BDrive®
Power for Intelligent Collaborative Infrastructures™

ARTICLE
Architecting the Generic Graphing Package

Authors:Nitin Uchil, Survi Gopal, Nanda Motikane, Rahul Phadnis

About
This article presents the plan for a generic charting tool that enterprises can use to access local/remote data and enable its viewing and charting in a web environment.

Package Modules and their Description
The Generic Graphing Package (GGP) consists of the following modules:

- **Header** for navigation
- **Home** - the home page for the utility
- **Create** - this is the work horse module for this application that connects to local and remote sources, lists the data and enables the channel selections for plotting
- **Open** - This lists existing chart definitions for overlays and quick plotting
- **Plot** - (embedded module) This takes care of setting the display and plotting graphs
- **Analytics** - This includes statistical and mathematical processing of the data
- **Admin** - for administrative features

The **Create** module is made up of sub-modules as follows:

- **Select** - this will enable the selection of the source
  - Local File
  - Remote File (pre-configured list of file servers/configure on demand)
  - Remote Database (pre-configured list of database servers/connect to new schema on demand)
  - Legacy System
  - Previously stored table (more on this later)
- **Initiate** - Based on the previous selection it will initiate the file loader/connection to the database
- **Access** - It will list out the files/tables/allow you to create a select statement and enable you to pick on what you want. Please note that in case of a Local File selection, this is handled when the Browse feature is automatically initiated.
- **List** - once you have specified what to access list will "depict" it. The way this will be done is that it will create a temporary table on a local "Hypersonic" database based on the access file/table fields with the ability that this can be saved permanently by the user (check - more on this later above).
- **Graph** - Here one would have the ability to choose channels and the graph type to plot. Nanda has done extensive work on this and we will reuse the code from the GraphPattern.
- **Analyze** - Here we will add statistical and mathematical routines to manipulate the List.

The **Admin** module will **Manage users**, their roles and tables created by them. It will also have the ability to **Add Sources** with defined properties for file servers and remote databases.

Managing State in the Create Module
The create module will go thru the steps of:

SELECT > INITIATE > ACCESS > LIST > SET DISPLAY > GRAPH > ANALYZE

On first use, subsequent states will not be available, but once the hypersonic table is loaded, the user should have the ability to move back and forth between LIST, GRAPH and ANALYZE states. If the user re-clicks on Access to redefine the table, subsequent states are not available till the list is re-generated.

Local Database Needs
The Local Hypersonic database running on localhost has the following tables

user - to manage user properties
log - to register user activity
source - to permanently store database and file server properties that will be available in the Initiate module
user_tables - names of tables stored by user, date created/modified source.

It also creates temporary/named tables when the users List items from the blob extracted. Last five temporary tables created by the user should be help - name of table CDSID_MMDDYYHHMMSS.

Naming Fields
Typical tabular data do not have more than 50 fields. But I checked with TASE engineers at Ford and in one of the tests they go up to 66 fields. SO we will assume a maximum of 100 columns in tabular data. This information is not necessary when creating the user table, but when we start naming the columns, it becomes important.

How do we name fields? When the data is uploaded, we explicitly ask the user if the first row is Header. If the answer is yes, then we create the temporary table with these as field names. However, this row may contain names that have blank spaces. To map the original names to the one's we use when creating the user_table, we need a table called table_names that by table explicitly stores the names of the fields. This table, since it is defined apriori, has to be of definite length - seq_id, table_name, user, date_created, field1 - field100.

Rendering Plots

Currently we can plot using NetChart applets from Visual Mining. GM does not allow applet rendering in their applications (because of their perception that security is a problem). We need to see how Visual Minings NetChart Server that also generates gif images can be used instead. Also, check on Sitraka and other plotting ways - maybe using SVG also. Just like Survi has a flavor for FTP (UFTP or JScape), we need to flavor out GraphPattern.

Packages Reviewed

- NetCharts from Visual Mining
- JClass Chart from Sitraka (Formerly KL Group). Other products: Entire JCLass Suite
- JFree Chart from Object Refinery. Other products: JWorkbook, JFinance and JCommon
- EasyCharts from Object Planet. Other products: Network Analysis and Survey Software.
- CoPlot from Cohort Software

We reviewed the plotting capabilities of NetCharts, Sitraka, Popcharts and CoHort. They all had a good image generation capabilities. NetCharts appeared to be better of the four chart vendors because of their chart definition language (CDL). NetCharts CDL could be easily understood and also was easy to create an XML file using their parameters as the XML elements and the attributes as the XML attributes. By abstracting out the chart definitions into an XML file and building intelligent parsing agents we are now able to add any number of their optional parameter information into the XML file and see the result instantly, without having to modify the code. The XML file designed on the basis of the NetCharts CDL served as the standard, using which we could also test JClass from Sitraka. We can easily include charting packages from other vendors by specifying a parsing agent that views a standard XML chart definition, and creating the vendor specific parameters and attributes out of it.

In order for the application to create graphs as images we had to install a version of XVFB (Xterm Virtual Frame Buffer). Earlier to Java 1.4 servers that had an terminal could only produce dynamic images by using the graphic card of the machine on which the application is hosted. But lately servers are headless, ie without a terminal. Hence applications could not be ported to such machines. But in Java 1.4 and above there is an ability in Java to use software loaded on the system that simulates a graphics card. For our application to work seamlessly we had to do the following steps:

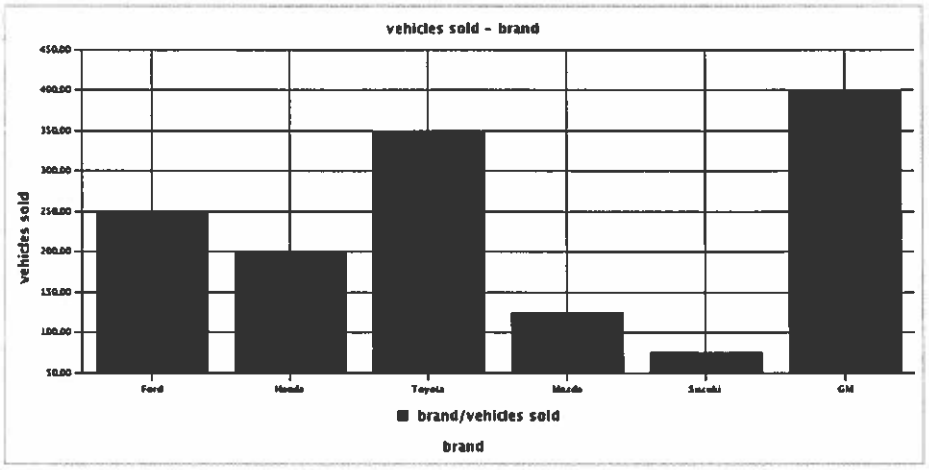
```
1. Shutdown the webserver
2. Install from the Linux CD the XVFB software
3. Login as root
4. Enter - export DISPLAY=:0.0
5. Enter - xhost +
6. Startup the webserver
```

At this stage the application is ready to create the required dynamic images.

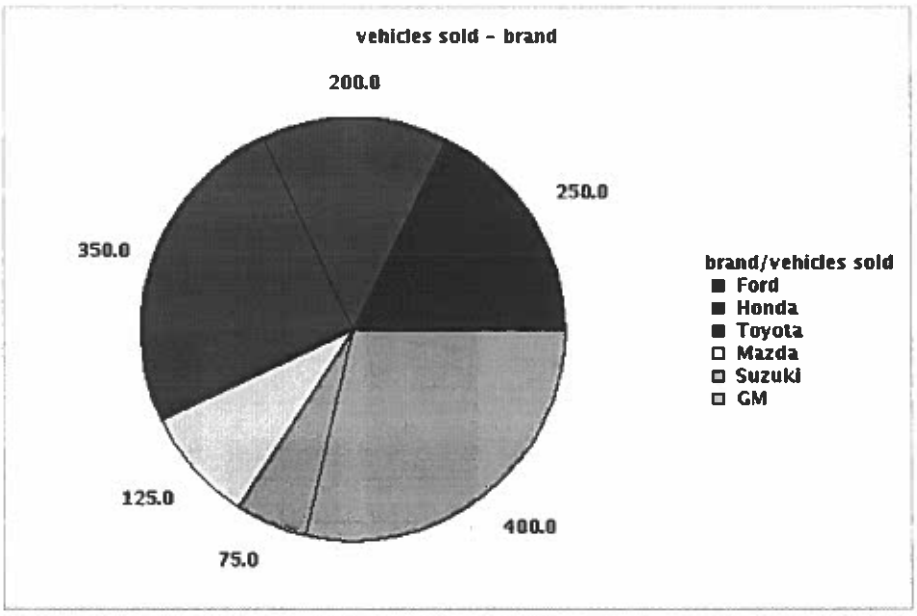
A Sampling of the Plots

Following are sample graph images created using the application:

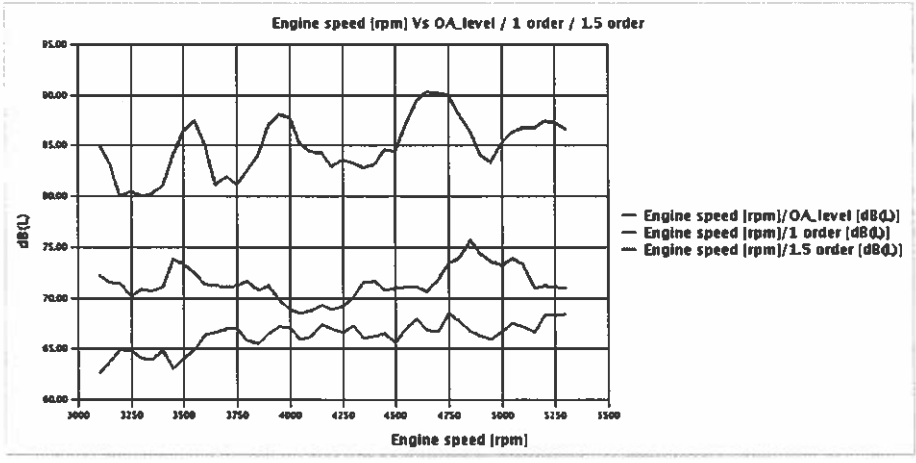
Bar Chart



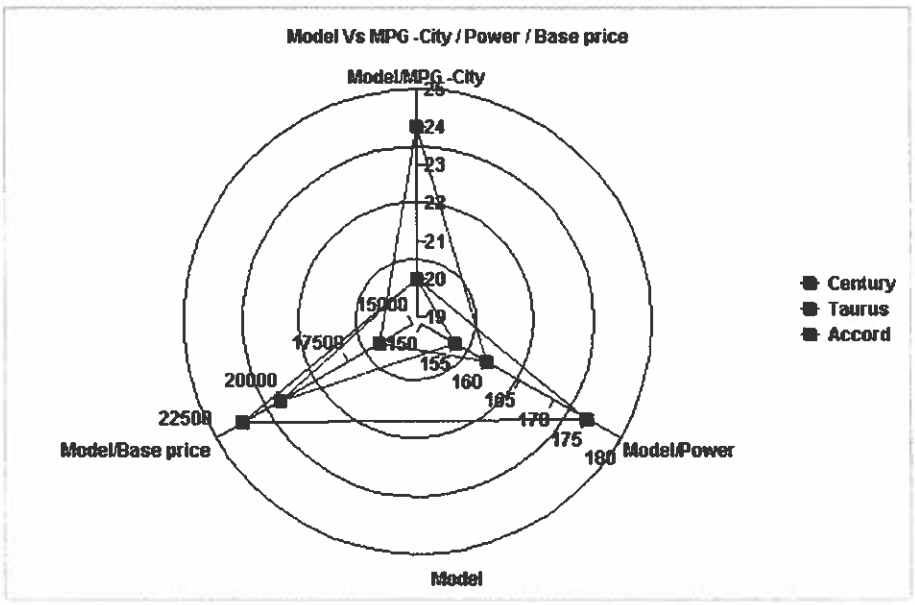
Pie Chart



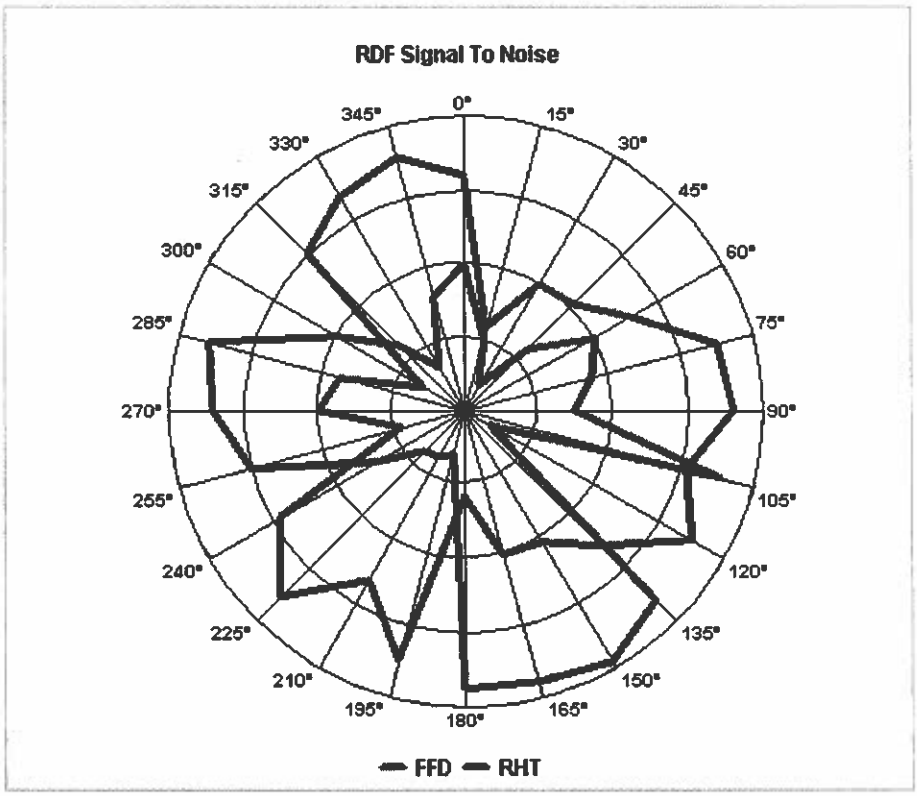
Cartesian Chart



Radar Chart


Model Vs MPG -City / Power / Base price
Model/MPG -City
24
23
22
21
20
19
15000
17500
20000
22500
150
155
160
165
170
175
180
Model/Base price
Model/Power
Model
Century
Taurus
Accord

Polar Chart


RDF Signal To Noise
0°
15°
30°
45°
60°
75°
90°
105°
120°
135°
150°
165°
180°
195°
210°
225°
240°
255°
270°
285°
300°
315°
330°
345°
FFD
RHT