Министерство информационных технологий и связи РФ

Сибирский государственный университет телекоммуникаций и информатики

Факультет информатики и вычислительной техники

Кафедра вычислительных систем

# Курсовая работа

по курсу Моделирование

Выполнили: Степанов Е.Е.

Гордеев С.А.

Гомзяков А.В.

студенты гр.ВМ-37

Проверил: Рудых Я.И.

Новосибирск 2005

**Содержание**

1. Постановка задачи
2. Описание модели в терминах PDEVS формализма
3. Атомарные компоненты
4. Полученные результаты
5. Основные фрагменты кода

Вывод

1. **Постановка задачи**

Модель состоит из трех обслуживающих серверов. Каждый сервер имеет очередь, в которой заявки могут ожидать своей очереди. Также есть генератор сообщений. Первая очередь бесконечная, остальные конечные. Обработки заявок всех серверов распределены экспоненциально. В начальный момент времени очереди. Необходимо построить модель в терминах PDEVS-формализма и произвести эксперименты над моделью с помощью пакета DEJaView.

Необходимо ответить на следующие вопросы:

1. Выдать статистику по всем очередям.
2. Максимальную длину первой очереди.
3. Сколько процентов сообщений прошло через очередь без задержек - «сквозняки».

Рис. 1. Схематическое изображение модели

1. **Описание модели в терминах PDEVS формализма**

В PDEVS-модели существует 7 компонент:

Queue1, Queue2 и Queue3 – это три очереди.

Server1, Server2 и Server3 – это три прибора (сервера). Время обслуживания распределено экспоненциально.

MessageGenerator - генератор сообщений.

Рассмотрим более подробно логику работы компонент.

1. Генератор подает сообщение в очередь Queue1.

2. В начальный момент времени все серверы находятся в состоянии free (свободен). Cерверы Server1, Server2 посылают сообщение на очереди Queue2 и Queue3 соответственно, это говорит о готовности серверов принимать сообщения.

1. Приняв сообщение каждая очередь подаёт сообщение на выход toNext и он приходит на вход fromPrev соответствующего каждой очереди сервера.
2. Сервет меняет своё состояние с free (свободен) на busy (занят) и через некоторое время подаёт сообщение на выход toNext и после этого ждёт от следующей очереди подтверждения что в данной очереди ещё есть хотя бы одно место.
3. После того как сервер отправил сообщение слёдующей очереди, он переходит в состояние free (свободен) и посылает предыдущей очереди сообщение, которое говорит о готовности сервера принимать следующее сообщение.
4. Также существует возможность перехода сообщения с Серверов в очередь Queue1 с вероятностями P1, P2, P3.

**3.Атомарные компоненты**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Класс атомарной компоненты | Объекты класса атомарной компоненты | Возможные состояния компоненты | Входные порты | Выходные порты |
| Server1 | Server1 | Free, Busy | FromPrev,FromNext | ToPrev,toNext,Vozvrat |
| Server2 | Server2 | Free, Busy | FromPrev,FromNext | ToPrev,toNext,Vozvrat |
| Server3 | Server3 | Free, Busy | FromPrev,FromNext | ToPrev,Vozvrat |
| Queue1 | Queue1 | Free, Full | FromPrev,FromNext,Vozvrat | ToPrev,toNext |
| Queue2 | Queue2 | N=={1,2,3,4,5} | FromPrev,FromNext | ToPrev,toNext |
| Queue3 | Queue3 | N=={1,2,3,4,5} | FromPrev,FromNext | ToPrev,toNext |

Алгоритмы функционирования компонент:

Компоненты классов Server1 и Server2:

Delta\_int:

Остаемся в текущем состоянии

Delta\_exp

Если (пришел новый пакет от очереди)

{

переходим в состояние “busy”

}

Lambda

Если(Сообщение от предыдущей очереди){

Если пакет обработан, то пытаемся отправить его следующей очереди.

Случайно определяем куда пойдёт сообщение – или в начало цепи, или в следующую очередь.

Отсылаем предыдущей очереди уведомление о том что сервер освободился.

}

Компоненты класса Queue2,Queue3:

Delta\_int:

остаемся в текущем состоянии

Delta\_exp

Если (пришёл новый пакет)

{

Если (Флаг ожидания, ожидания освобождения сервера)

 {

Увеличиваем счётчик поступивших пакетов на 1

Так же выполняем действия необходимые для вычисления средней длины очереди

}

Иначе

Если (Очередь не ждёт освобождения сервера )

{

«Сквозняк»

Выполняем действия по вычислению доли сквозняков

}

Иначе

Если (пришло сообщение от сервера)

{

Флаг готовности сервера ставим в значение истина

}

Lambda

Если (пришёл новый пакет)

{

Если (Сервер свободен)

{

Отсылаем сообщение серверу

«Сквозняк»

Выполняем операции по вычислению доли сквозняков

}

Если (Ожидаем сервер)

{

Если очередь переполнилась посылаем сообщение серверу

}

}

Если (пришло сообщение от сервера)

{

Если(Очередь не пуста){

Посылаем пакет на порт toNext очереди

Уменьшаем длину очереди на 1

}

}

Компоненты классов Server3:

Delta\_int:

Остаемся в текущем состоянии

Delta\_exp

Если (пришел новый пакет от очереди)

{

переходим в состояние “busy”

}

Lambda

Если (сообщение от предыдущей очереди){

Если пакет обработан, то пытаемся отправить его следующей очереди.

Случайно определяем куда пойдёт сообщение – или в начало цепи, или на выход из цепи.

Отсылаем уведомление предыдущей очереди о том, что сервер освободился.

}

Компонент класса Queue1:

Delta\_int:

остаемся в текущем состоянии

Delta\_exp

Если (пришёл новый пакет(или из цепи, или из генератора сообщений))

{

Если (Флаг ожидания, ожидания освобождения сервера)

{

Увеличиваем счётчик поступивших пакетов на 1

Выполняем действия по нахождению макс максимальной длины очереди

}

Иначе

Если (Сервер свободен )

{

«Сквозняк»

Выполняем действия по вычислению доли сквозняков

}

Иначе

Если (пришло сообщение от сервера)

{

Флаг готовности сервера ставим в значение истина

}

Lambda

Если( пришёл новый пакет(из генератора или из цепи) )

{

Если (Сервер свободен)

{

Отсылаем сообщение серверу

«Сквозняк»

Выполняем операции по вычислению доли сквозняков

}

Если (Ожидаем сервер)

{

}

}

Если (пришло сообщение от сервера)

{

Если(Очередь не пуста){

Посылаем пакет на порт toNext очереди

Уменьшаем длину очереди на 1

Выполняем действия по вычислению максимальной длины очереди

}

}

Компонент класса: MessageGenerator.

Если(пришло системное сообщение)

{

Отправляем сообщение первой очереди

Увеличиваем число сгенерированных сообщений на 1

}

**4. Полученные результаты**

Рис. 3. Результаты работы модели

1. **Основные фрагменты кода**

1.QueueModel.java

package DEJaView.modelLibs.a;

import DEJaView.modelLibs.a.MessageGenerator;

import DEJaView.modelLibs.a.Queue1;

import DEJaView.modelLibs.a.Server1;

import DEJaView.modelLibs.a.Queue2;

import DEJaView.modelLibs.a.Server2;

import DEJaView.modelLibs.a.Queue3;

import DEJaView.modelLibs.a.Server3;

import DEJaView.core.\*;

import java.util.\*;

public class QueueModel extends MULC {

 public static void main(String args[]) {

 MULC queuemodel = new MULC("queuemodel");

 MessageGenerator MessageGenerator1 = new MessageGenerator("MessageGenerator1");

 Queue1 Queue1 = new Queue1("Queue1");

 Queue2 Queue2= new Queue2("Queue2");

 Queue3 Queue3= new Queue3("Queue3");

 Server1 Server1 = new Server1("Server1");

 Server2 Server2=new Server2("Server2");

 Server3 Server3=new Server3("Server3");

 queuemodel.AddComponent(MessageGenerator1);

 queuemodel.AddComponent(Queue1);

 queuemodel.AddComponent(Queue2);

 queuemodel.AddComponent(Queue3);

 queuemodel.AddComponent(Server1);

 queuemodel.AddComponent(Server2);

 queuemodel.AddComponent(Server3);

 MessageGenerator1.addOutPort("toQueue","toQueue");

 Queue1.addInPort("fromMessageGenerator","fromMessageGenerator");

 Queue1.addInPort("fromNext","fromNext");

 Queue1.addInPort("Vozvrat","Vozvrat");

 Queue1.addOutPort("toNext","toNext");

 Queue2.addInPort("fromPrev","fromPrev");

 Queue2.addInPort("fromNext","fromNext");

 Queue2.addOutPort("toPrev","Prev");

 Queue2.addOutPort("toNext","toNext");

 Queue3.addInPort("fromPrev","fromPrev");

 Queue3.addInPort("fromNext","fromNext");

 Queue3.addOutPort("toPrev","toPrev");

 Queue3.addOutPort("toNext","toNext");

 Server1.addInPort("fromPrev","fromPrev");

 Server1.addOutPort("toPrev","toPrev");

 Server1.addOutPort("toNext","toNext");

 Server1.addOutPort("Vozvrat","Vozvrat");

 Server1.addInPort("fromNext","fromNext");

 Server2.addInPort("fromPrev","fromPrev");

 Server2.addOutPort("toPrev","toPrev");

 Server2.addOutPort("toNext","toNext");

 Server2.addOutPort("Vozvrat","Vozvrat");

 Server2.addInPort("fromNext","fromNext");

 Server3.addInPort("fromPrev","fromPrev");

 Server3.addOutPort("toPrev","toPrev");

 Server3.addOutPort("Vozvrat","Vozvrat");

 queuemodel.getIC().addCouple("MessageGenerator1","toQueue","Queue1","fromMessageGenerator");

 queuemodel.getIC().addCouple("Queue1","toNext","Server1","fromPrev");

 queuemodel.getIC().addCouple("Server1","toNext","Queue2","fromPrev");

 queuemodel.getIC().addCouple("Server1","Vozvrat","Queue1","Vozvrat");

 queuemodel.getIC().addCouple("Server1","toPrev","Queue1","fromNext");

 queuemodel.getIC().addCouple("Queue2","toPrev","Server1","fromNext");

 queuemodel.getIC().addCouple("Queue2","toNext","Server2","fromPrev");

 queuemodel.getIC().addCouple("Server2","toNext","Queue3","fromPrev");

 queuemodel.getIC().addCouple("Server2","Vozvrat","Queue1","Vozvrat");

 queuemodel.getIC().addCouple("Server2","toPrev","Queue2","fromNext");

 queuemodel.getIC().addCouple("Queue3","toPrev","Server2","fromNext");

 queuemodel.getIC().addCouple("Queue3","toNext","Server3","fromPrev");

 queuemodel.getIC().addCouple("Server3","Vozvrat","Queue1","Vozvrat");

 queuemodel.getIC().addCouple("Server3","toPrev","Queue3","fromNext");

 queuemodel.init();

 Date d1 = new Date();

 double time =10000.0;

 while ( !(queuemodel.getLocalTime()>time)) {

 queuemodel.getProcessor().Simulate();

 }

 System.out.println("Пакетов отправлено: " +MessageGenerator1.num);

 System.out.println("Число отказов во 2 ой очереди: " +Queue2.numOfRej);

 System.out.println("Число отказов в 3 ей очереди: " +Queue3.numOfRej);

 /\*System.out.println("Осталось в первой очереди " +Queue1.numOfMessages);\*/

 System.out.println("Число возвратов с первого сервера " +Server1.vozvrat);

 System.out.println("Число возвратов со второго сервера " +Server2.vozvrat);

 System.out.println("Число возвратов с третьего сервера " +Server3.vozvrat);

 System.out.println("Число возвратов в первую очередь " +Queue1.vozvrat);

 System.out.println("Средняя длина 2 ой очереди " +(double)Queue2.Dlina/(double)Queue2.Chislo);

 System.out.println("Средняя длина 3 ей очереди " +(double)Queue3.Dlina/(double)Queue3.Chislo);

 System.out.println("Максимальная длина 1 ой очереди " +Queue1.max);

 System.out.println("Процент сквозняков в первой очереди "+(double)Queue1.skvoz\*100/(double)Queue1.num +"%");

 System.out.println("Процент сквозняков во второй очереди "+(double)Queue2.skvoz\*100/(double)Queue2.num +"%");

 System.out.println("Процент сквозняков в третей очереди "+(double)Queue3.skvoz\*100/(double)Queue3.num +"%");

 Date d2=new Date();

 long d = d2.getTime()-d1.getTime();

 System.out.println("Время моделирования:" +d);

 }

}

2.Queue1.java.

package DEJaView.modelLibs.a;

import DEJaView.core.\*;

import java.util.\*;

/\*\* Класс, реализующий работу очереди сообщений (требований) \*/

public class Queue1 extends AtomicPDEVS {

 /\*\*Счётчик, подсчитывающий число пакетов в очереди\*/

 public int numOfMessages = 0;

 /\*\* Флаг, показывающий свободен ли сервер \*/

 private boolean serverIsFree = true;

 public int vozvrat;

 public int skvoz=0;

 public int num=0;

 public int max=0;

 /\*\* Создает объект Queue с заданным именем

 \* @param name имя создаваемого объекта ксласса Client \*/

 protected Queue1(String name) {

 super(name);

 /\* Объекты класса Queue могут находится в одном из 5-тии

 \* состояний, в зависимости от количесва сообщений в очереди

 \*/

 addState("free");

 addState("full");

 }

 /\*\* Инициализация компонента \*/

 protected void init() {

 /\* Описание системного порта \*/

 Port p;

 /\* Задание начального времени \*/

 this.setLastTime(0);

 /\* Задание начального состояния \*/

 this.setPresentState(findState("free"));

 /\* Далее генерируем начальное системное сообщение \*/

 MessagePDEVS init\_m = new MessagePDEVS("", Double.POSITIVE\_INFINITY, this.getLastTime());

 /\* Далее передаем системное сообщение сообщение в очередь сообщений

 \* мультикомпонента, непосредственно содержащего данный компонент: \*/

 /\* 1. Назначение порта \*/

 p = resolveOutPort("system");

 /\* 2. Назначение сообщения \*/

 p.setMessage(init\_m);

 /\* 3. Собственно передача сообщения \*/

 this.getParentMULC().getProcessor().PassMessage(this.getName(), p.getName());

 }

 /\*\* Функция продвижения времени \*/

 protected double ta() {

 return Double.POSITIVE\_INFINITY;

 }

 /\*\* Внутренняя функция транзакции \*/

 protected State delta\_int() {

 return this.getPresentState();

 }

 /\*\* Внешняя функция транзакции \*/

 protected State delta\_ext() {

 State newState = null;

 /\* Если сообщение пришло от сервера, то оно означает, что сервер готов обслуживать

 \* следующее сообщение (требование). В таком случае, если в очереди есть сообщения

 \* (требования), переходим в очереди новое состояние: уменьшаем количество сообщений

 \* (требований), ожидающих в обслуживания очереди на 1. \*/

 if (this.getCurrentPort().getName().equals("fromNext")) {

 serverIsFree = true;

 /\* Если в очереди было одно сообщение (требование), то теперь там не будет ни одного \*/

 if (this.getPresentState().getName().equals("full")) {

 newState = this.getPresentState();

 if(numOfMessages==1)

 newState = findState("free");

 }

 else

 /\* Если в очереди было два сообщения (требования), то теперь там будет одно \*/

 if (this.getPresentState().getName().equals("free")) {

 newState = this.getPresentState();

 }

 }

 else

 /\* Если сообщение (требование) пришло от клиента, то если очередь не заполнена,

 \* "вставляем" это сообщение (требование) в очередь, переходя в новое состояние \*/

 if (this.getCurrentPort().getName().equals("fromMessageGenerator")){

 num++;

 if(!serverIsFree){

 /\* Если в очереди было пять сообщений (требований), то их там и останется пять \*/

 if (this.getPresentState().getName().equals("full")){

 numOfMessages++;

 newState=this.getPresentState();

 }

 else

 /\* Если в очереди не было сообщений (требований), то теперь там будет одно \*/

 if (this.getPresentState().getName().equals("free")) {

 numOfMessages++;

 newState=findState("full");

 }

 }

 else

 if(serverIsFree){

 /\*проверить ещё надо\*/

 newState=findState("free");

 /\* serverIsFree=false;\*/

 /\*serverIsFree=false;\*/

 }

 }

 else

 if(this.getCurrentPort().getName().equals("Vozvrat")){

 num++;

 if(!serverIsFree){

 /\* Если в очереди было пять сообщений (требований), то их там и останется пять \*/

 if (this.getPresentState().getName().equals("full")){

 numOfMessages++;

 newState=this.getPresentState();

 }

 else

 /\* Если в очереди не было сообщений (требований), то теперь там будет одно \*/

 if (this.getPresentState().getName().equals("free")) {

 numOfMessages++;

 newState=findState("full");

 }

 }

 else

 if(serverIsFree){

 /\*проверить ещё надо\*/

 newState=findState("free");

 }

 }

 else newState = this.getPresentState();

 return newState;

 }

 /\*\* Выходная функция (создания списка выходных событий) \*/

 protected LinkedList lambda() {

 LinkedList list = new LinkedList();

 MessagePort mp1 = new MessagePort();

 MessagePDEVS msg1 = new MessagePDEVS();

 /\* System.out.println("Очередь 1");\*/

 /\* Реакция на сообщения от клиента \*/

 if (this.getCurrentPort().getType().equals("fromMessageGenerator")) {

 /\* Если сообщение от клиента приходит в тот момент, когда очередь была пуста, а сервер

 \* свободен, тогда формируется и отправляется на обработку сообщение серверу \*/

 if ((this.getPresentState().getName().equals("free")))

 {

 if(serverIsFree) {

 /\* Занимаем сервер \*/

 serverIsFree = false;

 skvoz++;

 /\* Установка метки времени \*/

 msg1.setTimeStamp(this.getCurrentPort().getMessage().getTimeStamp());

 /\* Назначение выходного порта \*/

 mp1.setPort("toNext");

 /\* Текст для отладки и трассировки \*/

 msg1.setData("from Queue to Server");

 /\* Назначение сообщения на выходной порт \*/

 mp1.setMessage(msg1);

 /\* Добавление в список выходных событий \*/

 list.add(mp1);

 return list;

 }

 }

 }

 else

 /\* Реакция на сообщение от сервера. Очередь реагирует на сообщения от сервера о том,

 \* что сервер свободен, только тогда, когда очередь не пуста \*/

 if(this.getCurrentPort().getType().equals("fromNext")) {

 if(this.getPresentState().getName().equals("full") || (numOfMessages==1)){

 /\* Занимаем сервер \*/

 serverIsFree = false;

 /\* Установка метки времени \*/

 msg1.setTimeStamp(this.getCurrentPort().getMessage().getTimeStamp());

 /\* Назначение выходного порта \*/

 mp1.setPort("toNext");

 /\* Текст для отладки и трассировки \*/

 msg1.setData("from Queue to Server");

 /\* Назначение сообщения на выходной порт \*/

 mp1.setMessage(msg1);

 /\* Добавление в список выходных событий \*/

 list.add(mp1);

 numOfMessages--;

 return list;

 }

 }

 else

 if(this.getCurrentPort().getType().equals("Vozvrat")) {

 vozvrat++;

 /\* Если сообщение от клиента приходит в тот момент, когда очередь была пуста, а сервер

 \* свободен, тогда формируется и отправляется на обработку сообщение серверу \*/

 if ((this.getPresentState().getName().equals("free")) && (serverIsFree)) {

 /\* Занимаем сервер \*/

 serverIsFree = false;

 skvoz++;

 /\* Установка метки времени \*/

 msg1.setTimeStamp(this.getCurrentPort().getMessage().getTimeStamp());

 /\* Назначение выходного порта \*/

 mp1.setPort("toNext");

 /\* Текст для отладки и трассировки \*/

 msg1.setData("from Queue to Server");

 /\* Назначение сообщения на выходной порт \*/

 mp1.setMessage(msg1);

 /\* Добавление в список выходных событий \*/

 list.add(mp1);

 return list;

 }

 }

 if(max<numOfMessages)

 max=numOfMessages;

 return list;

 }

 /\*\* Конфликтная функция транзакции (пуста) \*/

 protected String confluent() {

 return "external";

 }

}

3.Server1.java.

package DEJaView.modelLibs.a;

import DEJaView.core.\*;

import java.util.\*;

import java.util.Random;

/\*\* Класс, реализующий работу сервера, обрабатывающего сообщения (требования) \*/

public class Server1 extends AtomicPDEVS {

 /\*\* Параметр распределения, интенсивность потока \*/

 private final static double sigma = 1.0;

 private final static double P = 0.95;

 private double V;

 public int vozvrat;

 /\*\* Вспомогательная переменная \*/

 private double ta;

 Random ra = new Random();

 /\*\* Создает объект Server с заданным именем

 \* @param name имя создаваемого объекта ксласса Server \*/

 protected Server1(String name) {

 super(name);

 /\* Объекты класса Server могут находиться в одном из двух состояний, в зависимости

 \* от того, занят сервер обработкой сообщения (требования) или нет \*/

 addState("busy");

 addState("free");

 }

 /\*\* Инициализация компонента \*/

 protected void init() {

 /\* Описание системного порта \*/

 Port p;

 /\* Задание начального времени \*/

 this.setLastTime(0);

 /\* Задание начального состояния \*/

 this.setPresentState(findState("free"));

 /\* Далее генерируем начальное системное сообщение \*/

 MessagePDEVS init\_m = new MessagePDEVS("", Double.POSITIVE\_INFINITY, this.getLastTime());

 /\* Далее передаем системное сообщение сообщение в очередь сообщений

 \* мультикомпонента, непосредственно содержащего данный компонент: \*/

 /\* 1. Назначение порта \*/

 p = resolveOutPort("system");

 /\* 2. Назначение сообщения \*/

 p.setMessage(init\_m);

 /\* 3. Собственно передача сообщения \*/

 this.getParentMULC().getProcessor().PassMessage(this.getName(), p.getName());

 }

 protected double ta() {

 if (this.getPresentState().getName().equals("busy")) {

 ta = Generator.genExp(sigma);

 return ta;

 }

 else

 return Double.POSITIVE\_INFINITY;

 }

 protected State delta\_int() {

 return this.getPresentState();

 }

 protected State delta\_ext() {

 State newState = findState("busy");

 return newState;

 }

 protected LinkedList lambda() {

 LinkedList list = new LinkedList();

 MessagePort mp1 = new MessagePort();

 MessagePort mp2 = new MessagePort();

 MessagePDEVS msg1 = new MessagePDEVS();

 MessagePDEVS msg2 = new MessagePDEVS();

 /\* System.out.println("Сервер 1");\*/

 if (this.getCurrentPort().getType().equals("fromPrev")) {

 V=ra.nextDouble();

 if((V<P)||(V==P)){

 msg1.setTimeStamp(this.getLastTime() + ta);

 mp1.setPort("toNext");

 msg1.setData("from Server to Client: Message have being processing from" + this.getLastTime() + " till " + msg1.getTimeStamp());

 mp1.setMessage(msg1);

 list.add(mp1);

 msg2.setTimeStamp(this.getLastTime()+ ta);

 msg2.setPriority(1);

 mp2.setPort("toPrev");

 msg2.setData("from Server to Queue: Server is free");

 mp2.setMessage(msg2);

 list.add(mp2);

 }

 else

 if(V>P){

 msg1.setTimeStamp(this.getLastTime() + ta);

 mp1.setPort("Vozvrat");

 vozvrat++;

 /\* System.out.println("Возврат с первого сервера" +V);\*/

 msg1.setData("from Server to Queue1");

 mp1.setMessage(msg1);

 list.add(mp1);

 msg2.setTimeStamp(this.getLastTime()+ ta);

 msg2.setPriority(1);

 mp2.setPort("toPrev");

 msg2.setData("from Server to Queue: Server is free");

 mp2.setMessage(msg2);

 list.add(mp2);

 }

 }

 return list;

 }

 protected String confluent() {

 return "external";

 }

}

**Вывод**

В ходе проделанной работы были изучены основы моделирования. Также мы получили практические навыки имитационного моделирования.

Подробно был изучен PDEVS-формализм и пакет моделирования систем с дискретными событиями DEJaView. Исследованы принципы функционирования простейших моделей теории массового обслуживания. Разработан и реализован алгоритм функционирования одной из моделей теории массового обслуживания, описанной в терминах PDEVS под DEJaView.