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

<table>
<thead>
<tr>
<th>Geometry Type</th>
<th>Text Literal Representation</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Point (10 10)</td>
<td>a point</td>
</tr>
<tr>
<td>LineString</td>
<td>LineString (10 10, 20 20, 30 30)</td>
<td>a linestring with 3 points</td>
</tr>
<tr>
<td>Polygon</td>
<td>Polygon (10 10, 20 20, 30 30)</td>
<td>a polygon with 3 points</td>
</tr>
<tr>
<td>MultiPoint</td>
<td>MultiPoint (10 10, 20 20)</td>
<td>a multipoint with 2 points</td>
</tr>
<tr>
<td>MultiLineString</td>
<td>MultiLineString (10 10, 20 20, 30 30)</td>
<td>a multilinestring with 3 lines</td>
</tr>
<tr>
<td>MultiPolygon</td>
<td>MultiPolygon (10 10, 20 20, 30 30)</td>
<td>a multipolygon with 3 polygons</td>
</tr>
</tbody>
</table>

relational to generic features mapping

generic to application specific mapping
Integrated Programmatic and Cartographic Access to Spatial Data in a Web GIS Platform

Web GIS → Geodata API → DB4O with spatial objects

ODBMS ↔ Synchronization module ↔ RDBMS

API for geodata access → web map publishing
Proof-of-Concept Web GIS – Loading Data

Image of a web GIS interface showing the process of uploading a shapefile.
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

```java
public static void main(String[] args) {
    // 2000 auto-generated method stub
    try {
        GeoObjects.open();
    } catch (IOException e) {
        // 1000 auto-generated catch block
        e.printStackTrace();
    }

    Countries proto;
    Countries found = null;
    List<countries> ctry_list = null;
    try {
        GeoObjects.commit();
        System.out.println("Retrieve by attribute ");
        proto = new Countries();
        proto.setCountry_code("CH");
        ctry_list = GeoObjects.query(proto);
        for (Countries o : ctry_list) {
            System.out.println(o.getCountry_code());
            System.out.println(o.getCountry_name());
            System.out.println(o.getShape_len());
            System.out.println(o.getShape_area());
            if (o.getCountry_name().equals("Switzerland")) {
                o.getCountry_name("Elvetia");
                GeoObjects.store(o);
            }
        }
    }
}
```

- buffer(double distance) : Geometry - Geometry
- buffer(double distance, int quadrantSegments) : Geometry
- buffer(double distance, int quadrantSegments, int endCap) : Geometry
- close() : Object - GeometryCollection
- compareTo(Object a) : int - Geometry
- compareTo(Object o, CoordinateSequenceComparator co) : Geometry
- contains(Geometry g) : boolean - Geometry
- contains(null) : Geometry - Geometry
- coveredBy(Geometry g) : boolean - Geometry
- covers(Geometry g) : boolean - Geometry
- crosses(Geometry g) : boolean - Geometry
- difference(Geometry other) : Geometry - Geometry
- disjoint(Geometry g) : boolean - Geometry
- distance(Geometry g) : double - Geometry
- equals(Geometry g) : boolean - Geometry
- equals(Object obj) : boolean - Object
- equalsExact(Geometry other) : boolean - Geometry
- equalsExact(Geometry other, double tolerance) : boolean - Geometry
- getArea() : double - GeometryCollection
- getBoundary() : Geometry - MultiPolygon
- getBoundaryDimension() : int - MultiPolygon
- getCentroid() : Point - Geometry
- getClass() : Class<?> - Object
- getCoordinates() : Coordinate - GeometryCollection
- getCoordinates0() : Coordinate[] - GeometryCollection
```
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();
WKTRReader reader = new WKTRReader( 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