Формула изобретения

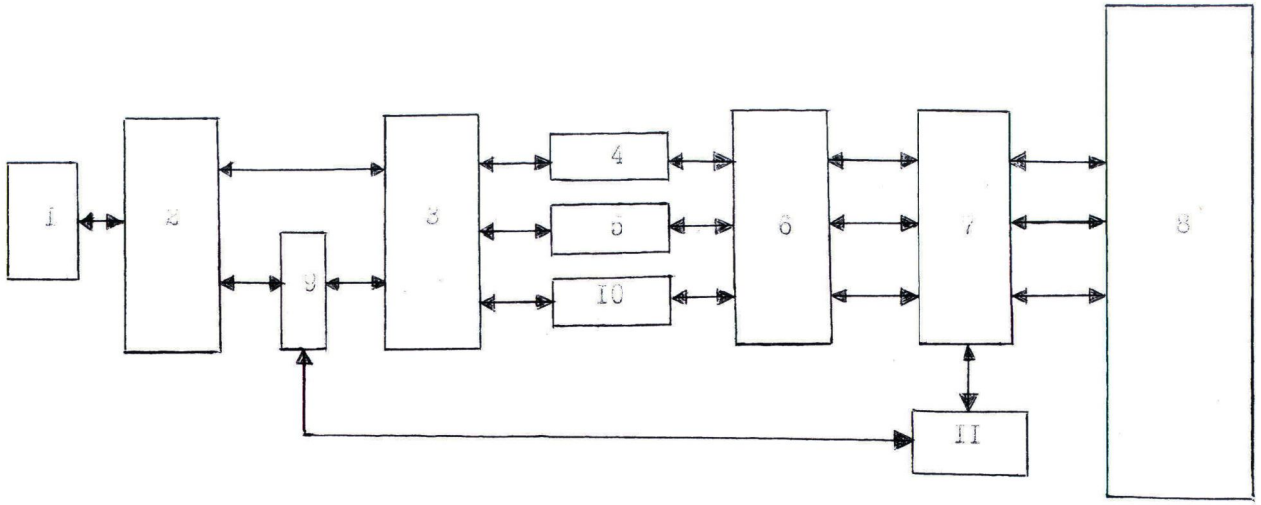
Многоканальная система передачи информации повышенной надёжности на базе лазерной и радио технологий, включающая передачу через беспроводную оптическую связь и отличающаяся тем, что последовательно соединёны источник информации, первый коммутатор и первый коллектор через которые входной сигнал подаётся на присоединённые к коллектору входы параллельно включённых линий связи, оптической и высокоскоростной радиочастотной миллиметрового диапазона, к выходам указанных линий связи присоединены второй коллектор и второй коммутатор через которые сигнал поступает на серверы пользователей; дополнительно, между первым коммутатором и вторым коллектором через синхронизатор включёна высоконадёжная низкоскоростная радиолиния связи сантиметрового диапазона; причём, ко входу/выходу управления второго коллектора присоединён селектор скорости передачи данных, вход/выход которого присоединён ко входу/выходу указанного синхронизатора.

Реферат

Многоканальная система передачи информации повышенной надёжности на базе лазерной и радио технологий, в которой последовательно соединёны источник информации, первый коммутатор и первый коллектор через которые входной сигнал подаётся на присоединённые к коллектору входы параллельно включённых линий связи, оптической и высокоскоростной радиочастотной миллиметрового диапазона, к выходам указанных линий связи присоединены второй коллектор и второй коммутатор через которые сигнал поступает на серверы пользователей; дополнительно, между первым коммутатором и вторым коллектором через синхронизатор включёна высоконадёжная низкоскоростная радиолиния связи сантиметрового диапазона; причём, ко входу/выходу управления второго коллектора присоединён селектор скорости передачи данных, вход/выход которого присоединён ко входу/выходу указанного синхронизатора.

Применение изобретения позволяет: повысить результирующую скорость передачи мультимедийной информации; достигнуть высокую операторскую надёжность; без существенного увеличения стоимости передачи информации обеспечить высокую пропускную способность и надёжность линий связи.

МНОГОКАНАЛЬНАЯ СИСТЕМА ПЕРЕДАЧИ ИНФОРМАЦИИ ПОВЫШЕННОЙ НАДЕЖНОСТИ НА БАЗЕ ЛАЗЕРНОЙ И РАДИО ТЕХНОЛОГИИ



Фиг. 1

Приложение 3.

Заявка на свидетельство о регистрации программ для ЭВМ «Пакет программ аналитического и машинного (имитационного) моделирования гибридного канала связи на базе лазерной и радио технологий»

Авторы: Вишневский В.М., Гречишкина Н.А., Шаров С.Ю., Семенова О.В.

ЗАО Научно-производственная фирма
«Информационные и сетевые технологии»
ЗАО НПФ «ИНСЕТ»

129626, г.Москва, ул. Староалексеевская, д. 5, офис 215, тел./ф.(495)7205129, (499)5798522

14.10.11 № 31/10

В Отдел регистрации
программ для ЭВМ

Направляем Вам на регистрацию программу для ЭВМ «Пакет программ аналитического и машинного (имитационного) моделирования гибридного канала связи на базе лазерной и радио технологий».

Авторы: Вишневский В.М., Гречишкина Н.А., Шаров С.Ю., Семенова О.В.

Приложение: 1. Листинг Пр ЭВМ
2. Реферат
3. Квитанция об уплате регистрационного сбора.

Генеральный директор
ЗАО НПФ «ИНСЕТ» Вишневский В.М.

Реферат

Название	Пакет программ аналитического и машинного (имитационного) моделирования гибридного канала связи на базе лазерной и радио технологий
Правообладатель	ЗАО Научно-производственная фирма «Информационные и сетевые технологии»
Авторы	Вишневский В.М., Гречишкина Н.А., Шаров С.Ю., Семенова О.В.
Аннотация	Пакет программ предназначен: для проведения численных расчётов характеристик производительности; выбора оптимальных параметров гибридного канала связи, реализованных на базе лазерных и радиотехнологий миллиметрового и сантиметрового диапазонов длин волн. Решает задачу исследования характеристик: лазерного канала, резервированного радиоканалом сантиметрового диапазона волн, находящемся в холодном резерве; лазерного канала, резервированного радиоканалом миллиметрового диапазона волн, находящемся в горячем резерве; высоконагруженной линии передачи мультимедийной информации, включающей параллельно работающие лазерный и миллиметровый радио каналы, зарезервированные радиоканалом сантиметрового диапазона волн, находящемся в холодном резерве. Может быть использована: при проектировании и разработке высокоскоростных и надёжных гибридных каналов передачи мультимедийной информации (голос, данные, видеоинформация) операторского класса; при выполнении раздельных задач, с представлением иерархии классов, определяющих множество исключений с возможностью их расширения.

Приложение 4.

Свидетельство о государственной регистрации программ для ЭВМ

Листинг пакета программ аналитического и машинного (имитационного) моделирования гибридного канала связи

AnalyticMain

```
#
#Файл model\hybrid\analytic\v2\AnalyticMain.java
#Отвечает за вызов аналитических моделей
package model.hybrid.analytic.v2;

import model.hybrid.v3.config.Config;
import model.hybrid.v3.impl.Context;
import model.hybrid.v3.util.Helper;

import java.io.*;


public class AnalyticMain
{
   static final String INPUT_FILE = "input.txt";

   public static void main(String[] args) throws Exception {
      double[] ip = loadInputParams();

      double lmb = ip[0];
      double[] mu = {0, ip[1], ip[2]};
      double[] Q = {0, ip[3], ip[4]};
      double[] p = {0, ip[5], ip[6]};
      double[] gm1 = {0, ip[7], ip[8]};
      double[] gm2 = {0, ip[9], ip[10]};

      double G_precision = ip[11];
      double pi_precision = ip[12];

      int maxIndex = (int)
            Math.round(lmb * 10);

      long startTime = System.currentTimeMillis();

      ScalarCalculator scalc =
            new ScalarCalculator(lmb, mu, Q, p, gm1, gm2);
      scalc.doCalculations(maxIndex);

      MatrixCalculator mcalc =
            new MatrixCalculator(scalc);
      mcalc.doCalculations(maxIndex,
            G_precision, pi_precision);

      System.out.println("Total calculation time: "
```

```
            + (System.currentTimeMillis() - startTime)/1000
            + " sec.");
    }


    public static double[] loadInputParams() throws IOException {
        BufferedReader reader = new BufferedReader(
                new FileReader(INPUT_FILE));

        double[] inputParams
                = new double[13];
        String line;
        int i = 0;

        while (true) {
            line = reader.readLine();

            if (line == null) break;
            if (line.equals("")) continue;

            inputParams[i] = Double.valueOf(
                    line.split("\t")[0]);
            i++;
        }

        return inputParams;
    }

    public static void loadParams() {
        Config conf = Context.getInstance().config;
        double lambda = 1000./conf.PACKET_INTERVAL; // 1/seconds

        int len = (int) (conf.PACKET_LEN); // bits
        double avSendingTimeO = Helper.convertPacketLengthToSendingTime(len, conf.THROUGHPUT_OPTICS); //seconds
        double muO = 1.0/avSendingTimeO; // 1/seconds

        double avSendingTimeR = Helper.convertPacketLengthToSendingTime(len, conf.THROUGHPUT_RADIO); //seconds
        double muR = 1.0/avSendingTimeR; // 1/seconds

        double gamma_1_Good = 1/(conf.GOOD_WEATHER_AV_1*60*60); // 1/seconds
        double gamma_2_Good = 1/(conf.GOOD_WEATHER_AV_2*60*60); // 1/seconds

        double gamma_1_Bad = 1/(conf.BAD_WEATHER_AV_1*60); // 1/seconds
        double gamma_2_Bad = 1/(conf.BAD_WEATHER_AV_2*60); // 1/seconds

        double lmb = lambda;
        double[] mu = {0, muO, muR};
        //double[] Q = {0, ip[3], ip[4]};
        double[] p = {0, conf.GOOD_WEATHER_P, conf.BAD_WEATHER_P};
        double[] gm1 = {0, gamma_1_Good, gamma_1_Bad};
        double[] gm2 = {0, gamma_2_Good, gamma_2_Bad};
    }
}
```

Matrix

```
#
#Файл model\hybrid\analytic\v2\Matrix.java
#Модель матрицы
package model.hybrid.analytic.v2;

public class Matrix {
    public int rows;
    public int cols;
```

```
public double[][] el;

public Matrix(int rows, int cols) {
   this.rows = rows;
   this.cols = cols;
   this.el = new double[rows][cols];
}


public static Matrix getNullMatrix(int rows, int cols) {
   Matrix matrix = new Matrix(rows, cols);

   for (int i=0; i < rows; i++) {
      for (int k=0; k < cols; k++){
         matrix.el[i][k] = 0;
      }
   }

   return matrix;
}

public static Matrix getIdentityMatrix(int size) {
   Matrix matrix = Matrix.getNullMatrix(size, size);

   for (int i=0; i < size; i++) {
      matrix.el[i][i] = 1;
   }

   return matrix;
}


public static double multiplyVectors(double[] v1, double[] v2) {
   double res = 0;

   for (int i=0; i < v1.length; i++) {
      res = res +
            v1[i] * v2[i];
   }

   return res;
}

public static double multiplyVectors(double[] v1, Matrix m2, int col) {
   double res = 0;

   for (int i=0; i < v1.length; i++) {
      res = res +
            v1[i] * m2.el[i][col];
   }

   return res;
}

public static double diffPrecision(Matrix m1, Matrix m2) {
   Matrix diff = m2.
         subtractMatrix(m1);
   double max = 0;
   double ratio;

   for (int i=0; i < m1.rows; i++) {
      for (int k=0; k < m1.cols; k++) {
         if (m2.el[i][k] != 0) {
            ratio = Math.abs(
                  diff.el[i][k] / m2.el[i][k]);
            max = Math.max(ratio, max);
         }
      }
```

```
    }

    return max;
  }


  public Matrix addMatrix(Matrix m2) {
    Matrix resMatrix =
          new Matrix(rows, cols);

    for (byte i=0; i < rows; i++) {
      for (byte k=0; k < cols; k++){
        resMatrix.el[i][k] =
              this.el[i][k] + m2.el[i][k];
      }
    }

    return resMatrix;
  }

  public Matrix subtractMatrix(Matrix m2) {
    Matrix resMatrix =
          new Matrix(rows, cols);

    for (byte i=0; i < rows; i++) {
      for (byte k=0; k < cols; k++){
        resMatrix.el[i][k] =
              this.el[i][k] - m2.el[i][k];
      }
    }

    return resMatrix;
  }

  public Matrix multiplyByIndex(double index) {
    Matrix resMatrix =
          new Matrix(rows, cols);

    for (byte i=0; i < rows; i++) {
      for (byte k=0; k < cols; k++){
        resMatrix.el[i][k] =
              this.el[i][k] * index;
      }
    }

    return resMatrix;
  }

  public Matrix multiplyByMatrix(Matrix m2) {
    Matrix resMatrix =
          Matrix.getNullMatrix(rows, m2.cols);

    for (byte row=0; row < rows; row++) {
      for (byte col=0; col < m2.cols; col++) {
        resMatrix.el[row][col] = multiplyVectors(
              this.el[row], m2, col);
      }
    }

    return resMatrix;
  }

  public Matrix powerMatrix(int power) {
    Matrix resMatrix =
          Matrix.getIdentityMatrix(rows);

    for (int i=1; i <= power; i++) {
      resMatrix = resMatrix.
```

```
            multiplyByMatrix(this);
    }

    return resMatrix;
}

public Matrix getInverseMatrix() throws Exception {
    Matrix A = this.
            multiplyByIndex(1);
    Matrix Inv = Matrix.
            getIdentityMatrix(rows);
    double diagEl, index;

    for (byte i=0; i < rows; i++) {
        swapMatrix(A, Inv, i);
        diagEl = A.el[i][i];

        if (diagEl == 0)
            throw new Exception("Inverting matrix error");

        for (byte k=0; k < cols; k++) {
            A.el[i][k] = A.el[i][k] / diagEl;
            Inv.el[i][k] = Inv.el[i][k] / diagEl;
        }

        for (byte k=0; k < rows; k++) {
            if (k == i) continue;
            index = A.el[k][i];
            A.subtractLine(i, k, index);
            Inv.subtractLine(i, k, index);
        }
    }

    return Inv;
}


private void swapRows(int i, int j) {
    double x;
    for (int k=0; k < cols; k++) {
        x = this.el[i][k];
        this.el[i][k] = this.el[j][k];
        this.el[j][k] = x;
    }
}

private void swapCols(int i, int j) {
    double x;
    for (int k=0; k < rows; k++) {
        x = this.el[k][i];
        this.el[k][i] = this.el[k][j];
        this.el[k][j] = x;
    }
}

private void swapMatrix(Matrix A, Matrix Inv, int i) {
    for (int i2 = i; i2 < rows; i2++) {
        for (int j2 = i; j2 < cols; j2++) {
            if (A.el[i2][j2] != 0) {
                A.swapRows(i, i2);
                A.swapCols(i, j2);
                Inv.swapRows(i, i2);
                Inv.swapCols(i, j2);
                return;
            }
        }
    }
}
```

```
private void subtractLine(int i, int k, double index) {
   for (int j=0; j < cols; j++) {
      this.el[k][j] = this.el[k][j] -
            this.el[i][j] * index;
   }
}


public void printStochastics(int precision) {
   double sum;

   for (int i=0; i < rows; i++) {
      sum = 0;
      for (int k=0; k < cols; k++){
         sum = sum + this.el[i][k];
      }

      if (precision > 0) {
         sum = Math.round(sum *
               Math.pow(10, precision));
         sum = sum /
               Math.pow(10, precision);
      }

      System.out.println("Line " + (i+1)
            + ", elements sum = " + sum);
   }
}

public double[] getStochastics(int precision) {
   double[] stoch
         = new double[rows];
   double sum;

   for (int i=0; i < rows; i++) {
      sum = 0;
      for (int k=0; k < cols; k++){
         sum = sum + this.el[i][k];
      }

      if (precision > 0) {
         sum = Math.round(sum *
               Math.pow(10, precision));
         sum = sum / Math.pow(10, precision);
      }

      stoch[i] = sum;
   }

   return stoch;
}

public boolean equals(Matrix matrix) {
   for (int i=0; i < rows; i++) {
      for (int k=0; k < cols; k++){
         if (this.el[i][k] != matrix.el[i][k])
            return false;
      }
   }

   return true;
}

public String toString() {
   String res = "";

   for (int i=0; i < rows; i++) {
```

```
            for (int k=0; k < cols; k++){
                res = res + "\t"
                        + this.el[i][k];
            }

            res = res + "\n";
        }

        return res;
    }
}
```

MatrixCalculator

```
#
#Файл model\hybrid\analytic\v2\MatrixCalculator.java
#Математические операции с матрицами
package model.hybrid.analytic.v2;


public class MatrixCalculator {
    private ScalarCalculator scalc;

    public Matrix[][] P;
    public Matrix G;
    public Matrix[][] Y;

    public Matrix[] F;
    public Matrix[] pi;


    public MatrixCalculator(ScalarCalculator scalc) {
        this.scalc = scalc;
    }

    public void doCalculations(int maxIndex, double G_precision, double pi_precision) throws Exception {
        System.out.println(
                "Calculating matrices P[i,l]...");
        P = calc_P(maxIndex);

        System.out.println("\n" +
                "Stochastic property of SUM(P[i,l]):");
        Psum(1,0).printStochastics(0);

        System.out.println("\n" +
                "Calculating matrix G...");
        G = calc_G(G_precision);

        System.out.println("\n" +
                "Stochastic property of G:");
        G.printStochastics(0);

        System.out.println("\n" +
                "Calculating matrices Y[i,l]...");
        Y = calc_Y(maxIndex);

        System.out.println("\n" +
                "Calculating vectors Pi[i]...");
        Matrix pi0 = calc_pi0();
        pi = calc_piVectors(
                pi0, maxIndex, pi_precision);

        System.out.println("\n" +
                "Calculating L value...");
```

```
    System.out.println("L = " + calc_L());

    System.out.println();
  }


  public Matrix[][] calc_P(int maxIndex) {
    Matrix[][] P_array
        = new Matrix[2][maxIndex + 1];

    for (int i = 0; i <= 1; i++) {
      for (int l=0; l < maxIndex + i; l++) {
        P_array[i][l] = scalc.Pmatrix(i,l);

        if (P_array[i][l].equals(
            Matrix.getNullMatrix(7,7))) {
          System.out.println(
              "P[" + i + ",l] = O when l>" + (l-1));
          break;
        }
      }
    }

    return P_array;
  }

  public Matrix calc_G(double precision) throws Exception {
    Matrix prevG =
        Matrix.getIdentityMatrix(7);
    Matrix nextG;

    for (int k=1; k <= Integer.MAX_VALUE; k++) {
      nextG = Matrix.getIdentityMatrix(7);
      nextG = nextG.subtractMatrix(
          PGsum(1, 1, prevG, 0));
      nextG = nextG.getInverseMatrix().
          multiplyByMatrix(P[1][0]);

      if (Matrix.diffPrecision(
          prevG, nextG) < precision) {
        System.out.println("\nMax k = " + k);
        return nextG;
      }

      if (k/10 == (double)k/10)
        System.out.print(".");

      prevG = nextG;
    }

    return null;
  }

  public Matrix[][] calc_Y(int maxIndex) throws Exception {
    Matrix[][] Y_array
        = new Matrix[2][maxIndex + 1];
    boolean searchNotNull;

    for (int i = 0; i <= 1; i++) {
      searchNotNull = true;
      for (int l = maxIndex; l > i; l--) {
        if (P[i][l] == null)
          continue;

        if (searchNotNull) {
          Y_array[i][l] = P[i][l];
          System.out.println(
              "Y[" + i + ",l] = O when l>" + (l-1));
```

```
            searchNotNull = false;
          }

          Y_array[i][l-1] = Y_array[i][l].
              multiplyByMatrix(G).
              addMatrix(P[i][l-1]);
        }
      }

      return Y_array;
    }


    public Matrix calc_pi0() throws Exception {
      Matrix pi0 = new Matrix(1,7);
      double[] stochFsum
          = Fsum().getStochastics(0);

      Matrix X = Matrix.getIdentityMatrix(7).
          subtractMatrix(Y[0][0]);

      for (byte k=0; k < 7; k++) {
        X.el[k][0] = stochFsum[k];
      }

      X = X.getInverseMatrix();
      pi0.el[0] = X.el[0];

      System.out.print("pi[0] =" + pi0);
      return pi0;
    }

    public Matrix[] calc_piVectors(Matrix pi0, int maxIndex, double precision) throws Exception {
      Matrix[] pi =
          new Matrix[maxIndex + 1];
      pi[0] = pi0;

      double curElSum;
      double elSum =
          pi0.getStochastics(0)[0];

      Matrix factor = Matrix.getIdentityMatrix(7).
          subtractMatrix(Y[1][1]).
          getInverseMatrix();

      for (int i=1; i <= maxIndex; i++) {
        pi[i] = pi0.
            multiplyByMatrix(
                Fmatrix(i, factor));

        curElSum = pi[i].getStochastics(0)[0];
        elSum = elSum + curElSum;

        if (i/100 == (double)i/100) {
          System.out.print("\n" +
              "pi[" + i + "] =" + pi[i]);
          System.out.println(
              "SUM(pi[i])*1 = " + elSum);
        }

        if (curElSum < 0) {
          System.out.println("\nCalculation error: " +
              "pi[" + i + "]*1 < 0 !!!");
          System.exit(1);
        }

        if (curElSum < precision) {
          System.out.println("\nMax i = " + i);
```

```
            System.out.println("SUM(pi[i])*1 = " + elSum);
            return pi;
        }
    }

    return pi;
}

public double calc_L() {
    double L = 0;
    double elSum;

    for (int i=1; i < pi.length; i++) {
        if (pi[i] == null)
            break;

        elSum = pi[i].getStochastics(0)[0];
        L = L + (i * elSum);
    }

    return L;
}


public Matrix Psum(int i, int init_l) {
    Matrix Psum = Matrix.getNullMatrix(7,7);

    for (int l = init_l; l < P[i].length; l++) {
        if (P[i][l] != null)
            Psum = Psum.addMatrix(P[i][l]);
        else break;
    }

    return Psum;
}

public Matrix PGsum(int i, int init_l, Matrix G, int init_Gpow) {
    Matrix PGsum = Matrix.getNullMatrix(7,7);
    Matrix powG = G.powerMatrix(init_Gpow);

    for (int l = init_l; l < P[i].length; l++) {
        if (l > init_l)
            powG = powG.multiplyByMatrix(G);

        if (P[i][l] != null)
            PGsum = PGsum.addMatrix(
                  P[i][l].multiplyByMatrix(powG));
        else break;
    }

    return PGsum;
}

public Matrix Ysum(int i) {
    Matrix Ysum = Matrix.getNullMatrix(7,7);

    for (int l=1; l < Y[i].length; l++) {
        if (Y[i][l] != null)
            Ysum = Ysum.addMatrix(Y[i][l]);
        else break;
    }

    return Ysum;
}

public Matrix Fsum() throws Exception {
    Matrix X, Fsum;
```

```
      X = Matrix.getIdentityMatrix(7).
            subtractMatrix(Ysum(1)).
            getInverseMatrix();
      Fsum = Ysum(0).
            multiplyByMatrix(X).
            addMatrix(Matrix.getIdentityMatrix(7));

      return Fsum;
   }

   public Matrix Fmatrix(int l, Matrix factor) throws Exception {
      if (F == null) {
         F = new Matrix[100000];
         F[0] = Matrix.getIdentityMatrix(7);
      }

      F[l] = Matrix.getNullMatrix(7,7);
      Matrix curY;

      for (int i=0; i <= l-1; i++) {
         if ((i == 0) &&
               (Y[0].length > l))
            curY = Y[0][l];
         else if ((i >= 1) &&
               (Y[1].length > l-i+1))
            curY = Y[1][l-i+1];
         else
            curY = null;

         if (curY != null)
            F[l] = F[l].addMatrix(
                  F[i].multiplyByMatrix(curY));
      }

      F[l] = F[l].
            multiplyByMatrix(factor);

      return F[l];
   }
}
```

ScalarCalculator

```
#
#Файл model\hybrid\analytic\v2\ScalarCalculator.java
#Математические операции в векторно-матричной форме
package model.hybrid.analytic.v2;

import java.math.BigDecimal;
import java.math.MathContext;

public class ScalarCalculator {
   private double lmb;
   private double[] mu;
   private double[] Q;
   private double[] p;
   private double[] gm1;
   private double[] gm2;

   public double alfa;
   public double[] s
         = new double[3];

   public BigDecimal[] factorial;
```

```
public double[][] f;
public double[][] g;
public double[][] h;
public double[] r;
public double[] m;

public double[][] v;
public double[][] fx;
public double[][] gx;


public ScalarCalculator(double lmb, double[] mu, double[] Q, double[] p, double[] gm1, double[] gm2) {
   this.lmb = lmb;
   this.mu = mu;
   this.Q = Q;
   this.p = p;
   this.gm1 = gm1;
   this.gm2 = gm2;

   alfa = p[1] * gm1[1] / (gm1[1] + Q[1])
         + (1 - p[1]) * gm2[1] / (gm2[1] + Q[1]);

   s[1] = p[1] * lmb / (lmb + gm1[1])
         + (1 - p[1]) * lmb / (lmb + gm2[1]);
   s[2] = p[2] * lmb / (lmb + gm1[2])
         + (1 - p[2]) * lmb / (lmb + gm2[2]);
}

public void doCalculations(int maxIndex) throws Exception {
   System.out.println(
         "Calculating f[i], g[i], h[i], r[i], m[i]...");

   factorial = calcFactorials(maxIndex);

   f = calc_f(maxIndex);
   g = calc_g(maxIndex);
   r = calc_r(maxIndex);
   m = calc_m(maxIndex);

   h = new double[3][];
   h[1] = calc_h1(maxIndex);
   h[2] = calc_h2(maxIndex);

   System.out.println();
   System.out.println(
         "Calculating v[i], f^[i], g^[i]...");

   v = calc_v(maxIndex);
   fx = calc_fx(maxIndex);
   gx = calc_gx(maxIndex);

   System.out.println();
}


public double[][] calc_f(int maxIndex) throws Exception {
   double[][] array
         = new double[3][maxIndex + 1];

   for (byte k=1; k <= 2; k++) {
      for (int i=0; i <= maxIndex; i++) {
         array[k][i] = checkVal(p[k] * mu[k] / (gm1[k] + mu[k] + lmb)
               * Math.pow(lmb / (gm1[k] + mu[k] + lmb), i)
               + (1 - p[k]) * mu[k] / (gm2[k] + mu[k] + lmb)
               * Math.pow(lmb / (gm2[k] + mu[k] + lmb), i), i);

         if (array[k][i] == 0) {
```

```
            System.out.println("f(" + k + ")=0 when i>" + (i-1));
            break;
         }
      }
   }

   return array;
}

public double[][] calc_g(int maxIndex) throws Exception {
   double[][] array
         = new double[3][maxIndex + 1];

   for (byte k=1; k <= 2; k++) {
      for (int i=0; i <= maxIndex; i++) {
         array[k][i] = checkVal(p[k] * gm1[k] / (gm1[k] + mu[k] + lmb)
               * Math.pow(lmb / (gm1[k] + mu[k] + lmb), i)
               + (1 - p[k]) * gm2[k] / (gm2[k] + mu[k] + lmb)
               * Math.pow(lmb / (gm2[k] + mu[k] + lmb), i), i);

         if (array[k][i] == 0) {
            System.out.println("g(" + k + ")=0 when i>" + (i-1));
            break;
         }
      }
   }

   return array;
}

public double[] calc_r(int maxIndex) throws Exception {
   double[] array
         = new double[maxIndex + 1];

   for (int i=0; i <= maxIndex; i++) {
      array[i] = checkVal(mu[2] / (Q[1] + mu[2] + lmb)
            * Math.pow(lmb / (Q[1] + mu[2] + lmb), i), i);

      if (array[i] == 0) {
         System.out.println("r=0 when i>" + (i-1));
         break;
      }
   }

   return array;
}

public double[] calc_m(int maxIndex) throws Exception {
   double[] array
         = new double[maxIndex + 2];
   array[0] = 0;

   for (int i=0; i <= maxIndex; i++) {
      array[i+1] = checkVal(Q[1] / (Q[1] + mu[2] + lmb)
            * Math.pow(lmb / (Q[1] + mu[2] + lmb), i), i);

      if (array[i+1] == 0) {
         System.out.println("m=0 when i>" + (i-1));
         break;
      }
   }

   return array;
}


public double[] calc_h1(int maxIndex) throws Exception {
   double[] array
```

```
        = new double[maxIndex + 1];

    BigDecimal res1, res2;
    MathContext mc = new MathContext(20);

    BigDecimal argLmb = BigDecimal.valueOf(lmb);
    BigDecimal argQ = BigDecimal.valueOf(Q[2]);
    BigDecimal arg1 = BigDecimal.valueOf(gm1[2] + lmb);
    BigDecimal arg2 = BigDecimal.valueOf(gm2[2] + lmb);

    BigDecimal exp = new BigDecimal(
        "2.71828182845904523536");
    BigDecimal exp1 = exp.pow((int)
        Math.round((gm1[2] + lmb) * Q[2]), mc);
    BigDecimal exp2 = exp.pow((int)
        Math.round((gm2[2] + lmb) * Q[2]), mc);

    BigDecimal sum1 = BigDecimal.ONE;
    BigDecimal sum2 = BigDecimal.ONE;

    for (int i=0; i <= maxIndex; i++) {
      if (i > 0) {
        sum1 = sum1.divide(arg1, mc).
            add(argQ.pow(i, mc).
                divide(factorial[i], mc), mc);
        sum2 = sum2.divide(arg2, mc).
            add(argQ.pow(i, mc).
                divide(factorial[i], mc), mc);
      }

      res1 = BigDecimal.ONE.divide(
          arg1.pow(i, mc), mc);
      res1 = res1.subtract(
          sum1.divide(exp1, mc), mc);
      res1 = res1.multiply(
          argLmb.pow(i, mc), mc);

      res2 = BigDecimal.ONE.divide(
          arg2.pow(i, mc), mc);
      res2 = res2.subtract(
          sum2.divide(exp2, mc), mc);
      res2 = res2.multiply(
          argLmb.pow(i, mc), mc);

      array[i] = checkVal(res1.doubleValue() *
          gm1[2] * p[2] / (gm1[2] + lmb)
          + res2.doubleValue() *
          gm2[2] * (1 - p[2]) / (gm2[2] + lmb), i);

      if (array[i] <= 0) {
        System.out.println("h(1)=0 when i>" + (i-1));
        array[i] = 0;
        break;
      }
    }

    return array;
  }

  public double[] calc_h2(int maxIndex) throws Exception {
    double[] array
        = new double[maxIndex + 1];

    BigDecimal res;
    MathContext mc = new MathContext(20);
    boolean nullCheck = false;

    BigDecimal exp = new BigDecimal(
```

```
        "2.71828182845904523536");
    exp = exp.pow((int)
        Math.round(lmb * Q[2]), mc);

    BigDecimal koef = BigDecimal.valueOf(
        p[2] * Math.exp(-gm1[2]*Q[2])
        + (1 - p[2]) * Math.exp(-gm2[2]*Q[2]));
    koef = koef.divide(exp, mc);

    BigDecimal arg = BigDecimal.
        valueOf(lmb * Q[2]);

    for (int i=0; i <= maxIndex; i++) {
      res = arg.pow(i, mc);
      res = res.divide(factorial[i], mc);
      res = res.multiply(koef, mc);

      array[i] = checkVal(
          res.doubleValue(), i);

      if ((array[i] > 0) && (!nullCheck)) {
        System.out.println("h(2)=0 when i<" + i);
        nullCheck = true;
      }

      if ((array[i] == 0) && (nullCheck)) {
        System.out.println("h(2)=0 when i>" + (i-1));
        break;
      }
    }

    return array;
}


public double[][] calc_v(int maxIndex) throws Exception {
    double[][] array
        = new double[3][maxIndex + 1];

    for (byte k=1; k <= 2; k++) {
      for (int i=0; i <= maxIndex; i++) {
        if (i > 0)
          array[k][i] = checkVal(s[k] * g[k][i-1], i);
        else
          array[k][i] = checkVal(1 - s[k], i);

        if (array[k][i] == 0) {
          System.out.println("v(" + k + ")=0 when i>" + (i-1));
          break;
        }
      }
    }

    return array;
}

public double[][] calc_fx(int maxIndex) throws Exception {
    double[][] array
        = new double[3][maxIndex + 1];
    double sum;

    for (byte k=1; k <= 2; k++) {
      boolean nullCheck = false;

      for (int i=0; i <= maxIndex; i++) {
        sum = 0;
        for (int l=0; l <= i; l++) {
          sum = sum +
```

```
                h[k][l] * f[k][i-l];
        }

        array[k][i] = checkVal(sum, i);

        if ((array[k][i] > 0) && (!nullCheck)) {
          if (i != 0)
            System.out.println("f^(" + k + ")=0 when i<" + i);
          nullCheck = true;
        }

        if ((array[k][i] == 0) && (nullCheck)) {
          System.out.println("f^(" + k + ")=0 when i>" + (i-1));
          break;
        }
      }
    }

    return array;
}

public double[][] calc_gx(int maxIndex) throws Exception {
    double[][] array
        = new double[3][maxIndex + 1];
    double sum;

    for (byte k=1; k <= 2; k++) {
      boolean nullCheck = false;

      for (int i=0; i <= maxIndex; i++) {
        sum = 0;
        for (int l=0; l <= i; l++) {
          sum = sum +
                h[k][l] * g[k][i-l];
        }

        array[k][i] = checkVal(sum, i);

        if ((array[k][i] > 0) && (!nullCheck)) {
          if (i != 0)
            System.out.println("g^(" + k + ")=0 when i<" + i);
          nullCheck = true;
        }

        if ((array[k][i] == 0) && (nullCheck)) {
          System.out.println("g^(" + k + ")=0 when i>" + (i-1));
          break;
        }
      }
    }

    return array;
}


public static BigDecimal[] calcFactorials(int maxIndex) {
    BigDecimal[] array
        = new BigDecimal[maxIndex + 1];
    array[0] = BigDecimal.ONE;

    for (int i=1; i <= maxIndex; i++) {
      array[i] = array[i-1].multiply(
          BigDecimal.valueOf(i),
          new MathContext(20));
    }

    return array;
}
```

```
public static byte I(boolean condition) {
   if (condition)
      return 1;
   else
      return 0;
}


public Matrix Pmatrix(int i, int l) {
   if (i == 0)
      return PmatrixCase1(l);
   else if (l == i-1)
      return PmatrixCase2();
   else if (l >= i)
      return PmatrixCase3(l-i);
   else
      return null;
}

private Matrix PmatrixCase1(int i) {
   Matrix matrix = Matrix.getNullMatrix(7,7);

   matrix.el[0][0] = s[1] * f[1][i];
   matrix.el[0][1] = v[1][i];
   matrix.el[1][0] = (s[1] * f[1][i] - f[1][i+1]) * h[1][0] + fx[1][i+1];
   matrix.el[1][1] = gx[1][i] + h[1][0] * (v[1][i] - g[1][i]);
   matrix.el[1][2] = (s[2] * f[2][i] - f[2][i+1]) * h[2][0] + fx[2][i+1];
   matrix.el[1][3] = gx[2][i] + h[2][0] * (v[2][i] - g[2][i]);
   matrix.el[2][2] = s[2] * f[2][i];
   matrix.el[2][3] = v[2][i];

   matrix.el[3][4] = lmb / (lmb + Q[1]) * r[i];
   matrix.el[3][5] = lmb * alfa / (lmb + Q[1]) * m[i-1+1]
         + Q[1] * alfa / (lmb + Q[1]) * I(i==0);
   matrix.el[3][6] = lmb * (1-alfa) / (lmb + Q[1]) * m[i-1+1]
         + Q[1] * (1-alfa) / (lmb + Q[1]) * I(i==0);
   matrix.el[4][4] = matrix.el[3][4];
   matrix.el[4][5] = matrix.el[3][5];
   matrix.el[4][6] = matrix.el[3][6];

   matrix.el[5][0] = s[1] * f[1][i];
   matrix.el[5][1] = v[1][i];
   matrix.el[6][2] = s[1] * f[2][i];
   matrix.el[6][3] = v[2][i];

   return matrix;
}

private Matrix PmatrixCase2() {
   Matrix matrix = Matrix.getNullMatrix(7,7);

   matrix.el[0][0] = f[1][0];
   matrix.el[1][0] = fx[1][0];
   matrix.el[1][2] = fx[2][0];
   matrix.el[2][2] = f[2][0];
   matrix.el[3][4] = r[0];
   matrix.el[4][4] = r[0];
   matrix.el[5][0] = f[1][0];
   matrix.el[6][2] = f[2][0];

   return matrix;
}

private Matrix PmatrixCase3(int i) {
   Matrix matrix = Matrix.getNullMatrix(7,7);

   matrix.el[0][0] = f[1][i+1];
```

```
        matrix.el[0][1] = g[1][i];
        matrix.el[1][0] = fx[1][i+1];
        matrix.el[1][1] = gx[1][i];
        matrix.el[1][2] = fx[2][i+1];
        matrix.el[1][3] = gx[1][i];
        matrix.el[2][2] = f[2][i+1];
        matrix.el[2][3] = g[2][i];
        matrix.el[3][4] = r[i+1];
        matrix.el[3][5] = alfa * m[i+1];
        matrix.el[3][6] = (1 - alfa) * m[i+1];
        matrix.el[4][4] = r[i+1];
        matrix.el[4][5] = alfa * m[i+1];
        matrix.el[4][6] = (1 - alfa) * m[i+1];
        matrix.el[5][0] = f[1][i+1];
        matrix.el[5][1] = g[1][i];
        matrix.el[6][2] = f[2][i+1];
        matrix.el[6][3] = g[2][i];

        return matrix;
    }


    private static double checkVal(double val, int i) throws Exception {
        if (Double.isNaN(val)
                || Double.isInfinite(val))
            throw new Exception(
                    "i=" + i + ", val=" + val);
        else
            return val;
    }
}
```

Channel

```
#
#Файл model\hybrid\v3\channels\Channel.java
#Абстрактная модель канала передачи данных
package model.hybrid.v3.channels;

import java.util.List;

import model.hybrid.v3.impl.Packet;
import model.hybrid.v3.impl.Context;
import model.hybrid.v3.util.Statistics;
import model.hybrid.v3.util.Helper;

public abstract class Channel
{
        public enum CHANNEL_STATE_TYPE {ACTIVE, UNAVAILABLE, SWITCHING_ON, SWITCHING_OFF, UNKNOWN}

    protected final double throughput;
    protected final String channelName;
    protected int channelNumber; //to be set by implementation of abstract class

    protected List<Packet> queue;
    protected Packet currentPacket;

    protected CHANNEL_STATE_TYPE state;
        //Time when current packet will be delivered or -1 if no packet is being sent
        protected double packetDeliveryTime;

    protected Context ctx;
    protected Statistics statistics;
    public Channel(double throughput, String channelName, List<Packet> commonQueue)
```

```
{
   this.throughput = throughput;
   this.channelName = channelName;
   this. queue = commonQueue;
   init();
}

public void init()
{
   state = CHANNEL_STATE_TYPE.ACTIVE;
   ctx = Context.getInstance();
   statistics = ctx.stat;
   currentPacket = null;
   packetDeliveryTime = -1;

   statistics.startSending(channelNumber);
}

public abstract Packet weatherChange();

            public void execute() {
   switch (state) {
      case ACTIVE:
         txEnd();
         break;
      case SWITCHING_ON:
         state = CHANNEL_STATE_TYPE.ACTIVE;
         packetDeliveryTime = -1;
         statistics.startSending(channelNumber);
         break;
      case SWITCHING_OFF:
         state = CHANNEL_STATE_TYPE.UNAVAILABLE;
         packetDeliveryTime = -1;
         break;
      default:
         throw new IllegalStateException("Illegal state of the "+channelName+" channel");
   }
            }

            public void packetCome()
{
                        if(state != CHANNEL_STATE_TYPE.ACTIVE)
      throw new IllegalStateException("Can't add packet to the queue since state of "+channelName+" channel is not Active");

   if(currentPacket != null)
      throw new IllegalStateException("Channel "+channelName+" is already sending a packet");

   currentPacket = queue.remove(0);
   statistics.startPacketSending(currentPacket, channelNumber);
                        txStart();
            }

            public void txEnd() {
                        //Statistics
   statistics.packetSent(channelNumber);
   //statistics.queueLengthAtServeEnd(queue.size()==0 ? 0 : queue.size()-1);
   statistics.queueLengthAtServeEnd(queue.size());

   //Packet is sent
   currentPacket = null;

                        if(queue.size() == 0) {
                                    packetDeliveryTime = -1;
                        }
                        else
   {
      packetCome();
   }
```

```
		}

		public void txStart()
	{
		packetDeliveryTime = ctx.scheduler.getCurrentTime()+
			Helper.convertPacketLengthToSendingTime(currentPacket.len, throughput);
		}

		public double getTime() {
			return packetDeliveryTime;
		}

	public boolean isActive()
	{
		return state == CHANNEL_STATE_TYPE.ACTIVE;
	}

	public boolean isChannelFree() {
			return currentPacket == null;
		}

	public void experimentFinished()
	{
		if( isActive() )
			statistics.stopSending(channelNumber);
	}
}
```

CompositeChannel

```
#
#Файл model\hybrid\v3\channels\CompositeChannel.java
#Модель «сложного» канала, объединяющего несколько каналов
package model.hybrid.v3.channels;

import java.util.List;
import java.util.ArrayList;

import model.hybrid.v3.impl.Context;
import model.hybrid.v3.impl.Packet;
import model.hybrid.v3.states.OpticsRadio;
import model.hybrid.v3.states.Radio;
import model.hybrid.v3.states.State;
import model.hybrid.v3.util.Helper;

public class CompositeChannel
{
	private Channel optic;
	private Channel radio;
	private List<Channel> channels;

	protected List<Packet> queue;

	protected Context ctx;

	public CompositeChannel()
	{
		init();
	}

	public void init()
	{
		queue = new ArrayList<Packet>(1000);
```

```
    ctx = Context.getInstance();
    optic = new OpticChannel(ctx.config.THROUGHPUT_OPTICS, queue);
    radio = new RadioChannel(ctx.config.THROUGHPUT_RADIO, queue);
    channels = new ArrayList<Channel>(2);
    channels.add(optic);
    channels.add(radio);
}

public void weatherChange() {
                Packet packet = optic.weatherChange();
    if(packet != null) packetCome(packet);

    //if(Config.getInstance().USE_STATES_STATISTICS) {
                //      Statistics.states.add(new StateStat(Scheduler.time, state));
                //}
        }

        public void execute() {
                if(optic.getTime() == ctx.scheduler.getCurrentTime()) {
                        optic.execute();
                }
    if(radio.getTime() == ctx.scheduler.getCurrentTime()) {
                        radio.execute();
                }

                //if(Config.getInstance().USE_STATES_STATISTICS) {
                //      Statistics.states.add(new StateStat(Scheduler.time, state));
                //}
        }

public void testQueue()
{
    if(queue.size() > 100000) {
                        StringBuilder out = new StringBuilder();
                        out.append("Error occurred!\nBuffer overflow:");
      double curTime = ctx.scheduler.getCurrentTime();
      out.append("time        =       ").append(Helper.convertTimeToStr(curTime));
                        out.append("\n");
      out.append("queue size : ").append(queue.size());
      System.out.println("out = " + out);
                        throw new NullPointerException(out.toString());
                }
}

        public void packetCome(Packet packet) {
                queue.add(packet);
    testQueue();
    for(Channel ch: channels)
    {
      if(ch.isActive() && ch.isChannelFree() )
      {
        ch.packetCome();
        break;
      }
    }
        }

public void experimentFinished()
{
    optic.experimentFinished();
    radio.experimentFinished();
}

        public double getTime() {
                if( optic.getTime() < 0 )
      return radio.getTime();
    if( radio.getTime() < 0 )
      return optic.getTime();
```

```
        return Math.min(optic.getTime(), radio.getTime());
                }

                public State getState() {
                        if( optic.isActive() && radio.isActive() )
          return OpticsRadio.getInstance();
        if( (!optic.isActive()) && radio.isActive() )
          return Radio.getInstance();
        throw new IllegalStateException("Unknown state");
                }

    public int getQueueSize()
    {
        return queue.size();
    }
}
```

OpticChannel

```
#
#Файл model\hybrid\v3\channels\OpticChannel.java
#Модель отпического канала
package model.hybrid.v3.channels;

import model.hybrid.v3.impl.Packet;
import model.hybrid.v3.config.Config;

import java.util.List;

/**
 * User: Sharov
 * Date: 06.02.2010
 * Time: 13:02:10
 * NetCracker
 */
public class OpticChannel extends Channel
{
    public OpticChannel(double throughput, List<Packet> queue)
    {
        super(throughput, "Optic", queue);
        if( ctx.config.TURN_OFF_OPTICS ) state = CHANNEL_STATE_TYPE.UNAVAILABLE;
    }

    @Override
    public void init()
    {
        channelNumber = Config.OPTIC_CHANNEL_NUM;
        super.init();
    }

    public Packet weatherChange() {
        Packet packetToBeDelivered = currentPacket;
        switch (state) {
            case ACTIVE:
                //TODO: change to SWITCHING_OFF if not needed
                //state = CHANNEL_STATE_TYPE.UNAVAILABLE;
                //packetDeliveryTime = -1;
                //currentPacket = null;
                //TODO: change to SWITCHING_OFF if needed
                state = CHANNEL_STATE_TYPE.SWITCHING_OFF;
                packetDeliveryTime = ctx.scheduler.getCurrentTime()+ctx.config.SWITCH_OR_TIME;
                currentPacket = null;
                statistics.stopSending(channelNumber);
                break;
```

```
            case SWITCHING_ON:
                state = CHANNEL_STATE_TYPE.UNAVAILABLE;
                packetDeliveryTime = -1;
                break;
            case SWITCHING_OFF: //state is not used yet
                state = CHANNEL_STATE_TYPE.ACTIVE;
                packetDeliveryTime = -1;
                statistics.startSending(channelNumber);
                break;
            case UNAVAILABLE:
                state = CHANNEL_STATE_TYPE.SWITCHING_ON;
                packetDeliveryTime = ctx.scheduler.getCurrentTime()+ ctx.config.SWITCH_RO_TIME;
                break;
            case UNKNOWN:
                System.out.println("Unknown state of the "+channelName+" channel");
            default:
                throw new IllegalStateException("Unknown state of the "+channelName+" channel");
        }
        return packetToBeDelivered;
                }
}
```

RadioChannel

```
#
#Файл model\hybrid\v3\channels\RadioChannel.java
#Модель радоиканала
package model.hybrid.v3.channels;

import model.hybrid.v3.impl.Packet;
import model.hybrid.v3.config.Config;

import java.util.List;

/**
 * User: Sharov
 * Date: 06.02.2010
 * Time: 13:14:30
 * NetCracker
 */
public class RadioChannel extends Channel
{
    public RadioChannel(double throughput, List<Packet> queue)
    {
        super(throughput, "Radio", queue);
        if( ctx.config.TURN_OFF_RADIO ) state = CHANNEL_STATE_TYPE.UNAVAILABLE;
    }

    @Override
    public void init()
    {
        channelNumber = Config.RADIO_CHANNEL_NUM;
        super.init();
    }

    public Packet weatherChange() {
        //do nothing since weather does not affect radio in this model
        return null;
                }
}
```

AbstractIterativeExperiment

```
#
#Файл model\hybrid\v3\experiment\AbstractIterativeExperiment.java
#Модель абстрактного итеративного эксперимента
package model.hybrid.v3.experiment;

import model.hybrid.v3.impl.Context;
import model.hybrid.v3.impl.Scheduler;
import model.hybrid.v3.util.Helper;

/**
 * User: Sharov
 * Date: 28.11.2010
 * Time: 15:44:39
 */
public abstract class AbstractIterativeExperiment
{
   protected Context ctx;
   protected SuperExperiment superExperiment;

   protected AbstractIterativeExperiment() {
      ctx = Context.getInstance();
      init();
   }

   protected AbstractIterativeExperiment(SuperExperiment superExperiment) {
      this();
      this.superExperiment = superExperiment;
   }

   public void execute()
   {
      final double EXP_LENGTH = ctx.config.EXPERIMENT_LEN*Scheduler.ONE_YEAR;
      System.out.println("Experiment: "+getExperimentName());
      System.out.println("Length = "+EXP_LENGTH+" sec. ("+ Helper.convertTimeToStr(EXP_LENGTH)+")");

      while(hasNext())
      {
         nextIteration();

         runIteration();

         storeIterationResults();
      }
      finalizeExperiment();
   }

   protected String getCSVHeader()
   {
      String result[] = ctx.stat.resultsToCSVString();
      return ( superExperiment == null ? "": superExperiment.getCSVHeaderPrefix() )+
            "\""+getVarName()+"\""+ctx.config.CSV_SEPARATOR+result[0];
   }

   protected String getCSVRecord()
   {
      String result[] = ctx.stat.resultsToCSVString();
      return ( superExperiment == null ? "": superExperiment.getCSVRecordPrefix() )+
            getCurrentVarValue()+ctx.config.CSV_SEPARATOR+result[1];
   }

   protected String getIterationInfo() {
      return ( superExperiment == null ? "": superExperiment.getIterationInfoPrefix() )+
            String.format(getVarName()+" %1$5s; av_len=%2$12g; av_wait=%3$12g; av_thrpt=%4$12g",
            getCurrentVarValue(), ctx.stat.getAverageQueueLen(), ctx.stat.getAveragePacketWaitingTime(), ctx.stat.getAverageThroughput());
   }
```

```
    protected void storeIterationResults()
    {
    }

    protected void finalizeExperiment()
    {
    }

    protected abstract void init();

    protected abstract boolean hasNext();

    protected abstract void nextIteration();

    protected abstract void runIteration();

    protected abstract String getVarName();

    protected abstract String getCurrentVarValue();

    protected abstract String getExperimentName();
}
```

MultipleRunExperiment

```
#
#Файл model\hybrid\v3\experiment\MultipleRunExperiment.java
#Модель итеративного эксперимента
package model.hybrid.v3.experiment;

import model.hybrid.v3.impl.Scheduler;
import model.hybrid.v3.util.Helper;
import model.hybrid.v3.util.Statistics;

import java.util.LinkedHashMap;
import java.util.Map;

/**
 * User: Sharov
 * Date: 28.11.2010
 * Time: 16:00:14
 */
public class MultipleRunExperiment extends AbstractIterativeExperiment{

    protected int iteration;
    protected Map<String, Object> sum;

    public MultipleRunExperiment() {
    }

    public MultipleRunExperiment(SuperExperiment superExperiment) {
        super(superExperiment);
    }

    @Override
    protected void init() {
        iteration = 0;
        sum = new LinkedHashMap<String, Object>();
    }

    @Override
    protected boolean hasNext() {
        return iteration < ctx.config.ITERATIONS_NUMBER;
```

```
    }

    @Override
    protected void nextIteration() {
        ctx.iteration = iteration;
        ctx.globalIteration++;
        iteration++;
    }

    @Override
    protected void runIteration() {
        ctx.stat = new Statistics();
        ctx.scheduler = new Scheduler();
        ctx.scheduler.init();
        ctx.scheduler.schedule();

        Helper.add(sum, ctx.stat.getResults());
    }

    @Override
    protected String getCurrentVarValue() {
        return String.valueOf(ctx.iteration);
    }

    @Override
    protected String getVarName() {
        return "Iteration";
    }

    @Override
    protected String getExperimentName() {
        return "Multiple Run Experiment";
    }

    @Override
    protected void finalizeExperiment()
    {
        Helper.average(sum, ctx.config.ITERATIONS_NUMBER);
        String result[] = Helper.mapsToCSVStrings(sum);

        if(ctx.globalIteration == ctx.iteration)
        {
            Helper.outputStringToFile(( superExperiment == null ? "": superExperiment.getCSVHeaderPrefix() )+result[0],
                    ctx.config.OUTPUT_FILE, false);
        }
        Helper.outputStringToFile( ( superExperiment == null ? "": superExperiment.getCSVRecordPrefix() )+result[1],
                ctx.config.OUTPUT_FILE, true);
    }

    @Override
    protected void storeIterationResults()
    {
        System.out.println(getIterationInfo());
        if(ctx.globalIteration == 0)
        {
            Helper.outputStringToFile(getCSVHeader(), ctx.config.OUTPUT_FILE_DETAILED, false);
        }
        Helper.outputStringToFile(getCSVRecord(), ctx.config.OUTPUT_FILE_DETAILED, true);
    }
}
```

VarBadWeatherAV1Experiment

#

```
#Файл model\hybrid\v3\experiment\VarBadWeatherAV1Experiment.java
#Эксперимент по варьированию параметра av1 плохой погоды
package model.hybrid.v3.experiment;

/**
 * User: Sharov
 * Date: 04.12.2010
 * Time: 15:30:19
 */
public class VarBadWeatherAV1Experiment extends SuperExperiment
{
   protected double badWeatherAV1;

   public VarBadWeatherAV1Experiment() {
      super();
   }

   @Override
   protected void init() {
      badWeatherAV1 = ctx.config.BAD_WEATHER_AV_1_FROM;
   }

   @Override
   protected boolean hasNext() {
      return badWeatherAV1<=ctx.config.BAD_WEATHER_AV_1_TO;
   }

   @Override
   protected void nextIteration() {
      ctx.config.BAD_WEATHER_AV_1 = badWeatherAV1;
      badWeatherAV1+=ctx.config.BAD_WEATHER_AV_1_STEP;
   }

   @Override
   protected void runIteration() {
      new MultipleRunExperiment(this).execute();
   }

   @Override
   protected String getCurrentVarValue() {
      return String.valueOf(ctx.config.BAD_WEATHER_AV_1);
   }

   @Override
   protected String getVarName() {
      return "Bad Weather AV1";
   }

   @Override
   protected String getExperimentName() {
      return "Bad Weather AV1 Variation";
   }
}
```

VarBadWeatherAV2Experiment

```
#
#Файл model\hybrid\v3\experiment\VarBadWeatherAV2Experiment.java
# Эксперимент по варьированию параметра av2 плохой погоды
package model.hybrid.v3.experiment;

/**
 * User: Sharov
 * Date: 04.12.2010
```

```
 * Time: 17:17:05
 */
public class VarBadWeatherAV2Experiment extends SuperExperiment
{
   protected double badWeatherAV2;

   public VarBadWeatherAV2Experiment() {
      super();
   }

   @Override
   protected void init() {
      badWeatherAV2 = ctx.config.BAD_WEATHER_AV_2_FROM;
   }

   @Override
   protected boolean hasNext() {
      return badWeatherAV2<=ctx.config.BAD_WEATHER_AV_2_TO;
   }

   @Override
   protected void nextIteration() {
      ctx.config.BAD_WEATHER_AV_2 = badWeatherAV2;
      badWeatherAV2+=ctx.config.BAD_WEATHER_AV_2_STEP;
   }

   @Override
   protected void runIteration() {
      new MultipleRunExperiment(this).execute();
   }

   @Override
   protected String getCurrentVarValue() {
      return String.valueOf(ctx.config.BAD_WEATHER_AV_2);
   }

   @Override
   protected String getVarName() {
      return "Bad Weather AV2";
   }

   @Override
   protected String getExperimentName() {
      return "Bad Weather AV2 Variation";
   }
}
```

VarBadWeatherAVsExperiment

```
#
#Файл model\hybrid\v3\experiment\VarBadWeatherAVsExperiment.java
#Эксперимент по варьированию параметров av1 и av2 плохой погоды
package model.hybrid.v3.experiment;

/**
 * User: Sharov
 * Date: 16.08.2011
 * Time: 15:30:19
 */
public class VarBadWeatherAVsExperiment extends SuperExperiment
{
   protected double avFactor;
   protected double currentAVFactor;
```

```
    public VarBadWeatherAVsExperiment() {
        super();
    }

    @Override
    protected void init() {
        avFactor = ctx.config.BAD_WEATHER_AV_FACTOR_FROM;
        currentAVFactor = avFactor;
    }

    @Override
    protected boolean hasNext() {
        return avFactor<=ctx.config.BAD_WEATHER_AV_FACTOR_TO;
    }

    @Override
    protected void nextIteration() {
        ctx.config.BAD_WEATHER_AV_1 = avFactor*ctx.config.BAD_WEATHER_AV_1_INITIAL;
        ctx.config.BAD_WEATHER_AV_2 = avFactor*ctx.config.BAD_WEATHER_AV_2_INITIAL;
        currentAVFactor = avFactor;
        avFactor+=ctx.config.BAD_WEATHER_AV_FACTOR_STEP;
    }

    @Override
    protected void runIteration() {
        new MultipleRunExperiment(this).execute();
    }

    @Override
    protected String getCurrentVarValue() {
        return String.valueOf(currentAVFactor);
    }

    @Override
    protected String getVarName() {
        return "Bad Weather AV Factor";
    }

    @Override
    protected String getExperimentName() {
        return "Bad Weather AVs Variation";
    }
}
```

VarBadWeatherPExperiment

```
#
#Файл model\hybrid\v3\experiment\VarBadWeatherPExperiment.java
#Эксперимент по варьированию параметра p плохой погоды
package model.hybrid.v3.experiment;

/**
 * User: Sharov
 * Date: 28.11.2010
 * Time: 22:13:04
 */
public class VarBadWeatherPExperiment extends SuperExperiment
{
    protected double badWeatherP;

    public VarBadWeatherPExperiment() {
        super();
    }
```

```
    @Override
    protected void init() {
        badWeatherP = ctx.config.BAD_WEATHER_P_FROM;
    }

    @Override
    protected boolean hasNext() {
        return badWeatherP<=ctx.config.BAD_WEATHER_P_TO;
    }

    @Override
    protected void nextIteration() {
        ctx.config.BAD_WEATHER_P = badWeatherP;
        badWeatherP+=ctx.config.BAD_WEATHER_P_STEP;
    }

    @Override
    protected void runIteration() {
        new MultipleRunExperiment(this).execute();
    }

    @Override
    protected String getCurrentVarValue() {
        return String.valueOf(ctx.config.BAD_WEATHER_P);
    }

    @Override
    protected String getVarName() {
        return "Bad Weather P";
    }

    @Override
    protected String getExperimentName() {
        return "Bad Weather P Variation";
    }
}
```

VarFlowExperiment

```
#
#Файл model\hybrid\v3\experiment\VarFlowExperiment.java
#Эксперимент по варьированию потока заявок
package model.hybrid.v3.experiment;

/**
 * User: Sharov
 * Date: 28.11.2010
 * Time: 19:43:31
 */
public class VarFlowExperiment extends SuperExperiment
{
    protected double packetInterval;

    public VarFlowExperiment() {
        super();
    }

    @Override
    protected void init() {
        packetInterval = ctx.config.PACKET_INTERVAL_FROM;
    }
```

```
    @Override
    protected boolean hasNext() {
        return packetInterval<=ctx.config.PACKET_INTERVAL_TO;
    }

    @Override
    protected void nextIteration() {
        ctx.config.PACKET_INTERVAL = packetInterval;
        packetInterval+=ctx.config.PACKET_INTERVAL_STEP;
    }

    @Override
    protected void runIteration() {
        new MultipleRunExperiment(this).execute();
    }

    @Override
    protected String getCurrentVarValue() {
        return String.valueOf(ctx.config.PACKET_INTERVAL);
    }

    @Override
    protected String getVarName() {
        return "Packet Interval";
    }

    @Override
    protected String getExperimentName() {
        return "Flow Intensity Variation (Packet Interval)";
    }
}
```

VarGoodWeatherAVsExperiment

```
#
#Файл model\hybrid\v3\experiment\VarGoodWeatherAVsExperiment.java
#Эксперимент по варьированию параметра av1 и av2 хорошей погоды
package model.hybrid.v3.experiment;

/**
 * User: Sharov
 * Date: 16.08.2011
 * Time: 15:30:19
 */
public class VarGoodWeatherAVsExperiment extends SuperExperiment
{
    protected double avFactor;
    protected double currentAVFactor;

    public VarGoodWeatherAVsExperiment() {
        super();
    }

    @Override
    protected void init() {
        avFactor = ctx.config.GOOD_WEATHER_AV_FACTOR_FROM;
        currentAVFactor = avFactor;
    }

    @Override
    protected boolean hasNext() {
        return avFactor<=ctx.config.GOOD_WEATHER_AV_FACTOR_TO;
    }
```

```
    @Override
    protected void nextIteration() {
        ctx.config.GOOD_WEATHER_AV_1 = avFactor*ctx.config.GOOD_WEATHER_AV_1_INITIAL;
        ctx.config.GOOD_WEATHER_AV_2 = avFactor*ctx.config.GOOD_WEATHER_AV_2_INITIAL;
        currentAVFactor = avFactor;
        avFactor+=ctx.config.GOOD_WEATHER_AV_FACTOR_STEP;
    }

    @Override
    protected void runIteration() {
        new MultipleRunExperiment(this).execute();
    }

    @Override
    protected String getCurrentVarValue() {
        return String.valueOf(currentAVFactor);
    }

    @Override
    protected String getVarName() {
        return "Good Weather AV Factor";
    }

    @Override
    protected String getExperimentName() {
        return "Good Weather AVs Variation";
    }
}
```

VarGoodWeatherPExperiment

```
#
#Файл model\hybrid\v3\experiment\VarGoodWeatherPExperiment.java
#Эксперимент по варьированию параметра p хорошей погоды
package model.hybrid.v3.experiment;

/**
 * User: Sharov
 * Date: 15.08.2011
 * Time: 22:13:04
 */
public class VarGoodWeatherPExperiment extends SuperExperiment
{
    protected double goodWeatherP;

    public VarGoodWeatherPExperiment() {
        super();
    }

    @Override
    protected void init() {
        goodWeatherP = ctx.config.GOOD_WEATHER_P_FROM;
    }

    @Override
    protected boolean hasNext() {
        return goodWeatherP<=ctx.config.GOOD_WEATHER_P_TO;
    }

    @Override
    protected void nextIteration() {
        ctx.config.GOOD_WEATHER_P = goodWeatherP;
        goodWeatherP+=ctx.config.GOOD_WEATHER_P_STEP;
```

```
    }

    @Override
    protected void runIteration() {
        new MultipleRunExperiment(this).execute();
    }

    @Override
    protected String getCurrentVarValue() {
        return String.valueOf(ctx.config.GOOD_WEATHER_P);
    }

    @Override
    protected String getVarName() {
        return "Good Weather P";
    }

    @Override
    protected String getExperimentName() {
        return "Good Weather P Variation";
    }
}
```

VarORSwitchTimeExperiment

```
#
#Файл model\hybrid\v3\experiment\VarORSwitchTimeExperiment.java
#Эксперимент по варьированию премени переключения с оптики на радио
package model.hybrid.v3.experiment;

/**
 * User: Sharov
 * Date: 28.11.2010
 * Time: 16:46:44
 */
public class VarORSwitchTimeExperiment extends SuperExperiment {

    protected double switchORTime;

    public VarORSwitchTimeExperiment() {
        super();
    }

    protected VarORSwitchTimeExperiment(SuperExperiment superExperiment) {
        super(superExperiment);
    }

    @Override
    protected void init() {
        switchORTime = ctx.config.SWITCH_OR_TIME_FROM;
    }

    @Override
    protected boolean hasNext() {
        return switchORTime <= ctx.config.SWITCH_OR_TIME_TO;
    }

    @Override
    protected void nextIteration() {
        ctx.config.SWITCH_OR_TIME = switchORTime;
        switchORTime+=ctx.config.SWITCH_OR_TIME_STEP;
    }
```

```
    @Override
    protected void runIteration() {
        new MultipleRunExperiment(this).execute();
    }

    @Override
    protected String getCurrentVarValue() {
        return String.valueOf(ctx.config.SWITCH_OR_TIME);
    }

    @Override
    protected String getVarName() {
        return "OR Switch Time";
    }

    @Override
    protected String getExperimentName() {
        return "OR Switch Time Variation Experiment";
    }
}
```

VarROSwitchTimeExperiment

```
#
#Файл model\hybrid\v3\experiment\VarROSwitchTimeExperiment.java
#Эксперимент по варьированию премени переключения с радио на оптику
package model.hybrid.v3.experiment;

/**
 * User: Sharov
 * Date: 28.11.2010
 * Time: 22:04:54
 */
public class VarROSwitchTimeExperiment extends SuperExperiment {

    protected double switchROTime;

    @Override
    protected void init() {
        switchROTime = ctx.config.SWITCH_RO_TIME_FROM;
    }

    @Override
    protected boolean hasNext() {
        return switchROTime <= ctx.config.SWITCH_RO_TIME_TO;
    }

    @Override
    protected void nextIteration() {
        ctx.config.SWITCH_RO_TIME = switchROTime;
        switchROTime+=ctx.config.SWITCH_RO_TIME_STEP;
    }

    @Override
    protected void runIteration() {
        new MultipleRunExperiment(this).execute();
    }

    @Override
    protected String getCurrentVarValue() {
        return String.valueOf(ctx.config.SWITCH_RO_TIME);
    }
```

```
    @Override
    protected String getVarName() {
        return "RO Switch Time";
    }

    @Override
    protected String getExperimentName() {
        return "RO Switch Time Variation Experiment";
    }
}
```

Flow

```
#
#Файл model\hybrid\v3\flows\Flow.java
#Модель входящего потока заявок
package model.hybrid.v3.flows;

import model.hybrid.v3.impl.Context;
import model.hybrid.v3.impl.Packet;
import model.hybrid.v3.util.Counters;
import cern.jet.random.Exponential;
import cern.jet.random.engine.MersenneTwister;

public class Flow {
        public final double INTERVAL; // seconds
        public final int LEN; // bits

        private Exponential interval; //in seconds
        private Exponential len; //in bits

        protected double time; // seconds

    public Flow()
    {
        INTERVAL = Context.getInstance().config.PACKET_INTERVAL/1000; // seconds
            LEN = (int) (8*Context.getInstance().config.PACKET_LEN); // bits
        interval = new Exponential(1.0/INTERVAL, new MersenneTwister(Counters.getNextSeed()));
            len = new Exponential(1.0/LEN, new MersenneTwister(Counters.getNextSeed()));
        time = interval.nextDouble();
    }

    public double getTime() {
                return time;
        }

        public void setTime(double time) {
                this.time = time + interval.nextDouble();
        }

        public Packet execute() {
                double arrival = time;
                time += interval.nextDouble();
                return new Packet(arrival, (int) len.nextDouble());
        }
}
```

Packet

```
#
#Файл model\hybrid\v3\impl\Packet.java
#Модель пакета данных
package model.hybrid.v3.impl;

public class Packet {
   /**
    * Arrival time in seconds
    */
   public final double arrival;
   /**
    * Packet length in bits
    */
		public final int len;

		public Packet(double arrival, int len) {
			this.arrival = arrival;
			this.len = len;
		}
}
```

Scheduler

```
#
#Файл model\hybrid\v3\impl\Scheduler.java
#Планировщик отвечающий за планирование событий при иммитационном моделировании
package model.hybrid.v3.impl;

import model.hybrid.v3.channels.CompositeChannel;
import model.hybrid.v3.flows.Flow;
import model.hybrid.v3.states.OpticsRadio;
import model.hybrid.v3.weather.Weather;
import model.hybrid.v3.weather.WeatherConstant;
import model.hybrid.v3.weather.WeatherHyperExp;
import model.hybrid.v3.util.Helper;

import java.math.BigInteger;

public class Scheduler
{
		public static final int ONE_YEAR = 365*24*60*60; //in seconds
   public double time = 0;

		private Flow flow;
		private CompositeChannel channel;
   private Weather weather;

   private Context ctx;
   private boolean useStatistics;
   private boolean useAnalytics;

   public Scheduler()
   {
      this(Context.getInstance());
   }

   public Scheduler(Context ctx)
   {
      this.ctx = ctx;
   }
```

```
public double getCurrentTime()
{
   return time;
}

public void init()
{
   useAnalytics = ctx.config.USE_ANALYTICS;
   useStatistics = ctx.config.USE_STATISTICS;

   time = 0;
                        channel = new CompositeChannel();
                        flow = new Flow(); //new FlowSimple();

   weather = ctx.config.TURN_OFF_WEATHER ? new WeatherConstant() : new WeatherHyperExp();
}

          public void schedule() {

                        int year = ONE_YEAR;
   final double EXP_LENGTH = ctx.config.EXPERIMENT_LEN*ONE_YEAR;
                        do {
      if(time > year) {
                                                //System.out.println("years passed              :              " + year/ONE_YEAR);
                                                year += ONE_YEAR;
                                    }
      double channelTime = channel.getTime();
      double weatherTime = weather.getTime();
      double flowTime = flow.getTime();

      // 1. Simulation period ended
      if(weatherTime < 0) { // simulation period end
                                                break;
                                    }
      // 2. There is no packets to be delivered
      else if(channelTime < 0) {
                                                time =  Math.min(
                flowTime,
                weatherTime
             );
                                    }
      // 3. Find next event
      else
      {
         time =  Math.min(
                Math.min(flowTime,weatherTime),
                channelTime
             );
      }
      performExecution();

                        } while(time < EXP_LENGTH);
   channel.experimentFinished();
          }

private void performExecution()
{
   if(useStatistics)
   {
      ctx.stat.queueLength(channel.getQueueSize());
   }

   if( time == weather.getTime() )
   {
      weather.execute(); // next change in the weather
                                    channel.weatherChange();

      ctx.stat.weatherChanged();
```

```
      //System.out.println("Weather Changed (time = "+Helper.convertTimeToStr(time)+")");
    }
    if( time == channel.getTime() )
    {
      channel.execute();
    }
    if( time == flow.getTime() )
    {
      channel.packetCome(flow.execute());
    }

    if( useAnalytics && channel.getTime() < 0)
    {
      if(channel.getQueueSize() > 0)
        throw new IllegalStateException("Queue is not empty!!!");

      if( channel.getState() instanceof OpticsRadio)
        twoChannelsAnalytics();
      else
        oneChannelAnalytics(ctx.config.THROUGHPUT_RADIO);
    }
  }

        public void oneChannelAnalytics(double throughput)
  {
    double avSendingTime = Helper.convertPacketLengthToSendingTime(flow.LEN, throughput); //seconds
    double mu = 1.0/avSendingTime; // 1/seconds
    double lambda = 1.0/flow.INTERVAL; // 1/seconds
                double regenerationPeriod = 1.0/(mu - lambda) + 1.0/lambda;
                //double rho = lambda/mu;
                double q = lambda/(mu-lambda); // average packets in system
    double n = lambda*lambda/(mu-lambda)/mu; // average queue length
                double w = 1.0/(mu-lambda); // average packet time in system (in seconds)
    double waitingTime = n/lambda; //w - avSendingTime; // average packet waiting time (in seconds)

                double interval = Math.min(ctx.config.EXPERIMENT_LEN*ONE_YEAR, weather.getTime()) - time;
                double periodsNum = Math.floor(interval/regenerationPeriod);
                double t = regenerationPeriod*periodsNum;

    int numberOfSentPackets = (int) (t*lambda);
    time += t;
    ctx.stat.analyticsPeriod(t, n, waitingTime, BigInteger.valueOf(numberOfSentPackets));

    flow.setTime(time);
        }

  public void twoChannelsAnalytics()
  {
    double lambda = 1.0/flow.INTERVAL; // 1/seconds

    double avSendingTimeO = Helper.convertPacketLengthToSendingTime(flow.LEN, ctx.config.THROUGHPUT_OPTICS); //seconds
    double muO = 1.0/avSendingTimeO; // 1/seconds

    double avSendingTimeR = Helper.convertPacketLengthToSendingTime(flow.LEN, ctx.config.THROUGHPUT_RADIO); //seconds
    double muR = 1.0/avSendingTimeR; // 1/seconds

    double rho = lambda/(muO+muR);
    double b = muO*muR/lambda/lambda*(1.0+ (lambda+muO)/(lambda+muR) );
    double a = b + muR/lambda*(1.0+muO/(lambda+muR))+ muO/(lambda+muR)+1.0/(1.0-rho);

                double regenerationPeriod = a/lambda/b;

                double q = ( a-b+1.0/(1.0-rho)/(1.0-rho) )/a; // average packets in system
    double n =  rho/(1. - rho)/(1. - rho)/a; // average queue length
    double w = q/lambda; // average packet time in system (in seconds)
    ///TODO check this: this is just approximation
    //double avSendingTime =
(avSendingTimeR*ctx.config.THROUGHPUT_RADIO+avSendingTimeO*ctx.config.THROUGHPUT_OPTICS)/(ctx.config.THROUGHPUT_RADIO+ctx.config.TH
```

```
ROUGHPUT_OPTICS);
	double waitingTime = n/lambda;//- avSendingTime; // average packet waiting time (in seconds)

				double interval = Math.min(ctx.config.EXPERIMENT_LEN*ONE_YEAR, weather.getTime()) - time;
				double periodsNum = Math.floor(interval/regenerationPeriod);
				double t = regenerationPeriod*periodsNum;

	int numberOfSentPackets = (int) (t*lambda);
	time += t;
	ctx.stat.analyticsPeriod(t, n, waitingTime, BigInteger.valueOf(numberOfSentPackets));

	flow.setTime(time);
		}
}
```

HyperExp

```
#
#Файл model\hybrid\v3\util\HyperExp.java
#Гиперэкспоненциальное распредеение
package model.hybrid.v3.util;

import cern.jet.random.Exponential;
import cern.jet.random.Uniform;
import cern.jet.random.engine.MersenneTwister;


public class HyperExp {
		private Exponential exp1;
		private Exponential exp2;
		private Uniform prob;

		private double p;
		private double mean1;
		private double mean2;

		private double time;

		public HyperExp(double p, double mean1, double mean2) {
			this.p = p;
			this.mean1 = mean1;
			this.mean2 = mean2;
			reset();
		}

		public double getTime() {
			return time;
		}

		public void execute() {
			if(prob.nextDouble() <= p) {
				time = exp1.nextDouble();
			} else {
				time = exp2.nextDouble();
			}
		}

		public void reset() {
			exp1 = new Exponential(1.0/mean1, new MersenneTwister(Counters.getNextSeed()));
			exp2 = new Exponential(1.0/mean2, new MersenneTwister(Counters.getNextSeed()));
			prob = new Uniform(0.0, 1.0, new MersenneTwister(Counters.getNextSeed()));
			execute();
		}
```

```
}
```

WeatherHyperExp

```
#
#Файл model\hybrid\v3\weather\WeatherHyperExp.java
#Модель погоды
package model.hybrid.v3.weather;

import model.hybrid.v3.config.Config;
import model.hybrid.v3.impl.Context;
import model.hybrid.v3.util.HyperExp;

public class WeatherHyperExp {
	private HyperExp distrGood;
	private HyperExp distrBad;

	private double time = 0; // seconds

	public WeatherHyperExp() {
		Config conf = Context.getInstance().config;
    distrBad = new HyperExp(conf.BAD_WEATHER_P, conf.BAD_WEATHER_AV_1, conf.BAD_WEATHER_AV_2); // min
		distrGood = new HyperExp(conf.GOOD_WEATHER_P, conf.GOOD_WEATHER_AV_1, conf.GOOD_WEATHER_AV_2); // hours
		reset();
	}

	@Override
	public double getTime() {
		return time;
	}

	@Override
	public void execute() {
		state = (state == State.GOOD ? State.BAD : State.GOOD);
		if(state == State.GOOD) {
			distrGood.execute();
			time += distrGood.getTime()*60*60;
		} else {
			distrBad.execute();
			time += distrBad.getTime()*60;
		}
	}

	@Override
	public void reset() {
		time = 0;
		state = State.BAD;
		distrGood.reset();
		distrBad.reset();
		execute();
	}

	@Override
	public State getState() {
		return state;
	}
}
```