Charting with dynamic symbolizers

Introduction

The dynamic symbolizers proposal opened the door to the possibility of developing custom, programmatic symbol generators that do accept feature parameters as inputs, be it in the category of marks or external graphics.
This alone already provides charting abilities to GeoTools when coupled with an external charting service such as Google Chart (thanks Chris Holmes and David Winslow for pointing this out in GeoServer land): the chart is seen as an external graphic that is loaded passing in feature attribute as chart data series elements.
An example of this is action is:

```
<ExternalGraphic>
  <OnlineResource xlink:type="simple"
xlink:href="http://chart.apis.google.com/chart?cht=p&chd=t:${100 * FEMALE /
PERSONS},${100 * MALE /   PERSONS}&chs=100x100&chf=bg,s,FFFFFF00"/>
<Format>image/png</Format>
</ExternalGraphic>
```

This combines feature attribute retrieval to show the percentage of female and male population in a pie chart for each state in the usual GeoServer topp:states demo.

However, relying on an external, remote service to generate charts has its own downsides:

1. each chart request takes extra time due to network delays, image building and decoding
2. the remote service is either free, and thus usually unreliable in terms of availability and performance, or requires one to pay per request
3. an external service does not usually have direct access to out of feature data series (think linked tables that can provide a list of titles and values for a certain feature)
4. bigger data series can make the request extend beyond the limits of what a GET request can do

Point 1 and 2 can be mitigated by deploying a local chart service. Having one has also the extra benefit of allowing chart production for other clients as well, such as cases in which the charts are too dense to be drawn on a map as smallish icons, but can be more naturally shown in map popups.

Point 4 can be mitigated by having the dynamic symbolizers use filter function that can query a known data source given a search key.

Point 3 should be addressed by having an in process charting system. This would also provide a definitive solution for point 1 and 2.

The DeeGree way

DeeGree has introduced charting abilities in its own WMS and presented it at FOSS4G 2008: http://conference.osgeo.org/index.php/foss4g/2008/paper/view/332/168

The general idea is quite close to the Google charts usage in dynamic symbolizers, that is:

```
http://localhost:8080/deegree/chart?chart=Pie3D&title=&legend=false&width=250&amp;heig
ht=220&format=image/png&GDD=$GDD$&HDD=$HDD$&CDD=$CDD$"
```

where $attribute$ extracts an attribute from the current feature, and GDD,HDD,CDD keys are the titles of the data elements in the pie chart.
Also interesting is how the system has been linked to a local data source, the SOS service, to provide time series charts (an example of leveraging a local data series to draw a custom chart).
The chart drawing library chosen is, not surprisingly, the very popular JFreeChart.

It is also interesting to see how the charts appearance is configured. The request provides only very generic information, but JFreeChart allows for very detailed control on how a chart is made, be it colors, fonts, positioning of elements and so on. DeeGree uses a custom xml template file format to specify how the chart is actually built. There are no examples, but the parsing code gives some ideas on what it might look like: https://svn.wald.intevation.org/svn/deegree/base/trunk/src/org/deegree/graphics/charts/ChartConfig.java

Interesting links

Eastwood charts reimplements the Google Charts API on top of the JFreeChart library: http://www.jfree.org/eastwood/. If Eastwood can be also run as a library component this gives us the best bang for the buck, it's already able to chew URL's, uses a well documented API, and can be directly exposed to the net as a service already.

Summary

To sum up:

- having an in process charting API seems the way to provide fast and reliable in map charting
- in a web setting, complementing it with a web accessible charting service provides better flexibility (just build the web layer on top of the in process library level)
- the charting library most likely provides a host of configuration, a text driven, comprehensive chart template system allows to keep the web api lean and leverage the full power of the charting library at the same time
- while allowing one to hand list labels and values for the chart, also allow the retrieval of those information directly from an external data source given a query and the attributes to be used for building up the data series

Pipe dream ideas

- Chart conflict resolution (mixed with labels, too)
- A custom charting drawer for faster chart drawing (JFreeChart is designed for flexibility, not for speed).