Geospatial API for the cloud

Martin Desruisseaux
Geomatys
martin.desruisseaux@geomatys.com
Agenda

- Data formats and web services standards
- Need for bringing algorithms to data
- GeoAPI for implementation independence
- Apache Spatial Information System (SIS)
Barrier in data exchange

- Incompatible data and systems
- Data fragmentation and redundancy

Need for common language for geospatial data and services
An OGC standards benefit

http://myserver/myservice?REQUEST=GetMap
&SERVICE=WMS&VERSION=1.3.0
&LAYER=myLayer&FORMAT=image/png
&CRS=EPSG:4326&BBOX=18,-161,23,-154
&WIDTH=981&HEIGHT=826
API popularity

Web API:
- SOAP
- REST

Programming language API:
- Remote Procedure Call
  - COM, CORBA, Java RMI
- Scripting languages
  - JavaScript, Python…

1990 - 2018
Web API implies data encoding

- May be large fraction of micro-services

- By contrast, API in programming languages often transfer only 4 or 8 bytes (a pointer)
  - More suited to fine-grain operations
Ways to process data

- Transferring data to algorithm
  - WFS / WCS / ...

- Transferring algorithm to data
  - GeoAPI / OpenEO

- WPS as an intermediate position
  - Transfer parameters for a process pre-existing on the server
  - Transfer SQL-like queries
Ways to bring algorithm to data

System Virtual Machine
- Include an operating system
- Costly to boot

Docker image
- More lightweight than system virtual machine
- Still relatively costly to boot
- Size is many Mb or Gb

Lambda function
- More lightweight than docker image
- Run in Process Virtual Machine (e.g. Java)
- Size can be a few kb

User provides his own complete environment
Google Earth Engine approach

Looping

Because the client doesn't know what's in server-side `ee.Thing` objects, JavaScript functionality such as conditionals and for-loops does not work with them. For that reason, and to avoid synchronous calls to `getInfo()`, use server functions to the extent possible. For example, consider the following two ways of creating a list:

Not recommended — client-side for-loop

```javascript
var clientList = [];
for(var i = 0; i < 8; i++) {
    clientList.push(i + 1);
}
print(clientList);
```

Recommended — server-side mapping

```javascript
var serverList = ee.List.sequence(0, 7);
serverList = serverList.map(function(n) {
    return ee.Number(n).add(1);
});
print(serverList);
```

The server-side mapping example is a little silly because you could make the same list simply with `ee.List.sequence(1, 8)`, but it illustrates some important concepts. The first concept is `map()` which simply applies the same function to everything in the list. Because this function is executed on the server, client-side functions such as `print()` won't work in a mapped function. For that reason, the `i + 1` code has to be replaced with the equivalent server-side code: `ee.Number(n).add(1)`. Importantly, `n` is an object that only exists on the server. Because the function doesn't know the type of its argument, it needs to be cast to an `ee.Number`.
OpenEO approach


- **Goal**: unified way to connect clients to Earth observation cloud back-ends
- High-level API (Python, Javascript, R)
- Various backends (Google Earth Engine, …)
- API not yet based on OGC standards
Another OGC standards benefit

A blueprint!

Latitudes and longitudes without CRS are ambiguous
3 km error in some parts of the world
From standards to interfaces

Generated from XSD

```java
CoordinateSystem cs;
if (crs instanceof GeodeticCRS) {
    GeodeticCRS geodeticCRS = (GeodeticCRS) crs;
    cs = geodeticCRS.getEllipsoidalCS();
    if (cs == null) {
        cs = geodeticCRS.getSphericalCS();
        if (cs == null) {
            cs = geodeticCRS.getCartesianCS();
        }
    }
} else if (crs instanceof VerticalCRS) {
    VerticalCRS verticalCRS = (VerticalCRS) crs;
    cs = verticalCRS.getVerticalCS();
} else if (crs instanceof EngineeringCRS) {
    EngineeringCRS engineeringCRS = (EngineeringCRS) crs;
    cs = engineeringCRS.getEllipsoidalCS();
    if (cs == null) {
        cs = engineeringCRS.getSphericalCS();
        if (cs == null) {
            cs = engineeringCRS.getCartesianCS();
            if (cs == null) {
                cs = engineeringCRS.getPolarCS();
                if (cs == null) {
                    cs = engineeringCRS.getCylindricalCS();
                    if (cs == null) {
                        cs = engineeringCRS.getPlaneCS();
                        if (cs == null) {
                            cs = engineeringCRS.getInversEarthCS();
                            if (cs == null) {
                                cs = engineeringCRS.getSquareCS();
                                if (cs == null) {
                                    cs = engineeringCRS.getRationalCS();
                                    if (cs == null) {
                                        cs = engineeringCRS.getRectangularCS();
                                        if (cs == null) {
                                            cs = engineeringCRS.getSphereCS();
                                            if (cs == null) {
                                                cs = engineeringCRS.getCircleCS();
                                                if (cs == null) {
                                                    cs = engineeringCRS.getPointCS();
                                                    if (cs == null) {
                                                        cs = engineeringCRS.getTypeCS();
                                                        if (cs == null) {
                                                            // etc.
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
} else // etc.

From abstract model

```
“We just need latitude and longitude”
Simple, isn't it?
Coordinate Reference Systems

In EPSG dataset 9.5.2:

- 50 ellipsoids
- 487 two-dimensional geographic coordinate reference systems
- 514 geodetic datums
- 4675 projected coordinate reference systems

Latitudes and longitudes without CRS are ambiguous
3 km error in some parts of the world

Images: NASA/USGS
Hellerick & al. CC BY-SA 3.0
“I don't need all this complexity”
Let just use the WGS84 datum everywhere

WGS84 = World Geodetic System 1984
This is the datum used by Global Positioning Systems (GPS)
Earth is changing

NAD83 is tied to North American Plate
WGS84 is averaged over the world

→ They do not move in the same way

30 years ago, NAD83 ≈ WGS84
Now, differ by about 1.5 metres

NAD83(86)  WGS 84(Transit)
NAD83(HPGN)  WGS 84(G730)
NAD83(CORS96)  WGS 84(G873)
NAD83(2007)  WGS 84(G1150)
NAD83(2011)  WGS 84(G1674)
NAD83(CSRS)  WGS 84(G1762)

Galileo uses a different reference frame
WGS84 not always suited

- Some boundaries in USA are still legally defined in NAD27

- Difference with geoidal height up to ±100 metres

Recommendation

- Keep data in their original system and transform only when needed.
- Prefer the reference system defined by the mapping agency of the country where the data are located.
“Keep transformation engine simple”
Can we use WGS84 as the pivot system?
Pivot system can put errors

- World Geodetic System 1984
- Nouvelle Triangulation Française
- Réseau Géodésique Français 1993
- World Geodetic System 1972
- North American Datum 1927
- North American Datum 1983
- Japanese Geodetic Datum 2000
- Japanese Geodetic Datum 2011

TOWGS84 is removed in WKT version 2
Map projections are only part of work

Proportion of map projection code in Apache SIS referencing module
“I just need a few metadata”
Coordinates and reference system, that’s all
Minimal metadata

• Popular library (GDAL) provide:
  - Title
  - Image size
  - Coordinate Reference System
  - Geographic or projected bounding box
  - Bands
ISO 19115: Geographic metadata

Metadata

Data identification
- Citation
  - Titles
  - Authors (creator, contributor…)
- Data format
- Spatiotemporal extent
  - Geographic bounding box
  - Vertical and temporal ranges
- Resolution

Content information
- Illumination elevation & azimuth angles
- Cloud cover percentage
- Attribute (band) group
  - Content type (physical measurement, …)
  - Attribute (band)
    - Description (coastal aerosol, …)
    - Peak response in nanometres
    - Transfer function
  - More attributes (bands)…
ISO 19115: Geographic metadata

Metadata

Spatial representation

Acquisition
- Platform & instruments
- Operation (status, events, …)

Distribution
- Format
- Digital transfer options

Lineage
- Processors (organization, …)
- Process steps (inputs, algorithm, …)

Data quality
- Completeness
- Consistency (logical, thematic, …)

Maintenance
- Scope (dataset, software, …)
- Dates & update frequency

Constraints (legal, security, …)
GeoAPI

- Initiated in 2002
- Open Geospatial Consortium (OGC) working group
- Java interfaces derived from OGC/ISO conceptual models
- `org.opengis.*` packages

**Versions:**

- Latest release is GeoAPI 3.0.1 (September 2017)
- New working group created at OGC for GeoAPI 3.1 and 4.0

Apache Spatial Information System (SIS)

- Initiated in 2010
- Top Level Apache project
- Strong focus on OGC/ISO standards (GeoAPI 3.0)

**Versions:**

- Latest release is Apache SIS 0.8 (November 2017)
- Current development is Apache SIS 1.0-SNAPSHOT

**Code:**

- 227,000 lines of Java code
- 262,000 lines of comments
- Progressive transfer from Geotk project (800,000 lines) to Apache SIS

http://www.geoapi.org

http://sis.apache.org
API layers

GeoAPI (temptative)

GDAL

libpng  libtiff  ...

Apache SIS
# from opengis.wrapper.gdal import DataSet
from apache.sis import DataSet

ds = DataSet("myRaster.tif")
md = ds.metadata()
axis0 = md.spatial_representation_info[0].axis_dimension_properties[0]
axis1 = md.spatial_representation_info[0].axis_dimension_properties[1]

print()
print("Resource title: ", md.identification_info[0].citation.title)
print("Resource scope: ", md.metadata_scope[0].resource_scope)
print("Name of first axis: ", axis0.dimension_name)
print("Size of first axis: ", axis0.dimension_size)
print("Name of second axis: ", axis1.dimension_name)
print("Size of second axis: ", axis1.dimension_size)
print()
print("Complete metadata as formatted by the implementation (non-normative):")
print(md)

ds.close()
Bridge between languages

Libraries
- GDAL
- UCAR netCDF
- Apache SIS

Applications
- GeoAPI in Python
- GeoAPI in Java

Languages
- Python
- Java
Bridge between languages

**Libraries**
- GDAL
- UCAR netCDF
- Apache SIS

**Applications**
- GeoAPI in Python
- GeoAPI in Java
- Java / Python bridge

APACHECON North America
Apache SIS usage example

- **Reference system by EPSG code**
  
  ```java
  CoordinateReferenceSystem myDataCRS = CRS.forCode("EPSG:3395");
  ```
  
  Apache SIS uses the complete EPSG geodetic dataset

- **Reference System by Well Known Text (WKT)**
  
  ```java
  CoordinateReferenceSystem myDataCRS = CRS.fromWKT("PROJCRS[...]");
  ```
  
  Apache SIS recognizes automatically both WKT 1 (OGC 01-009) and WKT 2

- **Reference System by Geographic Markup Language (GML)**
  
  ```java
  CoordinateReferenceSystem myDataCRS = CRS.fromXML("<gml:ProjectedCRS>...");
  ```
Let user know about issues!

Log non-conform axis order:

```cpp
CRS.fromWKT("GEOGCS[...definition with (lon,lat) axes..., AUTHORITY["EPSG", "4326"]]");
```

**WARNING:** The coordinate system axes in the given “WGS 84” description do not conform to the expected axes according “EPSG:4326” authoritative description.

Log if prime meridian probably in wrong units (or other mismatches):

```cpp
CRS.fromWKT("..., PRIMEM[...value in deg], ..., AUTHORITY["EPSG", "4807"]");
```

**WARNING:** The given “NTF (Paris)” description does not conform to the “EPSG:4807” authoritative description. Differences are found in prime meridian.

Log usage of deprecated EPSG code, with replacement proposal:

```cpp
CRS.forCode("EPSG:26747"); // NAD27 / California zone VII
```

**WARNING:** Code "EPSG:26747" is deprecated and superseded by 26799. Reason is: Error in dependent projection record.
Find coordinate operation

1) Get two CRS (source and target)
2) Get a coordinate operation from source to target
3) Verify domain of validity and positional accuracy

```java
import org.opengis.referencing.operation.CoordinateOperation;

// Class declaration omitted for brevity

CoordinateReferenceSystem sourceCRS = // any method shown in previous slides
CoordinateReferenceSystem targetCRS = // any method shown in previous slides
CoordinateOperation op = CRS.findOperation(sourceCRS, targetCRS, region);

// Verify domain of validity and accuracy
System.out.println("Valid in "+ CRS.getGeographicBoundingBox(op));
System.out.println("Accuracy "+ CRS.getLinearAccuracy(op) + " m");
```
Apply coordinate operation

Example for a two-dimensional map projection:
(number of rows or columns depend on the number of dimensions)

\[ \text{mt.transform}(\phi, \lambda) : \begin{pmatrix} x \\ y \end{pmatrix} \]

\[ \text{mt.derivative}(\phi, \lambda) : \begin{pmatrix} \partial x/\partial \phi & \partial x/\partial \lambda \\ \partial y/\partial \phi & \partial y/\partial \lambda \end{pmatrix} \]
Reproject bounding boxes

Envelopes transformed = Envelopes.transform(op, envelope);

4 corners transformation

Source

Not sufficient

Target

Better

\[ y = c_0 + c_1 x + c_2 x^2 + c_3 x^3 \]

approximation extremum

\[ \frac{\partial y_1}{\partial x_1}, \quad \frac{\partial y_2}{\partial x_2} \]
## Current state

<table>
<thead>
<tr>
<th>ISO</th>
<th>OGC</th>
<th>Topic</th>
<th>GeoAPI</th>
<th>Apache SIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO 19103</td>
<td></td>
<td>Conceptual schema language</td>
<td>3.0.0</td>
<td>0.3</td>
</tr>
<tr>
<td>ISO 19115</td>
<td></td>
<td>Metadata (including imagery and gridded data extension)</td>
<td>3.0.0</td>
<td>0.3 (updated in 0.5)</td>
</tr>
<tr>
<td>ISO 19139</td>
<td></td>
<td>Metadata — XML schema</td>
<td></td>
<td>0.3, 1.0-SNAPSHOT</td>
</tr>
<tr>
<td>JSR-363</td>
<td></td>
<td>Units of Measurement API</td>
<td>3.0.1</td>
<td>0.8</td>
</tr>
<tr>
<td>ISO 19111</td>
<td>08-015</td>
<td>Spatial referencing by coordinates</td>
<td>3.0.0</td>
<td>0.4, 0.5, 0.6, 0.7, 0.8</td>
</tr>
<tr>
<td>ISO 19162</td>
<td>12-063</td>
<td>Well Known Text (WKT) representation of reference systems</td>
<td></td>
<td>0.6 (updated in 0.7)</td>
</tr>
<tr>
<td>ISO 19136</td>
<td>07-036</td>
<td>Geographic Markup Language (GML)</td>
<td></td>
<td>0.6 (updated in 0.7)</td>
</tr>
<tr>
<td>ISO 19109</td>
<td></td>
<td>Rules for application schema (Features)</td>
<td>3.1-SNAPSHOT</td>
<td>0.5</td>
</tr>
<tr>
<td>14-084</td>
<td></td>
<td>Moving Features CSV encoding (read only)</td>
<td></td>
<td>0.7, 0.8</td>
</tr>
<tr>
<td>16-114</td>
<td></td>
<td>Moving Features NetCDF encoding</td>
<td></td>
<td>0.8, 1.0-SNAPSHOT</td>
</tr>
<tr>
<td>ISO 19107</td>
<td></td>
<td>Feature geometry (1 to 3 dimensional)</td>
<td></td>
<td>pending</td>
</tr>
<tr>
<td>ISO 19123</td>
<td>07-011</td>
<td>Coverage geometry and functions</td>
<td></td>
<td>pending</td>
</tr>
<tr>
<td>ISO 19156</td>
<td>10-004</td>
<td>Observation and measurement</td>
<td></td>
<td>pending</td>
</tr>
<tr>
<td>ISO 13249</td>
<td></td>
<td>SQL spatial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12-168</td>
<td></td>
<td>Catalog Services (CSW)</td>
<td></td>
<td>Google Summer of Code</td>
</tr>
<tr>
<td>ISO 19128</td>
<td>06-042</td>
<td>Web Map Service (WMS)</td>
<td></td>
<td>pending</td>
</tr>
</tbody>
</table>
The network is the computer

Remote Method Invocation (RMI) where introduced in Java 1.1, released in 1997.

OGC standards published in 2001 were RMI and CORBA ready.
THANK YOU

Martin Desruisseaux
martin.desruisseaux@geomatys.com