Paper 381-2013

Visualize your OLAP cubes on a map using a Stored Process

Frank Poppe, PW Consulting; Sjoerd Boogaard, Kiwa Prismant

ABSTRACT

What if you have an OLAP cube with a geo-dimension and you want a map from that, but you don't have ArcGIS? Enter this general stored process. It can read measures and dimensions from the cube, and it uses SAS/GRAPH® software to combine that with boundary data, creating a color-coded map. The map is clickable to navigate between the geographical levels. A selection pane offers measures, and non-geographical dimensions surface as (hierarchical) filters. Measures and dimensions are read from the metadata; values for the filters are read from the cube, using MDX. Boundaries are clipped to the right zoom level, and the picture gets a background with roads from a web service. HTML, CSS, and JS is generated to glue everything together and to deliver it to the portal.

SUMMARY

Kiwa Prismant, a healthcare consulting organization in the Netherlands, maintains a portal with several information tools for its clients. Many of these tools are based on OLAP cubes, and a lot of these cubes have a geographical dimension (levels comparable to country – state – county – zip code). To exploit this geographical dimension using only SAS/Base and SAS/GRAPH code a Stored Process has been developed that links these geographical levels to data sets with the respective boundaries.

The Stored Process generates a choropleth map for one of the measures in the cube. The map is clickable as long as there is lower geographical level present in the cube (and a data set for those geographical areas is available).

A selection pane is provided in which the user can activate filters for all the non-geographical dimensions in the cube using drop-down boxes. The availability of those dimensions is determined on the fly from the metadata on the cube, and the possible values for the filters are read from the cube using MDX queries. For a dimension with hierarchical levels drop-down boxes for a next level are being populated once a selection on the higher level has been made. Also in this pane other measures can be selected.

If the cube uses geographical levels for which boundaries are known the Stored Process can exploit any cube. Only a few additional parameters are needed, in particular the scheme to be used for each measure when mapping the measure to different colors.

This paper will describe the different parts of the Stored Process: code to query the SAS metadata, code to generate MDX queries that supply the filter values, the MDX query with the right filters to collect the measurements, SAS code to generate the HTML and CSS to combine all elements on the portal page, and of course the SAS code to generate the picture.

The page has a legend pane showing the active filters, and a navigation pane with select boxes, organized hierarchically according to the cube hierarchies, to change the. JavaScript links the elements of the page together.

The actual picture is comparable to the choropleth output of the GMAP procedure, but is entirely generated with the SAS annotate facility. That way the map can be cropped at the right point, but also depicting the areas that are only partially visible within the frame that is needed for the selected areas. Also this made it possible to enhance the picture with map features like roads en railways, collected in real-time from a public web service.

INTRODUCTION

The paper is set up as follows. We start by giving a short overview of the kind of business questions this application tries to answer. Next a description is given of the technical environment in which the application runs and how the end users operate it.

Before starting on a description of how the application operates a short description is given of the data sets used to draw the geographical areas. Then follow the different paragraphs discussing specific aspects of the application, in the order the Stored Process gathers and processes the information:

- Reading the metadata on a cube;
- Querying the cube (using MDX) for the members for each level;
- Building MDX where clauses to apply the filters selected by the user;

- Querying the cube for the selected measure;
- Selecting the areas that fall within the frame, and clipping those that are only partially visible;
- Determining the coloring for the measure;
- Generating the HTML for the URL's and the select boxes to put on the page for the next user action;
- Generating the picture containing the map and attaching those URL's to the picture;
- Putting al the material (picture, JavaScript and css) in a single HTML page.

During several of these steps two different things are done more or less in parallel. On the one hand information is being collected to generate the query for the measures on the subset of data to be used in the current map to be drawn. At the same time that information is being used to put together the code that will surface the drop-down boxes and clickable areas for the next request.

The paper will be concluded with a discussion of possibilities for further development, finishing with some conclusions.

BUSINESS BACKGROUND

Nowadays the Dutch hospital sector is a competitive market where hospitals are expected to compete on quality and efficiency. For hospital CEO's it is therefore important to have a clear view of the market dynamics. They want answers for questions like the following.

- Am I loosing market share on a particular treatment?
- Why – or why not – are patients redirected to my hospital?
- Where do my patients come from, what is the travel time and how important is this for my patients when it comes to highly specialized medical treatment?

Visualizing market information in geographical maps delivers much more actionable information. Changes in market shares can easily be linked to infrastructural changes like new bridges, highways or public transport availability. Identifying causes of changes in market share can be done more easily, which enables hospitals to anticipate.

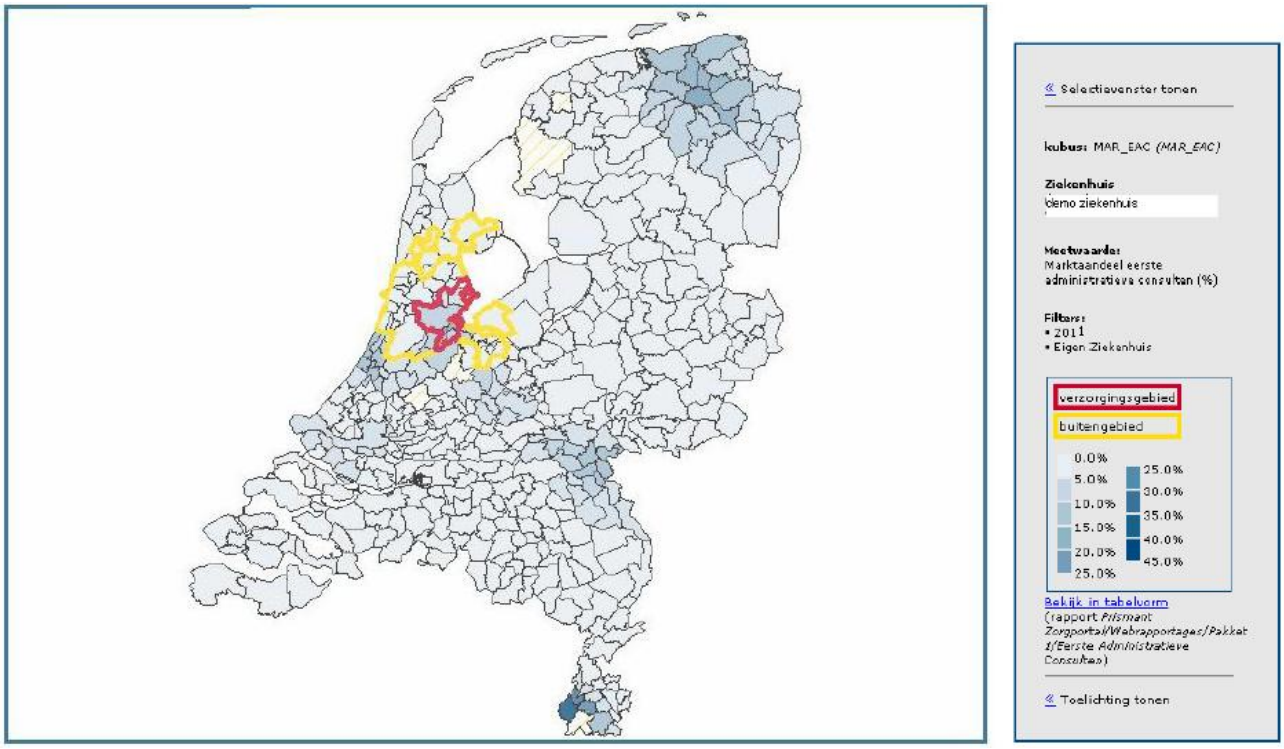
The OLAP cubes of Kiwa Prismant were being surfaced in web reports through the SAS® Information Delivery Portal. To accommodate the desire to show this information also in a geographical way it was decided to develop a tool that could use the cubes in that way. The tool is being made available to the clients of Kiwa Prismant as part of the so-called Prisma range of products: PrismaMaps.

TECHNICAL CONTEXT

SAS offers the SAS Bridge for ESRI, which makes it possible to combine the power of SAS OLAP Cubes with the spatial data of ArcGIS. But this is only an option for organisations that have ArcGIS as well, which is not the case for Kiwa Prismant.

Kiwa Prismant *does* have available data sets with the boundaries for Dutch ZIP-codes and municipalities. The goal therefore became the development of a Stored Process that would use the measures from the OLAP cubes and those boundary data sets to produce a choropleth map.

The Stored Process starts with a number of default choices for its parameters, using the cube that contains information for the topic under consideration, using a default measure and no filters activated except the geographical region of interest for the particular hospital under consideration. All actions that a user can choose, by clicking in the map, zooming out again to a larger area, selecting another measure or activating a filter, result in a new call to the same Stored Process with a changed set of parameters.



Display 1. All regions shown, map with legend

Most of the information needed by the Stored Process is determined on the fly from the metadata on the cube. Only a small amount of information is being provided through a separate data set of information. This means only a minimum amount of additional information has to be provided to the system whenever a new cube is added to the system. Throughout this paper these items will be pointed out.

THE DATA SETS FOR THE BOUNDARIES

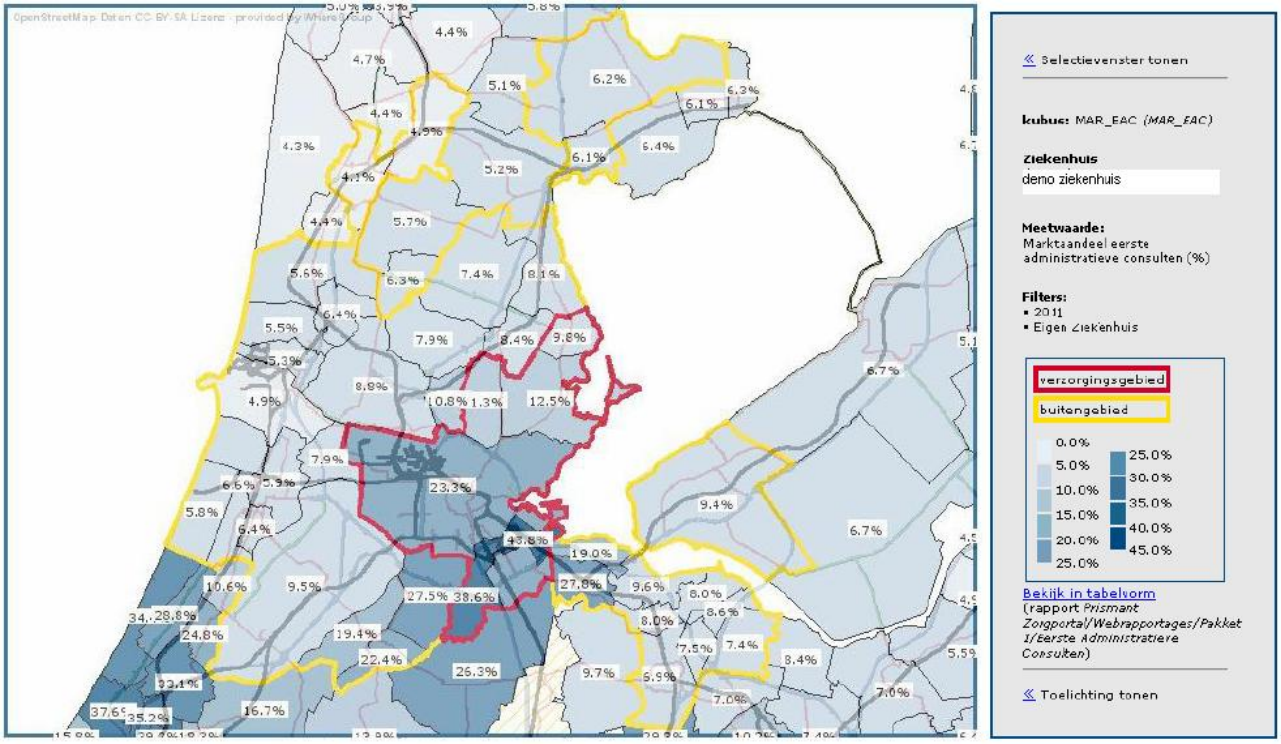
Basically these are the type of traditional data sets described in the SAS/Graph® documentation containing an ordered list of points that define the boundary of an area with the following variables:

- an identification variable, uniquely identifying an area;
- SEGMENT – used to separate different noncontiguous polygons for the same logical area (for instance areas separated by a large river, or an island before the coast);
- X and Y – the coordinates in a suitable coordinate system.

Also there is a response data set which for each value of the identification variable contains the membership of that area to the higher geographical levels. These data set is labeled here the 'response data set' because in normal SAS/GRAPH operations this data set would contain the statistics that are being used to color the map. In this Stored Process the statistics will be collected from the cube.

To facilitate the drawing process there is an additional statistical data set with for each combination of identification value and SEGMENT the minimum and maximum X- and Y-value. This is commonly named the bounding box for an area. This facilitates the clipping process described later on.

Also the acreage of each segment is computed and stored.



Display 2. View zoomed in to 'buitengebied' (larger target area, yellow outline), other areas clipped

Clipping is almost impossible to accomplish in the mapping procedure of SAS/GRAPH (PROC GMAP). An area is either in the picture, or it is out: it never is partly in the picture, the remainder being clipped off. So some data massage is in order. The bounding boxes can be used to make a quick separation into areas whose bounding boxes are completely within the map (and can be drawn without ado), that are completely outside (and can be ignored) and that overlap. For the latter group the individual points of the boundaries have to be examined. This is a rather elaborate process, but not very interesting within the context of this paper.

A small table that is accessed by the Stored Process defines for each geographical level that the Stored Process should be able to draw which are the three SAS data sets (coordinates, response and geo-statistics) and what the relevant id-variable is in those data sets.

READING THE METADATA ON A CUBE

The first query is for the description of the cube. This will be used as a label in the legend area. This is a rather simple metadata call using the macro variable &cube, one of the parameters with which the Stored Process is called:

```
rc=metadata_getattr("omsobj:cube?@Name='&cube'","Desc",desc);
```

More complex queries are needed to determine the hierarchies, the dimensions and the levels within those hierarchies and the measures that are available. For these more complex queries PROC METADATA is being used:

```
proc metadata in=query out=dhXML ;
run;
```

The fileref query refers to a file with the following kind of content.

```
<GetMetadataObjects>
   <Reposid>$METAREPOSITORY</Reposid>
   <Type>Dimension</Type>
   <Objects/>
   <Ns>SAS</Ns>
   <Flags>&flags</Flags>
   <Options>
    <Templates>
          <Dimension>
```

```
                <Hierarchies>
                       <Hierarchy />
                </Hierarchies>
            </Dimension>
    </Templates>
     <XMLSelect Search="*[Cubes/Cube[@Name='&cube']]" />
   </Options>
</GetMetadataObjects>
```

The macro references are being resolved by putting all the datalines through a DATA _NULL_ step first, that does nothing more then reading each line in, applying the RESOLVE function and writing the line out again. This is a handy trick that will be used at the end of the Stored Process as well, as the main method to combine everything on a single web pages.

The result is received as XML. The structure of this XML is known, so a an XML-map has been defined already that is used in a XML libref, 'translating' the XML into a SAS data set.

```
libname dimshier xml xmlfileref=dhXML xmlmap=dhmap;
```

Three requests are being made in this way:

- for the hierachies and dimension (as shown)
- for the properties of the hierarchies (similar)
- for the hierachies and the levels (similar)

This results in three tables which can be linked on the hierarchy-id, giving all the information on the levels.

The measures in the cube could be found this way as well. However, the available measures for each cube are read from a utility data set. For a new cube a small tool is available that extracts the measures in the way described above from the cube and stores these as an initial fill in the data set. Then the administrator has to to add some necessary information on the measures:

- if the measure should be available for selection;
- which measure is the default one to be displayed initially;
- the SAS format to be used when displaying values;
- the method to be used when categorizing the values and assigning colors (more on this in a next section).

QUERYING THE CUBE FOR THE MEMBERS FOR EACH LEVEL

Now a macro has being defined that queries the OLAP server for the values in a particular level. This macro is being called for each level found in the previous step, except for those levels that are part of a dimension that has been specified as type GEO (compared to STANDARD or TIME).The query using the geo-level is postponed until after examining all the other dimensions.

The macro uses an MDX query which is executed on the OLAP server using SQL Pass-Thru. Because we are not very knowledgable on MDX, the following code was arrived at by trial and error, using as examples the code that can be surfaced using Enterprise Guide and the *"Edit with MDX Editor…"* option.

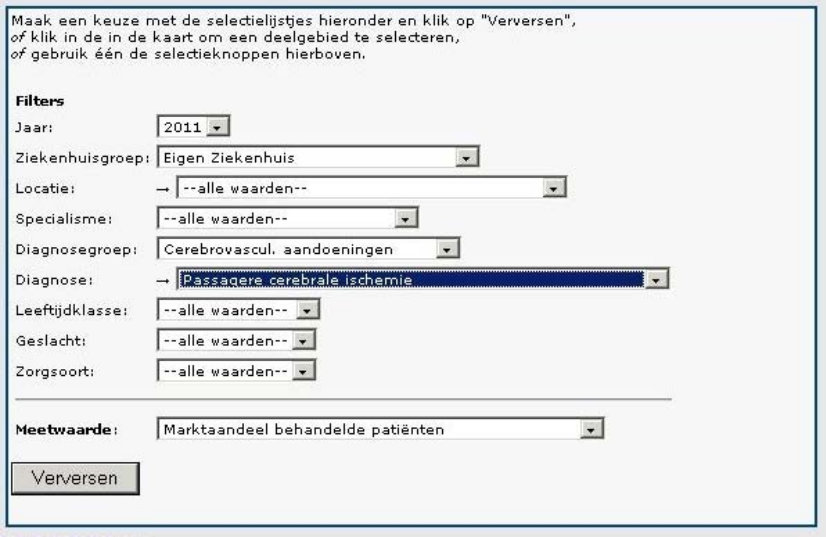
```
proc sql noprint;
  connect to olap ( host = &olapserver port = &olapport );
  create table members as select
    *
    from connection to olap (
      SELECT
        NON EMPTY [measures].[&defaultMeasure] on columns
        ,
        NON EMPTY[&dimension].[&levelName].AllMembers ON rows
        FROM [&cube]
    )
  ;
quit;
```

This generates a table with all members of that level. A small problem is that level names used in cubes not necessarily are valid SAS names. In the process above the names of the variables in the resulting table are changed slightly, if needed to arrive at valid SAS names. Luckily the original name of the level is contained in the label of the variable, so a small piece of extra code uses PROC DATASETS to determine how the variables actually are being called.

The macro compares the values returned from the OLAP server with the values from the filter that have been specified in the Stored Process call. If a match has been found this will lead to two different actions:

- the relevant part of the where clause for the subsequent MDX query is generated;
- in the HTML-code that is being generated to surface all the values as possible choices in a drop-down box on the web page this value will be marked as the selected value.

This latter action becomes more complicated for a multi-level dimension. The deeper levels can only get their values once the first-level value has been selected. To accomplish this javascript functions will be defined for each value containing the valid values the next level. This function will be called when a first-level value gets selected, and will then populate the next drop-down box.



Display 3. The selection pane, showing multi-level filter

So the drop-down boxes are populated using the following logic.

- For a first-level filter, create an entry "—alle waarden –" (i.e. "all values") and make that the default, except for some dimensions that always need a selection (like year, "Jaar" in the display above).
- For a second-level filter (and lower), if no selection has been made yet on the higher level, the filter is de-activated ('grayed') and the text "first choose a value from above" is shown.
- If on the first-level filter a selection is being made, activate the next drop-down box and populate it with valid values (as in "Diagnosegroep" and "Diagnose" in the display), also creating an "—alle waarden –" entry, making that the initial selection.

BUILDING MDX WHERE CLAUSES AND QUERYING THE CUBE

While cycling through all the levels it has been determined for which levels filters are active. For each active filter a piece of code is added to the where clause on the MDX query.

The code ensures that in the time dimension always a year has been selected. By default this is the last year. If the time dimension has smaller time units as well, a smaller time unit may be selected as well. Some cubes have another dimension across which aggregated measures produce meaningless of even misleading results. Also for this dimension the code ensures that at always a value is selected (for this one by default the first one). Currently this has been hard-coded – in a next version these extra properties of certain dimension will be stored in the SAS metadata or in a seperate small SAS data set.

Also it is always made sure that only the values for the hospital under consideration are being requested. This is only an extra security because the security on the cube and the organization dimension already enforces that one only gets data for which one is authorized.

The general form of the MDX query used to request the measures that are to be shown on the map is as follows.

```
SELECT
    ON EMPTY [Measures].[&measure] on columns ,
    NON EMPTY [&geodimensionName].[&geoHierName].[&idVar].AllMembers ON rows
    FROM
    [&cube]
&where
;
```

As shown previously this is encapsulated in SQL Pass-Thru. Again, there may be other ways to express the query, but this is an easily understandable form which also can be assembled in a somple way during the different steps of the Stored Process by assigning values to the macro variables.

The different macro variables are:

- &measure – the measure currently selected by the user;
- &geoDimensionName and &geoHierName – determined in one of steps above when the type “GEO” is encountered;
- &idvar – the current geographical level, either the default one or the value passed in as parameter when a user has drilled into the map by clicking an area;
- &cube – the name of the cube as used earlier;
- &where – the where clause.

It should be noted that this where clause in an MDX query is of a different kind then the ones in a SQL query. In a SQL query the where clause filters the observations that are being returned. In an MDX query the where clause specifies the slice of the cube. The form of the where clause is as follows:

```
[Hospital].[All Hospital].[Hospital Name XYZ] ,
[Time].[All Time].[2011]
```

So several *axes* can be combined by concatenating them with a comma, thus making this syntax very suitable for the 'incremental' approach used by going through all the dimensions. The name of the 'all' member (the middle part in each term) usually is “All” followed the dimenion name, but it can be defined differently in the cube. Therefore also this element is determined from the metadata.

A point of attention is the use of square brackets. The example shows that when selecting the slice for a particular member of a level (in this case, for the member “Hospital Name XYZ”) one has to use the formatted values (in the source data the hospitals are identified by id-values, but the cube recognizes them by there formatted name). Sometimes people feel the need to use square brackets when translating values into formatted texts. PROC OLAP does not complain when it encounters such values, but is is impossible to extract those members through the OLAP Server. The MDX language has an escape mechanism (involving double square brackets), but this has not been implemented in the SAS incarnation of it.

CLIPPING THE AREAS FOR THE MAP

If a filter is in effect for the geographical level only part of the map should be displayed. Say e.g. the state of California has been selected. The map has to be cropped such that the display shows the whole of California. Within that area the counties of California have to be shown, but also those counties of bordering states that fall completely or partially within that frame.

SAS does not have a procedure that automatically clips areas that fall only partial within a frame. Therefore some steps have been implemented to do just that.

As explained in the paragraph on “the data sets for the boundaries” a data set has been produced for each geographical level containing the *“bounding boxes”* for each segment. So for each disjunct area it is known what the minimum and maximum x- and y-values are.

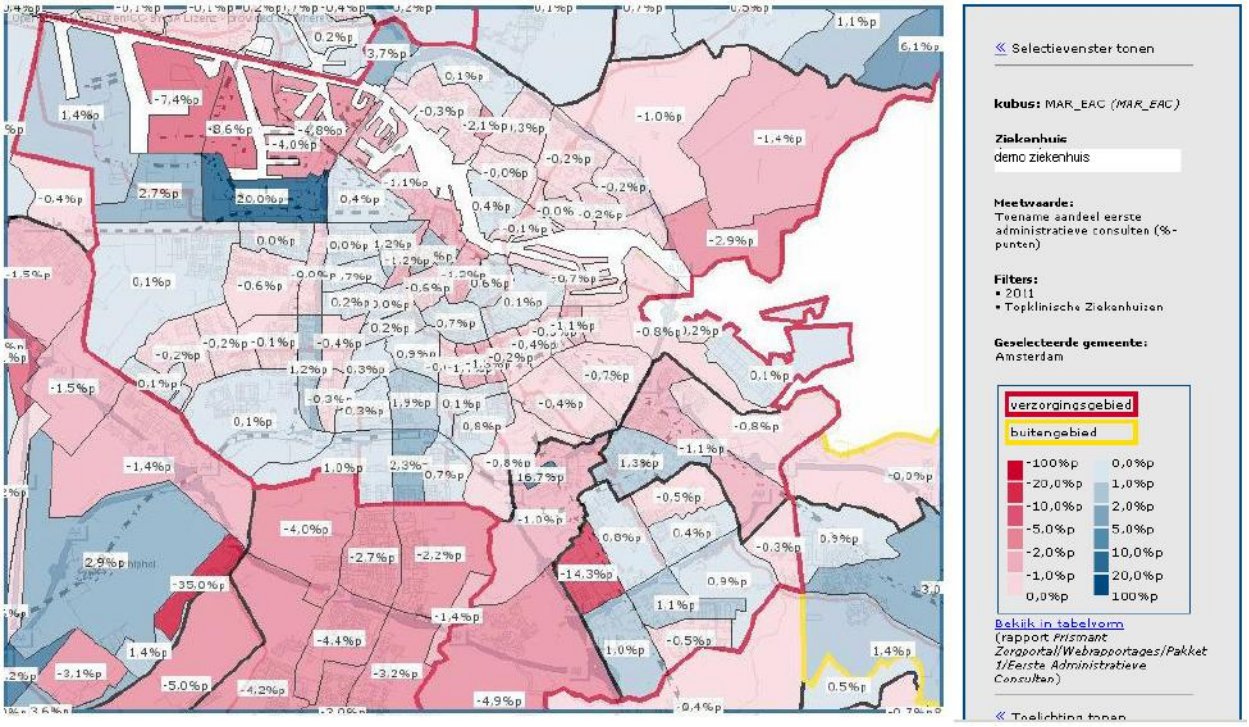
A comparison off the bounding box values for each segment with the bounding box of the map (the frame, i.e. in the example the minimum and maximum x- and y-values for California) makes it possible to make a quick break-down of segments that are completely within the frame, those that are definitely outside the frame, and those that might be partially within. For the latter category the boundaries are examined to determine exactly which part (if any) is within the frame, and to generate a modified boundary for the area within (i.e. clipped at the boundary of the frame). This is

done in a simple way, by moving all points outside the frame to the frame itself.

CREATING THE MAP

The first step in creating the map is determine the color to be used for each area.
A set of coloring schemes has been defined beforehand, and for each measure the relevant scheme has been fixed. The following basic schemes have been defined, using three basic colors. For this implementation two particular variants of red and blue have been selected, being colors of the corporate styles.

- Determine the minimum and maximum for the measure in the current subset, map the minimum on white and the maximum on blue, other values linearly between, or using a particular number of bands;
- Always map 0 on white and 100 on blue, other values linearly in between, or using a particular number of bands. This method is suitable for measure expressed as percentages.
- Map -100 to red, 0 to white and +100 to blue, using bands in between. This method is suitable for measures that express a change in percentage values.



Display 4. Two color scheme for display of change in market share

Using the method defined for the selected measure the color for each area is being determined.

The standard way of creating a map is by using PROC GMAP. But as already could be seen from the displays shown the map is being enhanced with some basic features like roads and railways. These are requested from a public web service by giving the bounding box of the map and the number of pixels the picture should have (width and height). These two sources, the features from the web service and the boundaries drawn by SAS, have to fit nicely. And alas, PROC GMAP is *not* very predictable in this respect. Therefore the ANNOTATE facility has been used to draw the areas. This sounds more complicated then it is. One simply works through the data set with all the coordinated for the boundaries, at the start of each area 'polygon start' function is generated specifying the color to be used for the area, an for each subsequent point a 'polygon continue' function is added. Only lakes (holes in the area) take some extra consideration.

At the same time HTML code has to be generated to make the picture clickable. For this an HTML map is needed that describes the same geographical areas, but this time in terms of pixels. This way the browser knows what URL to follow when the user clicks somewhere in the picture. The general syntax of an HTML-map is as follows.

```
<map name="&idvar">
<area shape="poly" coords="0,0,82,126, … " href="http://…" title="…">
<area shape="poly" coords="90,58,  …"
```

…
</map>

The map is linked to the picture by referencing it in the <img> tag.

```
<img src ="…" width= … height= … usemap="#&idvar">
```

To complicate things a bit the coordinate system for SAS/GRAPH has its reference point in the bottom-left corner, while the browser counts pixels from top-left, so each value has to be recomputed.

The title attribute in the map creates an mouse-over effect: the text is displayed by the browser when the mouse hovers over that area. For this special care had to be taken in situations where an area is completely surrounded by another, larger, area. The text shown will be the first for which the mouse position satisfies the list of coordinates in the <area > definition. If this is the larger area, the text for the smaller area will never be displayed.
Therefore the areas are ordered in increasing acreage, making sure smaller areas always come first. This is the reason for adding the acreage to the statistical data set with the bounding box information.

Most pieces of HTML code that are been produced in this Stored Process are stored in macro variables, or defined as macro's that will be resolved when everything is put together in the last step. But this HTML-map has a risk of getting too large for this process because the list of coordinates can become long. There is a limit to the number of characters that can be processed this way by the macro compiler. Therefore this code is written to a temporary file. In the next paragraph we will describe how to fit everything together.

The picture itself is also written to a temporary file. For this the well-tried method is used that SAS developed when Stored Processes were first deployed: the file resides within a temporary SAS catalog, and a special Stored Process is used in the `src` attribute in `<img >` tag to display the picture on the web page.

GENERATING HTML AND PUTTING ELEMENTS TOGETHER

The final web page is defined in the code as the DATALINES of a DATA _NULL_ step. The main statement in the data step is the RESOLVE function. This causes all macro variables and all macros to be resolved. The macros can be quite complicated, and any string of characters that they generate will become part of the dataline.

The result is written to _WEBOUT.

The complete code (without the datalines) is as follows.

```
DATA _NULL_ ;
    file _webout;
    infile cards4;
    length line $ 9999;
    input line & $char.;
    if upcase ( line ) =: '<SAS INCLUDE ' then do;
           file = scan ( line , 3 , ' >');
           insertFile = cats ( "&path/" , file , '.html' );
           infile insert filevar = insertFile end = done;
           do while  ( not done );
                  input line & $char.;
                  line = trim ( _INFILE_ );
                  put line;
           end;
    end;
    else do;
           line = resolve ( _INFILE_ );
           put line;
    end;
datalines4;
```

The code that needs further explanation is the whole `if – then do – end` construction. What happens here? If the line contains the string "<SAS INCLUDE" then the next word is taken as a filename, and everything in that file is read and inserted before continuing with the datalines. By this construction the possible limit on the number of characters that can be managed by resolving macros is overcome.

The web page displayed in this way has been built by honoring the specifications that were fed into the Stored Process as paramters. It contains all the URL's and FORMS with drop-down boxes to call the Stored Process again with new specifications. This process can be repeated endlessly.

POSSIBILITIES FOR FURTHER DEVELOPMENT

The Stored Process has been developed with the SAS Information Delivery Portal, used on a desktop pc, in mind. But is also works reasonably well on an iPad, or any other tablet, although the dimensions could be optimized. Also the user interaction has to be adjusted, because on a tablet the mouse has been replaced with swipe and drag movements.

Although almost all information is read from the cube metadata, some extra information still is needed. This makes that still some extra administration is needed to exploit an arbitrary new cube with a geographical dimension. It should be possible to develop some defaults that will make it possible to exploit any cube, even without additional information.

The way a measure is displayed (which bands are used, which colors, etc.) is fixed by the administrator. The user cannot influence this. The interaction with the user could be enhanced to give the user more possiblities in this area.

CONCLUSION

An OLAP cube is an excellent way to provide users with multi-dimensional data. This stored process makes it possible to visualize the geographical dimensions in a cube, using standard SAS code and standard SAS/GRAPH data sets. This gives the users a new view on their data, enabling new interpretations.

Only a minimum of extra information is needed to exploit a cube this way, because the SAS metadata can be queried for most information on the cube.

Using SAS it was possible to bring together many different 'languages'. Queries are being made using XML, SQL en MDX, the web page is being built using PNG, HTML and JS.

CONTACT INFORMATION

Your comments, suggestions and questions are valued and encouraged. Contact the authors at:

Frank Poppe
PW Consulting
Frank.Poppe@PWconsulting.nl
www.pwconsulting.nl

Sjoerd Boogaard
Kiwa Prismant
Sjoerd.Boogaard@kiwa.nl
www.kiwaprismant.nl