ibl software engineering

Making the Weather *Visual*, *Flexi*ble and *Online*

Jozef Matula, jozef.matula@iblsoft.com

Overview

- Using Python for Web Services in Visual Weather
- Lessons learned about WMS and tiling and making it effective
- The status and future of IBL web related products

Visual Weather

Simple & easy

DEVELOPMENT OF PYTHON WEB SERVICES IN VISUAL WEATHER

Python Web Services in Visual Weather

- Use of Visual Weather's Python API together with scripting allow rapid development and instant deployment of web services in *mod_python* style

```
import IBL.Kernel; from IBL.Net import apache
def getSYNOPTemperature(req, station):
  e = K.Expression(
      "@station v6[5,2] Fsynop_latest$sv52$C")
  e.setArgument("station", K.mkStation(station))
  tSYNOP = e.eval()
  req.write(str(tSYNOP["temperature"].
    toValue(K.u.T_CELS)))
  return apache.OK
```

http://my-server/service/getTemperature?station=LZIB

Python WS - a real example

https://ogcie.iblsoft.com/ria/helpers/distance?places=
EPSG:4326[48;19]+EPSG:4326[49;20]+EPSG:4326[48;19]

```
import math
import IBL.Geo as G

def __parseCoord(s_coord):
  a = s_coord.split('[', 2)
s_crs = a[0]
s_xy = a[1].rstrip(']')
(prj, tr) = \
  G.CRSFactory.getInstance().\
     createProjectionWindow( \
        s_crs, 1, -1, 1, 1)
if prj is None:
   raise Exception( \
     "Unsupported CRS"+s_crs)
a = s_xy.split(';', 2)
return prj.xy2lp(
  tr * G.Coord(float(a[0]), float(a[1])))
```

```
def distance(req, places):
  a_points = places.split(" ")
  f_distance = 0
  lastCoord = None
for s_coord in a_points:
  if lastCoord is None:
    lastCoord = __parseCoord(s_coord)
  else:
    c = __parseCoord(s_coord)
  f_distance += \
    lastCoord.distance(c)
  lastCoord = c
  return str(f_distance)
```

Visual Weather

Scalability vs.

MAKING WMS USE EFFECTIVE

Scalability vs. effectivity

- Scalability becomes a buzzword if it means that the software scales ***only*** with your wallet
- Consider - that 1 server should “feed”
  - 10 forecasters working
  - 10 “actions” per minute

  BUT:
  - A map may have multiple layers - times 5
  - Animations - times 15

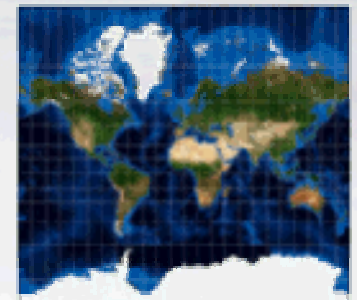
Total: 7500 req/h = 2 req/s
Expected response time 1-3s!

WMS caching?

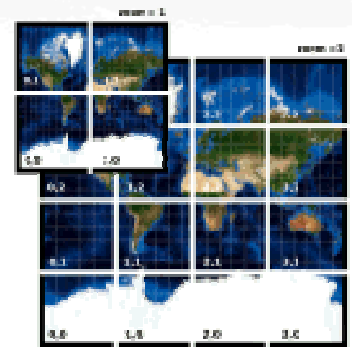
- Rendering a WMS image takes time T - assuming 200ms composed of:
  - Data retrieval
  - Data reprojecting and display
  - Rasterisation
  - Encoding final image
- Caching means, skipping all the phases and returning final image - typically 1-10ms
- Caching assumes high probability of exactly same request BUT in WMS the BBOX can have infinite number of variables (due to zooming, panning, etc.)

Image tiling in theory

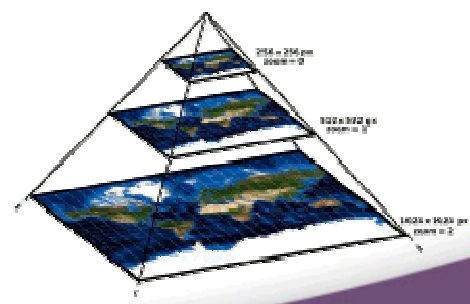
- http://www.maptiler.org/google-maps-coordinates-tile-bounds-projection
- Step 1. Define the “World extent”



- Step 2. Split the World into tiles (usually squared 256x256px)



- Step 3. Build a “pyramid” of tiles (1 tile as very top zoom level). For example split each tile into 4 sub-tiles



WMS tiling

- Why tiling?

  It's well accepted theory that tiling is the only way how to make WMS scalable.
- Tiling separates world view into finite number of "tiles".
- Various tiling WMS extension proposals...
- Finally went for yet another - for the back-end web service for the UK MetOffice Invent project - `GetGTile` request - mimics the Google Maps API-like tiling via WMS `GetMap` like interface.
- Note: Tiling requires not only CRS but also maximum extent BBOX!

Tiling - The success stories

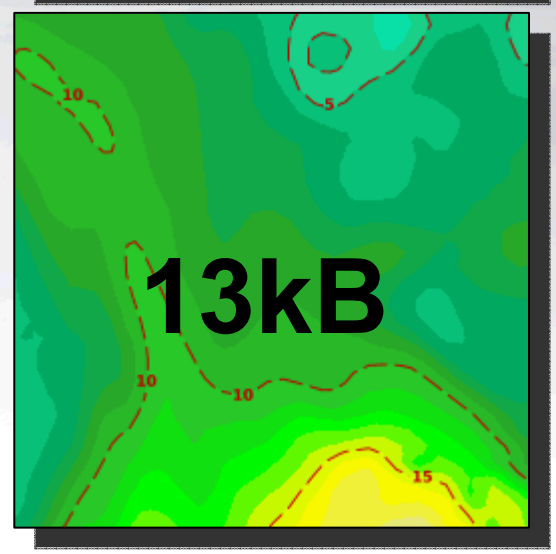
- Google Maps, MS Virtual Earth, Yahoo! Maps and others:
  - Due to stateless protocol very distributable & cache-able even in browser
- BUT: They have just 4 “solid layers”

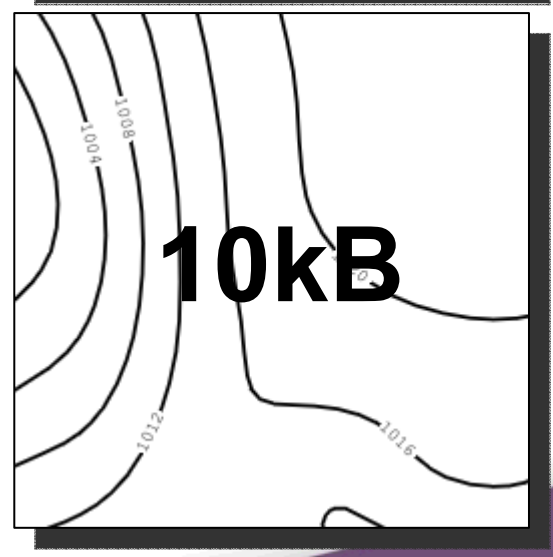
Disk space needed

- Imagine:
  - 100 layers x 20 vertical levels
  - 10 time steps
  - Average 3 styles per layer
  - Only 6 “QuadTree” zoom levels
    1+4+16+256+1024+4096 = 5397 tiles
  - Only PNGs (JPEG is not transparent)
  - Average tile size 20kB

Total 630GB

I almost forgot - each 6 hours!





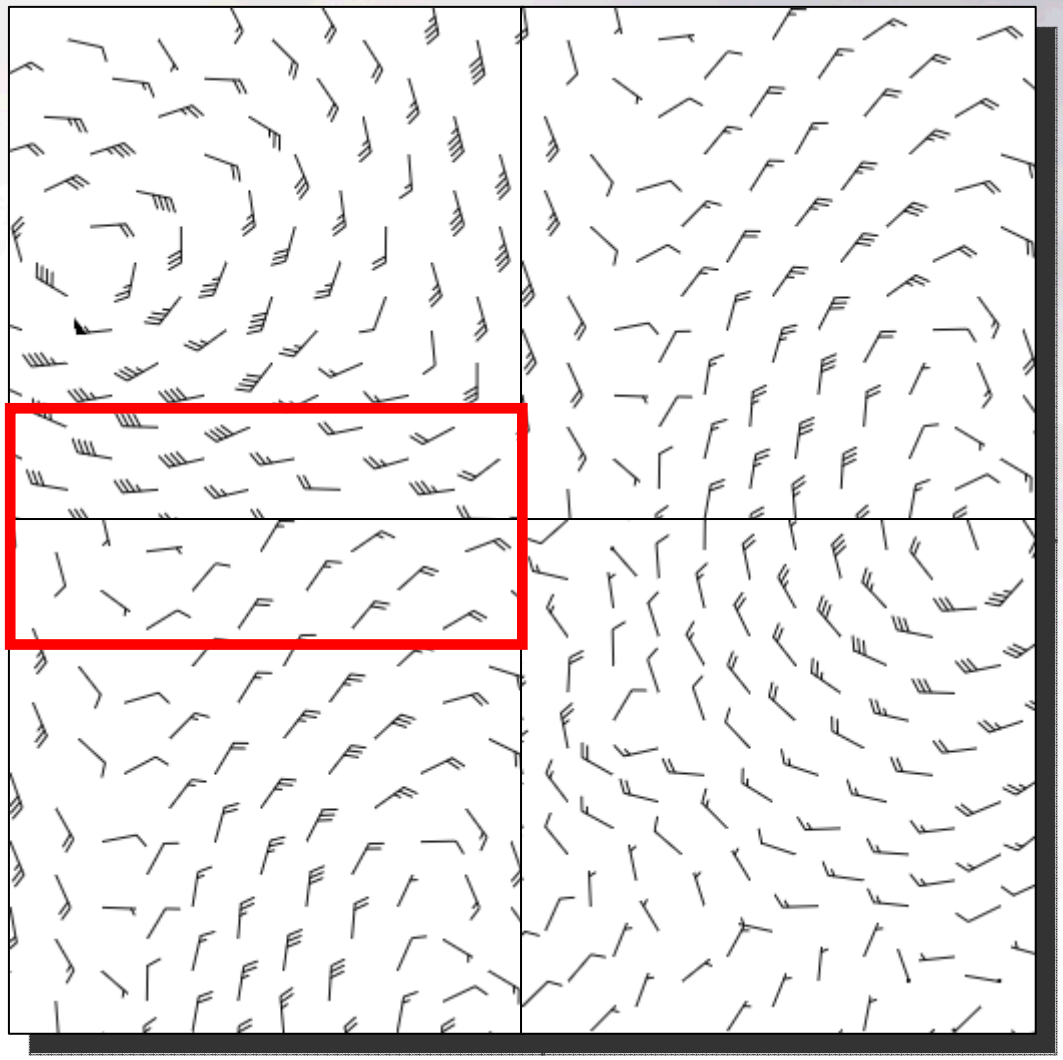
Real life

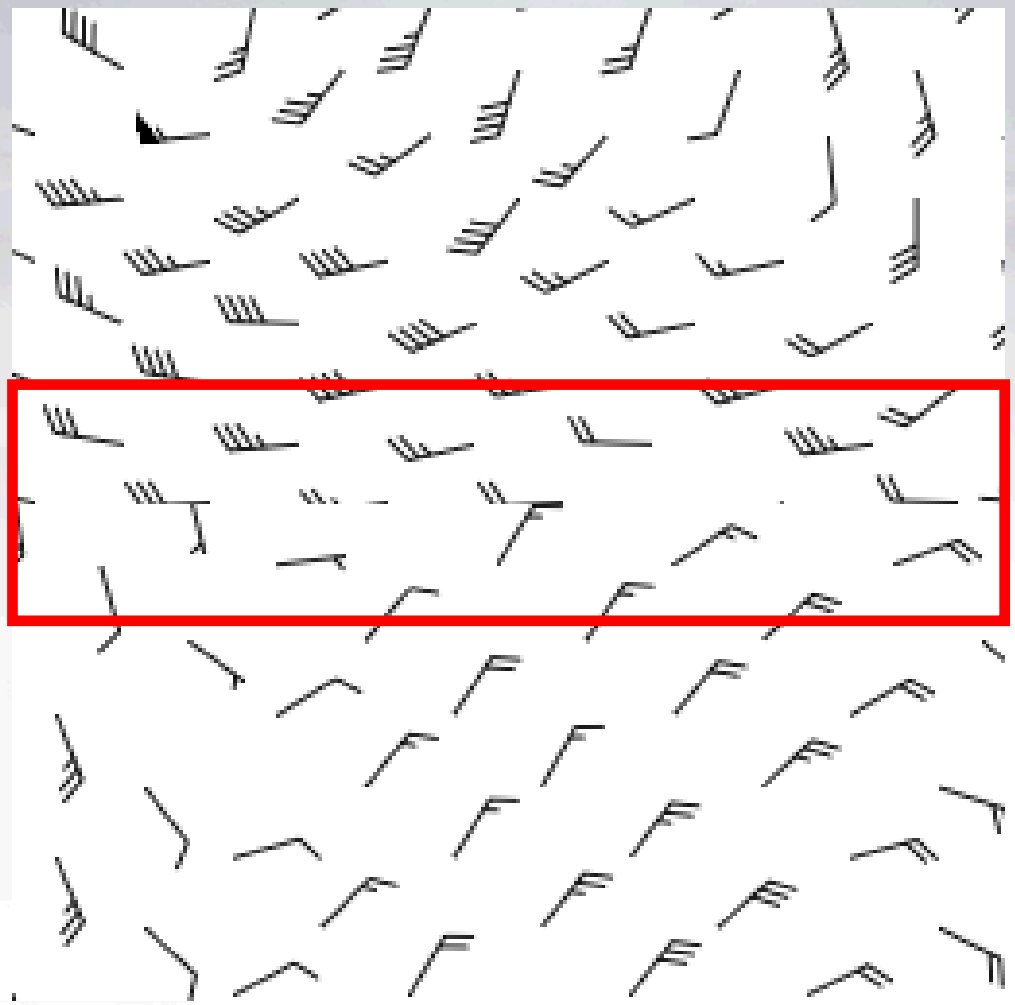
PROBLEMS WITH TILING

Major problems with tiling we faced

- Tiles have to be generated on-demand
- Problems related to tiles rendering (such as edge effects)
- Time consistency issues
- Tile invalidation

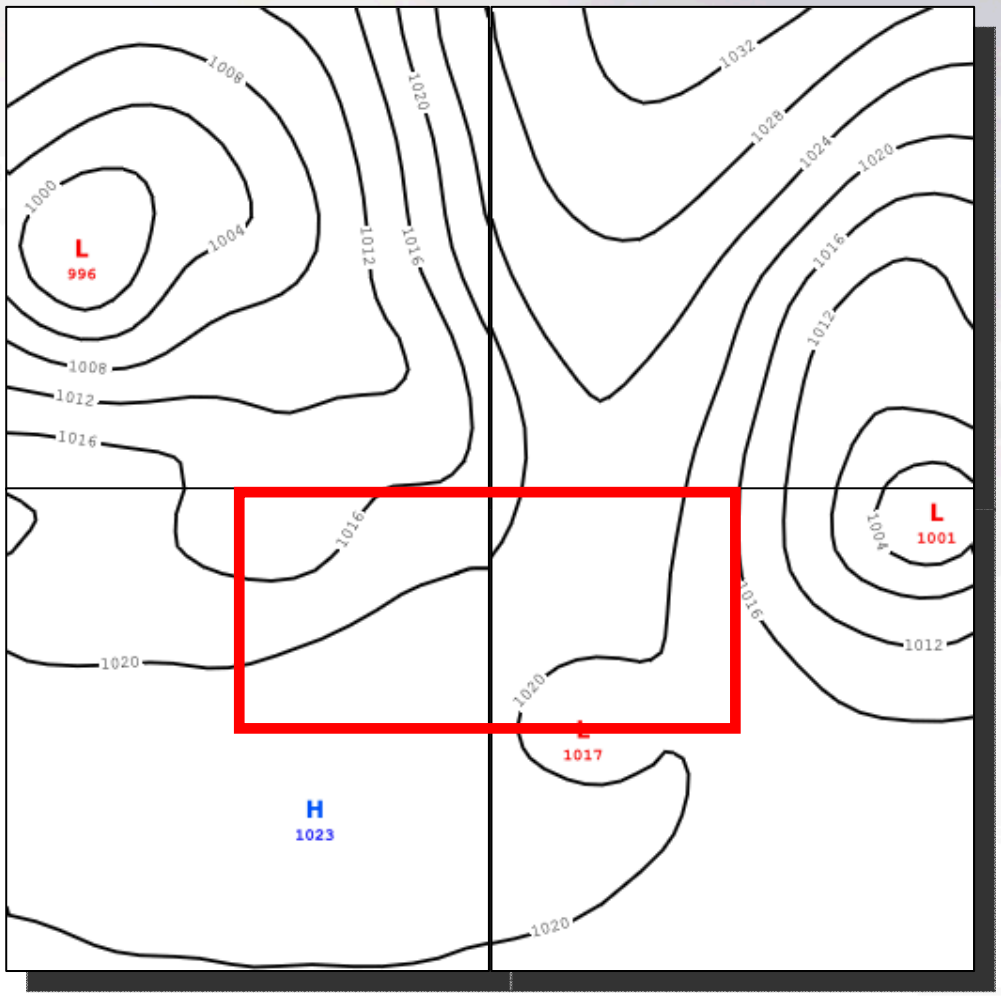
Edge effects - Symbol cutting

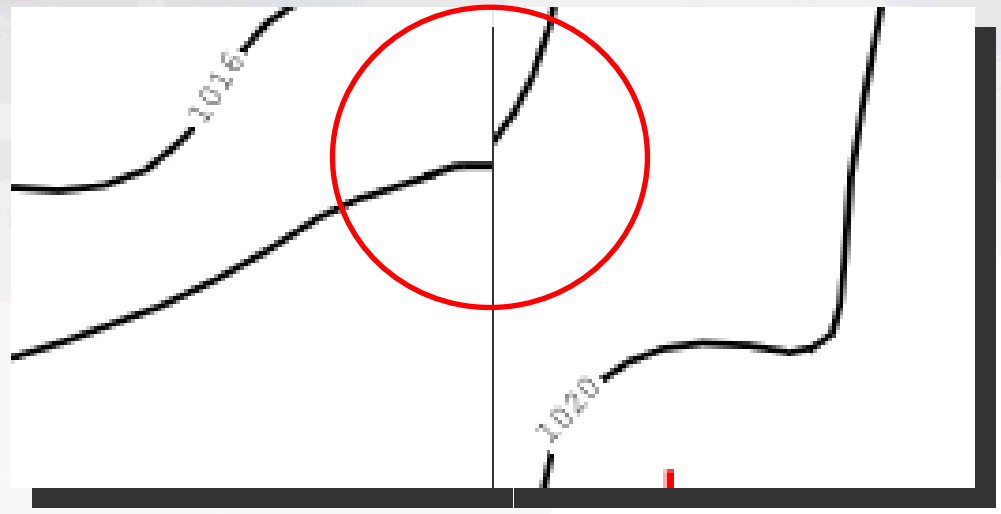




Edge effects - Non-matching of isolines

Grid of tiles is not parallel underlying data \grid





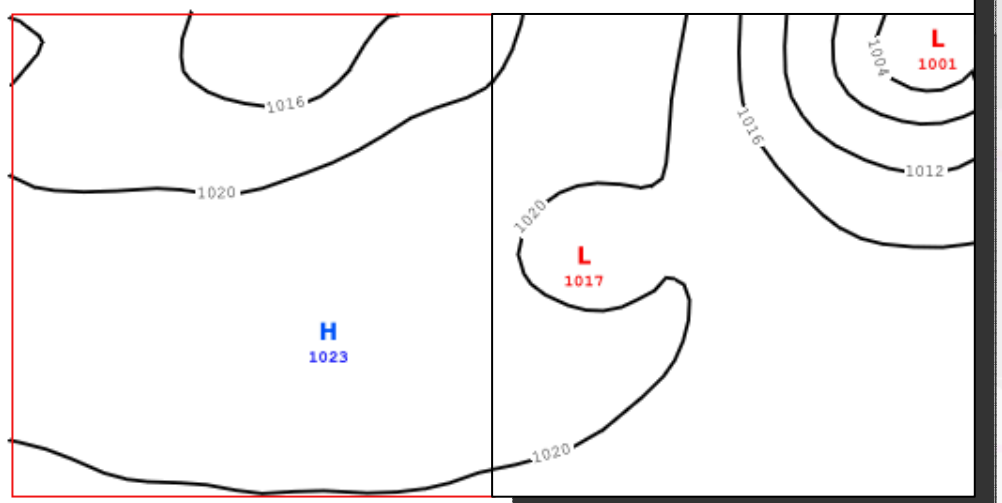
If interpolation is performed on reduced grid than at the edge the isoline is “extrapolated”.

Edge effect solutions

- Ideally we could render the image for the whole zoom level and just cut it into tiles.
- BUT we considered:
  - For higher zoom levels the whole image resolution is huge.
  - For some data types such observations - rendering of tiles is very effective because it requires only limited set of data, therefore it makes sense think of some tricks to really cheat this effects.

Solution - Isoline edge effects

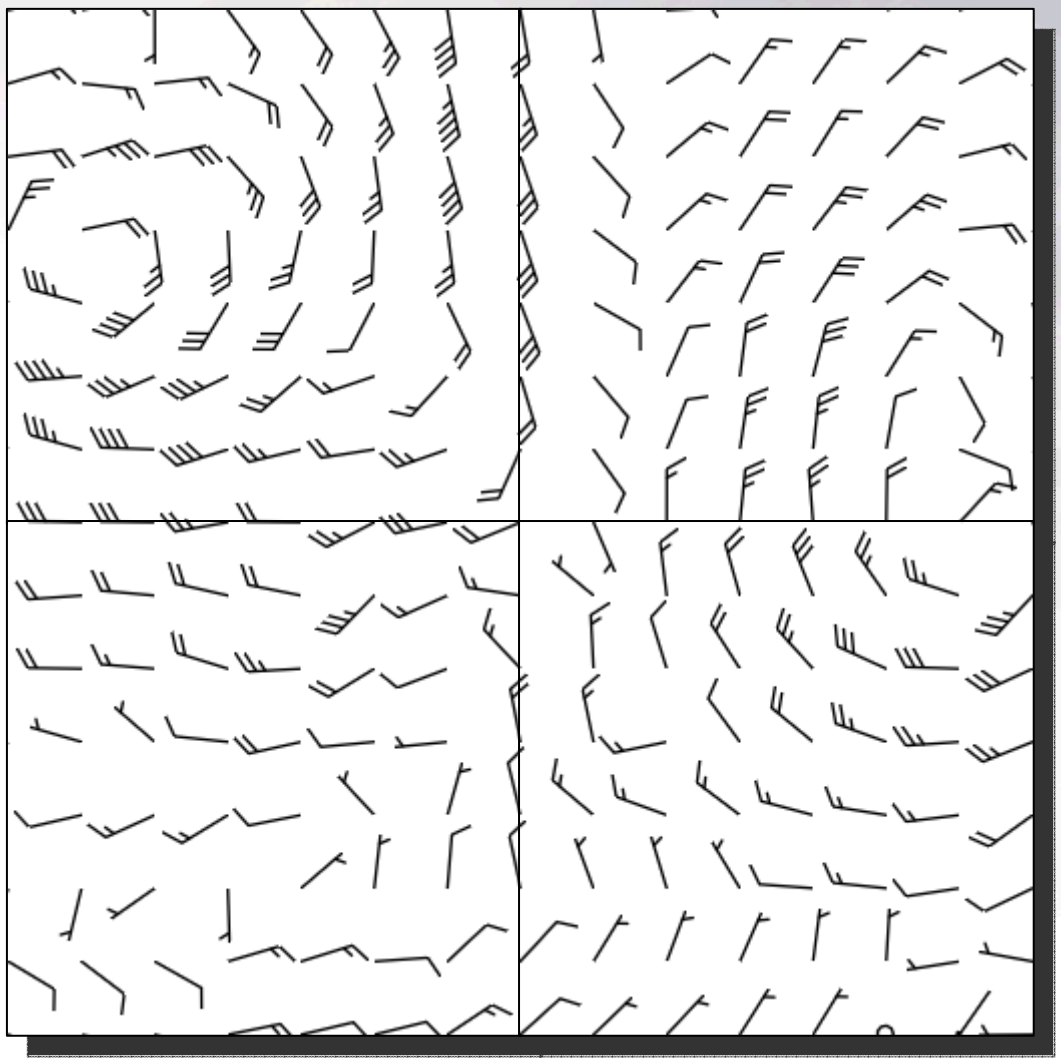
Finally cut off the rendering margin



Sure, this increases the rendering time.

Rendering the isolines with a “margin” ideally that big as the surrounding tiles.
This works well with observation decluttering too.

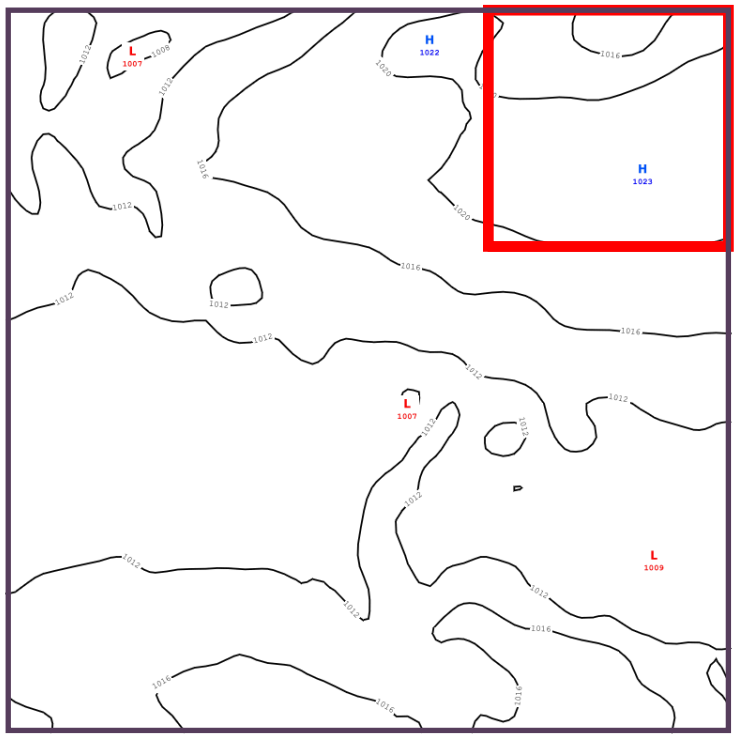
Solutions - Windbarb edge effects



1. Use rendering margin and draw “all” symbols
2. or interpolate wind into regular grid matching the reference system of tiles and trick the size of windbarbs to fit to the tile.

Multi-tile rendering

Cut off the rendering margin



Store ALL tiles in the cache

- Data retrieval and reprojection is costly.
- Once you retrieve data for the tile request it's "cheap" to render "surrounding" tiles.
- This make single tile request expensive
- This also reduces the tile edge effects

Browser caching and time consistency

- Cache in any browser is designed for “resources”

  `http://khm0.google.com/kh/v=62&x=560&y=355&z=10&s=Gal`

- To utilise this, each tile have to be turned into such resource (in the REST terminology):
  - URL expresses time-independent unique identifier This mostly implies that time related default assumptions SHOULD NOT be taken (such as best model run time or default validity time)
  - Proper HTTP headers have to used (`GET` request, `Cache-Control`)

Inter-tile time consistency

- What if the data change between rendering of tiles? (new observation data arrives, data amendments)

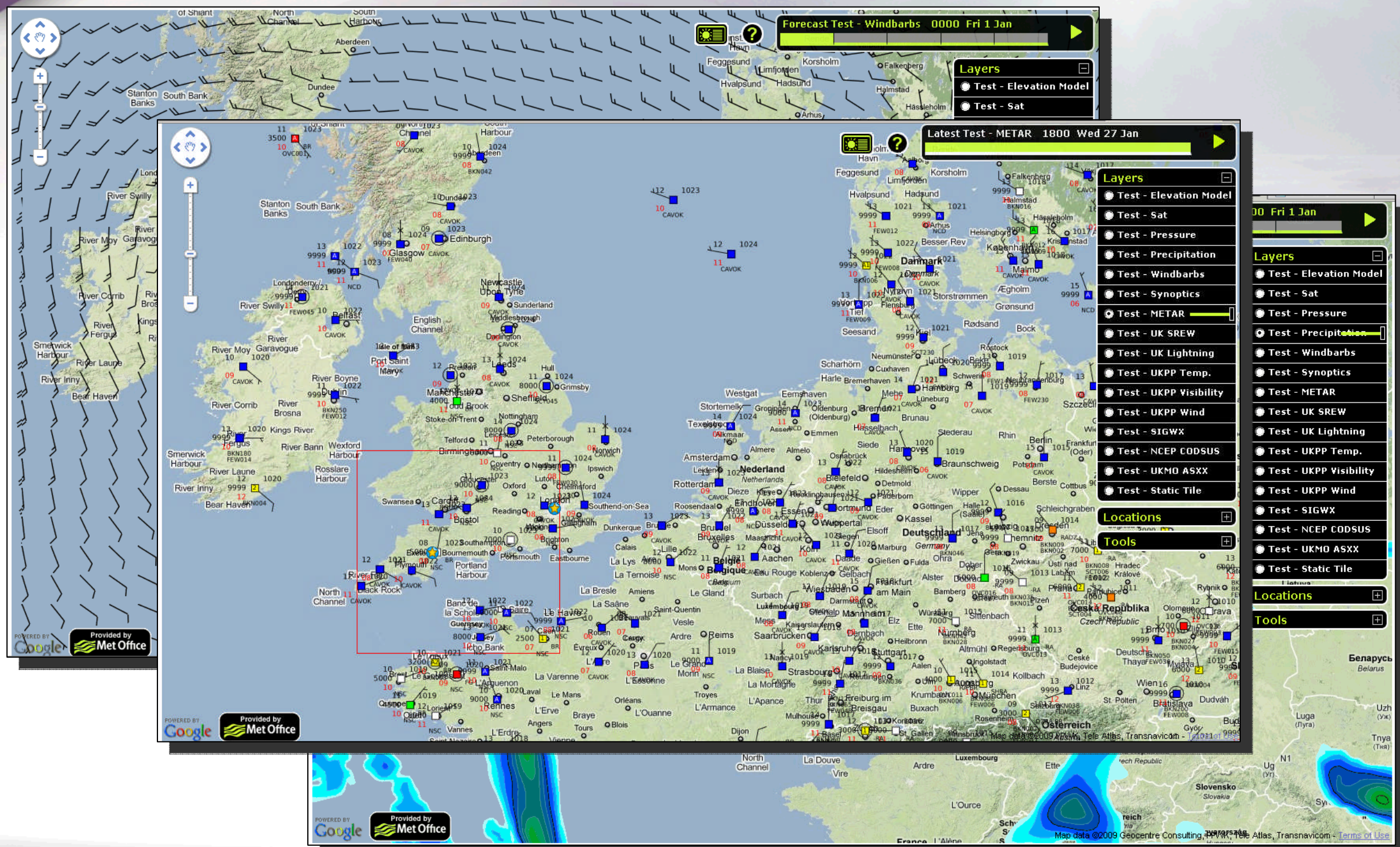
This is an issue without any *simple* solution.

- Can be solved only by introducing some state/transaction mechanism to the tile requests. e.g. rendering of observations in “time snapshots”, but this complicates the client’s access significantly.

- Then it’s much easier to use traditional WMS `GetMap` request with CPU penalty on the server side.

Visual Weather

MetOffice “Invent” Tests



Cost of a map request

- Is the tile based access to a map quicker than `GetMap` request?

It depends...

- Tiling produces lot of requests increasing high network latency.
- Tiling in longer term view reduces server CPU load significantly.
- Simple (and fast and tuned) WMS layers can provide comparable performance to tiling
  (200ms == 20 tiles * 10ms)
  This is visible in animations of simple layers such as radar.

Visual Weather

The web synergie of

VISUAL, FLEXI, ONLINE

Visual Weather OGC WS status

- WMS 1.1.0-1.3.0:
  - `GetGTile` extension

```
...&REQUEST=GetGTile&LAYER=MSGIR&
 CRS=EPSG:900913&
 TILEZOOM=4&TILEROW=5&TILECOL=7
```

  - `GetLegendGraphic`
- WCS 1.0.0 for grid data
- WFS 1.0.0-1.1.0 for any observed or derived data, ability to define custom feature types.

```
https://ogcie.iblsoft.com/obs?
    SERVICE=WFS&REQUEST=GetCapabilities
```

- WFS-T 1.0.0-1.1.0 for weather features

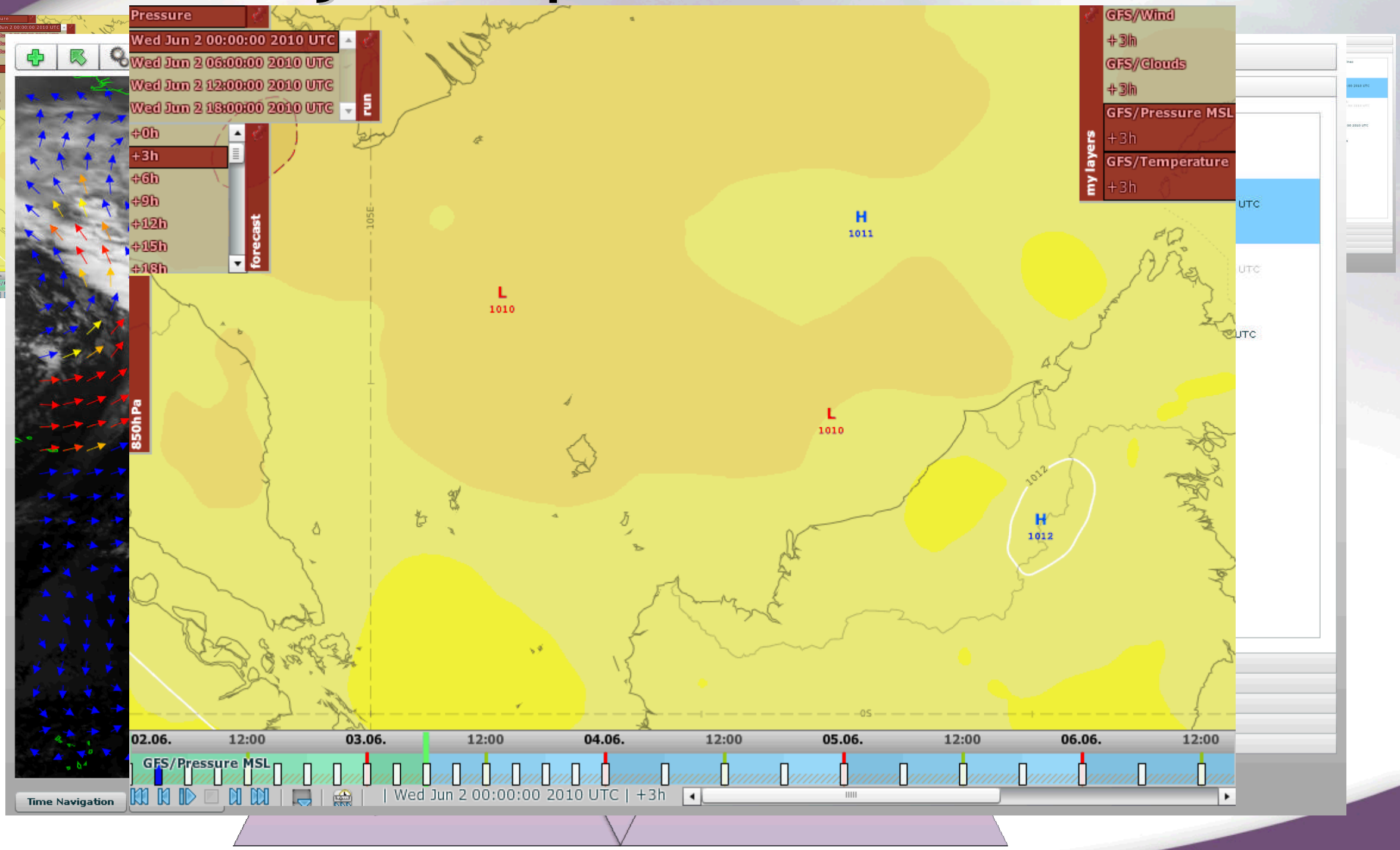
Latest TAF for EGLF via WFS

- https://ogcie.iblsoft.com/obs?SERVICE=WFS&
  VERSION=1.1.0&REQUEST=GetFeature&
  TYPENAME=TAFReportLatest&ICAO=EGLF

```
<wfs:FeatureCollection …>
  <gml:featureMember>
    <TAFReportLatest>
      <ICAO>EGLF</ICAO>
      <country>EG</country>
      <issueTime>2010-06-03T06:27:00</issueTime>
      <reportText>
EGLF 030627Z 0306/0315 09008KT 9000 NSC BECMG 0306/0309 CAVOK=
      </reportText>
      <gml:location>
        <gml:Point>
          <gml:pos srsName="EPSG:4326">-0.78000000000000236 51.280000000000001</gml:pos>
        </gml:Point>
      </gml:location>
      <valid>true</valid>
      <validFrom>2010-06-03T06:00:00</validFrom>
      <validTo>2010-06-03T15:00:00</validTo>
    </TAFReportLatest>
  </gml:featureMember>
</wfs:FeatureCollection>
```

Visual Weather

Major Components & Protocols



Key protocol decisions (taken)

- OGC WMS - to access any imagery:
  - Tiled for all “time-constant” data
  - Traditional WMS for all time-varying data
- OGS WFS - to access all “object” data (aka SQL table) - for example:
  - Weather Features
  - Observations
  - But also Station Catalogue, and may be NWP data
- Python bespoke Web Services - for anything else
- Security: HTTPS & Basic Auth
- http://code.google.com/p/flexiweather

Thank you
for your attention!

Questions?

ibl
software engineering