Using Java Technologies in Statistics Applications Data Analysis Graphic Generator

Felix FURTUNĂ, Bucharest, Romania
Marian DÂRDALĂ, Bucharest, Romania

This paper proposes an idea for building a Java Application Programming Interface (API) that allows generating statistics graphics used in Data Analysis. The core of this API is a Java 2D library, and some classes which implement the 2D geometric transformations. The classes are small, fast, easy to use and can be integrated into your projects, and are completely written in pure Java. It allows users to easily develop and deploy sophisticated reports across any platform.
***Keywords**: Java API, Data Analysis, Graphics.*

Introduction

The key idea of the proposed API is inspired by two of the data analysis methods specific features:

- data multivariate feature that involves memorizing (storing) data in tables of large sizes;
- analyses are based a lot on graphical representations, which better express the data fundamental features and better comply descriptive features of the methods.
- Similar approaches are used by other Java libraries which can by applied in Statistics applications, such as:
- Java Report Free – a Java libraries for generating reports [3];
- Java SPSS Writer – Java API for generating SPSS files [10];
- Java Free Chart – Java API designed for generating pie charts, bar charts, line charts, scatter plots etc [9].

The Data

Data Analysis Graphic Generator data is sourced via *Swing's TableModel* interface [1][2][6]. The `TableModel` interface specifies the methods the `JTable` will use to interrogate a tabular data model. In standard usage, there are three major tasks in generating reports with Data Analysis Graphics Generator (DAGG):

- arrange for some data that can be accessed via the TableModel interface (that is, the model used by Swing's JTable class);
- create a DAGG object with a TableModel instance that contains the data;
- rendering the graphics with the DAGG instance and pass the report to a print or save them to the JPEG file.

The Data is designed to work with data that is accessible via the TableModel interface.
In order to create a TableModel the following possibilities could be taken into account [1] [2] [6]:

- Implementing of the *TableModel* interface;
- Extending of the *AbstractTableModel*;
- Using of the *DefaultTableModel*.

For implementing the DAGG we used the second possibility shown above, extending of the *AbstractTableModel*. Any other *TableModel* could be converted to this one. The *TableModel* implementing class looks like this:

```
import javax.swing.table.*;
import java.util.*;
import javax.swing.*;
public class TabelDeDate extends AbstractTableModel {
private Vector linii=new Vector(),coloane;
public TabelDeDate(Vector coloane)
{ this.coloane=coloane; }
public Class getColumnClass(int indiceColoana) {
        Vector v=(Vector)linii.elementAt(0) ;
        return
v.elementAt(indiceColoana).getClass();
}
public int getRowCount(){ return linii.size();
```

```
}
public int getColumnCount() { return coloane.size(); }
public Object getValueAt(int row,int column){
        Vector v=(Vector)linii.elementAt(row) ;
        return v.elementAt(column);
}
public String getColumnName(int indiceCol)
{                                   return coloane.elementAt(indiceCol).toString(); }
public void aduagare(Vector v) { linii.addElement(v); }
}
```

The conversion from some *TableModel* to the main *TableModel* could be as follows:

```
public DataAnalysisGG(TableModel model)
{
  Vector v=new Vector();
  for(int i=0;i<model.getColumnCount();i++)
v.addElement(model.getColumnName(i));
  tabel=new TabelDeDate(v);
  for(int i=0;i<model.getRowCount();i++){
        v=new Vector();
        for(int j=0;j<model.getColumnCount();j++)
v.addElement(model.getValueAt(i,j));
                tabel.adaugare(v);}}
```

For further information about *TableModel* go to Sun's Java website: http://java.sun.com/.

In order to generate the graphics using data accessed via JDBC, DAGG can be used for generating a *TableModel* instance from a *JDBC ResultSet*.

Building graphic representations using Java2D

The Java 2D library belongs to Java Foundation Classes (JFC) [4]. The Java Foundation Classes are a graphical framework for building portable Java-based graphical user interfaces (GUIs). JFC consists of the Abstract Window Toolkit (AWT), Swing and Java 2D. Together, they provide a consistent user interface for Java programs, regardless whether the underlying user interface system is Windows, Mac OS X or Linux. The Java 2D API is a set of classes for advanced 2D graphics and imaging [5]. The most important Java2D class is Graphics2D. This class inherits the ancient Graphics class. The following are some of the properties that the Graphics2D class provides:

- *Background* – Allows a *Color* object to be specified as the default color that appears when portions of the graphics context are erased.
- *RenderingHints* – Controls the quality of graphics rendering.
- *Paint* – Instead of a simple color, the *Paint* interface can be used to fill shapes with any solid, gradient or pattern. Under Java 2D, the *Color* class has been modified to implement the *Paint* interface.
- *Stroke* – Java 2D supports the *Stroke* interface to describe a virtual pen to draw lines and curves with various widths, colors and patterns.
- *Transform* – Supports the use of Transformations. The Transform property of *Graphics2D* is of type *java.awt.geom.AffineTransform* and represents a mathematical rule expressed as a 3-by-3 matrix. This transformation is applied to all operations as they are rendered by the *Graphics2D* object.
- *Composite* – Controls what happens when drawing operations overlay already colored-in pixels. The value of the property is of type *java.awt.Composite*, which is an interface that describes how colors are supposed to combine.

Case Study

In the following lines we are bringing up a graphical example of Principal Components Analysis [7][8]. It's a graphical projection of the cases on the factor-plane into the first two axes corresponding to the first two principal components. Data are sourced from a MS SQL Server database table and have the following structure: *ch_hr* – food expenses, *ch_b* – drink expenses, *ch_int* – utilities expenses, *ch_ii* – footwear and clothing expenses, *ch_c* – domestic expenses, *ch_m* – furniture expenses, representing the main expenses of a common family in some UE countries. The Java sequence code for building the data model and its graphical represen-

tation are:

```
try{
        Connection
c=DriverManager.getConnection(urlConexi
une);
        Statement
s=c.createStatement(ResultSet.TYPE_SCRO
LL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
        ResultSet r=s.executeQuery
("select    ch_hr,ch_b,ch_int,ch_ii,ch_c,ch_m
from chetuieli");
        DataAnalysisGG d=new DataAna-
lysisGG(r);
        r.close();
        r=s.executeQuery("select nume from
tari");
        String et[]=new String[r.getRow()];
        int i=0;

while(r.next()){et[i]=r.getString(1);i++;}
        r.close();
        d.ACPIndivizi(0,1,et);
       }
       catch(Exception
e){javax.swing.JoptionPane.showMessageDi
alog
(null,"Eroare!!!"+e);}
```

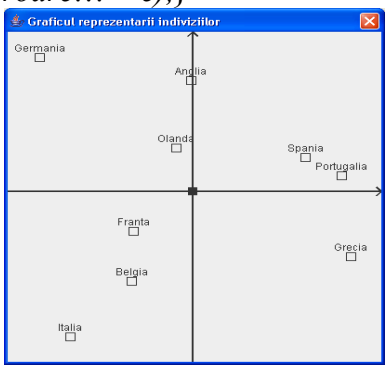

Figure 1. The graphical representation

Conclusions

Data Analysis Graphic Generator covers graphical representations for Principal Components Analysis, Cluster Analysis, Factor Analysis, Discriminant Analysis, Canonical Analysis, Correspondence Analysis, Multidimensional Scaling, but it is open for any kind of development based on graphical representation of data stored in matrix format. MVC philosophy that the library was built on allows this approach.

References

- * *, *Java Platform Enterprise Edition 5. API Specifications* , http://java.sun.com/javaee/5/ docs/api
- * *, The Java Tutorials, http://java.sun.com/docs/books/ tutorial/uiswing/ index.html
- ***, *Java Free Report Documentation* , http://www.jfree.org/jfreereport/ index.php
- * *, Java Foundation Classes, http://java.sun.com/products/jfc/index.jsp/
- Fraizer, C., Bond, J., *Java API: manualul interfeţei de programare a aplicaţiilor*, Bucureşti, Teora, 1998.
- Tănasă, T., Olaru, C., Andrei, S., *Java de la 0 la expert*, Iaşi, Polirom, 2003
- Jobson, J.,D., *Applied multivariate data analysis*, New York, Springer, 1992
- Voineagu, V., Furtună, F.,Voineagu, M, Stefănescu, C., *Analiza factorială a fenomenelor social-economice în profil regional*, Editura Aramis, Bucureşti, 2002
- * *, JfreeChart Documentation, http://www.jfree.org/jfreechart/index.html
- * *, Java SPSS Writer, http://spss.pmstation.com/