ESRI Nederland

RGI 233 WP4

SILVERLIGHT TGAPVIEWER

Customer	TU Delft	***Contractor***	ESRI Nederland B.V.
Customer contact	Theo Tijssen	***Author***	Ben de Vries
		Date	December 2008
		Version	0.1
		Status	concept

Version history

Version	Date	Update	Author
0.1	18-dec-2008	WP4	BdV

Distribution list

Date	Name	Department
18-dec-2008	Theo Tijssen	TU Delft, OTB

Contents

1. Introduction 1

1.1 System context 1

1.2 Original goals WP2 1

1.3 Accomplished goals in WP2 2

1.4 Accomplished goals in WP4 2

2. TGap service capabilities 4

2.1 GetTGapData 4

2.2 GetProgressiveTGapData 5

3. Client implementation 7

3.1 High level design 7

3.2 Detail design 8

3.2.1 Map and rendering 8

3.2.2 Map refresh cycle 9

3.2.3 Configuration 10

3.2.4 TGapConfigEditor 11

3.2.5 Map element classes 12

3.2.6 GetProgressiveTGapData 13

4. Screenshots 15

5. Development tools 18

6. Conclusions 19

6.1 Performance GetTGapData 19

6.2 Performance GetProgressiveTGapData 19

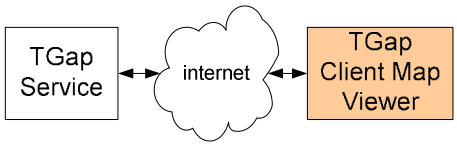
6.3 Silverlight and graphics 19

1. Introduction

This report describes the implementation of a map viewer which can visualize tGAP (topological Generalized Area Partitioning) data structures. The first implementation of this viewer was done last year in WP2 with the ArcGIS Server Mobile toolkit. That toolkit had little or no facilities for animations. Since the requirements for animations were still high, a new viewer has been developed with the Microsoft Silverlight 2.0 SDK. That SDK is available since june 2008.

Only the client side implementation is described here. The tGAP data structures are created and maintained in a tGAP server environment. The server side technology for tGAP is described in a separate report.

1.1 System context



- tGAP data structures are published on the server through a tGAP service (WFS/SOAP).
- The map viewer client application sends requests to the tGAP service and displays the results.

1.2 Original goals WP2

- Service
  - The tGAP data would originally be published through a WFS service.
- Dynamic map refresh
  - The objective was to implement a map viewer on a mobile platform that, dependent of the required map scale, would dynamically refresh itself while connected with the tGAP service.
- Animated rendering
  - The map view should render in an animated way. The map's image should not be displayed at once but instead should build up in a smooth way. Map elements should fade out/fade in gradually when the map image changes as a result of zooming or panning. Also map labels should smoothly fade in and fade out.
- Testdata
  - Several different TGap datasets will be available through a service for testing from OTB at TU Delft.

- Mobile platform
    - The map viewer must be deployed on a windows mobile device. For this reason the initial development environment (for WP2) chosen was the ESRI ArcGISServer Mobile Application Development Framework for .NET.

1.3 Accomplished goals in WP2

- Service
    - The ESRI client technology does not yet support WFS services. The support for WFS services is planned for the next major release of ArcGIS. TU Delft has therefore created a standard SOAP service which publishes the same information as the WFS service did. The geometries are packaged in GML format. At the client side logic has been implemented to interpret the soap response and unpack the GML geometries.
- Dynamic map refresh
    - The tGAP database was not fully prepared to receive map requests for an absolute map scale. Instead it was possible to request a map for a certain set of predefined scale levels. A scale level is an integer value that does not correspond to a map scale. Valid scale levels range from 2 – 160. Since there was no logic to relate a certain map scale to a predefined scale level, the dynamic aspect of map viewing was not implemented.
- Animated rendering
    - All requirements regarding animated rendering can not be implemented by the ESRI technology at this moment. Therefore all these requirements have been left out and shifted forward to work package 4. It means that the current implementation of the map viewer renders its graphics in a standard way.
- Testdata
    - The available testdata in tGAP consists of a set of cadastral polygons. Not too much data but just good enough to demonstrate the concepts.
- Mobile device
    - Since there was no suitable mobile device available, the work package team decided that the map viewer could be deployed on a standard windows desktop platform. The program logic can easily be ported to a Windows Mobile 5 or 6 deployment platform. The user interface would have to be redesigned.

1.4 Accomplished goals in WP4

- Service
    - The SOAP service that was used in WP2 is still functioning and was used for the new client implementation.
- Dynamic map refresh

  - In this WP there is no more need to use scale ranges in the TGap service requests. Given a desired mapextent, the service chooses the proper scale level.

- Animated rendering
  - Using animated rendering in a web browser is only possible when using FLEX (Adobe Flash) or Silverlight (Microsoft Windows Presentation Foundation, WPF) technology. Since the Microsoft Silverlight 2.0 SDK was just released recently, it was decided to build the map viewer with Silverlight 2.0.
  - Silverlight offers, just like WPF, various ways to do animations:
    - Fading in and out, translating, rotating and scaling of graphic objects given a "from" and a "to" value. The time that an animation should last is given by a "duration" value.
  - A Silverlight application runs inside any standard web browser. The TGap viewer has been tested in Internet Explorer 7.0 and FireFox 3.0.
- Testdata
  - The number of test datasets on the OTB server has been increased for WP4. The performance of the TGap service was improved.
  - The only negative aspect is still the amount of data in the response of a TGap request. The TGap service does not really generalize the geometries of the objects that are sent back from the TGap service. It would be nice if some form of "weeding" could be implemented at the TGap side. As a result the current transmission time of a TGap response over the internet takes relatively long. Sometimes seconds. While the rendering, including animation, at the client side takes just a fraction of that time.
- Mobile device
  - In WP4 we still have no mobile device available. The intention was there to use a mobile flavor of the Silverlight SDK to implement the viewer on a Windows Mobile 6.0 device. However, the release of the Silverlight Mobile SDK is postponed by Microsoft to Q2 - 2009.
  - The current implementation only functions in a web browser on a desktop machine. The plan is to convert the WP4 Silverlight viewer to a mobile version as soon as the Silverlight Mobile SDK becomes available.

2. TGap service capabilities

The TGap service capabilities in WP4 as they are published by OTB have changed since WP2. These are some of the most important capabilities:

2.1 GetTGapData

```
<xsd:element name="getTgapDataElement">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="tgapName" nillable="true" type="xsd:string" />
      <xsd:element name="windowXmin" nillable="true" type="xsd:decimal" />
      <xsd:element name="windowYmin" nillable="true" type="xsd:decimal" />
      <xsd:element name="windowXmax" nillable="true" type="xsd:decimal" />
      <xsd:element name="windowYmax" nillable="true" type="xsd:decimal" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

A GetTGapData request takes the name of the desired tgap plus the extent for which tgap objects must be returned. The response from the service contains lists with edges and faces that look like this:

```
<xsd:complexType name="TgapLayerUser">
  <xsd:complexContent mixed="false">
    <xsd:extension base="tns:TgapLayerBase">
      <xsd:sequence>
        <xsd:element name="edges" nillable="true" type="tns:EdgeList" />
        <xsd:element name="faces" nillable="true" type="tns:FaceList" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Each edge in an edgelist contains these attributes:

```
<xsd:extension base="tns:EdgeObjectBase">
  <xsd:sequence>
    <xsd:element name="startNodeId" nillable="true" type="xsd:decimal" />
    <xsd:element name="rightFaceId" nillable="true" type="xsd:decimal" />
    <xsd:element name="endNodeId" nillable="true" type="xsd:decimal" />
    <xsd:element name="edgeId" nillable="true" type="xsd:decimal" />
    <xsd:element name="leftFaceId" nillable="true" type="xsd:decimal" />
    <xsd:element name="coordinates" nillable="true" type="xsd:string" />
  </xsd:sequence>
</xsd:extension>
```

The coordinates of an edge are used to construct the geometry of an edge. The left- and right-faceID's are used to reconstruct the topological structure for the faces.

Each face in the facelist contains these attributes:

```
<xsd:extension base="tns:FaceObjectBase">
  <xsd:sequence>
    <xsd:element name="featureClassId" nillable="true" type="xsd:decimal" />
    <xsd:element name="faceId" nillable="true" type="xsd:decimal" />
  </xsd:sequence>
</xsd:extension>
```

The geometry for a face can be constructed by collecting the edges belonging to that face (left or right) and chaining the edges in the correct order and in the correct direction based on the start- and end-nodes of the edges for that face.

Faces can be classified based on featureClassID. During rendering in the viewer the fillcolor for a face can be assigned based on featureClassID.

2.2 GetProgressiveTGapData

Besides the GetTGapData request the WP4 service also supports a GetProgressiveTGapData request. This request does not respond by sending all the edges and faces in a given extent but only those edges and faces that have changed compared to the previous extent. The idea behind this is to reduce the amount of data that is transferred over the internet for each response. The response consists of a number of "transactions". Each transaction consists of a set of changes in edges and faces. See the following descriptions for more details.

```
<xsd:element name="getProgressiveTgapDataElement">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="tgapName" nillable="true" type="xsd:string" />
      <xsd:element name="windowCurrentXmin" nillable="true" type="xsd:decimal" />
      <xsd:element name="windowCurrentYmin" nillable="true" type="xsd:decimal" />
      <xsd:element name="windowCurrentXmax" nillable="true" type="xsd:decimal" />
      <xsd:element name="windowCurrentYmax" nillable="true" type="xsd:decimal" />
      <xsd:element name="windowNewXmin" nillable="true" type="xsd:decimal" />
      <xsd:element name="windowNewYmin" nillable="true" type="xsd:decimal" />
      <xsd:element name="windowNewXmax" nillable="true" type="xsd:decimal" />
      <xsd:element name="windowNewYmax" nillable="true" type="xsd:decimal" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

A GetProgressiveTGapData request needs these arguments:

- The desired tgapname from which to extract the tgap objects
- The current (old) map extent and the new map extent. The tgap request will determine the differences between the two extents and send those changes back in the response.

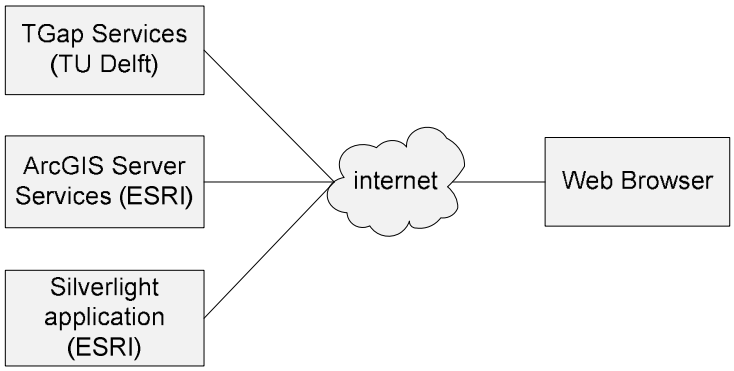
The response consists of a list with transactions. Each transaction contains the following:

```
<xsd:extension base="tns:FaceUpdateBase">
  <xsd:sequence>
    <xsd:element name="transactionId" nillable="true" type="xsd:decimal" />
    <xsd:element name="edges" nillable="true" type="tns:EdgeList" />
    <xsd:element name="faces" nillable="true" type="tns:FaceList" />
  </xsd:sequence>
</xsd:extension>
```

The lists with edges and faces are identical to the lists with edges and faces described earlier with the GetTGapData response. The logic, needed to reconstruct the face's geometries, is therefore identical to the logic that was used to reconstruct the geometries from the GetTGapData response.

3. Client implementation

3.1 High level design



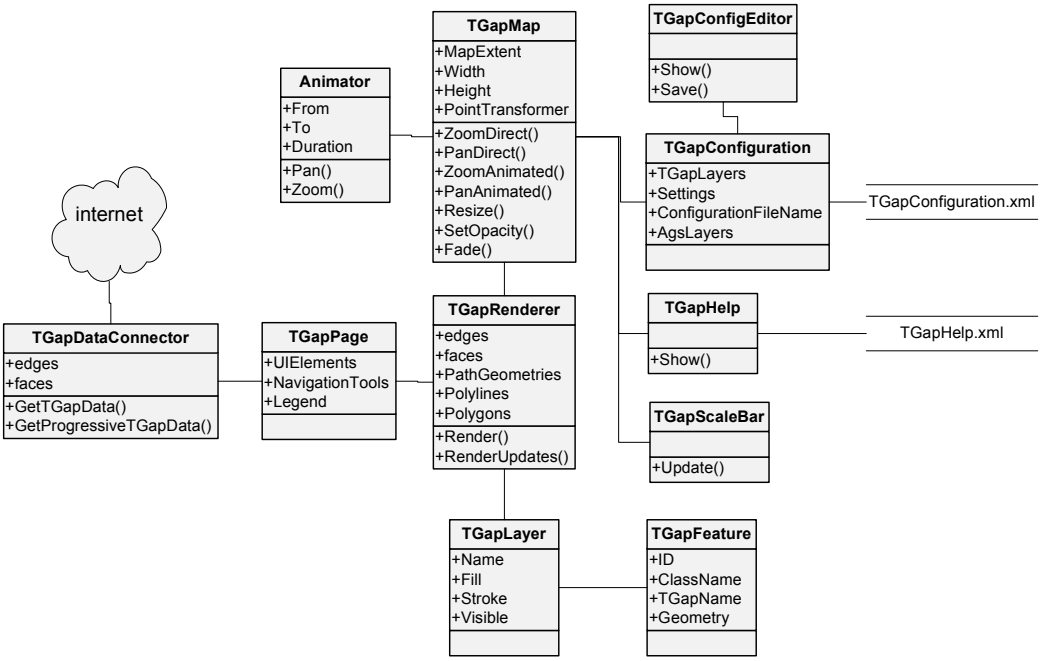
The TGap service(s) are published by OTB at TU Delft. A Silverlight application is published by ESRI NL. The application is accessed through a URL which can be requested from any web browser. The application is then shipped to the client, inserted in the browser and starts running. The first time the application is used, the browser will install the Silverlight plugin. After the plugin is installed, the first thing the application will do is to load a configuration file that is shipped with the application. The configuration file contains all sorts of settings for display options, the URL's for the TGap service and optionally URL's for additional map services from a ArcGIS Server at ESRI NL.

The big advantage of an architecture like this is the easy way for deploying the application and keeping the application uptodate. Each time the user references the application's URL, the whole application will be shipped to the client's web browser. This shipping mechanism is very efficient. The application is packaged in a zip file (TGap_Silverlight_Map.xap). So, if any change is made to the application on the ESRI server, the client browser will automatically use that change when connecting to the application's URL the next time.

The application can be accessed here:
http://arcgis93.esri.nl/tgap_silverlight_mapweb/TGap_SilverLight_MapTestPage.aspx

For a description of what Silverlight exactly is and how it works, see here:
http://msdn.microsoft.com/en-us/library/bb404700(VS.95).aspx

3.2 Detail design



Silverlight runs in a mini version of .NET. The most obvious way for implementing the code logic is to make use of the C# programming language. The above diagram shows a number of functional classes that were implemented with C#.

3.2.1 Map and rendering

The central part of the application is the web page class called TGapPage. This class contains the user interface elements and basic logic to do web requests through the TGapDataConnector class. All activity is triggered from the TGapMap class. This class inherits from a Silverlight Canvas class. The geometries for the edges and faces are drawn on this canvas.

The user triggers events from the TGapMap class. For example from zoom or pan actions. These zoom and pan events trigger the TGapRenderer to request new data from the TGapDataConnector. This class does the actual SOAP requests to the TGap service at OTB in TU Delft. The results from the TGapDataConnector are consumed by the TGapRenderer. This class creates geometries for the edges and faces from the topological structures that came from the TGap service. The actual creation of geometry for one edge or one face is done in a helper class TGapFeature.

During the rendering the geometries are given drawing attributes like FillColor, StrokeColor, StrokeWidth and Transparency. The values for these attributes are derived from the TGapLayer class. At the start of the application TGapLayer objects are created for each layer described in the TGapConfiguration file. A more detailed description of the configuration contents will follow later.

As soon as the TGapRenderer has finished with creating the correct geometries for edges and faces the TGapMap takes over and starts to refresh. The actual view for the user consists of two TGapMap objects. There is a map1 and a map2. Map1 is the map that the user normally sees and performs navigation on. Map2 normally lies behind map1 and is invisible.

When the user pans in map1, the map1 is translated along the panned distance in an animated way. The animation keeps going during the pan movement until the mousecursor is released. Releasing the mouse cursor or just clicking one of the pan buttons is the trigger to start a full map refresh.

All animations in the application are done by a generic Animator class. This class is not only used for doing an animated pan or zoom in the map but is also used with userinterface elements such as a menu that can hide/unhide when the mouse cursor hovers over a certain area in the map.

3.2.2 Map refresh cycle

The same concept is valid for zooming. There are several different ways to perform zooming in the map.

- Click on one of the zoom buttons. The map will zoom instantly in an animated way according to a fixed zoom factor.
- Roll the mousewheel. Each mousewheel click will generate an animated zoom according to a fixed zoom factor. There is a delay after each mousewheel click before the map will start a refresh sequence. This is to prevent that a complete map refresh cycle will start after each mousewheel click. The delay time is specified in the configuration file along with other pan/zoom factors.

As soon as map1 needs a refresh (as a result from panning, zooming or switching layer visibility on/off) a socalled refresh cycle is started. At the start of the refresh cycle all geometries for edges and faces are available in the TGapRenderer. A refresh cycle consists of these steps:

- Map1 is visible and shows the panned or zoomed result. This is just a translated (after pan) or scaled (after zoom) version of map1. The map needs to be refreshed. But in an animated way so that the new situation will gradually blend with the old situation and at the end of the animation will show the final result as received from the renderer.
- Clear all contents from map2. Note: map2 is currently invisible so the user will not notice that.
- Add new contents from the renderer to map2.
- Start an animation where map1 will slowly fade out and map2 will slowly fade in. At the end of this animation map1 will be invisible and map2 with the new contents will be visible.
- The end of the previous animation triggers the following steps:
    - Clear all contents from map1 (which is currently invisible)
    - Add new contents from the renderer to map1. The contents of map1 and map2 are now identical. Only map1 is still invisible.

- Instantly change the visibility of the maps: make map1 visible and make map2 invisible. This is not noticed by the user because the contents of both maps are identical.

- This ends the map refresh cycle. Map2 is invisible and map1 is visible and ready for the next user action and map refresh.

3.2.3 Configuration

The TGapConfiguration class reads a TGapConfiguration.xml file at the start of the application. The configuration file can come from two different sources:

- Application resource
  If the application is used for the first time the configuration data is read from the embedded resource inside the shipped application. This is the configuration contents as is defined by the developer of the application. The configuration is immediately stored in the socalled isolated storage of the user.

- Isolated storage
  If the application was used before or if the user has made use of the TGapConfigEditor and saved the configuration in its personal profile, the configuration will be read from the user's personal configuration file in the user's isolated storage.

Isolated storage is a secure place on the file system of the user's computer. This is the only place where a Silverligth application can acces files. For security reasons a Silverlight application is not allowed to access any other place on the file system.

This is the contents of the configuration file:

```
<ApplicationSettings>
 <!--maps the response from the esri proxy server to the correct ArcGIS server output url-->
 <PROXY_RESPONSE_MAPPING response="demo93" mapping="arcgis93.esri.nl"></PROXY_RESPONSE_MAPPING>
 <!--duration for background raster image fade in during map refresh-->
 <DURATION_IMAGE_FADE_IN>0.5</DURATION_IMAGE_FADE_IN>
 <!--the initial transparency for the tgap layers. Can be adjusted with transparency slider-->
 <LAYER_OPACITY_INITIAL>0.9</LAYER_OPACITY_INITIAL>
 <!--duration for map fade in and fade out during map refresh-->
 <DURATION_MAP_FADE_OVER>0.5</DURATION_MAP_FADE_OVER>
 <DURATION_PROGRESSIVE_MAP_FADE_OVER>0.05</DURATION_PROGRESSIVE_MAP_FADE_OVER>
 <!--duration for animations during map zoom and pan-->
 <DURATION_ZOOM_PAN_ANIMATION>0.5</DURATION_ZOOM_PAN_ANIMATION>
 <!--duration for the animated hide/unhide of the menu TOC-->
 <DURATION_HIDE_TOC>0.5</DURATION_HIDE_TOC>
 <!--the zoom factor when clicked on a zoom button or zoom key-->
 <ZOOM_FACTOR_CLICK>1.5</ZOOM_FACTOR_CLICK>
 <!--the zoom factor to apply when using the mousewheel-->
 <ZOOM_FACTOR_WHEEL>1.25</ZOOM_FACTOR_WHEEL>
 <!--the pan factor to apply when clicked on a pan button or pan key-->
 <PAN_FACTOR_CLICK>0.5</PAN_FACTOR_CLICK>
 <!--the delay before map is refreshed after resize of the parent screen-->
 <WAIT_TIME_AFTER_RESIZE>1.0</WAIT_TIME_AFTER_RESIZE>
 <!--the delay befaore map is refreshed after using the mousewheel for zooming-->
 <WAIT_TIME_AFTER_MOUSEWHEEL_ZOOM>0.75</WAIT_TIME_AFTER_MOUSEWHEEL_ZOOM>
 <!--the initial map extent in the center of the original extent. Used for better initial performance-->
 <INITIAL_MAP_WIDTH>400</INITIAL_MAP_WIDTH>
</ApplicationSettings>
```

The element ConfigurationSettings contains pan-, zoom-factors and duration values for the various animations that are used. Each ConfigurationSettings element has a comment line in front of it with an explanation about the element's meaning.

```
<!--ArcGIS Server layers accessed through REST interface-->
<ArcGISServerLayers>
 <ArcGISServerLayer featureClassId="999001" description="Background raster" geometryType="backgroundRaster" url="http://arcgis93.esri.nl/arcgis/rest/
services/Euroseas/Euroseas01/MapServer" strokeThickness="1" strokeColor="255,0,0,0" fillColor="255,0,0,0" visible="0"/>
 <ArcGISServerLayer featureClassId="999002" description="Poi sample" geometryType="point" url="http://arcgis93.esri.nl/ArcGIS/rest/services/RGI/Poiz/
MapServer" strokeThickness="1" strokeColor="255,0,0,0" fillColor="255,255,200,200" visible="0"/>
</ArcGISServerLayers>
```

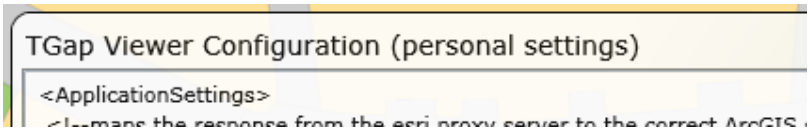
The section ArcGISServerLayers contains definitions and references to mapserver layers in an ArcGIS Server. These layers can be used as background layers or as POI layers in the TGap map.

```
<!--Color definitions format: A,R,G,B. A is transparency. Valid values between 0 and 255.-->
<TGapLayers>
 <TGapLayer featureClassId="0" description="Edges" geometryType="polyline" strokeThickness="1" strokeColor="255,0,0,0" fillColor="50,0,0,0"
visible="0"/>
 <TGapLayer featureClassId="1000" description="Built-up area" geometryType="polygon" strokeThickness="1" strokeColor="50,100,100,100"
fillColor="255,255,222,138" visible="1"/>
 <TGapLayer featureClassId="2000" description="Road" geometryType="polygon" strokeThickness="1" strokeColor="50,100,100,100"
fillColor="255,200,200,200" visible="1"/>
 <TGapLayer featureClassId="4000" description="Railroad" geometryType="polygon" strokeThickness="1" strokeColor="50,100,100,100"
fillColor="255,100,100,100" visible="1"/>
 <TGapLayer featureClassId="5000" description="Terrain" geometryType="polygon" strokeThickness="1" strokeColor="50,100,100,100"
fillColor="255,150,200,100" visible="1"/>
 <TGapLayer featureClassId="6000" description="Water" geometryType="polygon" strokeThickness="0" strokeColor="50,100,100,100"
fillColor="255,150,200,255" visible="1"/>
 <TGapLayer featureClassId="-9999" description="Unknown" geometryType="polygon" strokeThickness="1" strokeColor="50,100,100,100"
fillColor="255,255,100,80" visible="1"/>
```

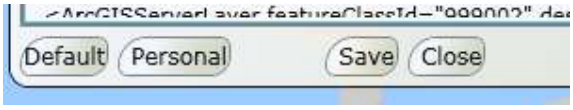
The section with TGapLayers defines attributes for each TGap layer. For example: layer name and color values.

3.2.4 TGapConfigEditor

The user can access the configuration data via an editor screen. Activate the editor via the menu botton "Configuration".



Initially the TGapConfigEditor starts up with the configuration contents from the user's personal settings (from the user's isolated storage). The title in the top frame indicates that.



The user can make changes to the configuration by just editing the values in the editor's screen. It is a very simple editor, there is only very simple checking performed during editing. So, the user must be very careful with editing and needs basic understanding of valid XML structures.

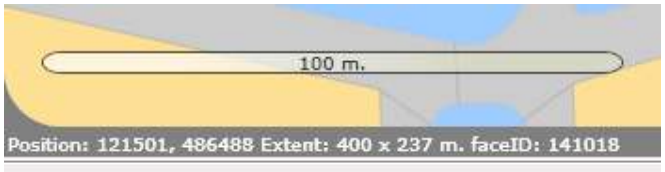
Do NOT change element or attribute names, only edit the element and attribute values.

Changes can be saved by clicking the Save button. The configuration is then saved in the personal isolated storage.

If for some reason the user is not happy with the changes made after saving, he can still recall the original configuration settings by clicking the Default button.

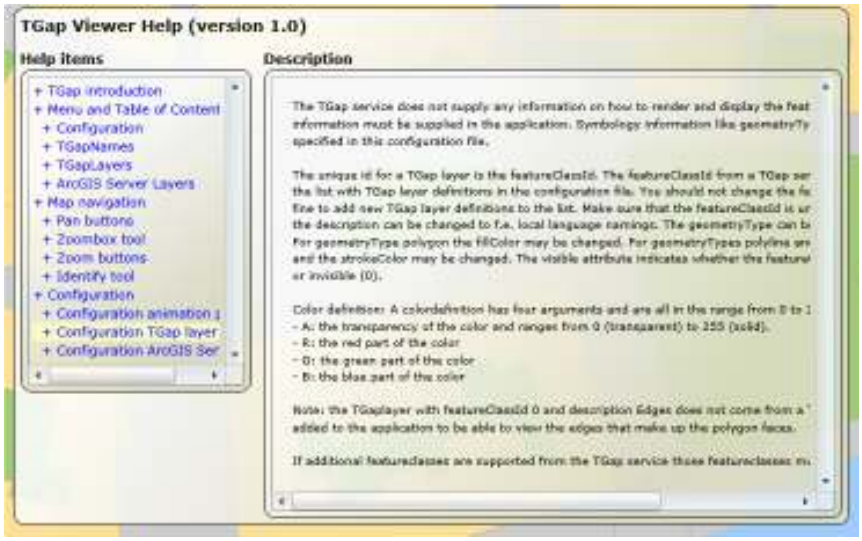
3.2.5 Map element classes

The map navigation tools are on a fixed position on the upper left part of the map. The tool buttons have a tooltip that gets visible when hovering over a tool with the mouse cursor. When hovering over the menu button a menu screen will show.



The TGapScaleBar class shows itself on the map in the lower left part. The scalebar is automatically adjusted after each map refresh. A status bar in the lower left part, just under the map frame, shows the current cursor position in RD coordinates, the current map extent in meters and optionally the ID of the face (polygon) under the current cursor position.

From the menu screen you can activate a Help screen that gives compact help instructions for all main functions.



When hovering with the mouse cursor over a help item in the left pane, the corresponding help description is shown in the right pane. The help screen closes when the mouse cursor leaves the help screen.

3.2.6 GetProgressiveTGapData

As explained before, this request returns a set of transactions where each transaction contains a set of changed edges and faces. In each transaction's list with changes some edges and/or faces will be indicated as "to be removed" while others are indicated as "to be added". The edges and/or faces to be removed have a negative ID. The GetProgressiveTGapData request needs two geographic extents as input arguments:

- The old extent. As it was during the last GetTGapData or GetProgressiveTGapData query.
- The new extent. As it is after a user has panned or zoomed the map.

The idea behind the GetProgressiveTGapData request is to send only the changes between the two extent (scale) levels back to the client. The client should start refreshing the map as soon as the first transaction with changes has arrived at the client side. That means: apply the contents of the first transaction to the contents of the map. After that the next transactions should be applied one by one.

The theory (of OTB) is that by applying the changes in this way it will give the user a nicer experience during a map refresh. Instead of seeing all the changes applied at once it would be nicer to see the changes applied one by one.

For the user the total wait time will not change much. After all, for both methods (GetTGapData and GetProgressiveTGapData) the total amount of data sent over the internet should be more or less the same.

The wait time for the user is caused by the data transfer over the internet and not by the rendering process at the client side. The size of the TGap service responses is relatively big.

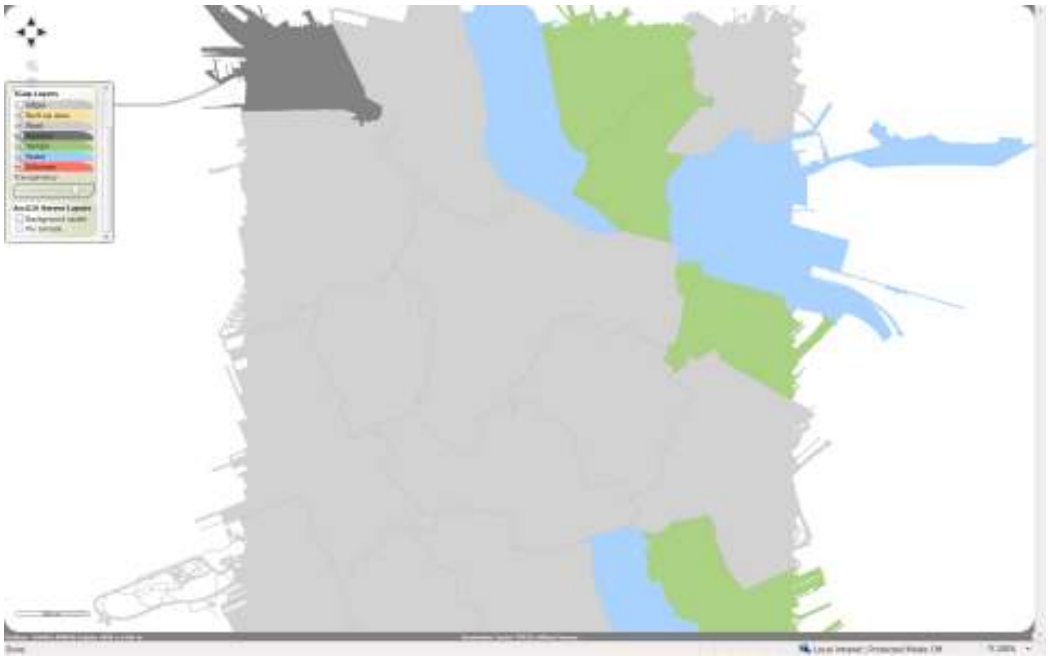
Since there is no coordinate weeding taking place at the TGap server, a large extent response is actually bigger then a small extent response while the total amount of TGap objects in the response stays the same.

The first tests with the GetProgressiveTGapData request were not very promising. First of all it appeared that a progressive response after zooming in on the map was actually larger than a normal response. Meaning that a progressive request takes longer than a normal request. Secondly the contents of the transaction updates don't seem to be correct. If the edges and faces that are marked as "to be removed" are actually removed from the map there will be some white spots on the map after all the transactions are applied. This means that not all "removed" faces are replaced with new "added" faces in the transaction updates. It seems the contents of the transactions is not correct.

Since the GetProgressiveTGapData mode does not work properly it is not enabled by default. To use this mode you have to explicitly activate this mode via a checkbox button in the menu.

After having checked this button, all subsequent map refreshes will be handled by the GetProgressiveTGapData request.

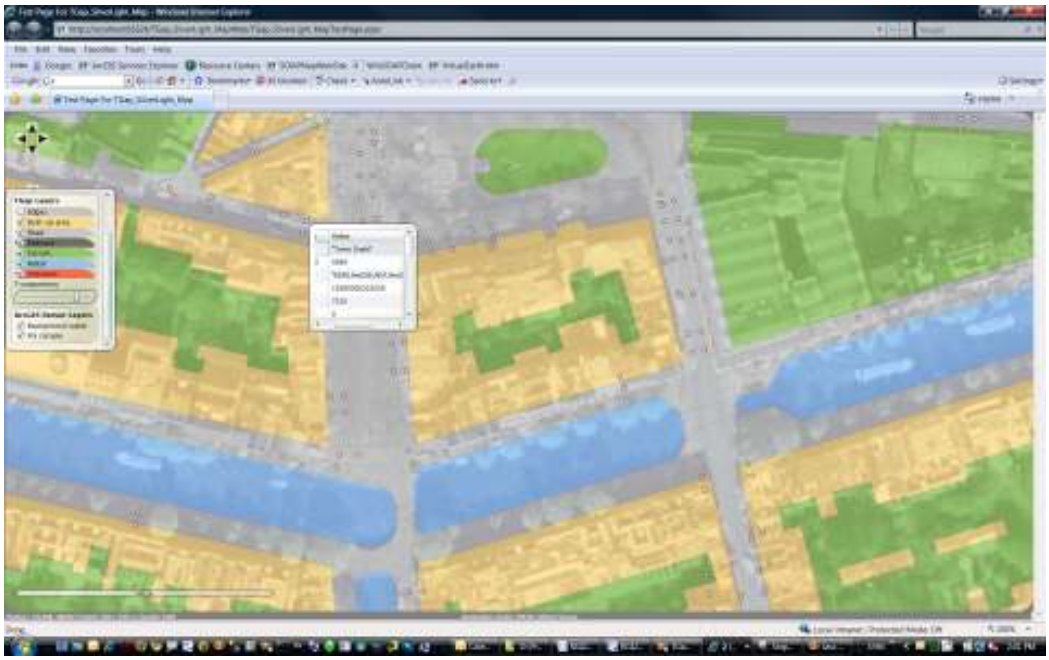
4. Screenshots



Zoomed out to the maximum extent of a TGap dataset. In this case the centre of Amsterdam. Only a few TGap objects are shown in the map. Due to the small scale many objects are unioned to larger objects. The legend explains the colors in the map.

Zoomed in to an extent of about 1000 meter in the centre of the map. The objects in the map show more detail then in the full extent map. Some of the map area is now classified as build-up area.

Zoomed in to an extent of 400 meter shows clearly how the details in the map have changed compared to the previous extent.



The two ArcGIS Server layers (defined in the configuration) are now visible. The areal picture serves as background image. Together with the background image it becomes clear that the green faces are the inner gardens within the building blocks.

The points layer represents points of interest. That layer is rendered on top of the background and TGap layers. The attributes for a poi become visible when hovering the mouse cursor over a poi in the map. The symbology of the pois in this case is very simple. The symbology could be customized by defining such in the configuration file.

5. Development tools

For the development of the client application, the following tools have been used:

- Development environment: Microsoft Visual Studio 2008
- Development language: C#
- Web development: Microsoft Silverlight 2.0
- ESRI ArcGIS Server map services through the REST endpoint. These are only used to add the background- and poi-layer to the map.

6. Conclusions

6.1 Performance GetTGapData

A GetTGapData response for a full extent request on the dataset with the centre of Amsterdam is 630KB and takes 3 seconds to transfer over the internet from the server to the client. When zoomed in to an extent of 500 meter the response is 86KB and takes 0.3 seconds to transfer. The total number of TGap objects in both cases is about the same. It shows that the user experience, a wait time of 3 seconds, can be greatly improved by applying a simple form of generalization (coordinate weeding) at the TGap server side. This would improve the data transfer time dramatically.

6.2 Performance GetProgressiveTGapData

Updating a map with the GetProgressiveTGapData request mode after having displayed the map initially using the default GetTGapData request mode, it appears that the second mode produces a larger response than the first mode. In other words, updating the map with progressive updates takes more time than creating the map initially. For example: the initial map with an extent of 1200 meters was created from a response with a size of 347KB. Zooming in to the map with a factor of 1.5 using the progressive update mode created a response with a size of 465KB. Maybe the idea behind the progressive update is good but the implementation of it is not good yet and therefore not suited to be tested.

6.3 Silverlight and graphics

The use of a Silverlight client proved to be successful. Especially the performance of the graphics system is acceptable. Rendering about 100 TGap objects with some 10.000 vertices and dispaying them on a canvas takes less than 0.25 seconds. The interactive behavior of the map (dynamic panning and zooming) stays very good with these amounts of graphics on the canvas.