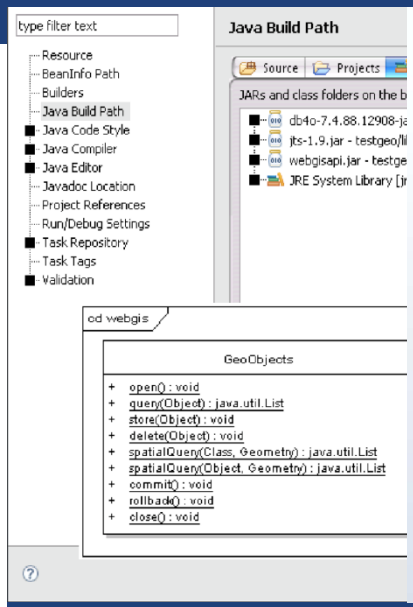
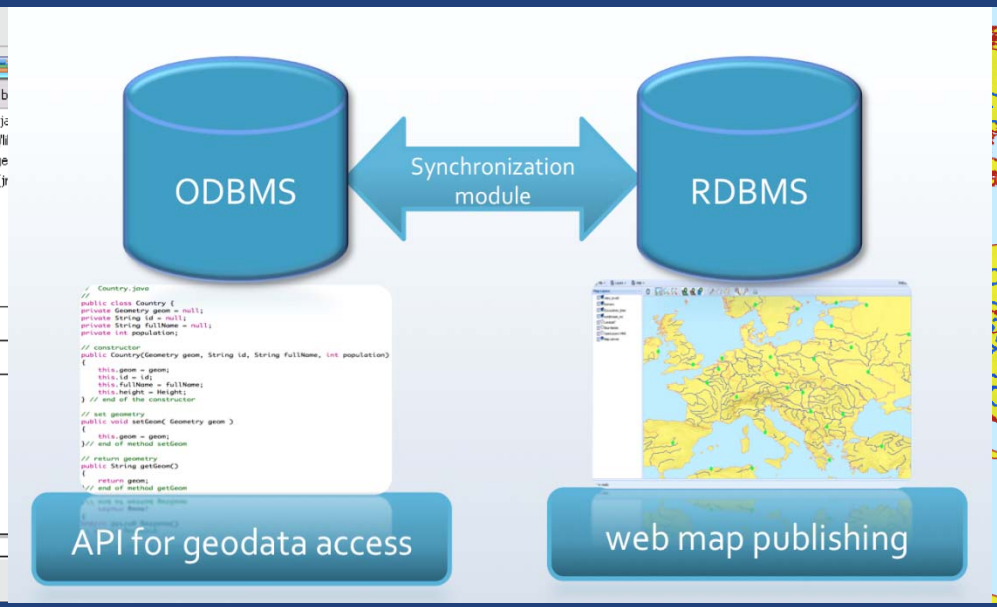
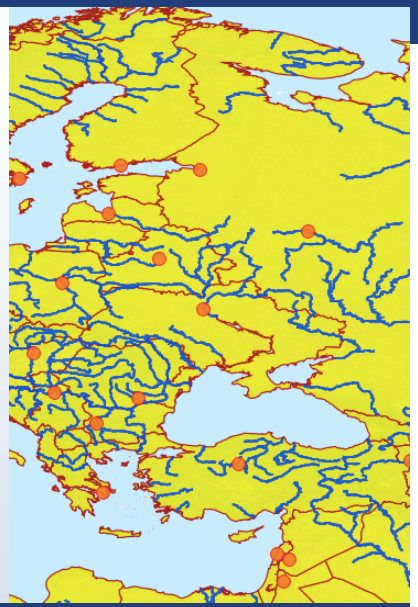






Integrated Cartographic and Programmatic Access to Spatial Data using Object-Oriented Databases in an online Web GIS

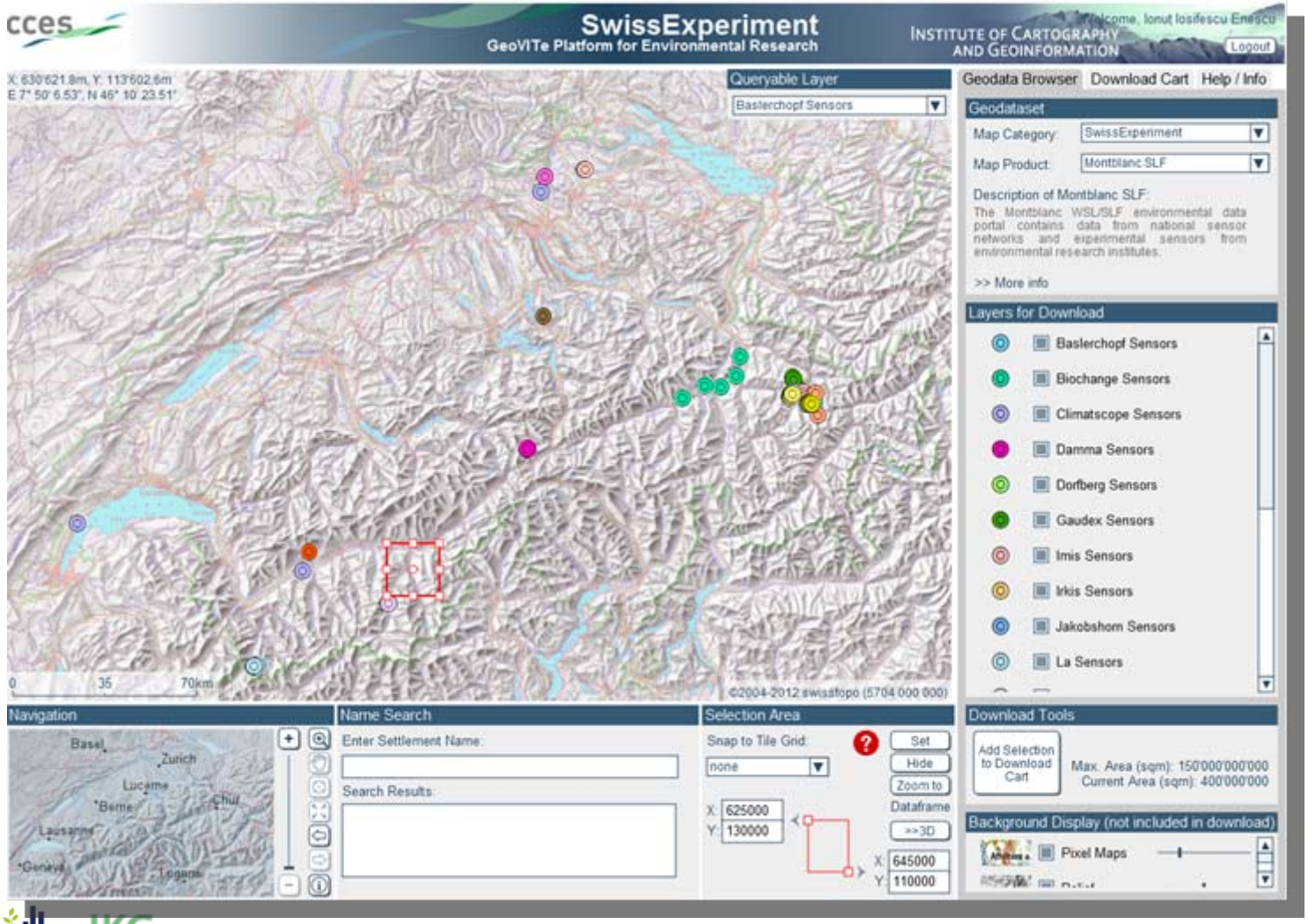
ICC Dresden 2013

I. Iosifescu & L. Hurni
Institute of Cartography and Geoinformation, ETH Zurich, Switzerland

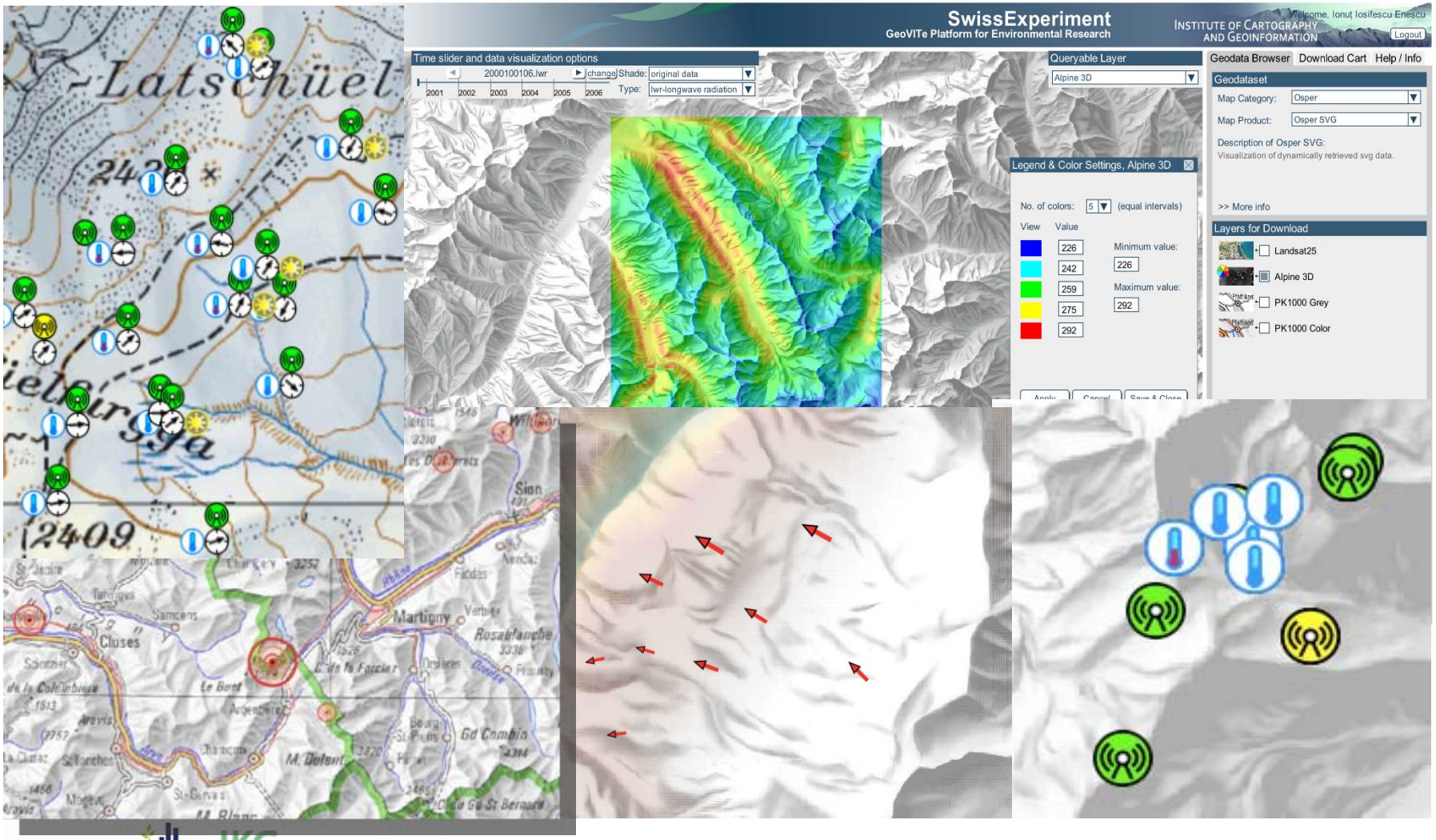
Motivation and Goal

- Modern Cartographic Systems / Web GIS imply:
  - Cartographic development AND custom software programming
  - Efficient and **programmer-friendly access and handling of spatial data can become equally important to visualization**
- Requirements from environmental sciences:
  - Projects SwissExperiment and OSPER (Open Support Platform for Environmental Research)
  - Real-time environmental monitoring through a common, modern and generic cyber-infrastructure
  - Platform for large-scale sensor network deployment and information retrieval and exploitation
  - **Service-driven Web-GIS platform** (for Environmental Research)

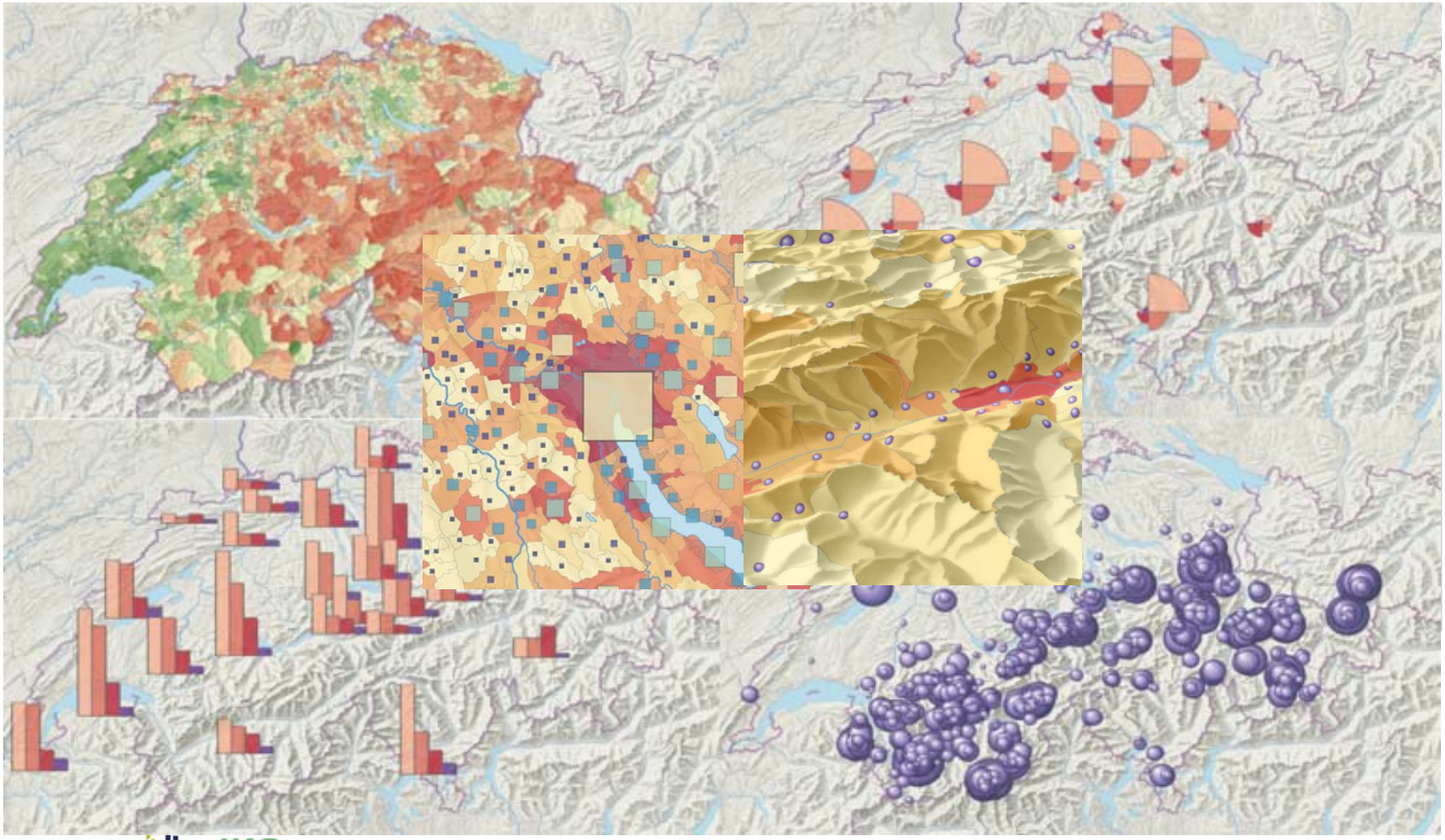
GIS Platform for Environmental Research



Cartography in the GIS Platform



Service-driven Cartography at Large



Programming for the GIS Platform

- Cartographic visualization requires:
  - Real-time processing: mainly selection/filtering and joins based on attributes and spatial location
  - Performance in processing and visualization for many sensor data
  - Communication with additional sensor middleware (Global Sensor Networks - GSN, SwissExperiment Semantic Metadata Wiki)

- And developers would like to have:
  - Displayed sensor data as objects in programming language (Java)
  - Eliminate the serialization and deserialization overhead
  - Automatic synchronization between the Java middleware (GSN and the relational PostGIS database used for visualization)

Main Programming Hurdle: Object-Relational Impedance Mismatch for Spatial Data

recurring mappings and conversions
&
absence of compile safety

Geometry Type	Text Literal Representation	Comment
Point	Point (10 10)	a Point
LineString	LineString (10 10, 20 20, 30 40)	a LineString with 3 points
Polygon	Polygon ((10 10, 10 20, 20 20, 20 15, 10 10))	
Multipoint	MultiPoint ((10 10), (20 20))	
MultiLineString	MultiLineString ((10 10, 20 20), (15 15, 30 15))	a MultiLineString with 2 linestrings
MultiPolygon	MultiPolygon (((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60)))	a MultiPolygon with 2 polygons

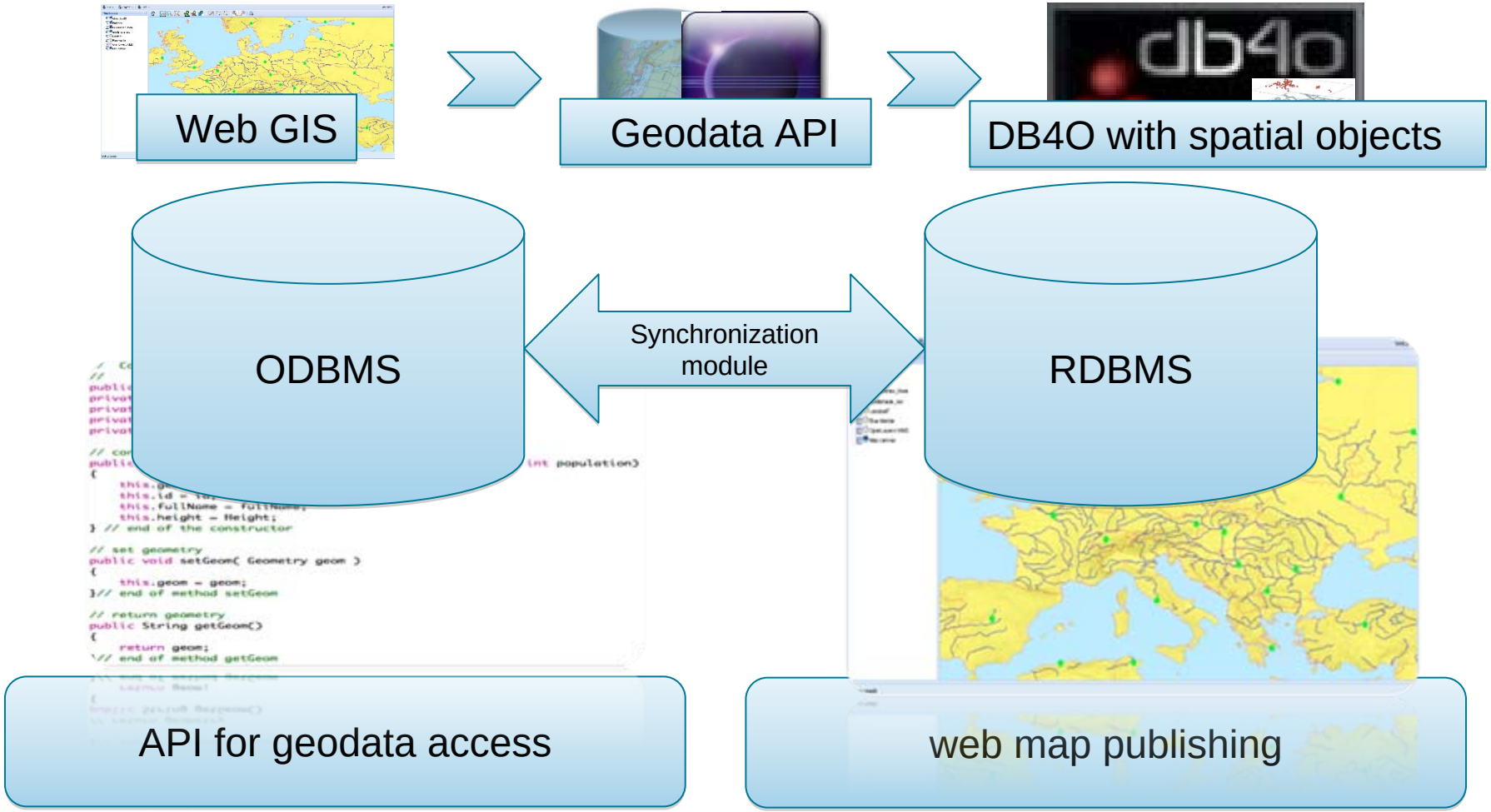
```
// connect to data store
DataStore dataStore = DataStoreFinder.getDataStore(connectParameters);
// read type names
String[] typeNames = dataStore.getTypeNames();
// declaration of generic Model
FeatureSource<SimpleFeatureType, SimpleFeature> featureSource;
FeatureCollection<SimpleFeatureType, SimpleFeature> collection;
FeatureIterator<SimpleFeature> iterator;

e.getFeatureSource(typeN
tures
ce.getFeatures();

atures();

    while (iterator.hasNext()) {
        SimpleFeature feature = iterator.next();
        // here map feature to your application dependent class
    }
}
finally {
    if( iterator != null ){
        // CLOSE THE ITERATOR!
        iterator.close();
    }
}
```

```
//  Country.java
//
public class Country {
private Geometry geom = null;
private String id = null;
private String fullName = null;
private int population;

// constructor
ring id, String fullName, int population)

// set geometry
public void setGeom( Geometry geom )
{
    this.geom = geom;
}// end of method setGeom

// return geometry
public String getGeom()
{
    return geom;
}// end of method getGeom
```

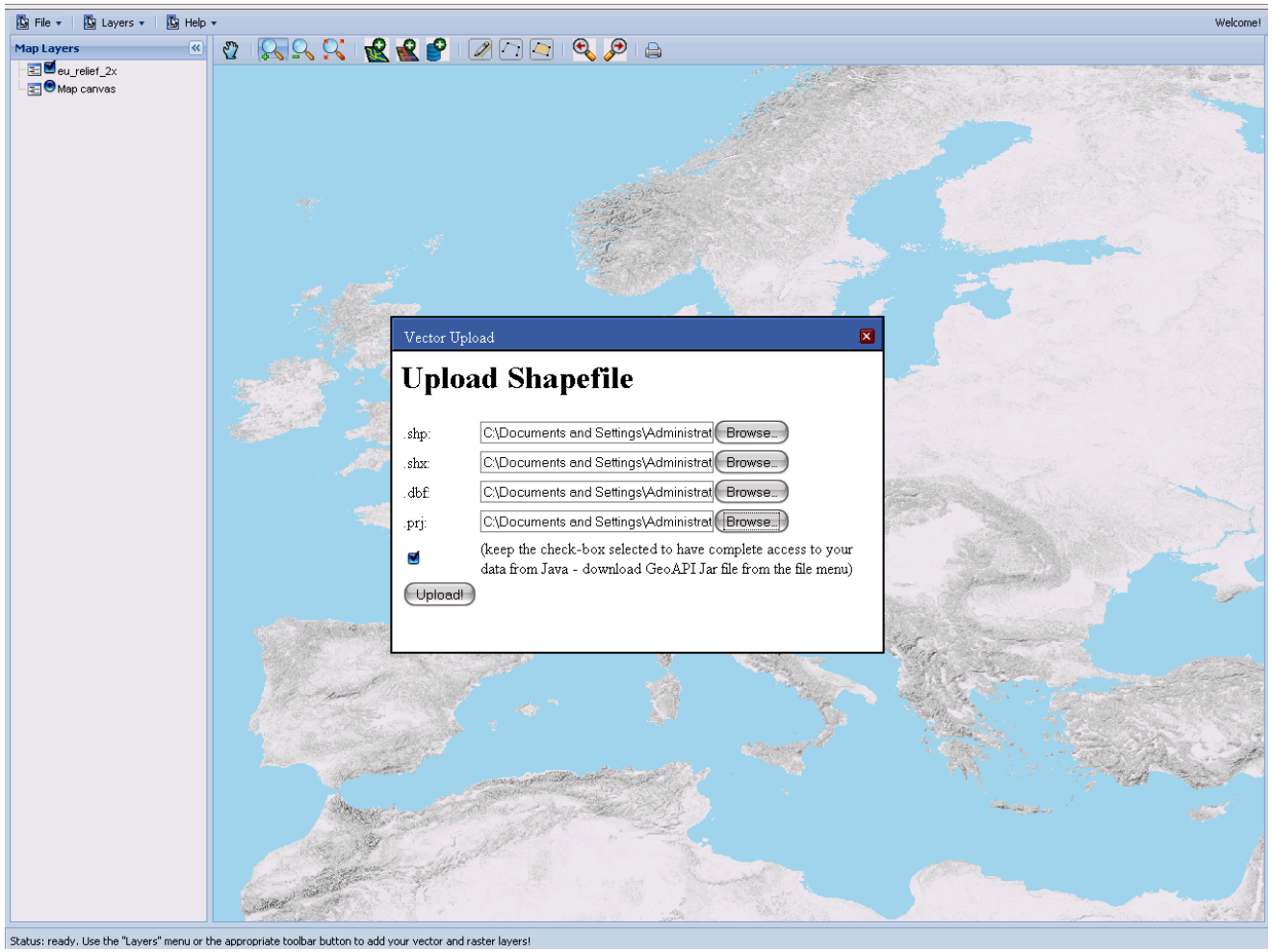
relational to generic features mapping

generic to application specific mapping

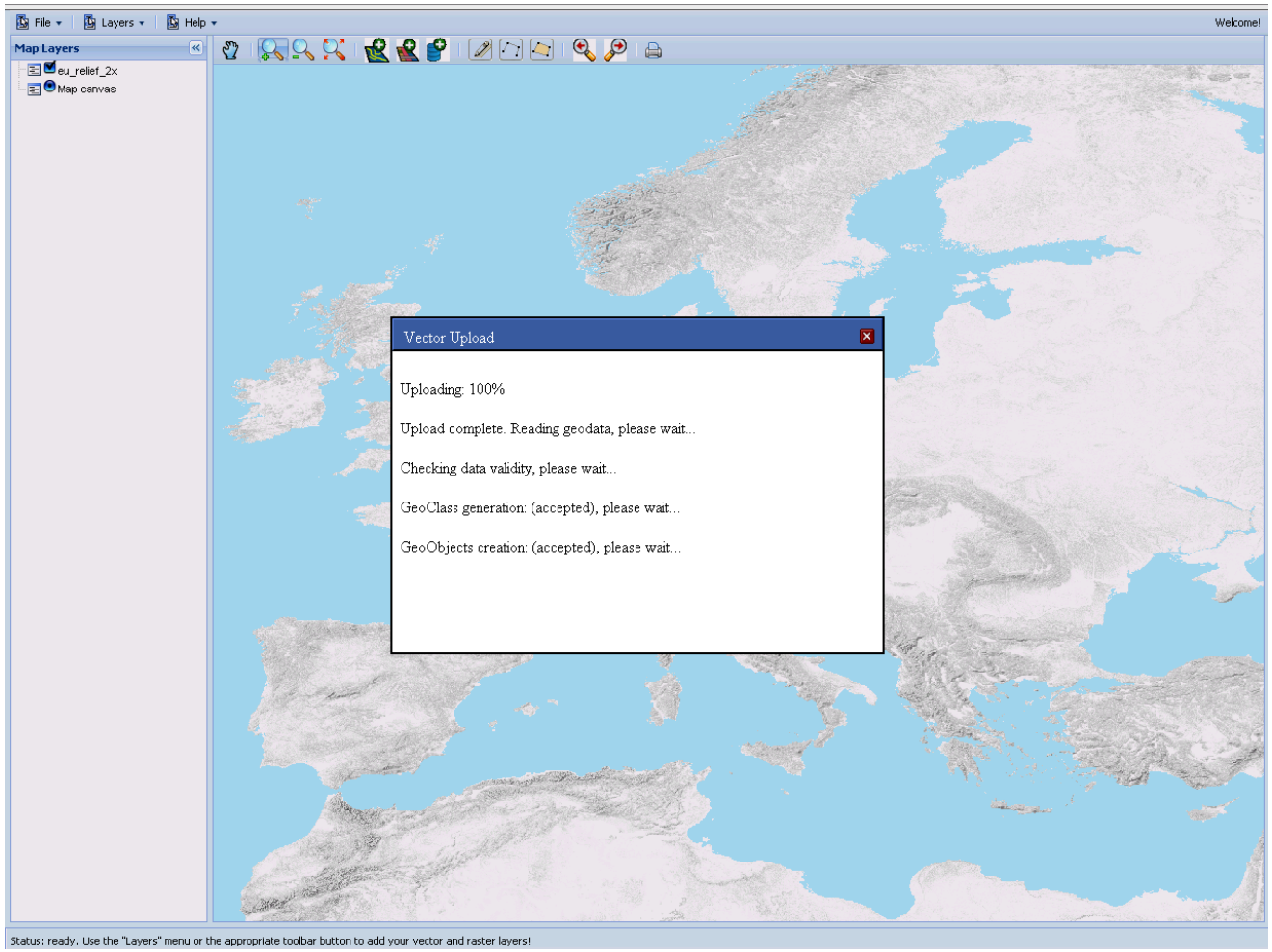
Integrated Programmatic and Cartographic Access to Spatial Data in a Web GIS Platform



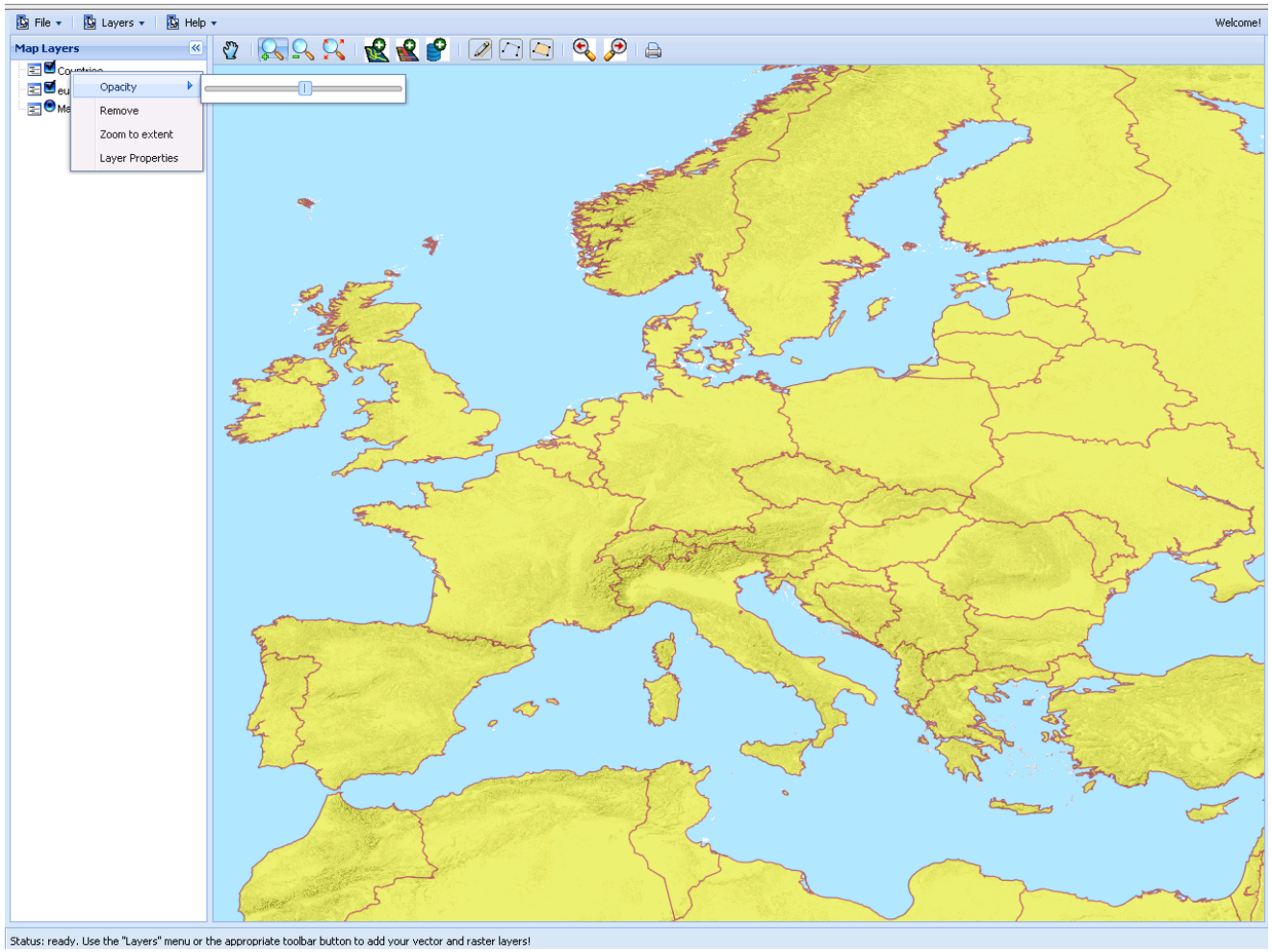
Proof-of-Concept Web GIS – Loading Data



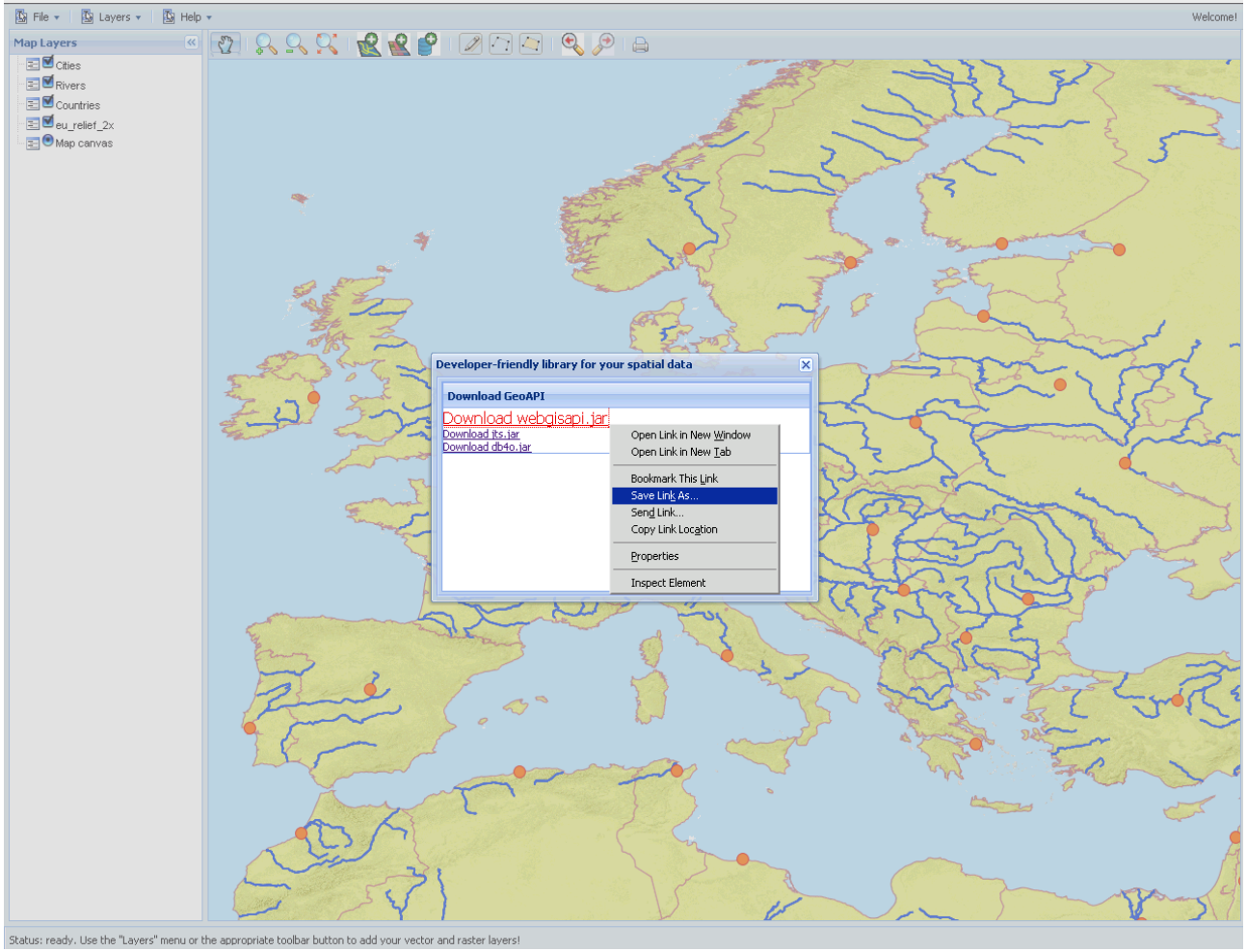
Proof-of-Concept Web GIS – Object Generation



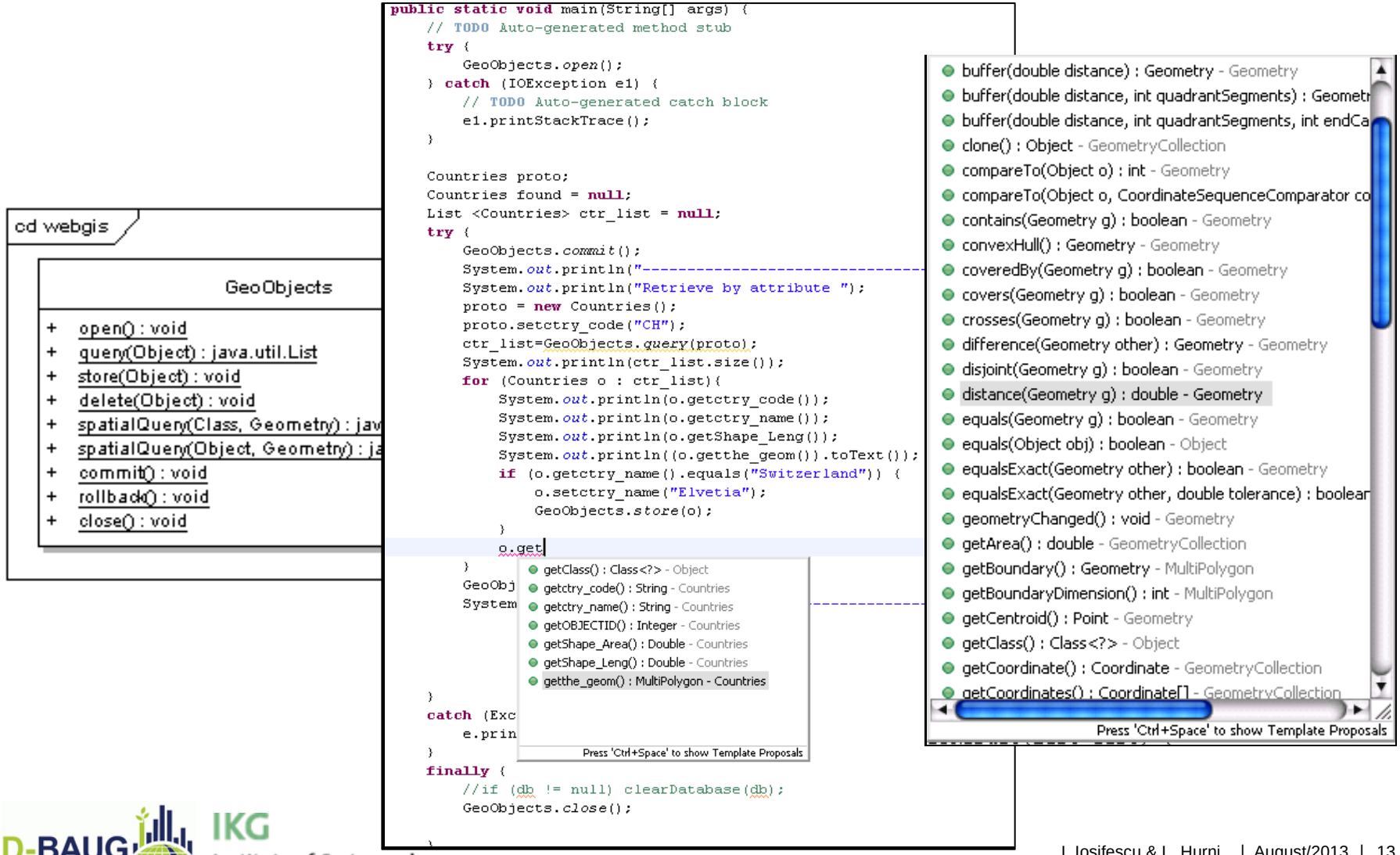
Proof-of-Concept Web GIS – Visualization



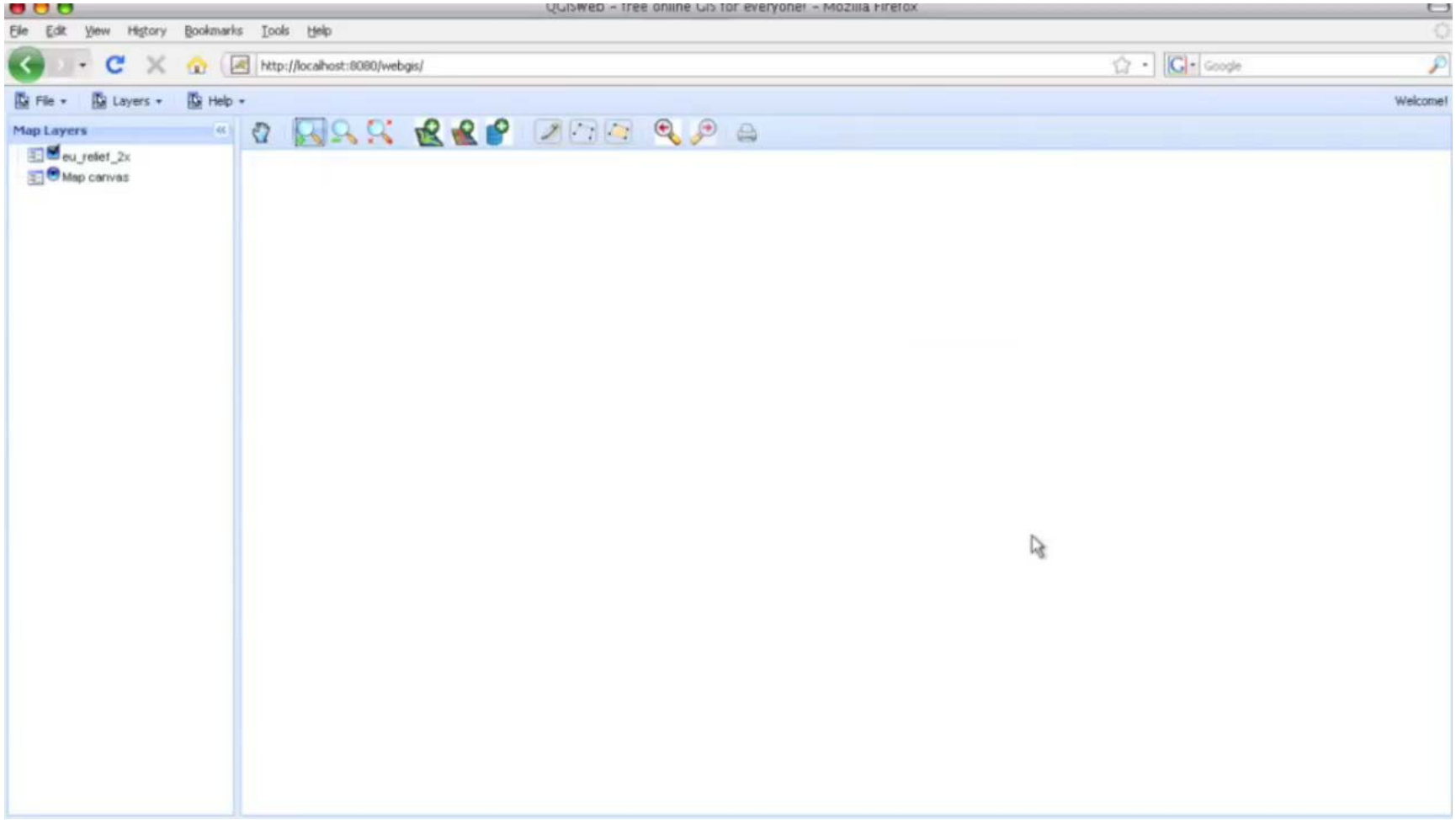
Proof-of-Concept Web GIS – Programming



Object-oriented Data Access in Java Code



Demo Object-oriented Data Access in Java Code

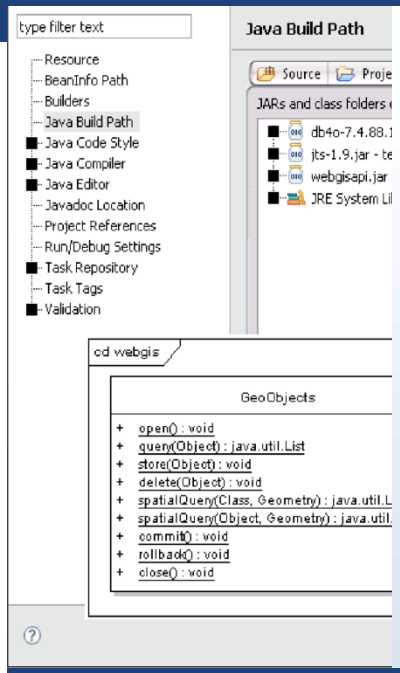


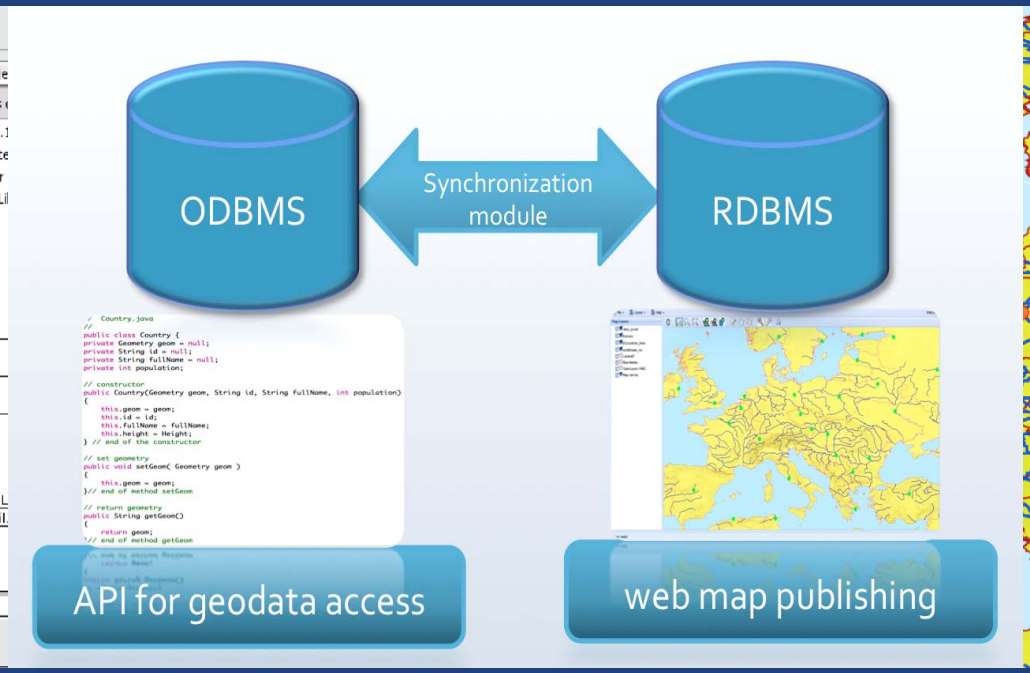
Usefulness of Programming API for Spatial Queries

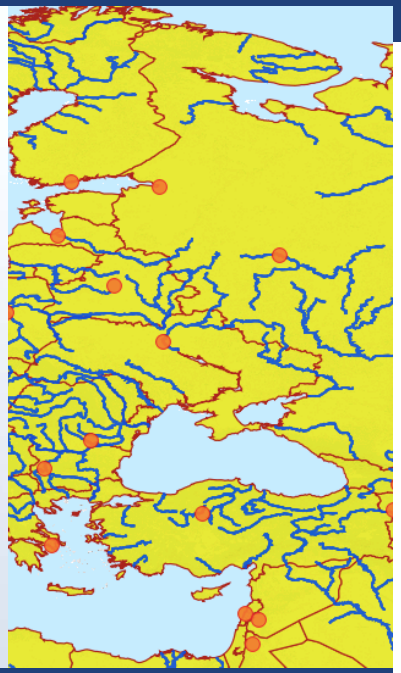
```
// opening the API
GeoObjects.open();
// create a new point geometry for the query
GeometryFactory geometryFactory = new GeometryFactory();
WKTReader reader = new WKTReader( geometryFactory );
Point point = null;
try {
        // we assume that a GeocodedAddress object is available
        point = (Point) reader.read("POINT (" + GeocodedAddress.getLat() + " " +
GeocodedAddress.getLong() + ")");
}
catch (ParsingException e) {
 //handle exception in parsing the coordinates
}
if (point != null) {
        // define the geometry of the buffer for the neighbouring query area as desired
        Geometry geom = point.buffer(0.025);
        // retrieve the objects of interest from the immediate area
        sensors_list = GeoObjects.spatialQuery(Sensor.class, geom);
        // implement the application logic as needed or simply display the query results
}
// closing the API
GeoObjects.close();
```

Conclusions

- Integrated cartographic and programmatic access to spatial data very practical for a software developer
- Object-oriented spatial data handling reduces the development time
- Mappings and transformations of spatial data to objects in a programming language are no longer necessary
- Spatial data access and conversions are completely transparent and object-oriented
- It insures flexibility and integration with existing cartographic tools and technologies
- Object-oriented databases aided the implementation







Thank you for your attention!

I. Iosifescu & L. Hurni
Institute of Cartography and Geoinformation, ETH Zurich, Switzerland