no

// more instruction in the embedded instruction. load next

// instruction.

if (!current.isAtomic() && !current.hasNext()) {

if (!iter.hasNext()) return;

current = iter.next();

if (!current.isAtomic()) current.loadNextInstruction();

}

// Initialize resource demand.

initResourceDemand(current);

// Set next instruction accordingly.

if (current.isAtomic()) {

if (!iter.hasNext()) current = null;

else current = iter.next();

} else {

if (current.hasNext()) current.loadNextInstruction();

else current = null;

}

}

/\*\*

\* Manipulate nested instructions.

\*/

@Override

public Queue<Instruction> getNestedInstructions() {

return instructions;

}

@Override

public void setNestedInstructions(Queue<Instruction> instructions) {

this.instructions = instructions;

iter = this.instructions.iterator();

}

@Override

public boolean reset() {

return false;

}

}

package edu.buaa.sei.run;

import java.util.Queue;

public interface Instruction {

/\*\*

\* Test if this instruction is nested.

\* @return

\*/

public boolean isAtomic();

/\*\*

\* Test if this instruction is asynchronous message.

\* @return

\*/

public boolean isAsync();

/\*\*

\* Manipulate instruction command name.

\* @return

\*/

public String getName();

public void setName(String name);

/\*\*

\* Manipulate processor resource demand.

\*/

public int getProcessorDemand();

public void setProcessorDemand(int processorDemand);

/\*\*

\* Manipulate storage resource demand.

\* @return

\*/

public int getStorageDemand();

public void setStorageDemand(int storageDemand);

/\*\*

\* Manipulate logical resource demand.

\*/

public LogicalEntity getLogicDemand();

public void setLogicalDemand(LogicalEntity logicalDemand);

/\*\*

\* Manipulate communication demand.

\* @return

\*/

public CommuEntity getCommuDemand();

public void setCommuDemand(CommuEntity commuDemand);

/\*\*

\* manipulate resource type.

\*/

public RESOURCE\_TYPE getResourceType();

public void setResourceType(RESOURCE\_TYPE resourceType);

/\*\*

\* Load next instruction before get any resource demand. This

\* is for embedded instruction.

\*/

public void loadNextInstruction();

public boolean hasNext();

/\*\*

\* Manipulate list of instructions nested within this instruction. This

\* is for embedded instruction.

\* @return

\*/

public Queue<Instruction> getNestedInstructions();

public void setNestedInstructions(Queue<Instruction> instructions);

/\*\*

\* Reset status of this instruction.

\* @return

\*/

public boolean reset();

}

package edu.buaa.sei.run;

/\*\*

\* Entity stores logical resource requirement.

\* @author sei

\*

\*/

public class LogicalEntity {

private int size;

private String resourceId;

public int getSize() {

return size;

}

public void setSize(int size) {

this.size = size;

}

public String getResourceId() {

return resourceId;

}

public void setResourceId(String resourceId) {

this.resourceId = resourceId;

}

public LogicalEntity(String resourceId, int size) {

this.size = size;

this.resourceId = resourceId;

}

}

package edu.buaa.sei.run;

import java.util.Queue;

public class LoopInstruction extends AbstractInstruction {

private Instruction internal = null;

private int cycles = -1;

public LoopInstruction(Instruction internal) {

this.internal = internal;

}

@Override

public boolean isAtomic() {

return false;

}

@Override

public void loadNextInstruction() {

// assign an loop temporally.

if (cycles == -1) cycles = 10;

if (internal.isAtomic()) {

} else {

if (!internal.hasNext()) {

cycles--;

internal.reset();

}

internal.loadNextInstruction();

}

}

@Override

public boolean hasNext() {

// TODO Auto-generated method stub

return false;

}

@Override

public Queue<Instruction> getNestedInstructions() {

// TODO Auto-generated method stub

return null;

}

@Override

public void setNestedInstructions(Queue<Instruction> instructions) {

// TODO Auto-generated method stub

}

@Override

public boolean reset() {

// TODO Auto-generated method stub

return false;

}

}

package edu.buaa.sei.run;

import java.util.Queue;

public class NormalInstruction extends AbstractInstruction {

/\*\*

\* Method test if this is atomic instruction.

\*/

@Override

public boolean isAtomic() {

return true;

}

/\*\*

\* Get nested instructions.

\*/

@Override

public Queue<Instruction> getNestedInstructions() {

throw new UnsupportedOperationException("getNestedInstructions operation unsupported");

}

@Override

public void loadNextInstruction() {

throw new UnsupportedOperationException("loadNextInstruction operation unsupported");

}

@Override

public void setNestedInstructions(Queue<Instruction> instructions) {

throw new UnsupportedOperationException("setNestedInstructions operation unsupported");

}

@Override

public boolean hasNext() {

return false;

}

@Override

public boolean reset() {

return true;

}

}

package edu.buaa.sei.run;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Class record the execution of processes.

\* @author sei

\* The content of record includes:

\* 1. The execution time of an action;

\* 2. The require time of an logical resource;

\* 3. The release time of an logical resource;

\* 4. The require time of an linking resource;

\* 5. The complete time of an linking resource;

\* 6. The require time of an storage resource;

\* 7. The complete time of an storage resource;

\*/

public class ISchedulableProcessObserver {

List<ProcessRecord> records = null;

private class ProcessRecord {

public ProcessRecord(int time, RecordEvent event) {

this.time= time;

this.event = event;

}

public int time;

public RecordEvent event;

}

private enum RecordEvent {

ExecuteAction,

CompleteAction,

RequireLogicalResource,

AcquireLogicalResource,

ReleaseLogicalResource,

RequireStorageResource,

AcquireStorageResource,

CompleteStorageResource,

RequireLinkingResource,

AcquireLinkingResource,

CompleteLinkingResource,

}

public ISchedulableProcessObserver() {

records = new ArrayList<ProcessRecord>();

}

public void recordActionExcution(Instruction instr, int time) {

ProcessRecord record = new ProcessRecord(time, RecordEvent.ExecuteAction);

records.add(record);

}

public void recordActionCompletion(Instruction instr, int time) {

ProcessRecord record = new ProcessRecord(time, RecordEvent.CompleteAction);

records.add(record);

}

public void recordRequireLogicalResource(Instruction instr, int time) {

ProcessRecord record = new ProcessRecord(time, RecordEvent.ReleaseLogicalResource);

records.add(record);

}

public void recordAcquireLogicalResource(Instruction instr, int time) {

ProcessRecord record = new ProcessRecord(time, RecordEvent.AcquireLogicalResource);

records.add(record);

}

public void recordReleaseLogicalResource(Instruction instr, int time) {

ProcessRecord record = new ProcessRecord(time, RecordEvent.ReleaseLogicalResource);

records.add(record);

}

}

package edu.buaa.sei.run;

import java.util.Queue;

public interface ISchedulableProcess extends IResource {

/\*\*

\* Notifies the process to resume its execution. Only called if

\* has been passivated.

\*/

public void active();

/\*\*

\* Notifies the process to stop its execution.

\*/

public void passivate();

/\*\*

\* Notifies the process to start its execution.

\*/

public void start();

/\*\*

\* Test if this process has finished.

\* @return

\*/

public boolean isFinished();

/\*\*

\* Proceed this process with passed processor time resource.

\* If the passed time resource is all consumed, return -1,

\* else return the left time resource not used.

\* @param processed

\* @return

\*/

public int proceed(int time);

/\*\*

\* Proceed this process, satisfying all its request. And forward

\* with next resource require.

\*/

public void proceed();

/\*\*

\* Manipulate process state.

\*/

public PROCESS\_STATE getProcessState();

public void setProcessState(PROCESS\_STATE state);

/\*\*

\* Manipulate Embedded instructions.

\*/

public Queue<Instruction> getInstructions();

public void setInstructions(Queue<Instruction> instr);

/\*\*

\* Manipulate processor resource demand.

\*/

public int getProcessorDemand();

public void setProcessorDemand(int processorDemand);

/\*\*

\* Manipulate storage resource demand.

\* @return

\*/

public int getStorageDemand();

public void setStorageDemand(int storageDemand);

/\*\*

\* Manipulate logical resource demand.

\*/

public LogicalEntity getLogicDemand();

public void setLogicalDemand(LogicalEntity logicalDemand);

/\*\*

\* Manipulate communication demand.

\* @return

\*/

public CommuEntity getCommuDemand();

public void setCommuDemand(CommuEntity commuDemand);

/\*\*

\* Manipulate resource type.

\*/

public RESOURCE\_TYPE getResourceType();

public void setResourceType(RESOURCE\_TYPE resourceType);

/\*\*

\* Manipulate processor.

\*/

public ActiveResourceProcessor getProcessor();

public void setProcessor(ActiveResourceProcessor processor);

}

package edu.buaa.sei.run;

import java.util.Queue;

// Default process class simulating behavior of process.

public class SchedulableProcess extends AbstractResource implements ISchedulableProcess {

private PROCESS\_STATE state = PROCESS\_STATE.READY;

private boolean finished = false;

// required process time.

private int processorDemand = 0;

// required storage size.

private int storageDemand = 0;

// required communication resource.

private CommuEntity commuDemand = null;

// required logical resource.

private LogicalEntity logicalDemand = null;

// resource type.

private RESOURCE\_TYPE resourceType = RESOURCE\_TYPE.NONE;

// the instruction processed currently.

private EmbeddedInstruction instructions = null;

// Processor processing this process.

private ActiveResourceProcessor processor = null;

// Switch time for process.

private int switchTime;

/\*\*

\* Constructor for this class.

\* @param name

\* @param id

\* @param capacity

\*/

public SchedulableProcess(String name, String id, int capacity) {

super(name, id, capacity);

instructions = new EmbeddedInstruction();

}

public SchedulableProcess(String name, String id, int capacity,

Queue<Instruction> instructions) {

super(name, id, capacity);

this.instructions = new EmbeddedInstruction(instructions);

}

/\*\*

\* Start execution of this process.

\*/

@Override

public void start() {

instructions.loadNextInstruction();

recordAndLoadResourceDemand();

}

@Override

public void active() {

state = PROCESS\_STATE.READY;

throw new UnsupportedOperationException("Active not implemented.");

}

@Override

public void passivate() {

state = PROCESS\_STATE.READY;

throw new UnsupportedOperationException("Passivate not implemented.");

}

private void recordAndLoadResourceDemand() {

while (resourceType == RESOURCE\_TYPE.NONE) {

processorDemand = instructions.getProcessorDemand();

storageDemand = instructions.getStorageDemand();

logicalDemand = instructions.getLogicDemand();

commuDemand = instructions.getCommuDemand();

resourceType = instructions.getResourceType();

if (instructions.hasNext()) instructions.loadNextInstruction();

else break;

}

}

/\*\*

\* Proceed this process with passed processor time resource.

\*/

@Override

public int proceed(int time) {

while ( time > 0 ) {

// If allocated time is not enough.

if (time < processorDemand) {

processorDemand -= time;

return -1;

}

time -= processorDemand;

if (getResourceType() == RESOURCE\_TYPE.PROCESSOR)

recordAndLoadResourceDemand();

else break;

}

return time;

}

@Override

public void proceed() {

// Before proceed, processorDemand must be zero.

if (processorDemand > 0) throw new RuntimeException("invalide program");

recordAndLoadResourceDemand();

}

/\*\*

\* Manipulate nested instructions.

\*/

public Queue<Instruction> getInstructions() {

return instructions.getNestedInstructions();

}

public void setInstructions(Queue<Instruction> instructions) {

this.instructions.setNestedInstructions(instructions);

}

/\*\*

\* Test if this process has finished.

\*/

@Override

public boolean isFinished() {

return finished;

}

@Override

public int getProcessorDemand() {

return processorDemand;

}

@Override

public int getStorageDemand() {

return storageDemand;

}

@Override

public CommuEntity getCommuDemand() {

return commuDemand;

}

@Override

public LogicalEntity getLogicDemand() {

return logicalDemand;

}

@Override

public void setProcessorDemand(int processorDemand) {

this.processorDemand = processorDemand;

}

@Override

public void setStorageDemand(int storageDemand) {

this.storageDemand = storageDemand;

}

@Override

public void setLogicalDemand(LogicalEntity logicalDemand) {

this.logicalDemand = logicalDemand;

}

@Override

public void setCommuDemand(CommuEntity commuDemand) {

this.commuDemand = commuDemand;

}

@Override

public RESOURCE\_TYPE getResourceType() {

return resourceType;

}

@Override

public void setResourceType(RESOURCE\_TYPE resourceType) {

this.resourceType = resourceType;

}

@Override

public PROCESS\_STATE getProcessState() {

return state;

}

@Override

public void setProcessState(PROCESS\_STATE state) {

this.state = state;

}

@Override

public ActiveResourceProcessor getProcessor() {

return processor;

}

@Override

public void setProcessor(ActiveResourceProcessor processor) {

this.processor = processor;

}

public int getSwitchTime() {

return switchTime;

}

public void setSwitchTime(int switchTime) {

this.switchTime = switchTime;

}

}

package edu.buaa.sei.run;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

/\*\*

\* ResourceContainer contains resources within a node.

\* Do not support nested container currently.

\* @author sei

\*

\*/

public class ResourceContainer extends AbstractResource {

// list of processes in this container.

private List<ISchedulableProcess> processes;

// list of processors.

private List<ActiveResourceProcessor> processorResources;

// storage resource, only one in container.

private ActiveResourceStorage storageResource;

// list of communication resource

private List<ActiveResourceLinking> commuResources;

// list of passive resources, i.e. logical resource.

private List<IPassiveResource> passiveResources;

// load balancer manager processors of this container.

private ILoadBalancer loadBalancer;

// Public constructor.

public ResourceContainer(String name, String id, int capacity) {

super(name, id, capacity);

processes = new ArrayList<ISchedulableProcess>();

processorResources = new ArrayList<ActiveResourceProcessor>();

storageResource = null;

commuResources = new ArrayList<ActiveResourceLinking>();

passiveResources = new ArrayList<IPassiveResource>();

loadBalancer = new LoadBalancer(processorResources);

}

/\*\*

\* Add passive resource to this container.

\*/

public boolean addPassiveResource(IPassiveResource resource) {

if (passiveResources.contains(resource)) return false;

passiveResources.add(resource);

return true;

}

/\*\*

\* Start the resources in this container.

\*/

public void start() {

// start all processors.

Iterator<ActiveResourceProcessor> processorIter = processorResources.iterator();

while (processorIter.hasNext()) {

processorIter.next().start();

}

// start storage resource.

// start communication resource.

// Insert all processes into load balancer for process dispatch.

Iterator<ISchedulableProcess> processIter = processes.iterator();

while (processIter.hasNext()) {

ISchedulableProcess process = processIter.next();

process.start();

loadBalancer.addProcess(process);

}

}

/\*\*

\* Find the linking resource.

\* @param container

\* @return

\*/

private ActiveResourceLinking getLinkingResource(ResourceContainer container) {

Iterator<ActiveResourceLinking> iter = commuResources.iterator();

while (iter.hasNext()) {

ActiveResourceLinking resource = iter.next();

if (resource.getTo().equals(container)) return resource;

if (resource.getFrom().equals(container)) return resource;

}

return null;

}

/\*\*

\* Find the logical resource.

\*/

private IPassiveResource getPassiveResource(String resourceId) {

IPassiveResource resource = null;

Iterator<IPassiveResource> iter = passiveResources.iterator();

while (iter.hasNext()) {

resource = iter.next();

if (resource.getId() == resourceId) return resource;

}

return null;

}

public boolean rescheduleProcess(ISchedulableProcess process) {

if (process.getProcessorDemand() > 0) {

if (process.getProcessor() != null)

process.getProcessor().process(process);

else {

process.start();

loadBalancer.addProcess(process);

}

}

switch (process.getResourceType()) {

case PROCESSOR:

throw new RuntimeException("Invalide resource required");

case STORAGE:

storageResource.process(process);

break;

case COMMU:

CommuEntity demand = process.getCommuDemand();

ResourceContainer container = demand.getTo();

ActiveResourceLinking resource = getLinkingResource(container);

if (resource == null) throw new RuntimeException("Invalide resource required");

resource.process(process);

break;

case LOGIC:

LogicalEntity logicalEntity = process.getLogicDemand();

IPassiveResource logicalResource =

getPassiveResource(logicalEntity.getResourceId());

if (logicalResource == null) throw new RuntimeException("Invalide resource required");

logicalResource.acquire(process, logicalEntity.getSize(), -1);

break;

case NONE:

// The termination of this process. Left empty.

default:

// This should never happen.

throw new RuntimeException("This should never happend");

}

return true;

}

}

package edu.buaa.sei.resource.passive;

import java.util.ArrayDeque;

import java.util.Iterator;

import java.util.Queue;

/\*\*

\* Simulates a simple passive resource.

\*

\* @author sei

\*

\*/

public class PassiveResource extends AbstractResource implements IPassiveResource {

public PassiveResource(String name, String id, int capacity) {

super(name, id, capacity);

}

protected Queue<ISchedulableProcess> waitingProcesses;

private int available;

/\*\*

\* Test if this process can execute forward.

\* @param process

\* @param num

\* @return

\*/

private boolean canProceed(ISchedulableProcess process, int num) {

if (waitingProcesses.isEmpty() || waitingProcesses.peek().equals(process))

return num <= available;

return false;

}

private void grantAccess(ISchedulableProcess process, int num) {

available -= num;

process.proceed();

int time = SimulationController.instance.currentTime() + process.getProcessorDemand();

LogicalAcquireFinishedEvent event =

new LogicalAcquireFinishedEvent(this, process, time);

event.schedule();

}

//this is a sugar, add on later.

private void processTimeout(double timeoutValue,

ISchedulableProcess process) {

if (timeoutValue <= 0.0) {

throw new RuntimeException("timeout value invalide");

} else {

int time = (int) (SimulationController.instance.currentTime() + timeoutValue);

PassiveResourceTimeoutEvent event = new PassiveResourceTimeoutEvent(

process, this, time);

event.schedule();

}

}

/\*\*

\* Handle process require, timeout is not implemented current.

\* @param process

\* @param num

\* @param timeout

\* @param timeoutValue

\* @return

\*/

@Override

public boolean acquire(ISchedulableProcess process, int num,

double timeoutValue) {

if (num > getCapacity()) {

throw new RuntimeException("Too much resource required");

}

// grant num instance to process if available;

if (canProceed(process, num)) {

grantAccess(process, num);

return true;

} else {

process.passivate();

processTimeout(timeoutValue, process);

waitingProcesses.add(process);

return false;

}

}

/\*\*

\* Release this resource.

\* @param process

\* @param num

\*/

@Override

public void release(ISchedulableProcess process, int num) {

this.available += num;

notifyWaitingProcesses();

}

private void notifyWaitingProcesses() {

ISchedulableProcess process = waitingProcesses.peek();

while (process != null && canProceed(process, process.getLogicDemand().getSize())) {

grantAccess(process, process.getLogicDemand().getSize());

waitingProcesses.remove();

process.active();

process = waitingProcesses.peek();

}

}

@Override

public int getAvailable() {

return available;

}

@Override

public void setAvailable(int available) {

// TODO Auto-generated method stub

this.available = available;

}

@Override

public Queue<ISchedulableProcess> getWaitingProcesses() {

Queue<ISchedulableProcess> queue = new ArrayDeque<ISchedulableProcess>();

Iterator<ISchedulableProcess> iter = waitingProcesses.iterator();

while (iter.hasNext()) {

queue.add(iter.next());

}

return queue;

}

@Override

public boolean isWaiting(ISchedulableProcess process) {

return waitingProcesses.contains(process);

}

@Override

public void cancelWaiting(ISchedulableProcess process) {

if (waitingProcesses.contains(process))

waitingProcesses.remove(process);

}

}

package edu.buaa.sei.resource.passive;

import java.util.Queue;

/\*\*

\* A passive can be hold by a process for some time. As the number of available

\* instances is limited the process might has to wait until it gets the

\* requested number of instances.

\* @author sei

\*/

public interface IPassiveResource extends IResource{

/\*\*

\* Acquires num instances of the passive resources for the given process. The

\* process is blocked until it successfully receive the requested number of

\* resource instances.

\*

\* @param process

\* @param num

\* @param timeout

\* @param timeoutValue

\* @return

\*/

public boolean acquire(ISchedulableProcess process, int demand,

double timeoutValue);

/\*\*

\* Releases num instances of the passive resource form the given process.

\* @param process

\* @param num

\*/

public void release(ISchedulableProcess process, int num);

/\*\*

\* returns the maximal number of instances that can be acquired at the same

\* time.

\* @return

\*/

public int getCapacity();

public void setCapacity(int capacity);

/\*\*

\* Returns the number of remaining instances.

\* @return

\*/

public int getAvailable();

public void setAvailable(int available);

/\*\*

\* Return a queue containing the waiting processes for this passive resource.

\*/

public Queue<ISchedulableProcess> getWaitingProcesses();

/\*\*

\* Test if this process is waiting

\* @param process

\* @return

\*/

public boolean isWaiting(ISchedulableProcess process);

public void cancelWaiting(ISchedulableProcess process);

}

package edu.buaa.sei.resource.active;

import edu.buaa.sei.processes.ISchedulableProcess;

import edu.buaa.sei.resource.IResource;

/\*\*

\* An active resource can execute demands of schedulabe processes. Active

\* resource are shared by multiple processes so that they need to use

\* scheduling strategies to assign processing time of the resources to process.

\* @author sei

\*/

public interface IActiveResource extends IResource {

/\*\*

\* process the specified demand of the process.

\*/

public void process(ISchedulableProcess process);

/\*\*

\* Release the resource.

\*/

public void release(ISchedulableProcess process);

/\*\*

\*

\*/

public boolean canProcess(ISchedulableProcess process);

/\*\*

\* Creates the initial events for the resource.

\*/

public void start();

/\*\*

\* Creates the resume events for the resource.

\*/

public void resume();

/\*\*

\* Creates the terminate events for the resource.

\*/

public void stop();

}

package edu.buaa.sei.resource.active;

import edu.buaa.sei.controller.SimulationController;

import edu.buaa.sei.event.StorageFinishedEvent;

import edu.buaa.sei.processes.ISchedulableProcess;

public class ActiveResourceStorage extends AbstractActiveResource{

//speed of this storage resource.

int speed = 10;

// headTime of this storage resource.

int headTime = 0;

public ActiveResourceStorage(String name, String id) {

super(name, id, 1);

}

private int calculateTime(int demand) {

return headTime + demand / speed;

}

@Override

public void process(ISchedulableProcess process) {

waitingProcesses.add(process);

process();

}

private void process() {

ISchedulableProcess process = waitingProcesses.remove();

int time = calculateTime(process.getStorageDemand()) +

SimulationController.instance.currentTime();

process.proceed();

StorageFinishedEvent event = new StorageFinishedEvent(this, process, time);

event.schedule();

}

@Override

public void start() {

// left empty

}

@Override

public void resume() {

// left empty

}

@Override

public void stop() {

// left empty

}

@Override

public void release(ISchedulableProcess process) {

setCapacity(1);

}

}

package edu.buaa.sei.resource.active;

import edu.buaa.sei.controller.SimulationController;

import edu.buaa.sei.event.ProcessorFinishedEvent;

import edu.buaa.sei.processes.ISchedulableProcess;

import edu.buaa.sei.resource.RESOURCE\_STATE;

import edu.buaa.sei.resource.ResourceContainer;

/\*\*

\* Default class simulation processor with sharing strategy.

\* @author sei

\*/

public class ActiveResourceProcessor extends AbstractActiveResource {

// Time slice for each process.

int timeSlice = 20;

// Switch time of this processor.

int switchTime = 0;

// Current state of resource.

RESOURCE\_STATE state;

ResourceContainer container;

public ActiveResourceProcessor(String name, String id,

ResourceContainer container) {

super(name, id, 1);

this.container = container;

}

@Override

public void process(ISchedulableProcess process) {

waitingProcesses.add(process);

process();

}

/\*\*

\* Process next process in the waiting list.

\*/

private void process() {

ISchedulableProcess process = waitingProcesses.peek();

if (process == null) return;

if (!canProcess(process)) return;

// execute this process if processor is available.

waitingProcesses.remove();

setCapacity(0);

int currentTime = SimulationController.instance.currentTime();

int remainingTime = process.proceed(timeSlice);

ProcessorFinishedEvent event = new ProcessorFinishedEvent(this, process,

timeSlice - remainingTime + currentTime + switchTime);

event.schedule();

}

/\*\*

\* Add this process to correspondence demand according to its resource demand.

\*/

public void dispatchProcess(ISchedulableProcess process) {

container.rescheduleProcess(process);

}

@Override

public void start() {

state = RESOURCE\_STATE.RUN;

}

@Override

public void resume() {

state = RESOURCE\_STATE.SUSPEND;

}

@Override

public void stop() {

state = RESOURCE\_STATE.STOP;

}

@Override

public void release(ISchedulableProcess process) {

setCapacity(1);

process();

}

}

package edu.buaa.sei.resource.active;

import edu.buaa.sei.controller.SimulationController;

import edu.buaa.sei.event.LinkingFinishedEvent;

import edu.buaa.sei.processes.ISchedulableProcess;

import edu.buaa.sei.resource.ResourceContainer;

public class ActiveResourceLinking extends AbstractActiveResource implements Runnable {

private ResourceContainer from;

private ResourceContainer to;

private int speed = 10;

private int headTime = 0;

public ActiveResourceLinking(String name, String id, int capacity,

ResourceContainer from, ResourceContainer to) {

super(name, id, capacity);

this.from = from;

this.to = to;

}

public ResourceContainer getFrom() {

return from;

}

public void setFrom(ResourceContainer from) {

this.from = from;

}

public ResourceContainer getTo() {

return to;

}

public void setTo(ResourceContainer to) {

this.to = to;

}

private int calculateTime(int demand) {

return headTime + demand / speed;

}

@Override

public void process(ISchedulableProcess process) {

waitingProcesses.add(process);

process();

}

private void process() {

ISchedulableProcess process = waitingProcesses.peek();

if (!canProcess(process)) return;

waitingProcesses.remove();

int time = calculateTime(process.getCommuDemand().getSize()) +

SimulationController.instance.currentTime();

process.proceed();

LinkingFinishedEvent event = new LinkingFinishedEvent(this, process, time);

event.schedule();

setCapacity(0);

}

@Override

public void start() {

}

@Override

public void resume() {

// TODO Auto-generated method stub

}

@Override

public void stop() {

// TODO Auto-generated method stub

}

@Override

public void run() {

// TODO Auto-generated method stub

}

@Override

public void release(ISchedulableProcess process) {

setCapacity(1);

process();

}

}

package edu.buaa.sei.resource.active;

import java.util.ArrayDeque;

import java.util.Queue;

import edu.buaa.sei.processes.ISchedulableProcess;

import edu.buaa.sei.resource.AbstractResource;

public abstract class AbstractActiveResource extends AbstractResource

implements IActiveResource {

protected Queue<ISchedulableProcess> waitingProcesses;

public AbstractActiveResource(String name, String id, int capacity) {

super(name, id, capacity);

waitingProcesses = new ArrayDeque<ISchedulableProcess>();

}

public int getWaitingLength() {

return waitingProcesses.size();

}

@Override

public boolean canProcess(ISchedulableProcess process) {

if (getCapacity() > 0) return true;

return false;

}

}

package edu.buaa.sei.loadBalancer;

import java.util.List;

import edu.buaa.sei.processes.ISchedulableProcess;

import edu.buaa.sei.resource.active.ActiveResourceProcessor;

/\*\*

\* Load balancer for new created process only. Process is executed

\* in the same processor until finished. Currently, the load balancer

\* does not consume time.

\* @author sei

\*

\*/

public interface ILoadBalancer {

/\*\*

\* Add a processor to this load balancer. If this processor has been

\* added before, return false. Otherwise return true.

\* @param processor

\* @return

\*/

public boolean addProcessor(ActiveResourceProcessor processor);

/\*\*

\*

\* @param process

\* @return

\*/

public void addProcess(ISchedulableProcess process);

}

package edu.buaa.sei.resource;

import java.util.ArrayDeque;

import java.util.Queue;

import org.junit.Test;

import edu.buaa.sei.controller.ISimulationController;

import edu.buaa.sei.controller.SimulationController;

import edu.buaa.sei.instructions.Instruction;

import edu.buaa.sei.instructions.NormalInstruction;

import edu.buaa.sei.processes.ISchedulableProcess;

import edu.buaa.sei.processes.RESOURCE\_TYPE;

import edu.buaa.sei.processes.SchedulableProcess;

import edu.buaa.sei.resource.active.ActiveResourceProcessor;

public class SimulationTest {

@Test

public void test() {

Queue<Instruction> queue = buildInstructions();

ISimulationController controller = SimulationController.instance;

ISchedulableProcess process = buildProcess(queue);

ResourceContainer container = buildResource();

controller.addContainer(container);

container.addProcess(process)

controller.start();

}

Queue<Instruction> buildInstructions() {

Queue<Instruction> queue = new ArrayDeque<Instruction>();

for (int i = 0; i <= 20; ++i) {

Instruction instr = new NormalInstruction();

instr.setName("instr" + i);

instr.setResourceType(RESOURCE\_TYPE.PROCESSOR);

instr.setProcessorDemand(100);

queue.add(instr);

}

return queue;

}

ResourceContainer buildResource() {

ResourceContainer container = new ResourceContainer("container", "12345", 1000);

ActiveResourceProcessor processor = new ActiveResourceProcessor("processor", "123456", container);

container.addProcessorResource(processor);

return container;

}

ISchedulableProcess buildProcess(Queue<Instruction> queue) {

ISchedulableProcess process = new SchedulableProcess("process", "12345", 0);

process.setInstructions(queue);

return process;

}

}

package edu.buaa.sei.resource;

public class ThreadProcess implements Runnable {

Thread thread;

public void start() {

thread = new Thread(this);

thread.start();

}

public void suspend() {

try {

thread.wait();

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public void resume() {

thread.notify();

}

public void stop() {

}

@Override

public void run(){

try {

System.out.println("print before wait");

System.out.println("print after wait");

} catch (Exception e) {

System.out.println("Exception happends");

}

}

}

package edu.buaa.sei.run;

import java.util.ArrayDeque;

import java.util.Iterator;

import java.util.Queue;

/\*\*

\* Class manages list of instructions.

\* @author sei

\*/

public class EmbeddedInstruction extends AbstractInstruction {

// list of instructions stored in this class.

private Queue<Instruction> instructions = null;

// Current instruction.

private Instruction current = null;

// Iterator for the internal queue.

private Iterator<Instruction> iter = null;

/\*\*

\* Constructor for this class.

\*/

public EmbeddedInstruction() {

instructions = new ArrayDeque<Instruction>();

iter = instructions.iterator();

}

public EmbeddedInstruction(Queue<Instruction> instructions) {

this.instructions = instructions;

iter = instructions.iterator();

}

/\*\*

\* Test if this is atomic instruction.

\*/

@Override

public boolean isAtomic() {

return false;

}

private void initResourceDemand() {

this.setProcessorDemand(0);

this.setStorageDemand(0);

this.setCommuDemand(null);

this.setLogicalDemand(null);

this.setResourceType(RESOURCE\_TYPE.NONE);

}

private void initResourceDemand(Instruction instr) {

// Initialize resource type first.

this.setResourceType(instr.getResourceType());

// Set resource demand accordingly.

this.setProcessorDemand(instr.getProcessorDemand());

switch(instr.getResourceType()) {

case PROCESSOR:

// does nothing.

break;

case STORAGE:

this.setStorageDemand(instr.getStorageDemand());

break;

case COMMU:

this.setCommuDemand(instr.getCommuDemand());

break;

case LOGIC:

this.setLogicalDemand(instr.getLogicDemand());

break;

default:

throw new UnsupportedOperationException("Unsupported case occured.");

}

}

/\*\*

\* Test if this instruction have more.

\*/

public boolean hasNext() {

if (current != null) return true;

if (iter.hasNext()) return true;

return false;

}

public static int getKbFromStr(String str) {

char metric = str.charAt(str.length()-1);

int multiSize = 0;

if (metric == 'M' || metric == 'm')

multiSize = 1024;

else if (metric == 'G' || metric == 'g')

multiSize = 1024\*1024;

else if (metric == 'K' || metric == 'k')

multiSize = 1;

else

System.*out*.println("no such metric : " + metric);

instr.setResourceType(RESOURCE\_TYPE.PROCESSOR);

instr.setProcessorDemand(100);

String num = str.substring(0, str.length()-1);

int rv = Integer.*valueOf*(num);

return rv\*multiSize;

}

}