Managing Geo Data –
Usage of Open Street Map for Own
Services and Applications

Jörg Roth
Department of Computer Science
Univ. of Applied Sciences Nuremberg
Kesslerplatz 12, 90489 Nuremberg, Germany
Why geo data?

Applications that consider the user's current location:
- Tour guides
- Route planning (car, tourist)
- Where is…?
- Community services, social networks

Not only end-consumers:
- Market research
- Logistics
- Traffic planning
Service platforms

Service platforms, e.g. *Google Maps*:

- Map display
- Routing
- Address resolution
- Friend finders
- Mobile support
Service platforms

Why not simply using such a service platform?

- Only services that are available can be used – no modifications possible
- Services can be withdrawn
- Costs, licenses
- Service availability (access via mobile networks often a problem)
- No control over geo data
  - Quality or coverage of geo data
  - Not possible to change or add own data

Sometimes, we do not want to rely on other services
Typical functions

What to do with geo data?

- Paint maps
- Routing
  - shortest, fastest
  - car, pedestrian
  - also useful: train, bus
    (problem: timetables)
- Geocoding, reverse geocoding
  - geocoding: geo object $\rightarrow$ coordinate
  - reverse geocoding: coordinate $\rightarrow$ geo object
    (or $\rightarrow$ postal address)
Typical functions

- **Radius search, nearby search, city search:**
  - *Where is the nearest fuel station?*
  - *Where are hotels with a distance not exceeding 5 km?*
  - *Give me all parks in Lisbon.*

- **Spatial join:**
  - *Give me all open air baths in Bavaria that are close to a train station (nearer than 1km).*
  - *How many cities in Europe have rivers running through them?*
Geo data basics

Three major properties of geo data:

- **Geometry**: what is the shape and location of the object?
- **Topology**: how is an object related to other objects? (most important: street routing network)
- **Thematic properties**:
  - Object type, e.g. street, forest, lake, church, bus stop, bistro, tree, artwork…
  - Names (in different languages, for different purposes)
  - Further properties: opening hours, max. speed, tree species, restaurant type
We now focus on *vector* data

- Object geometries are points, line strings or polygons
- Also possible: Bezier curves or splines (usually not supported)
- Typically 2D (only plane) or 2.5D (height is an attribute), *not* full 3D
Open Street Map:

- A community project to collect, process and distribute world-wide geo data
- First ideas in 2004
- 2006: *Open Street Map Foundation*, operable infrastructure
- 2011: 1.2 billion points, 116 million ways, 1.1 million relations
Open Street Map

Sources:

- Privately collected GPS tracks, entered, processed and classified by participating users
- Open geo databases, e.g. the TIGER databases
- Aerial images used to manually georeference objects
- Users can enter objects without any GPS measurement only in relation to existing objects
Open Street Map

Ways to add data to Open Street Map:

- Online editors
- Stand alone editors
- Registration required

Online discussion (Wiki):

- Most important issues: How to classify objects? (see later)
- Also: workflow, process models
Open Street Map

Available data:

- Most important: vector data in 2D
- Map bitmaps (many renderers, also 3rd party) – not the original domain of OSM
- Not or only partly supported:
  - Height profiles: only experimental
  - Street topologies: can be derived from geometries
  - Is-in relationship: only if users enter tags
  - Postal addresses: only if users enter tags
Ways to access vector data from Open Street Map:

- **Online via HTTP** – only small amounts of data
- **Planet files**:
  - Export of the entire OSM database as XML
  - Also parts in different granularities available: continents, countries, states, districts
OSM files:

- bzip2 compressed (zip not possible due to size limitations)
- unpacked: XML
- Main structure `<osm>…</osm>`
- Inside: three types of entries (in this order)
  - Nodes: `<node>…</node>`: Point objects and line points
  - Ways: `<way>…</way>`: Objects with line string or area geometries
  - Relations: `<relation>…</relation>`: Objects that are built up by other objects
OSM files

Example: Germany.osm (May 2012)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bzip2 file size</td>
<td>2 GB</td>
</tr>
<tr>
<td>XML file size</td>
<td>22 GB</td>
</tr>
<tr>
<td>XML tags total</td>
<td>318,532,216</td>
</tr>
<tr>
<td>nodes</td>
<td>100,475,499</td>
</tr>
<tr>
<td>ways</td>
<td>14,996,507</td>
</tr>
<tr>
<td>relations</td>
<td>234,273</td>
</tr>
</tbody>
</table>
<?xml version='1.0' encoding='UTF-8'?>
<osm version="0.6" generator="pbf2osm">
  <node id="1" lat="51.2492152" lon="9.4317166" version="6"
    user="elllit" uid="24852"
    timestamp="2011-08-16T11:26:47Z"/>
  <node id="10" lat="51.3806531" lon="9.3599172"
    version="5" user="max60watt" uid="134914"
    timestamp="2011-04-26T20:50:36Z"/>
  <node id="12" lat="51.3400316" lon="9.4819956"
    version="2" user="max60watt" uid="134914"
    timestamp="2011-04-28T21:39:02Z"/>
  <node id="13" lat="51.3731042" lon="9.5130058"
    version="2" user="max60watt" uid="134914"
    timestamp="2011-05-08T22:06:06Z">
    <tag k="highway" v="bus_stop" />
    <tag k="name" v="Bleichplatz" />
    <tag k="shelter" v="yes"/>
  </node>
</osm>
OSM XML file (ways)

```xml
<way id="3591699" version="2" user="Bube"
     timestamp="2009-06-13T07:45:37Z">
    <nd ref="17410365"/>
    <nd ref="17410355"/>
    <tag k="created_by" v="JOSM" />
    <tag k="highway" v="track" />
    <tag k="tracktype" v="grade2" />
</way>

<way id="3593390" version="8" user="kanu_guenni"
     timestamp="2010-03-31T18:19:41Z">
    <nd ref="14539664"/>
    <nd ref="14556238"/>
    ...
    <nd ref="14539666"/>
    <tag k="bicycle" v="official" />
    <tag k="foot" v="official" />
    <tag k="highway" v="path" />
</way>
...
```
<relation id="330" version="6" uid="161619" user="FvGordon"
timestamp="2012-02-24T23:13:40Z">
  <member type="way" ref="49022711" role="inner"/>
  <member type="way" ref="24808645" role="outer"/>
  <tag k="area" v="yes" />
  <tag k="highway" v="pedestrian" />
  <tag k="name" v="Martin-Luther-Platz" />
  <tag k="type" v="multipolygon" />
</relation>

<relation id="2235" version="6" uid="39381" user="DD1GJ"
timestamp="2009-11-22T08:15:27Z">
  <member type="way" ref="4917826" role=""/>
  ... 
  <member type="way" ref="7942741" role=""/>
  <tag k="name" v="Lichtentaler Allee" />
  <tag k="route" v="road" />
  <tag k="type" v="route" />
</relation>

...
Nodes:

- Major property: latitude/longitude, *no* extent
- Two types (only implicitly defined):
  - Point-like objects (often called *Point of Interests*):
    Objects as such, e.g. shop, restaurant, traffic light, postbox, waste container
  - Part of a line string (i.e. only a coordinate)
- Point of Interests must have further properties to be useful
  - at least the object type

```xml
<node id="13" lat="51.3731042" lon="9.5130058"
     version="2" user="max60watt" uid="134914"
     timestamp="2011-05-08T22:06:06Z">
  <tag k="highway" v="bus_stop" />
  <tag k="name" v="Bleichplatz" />
  <tag k="shelter" v="yes" />
</node>
```
Tagging objects

Tags k, v:

- k="..." and v="..." are used to express non-geometric properties, e.g.
  
  ```
  <tag k="highway" v="bus_stop" />
  <tag k="name" v="Bleichplatz" />
  <tag k="shelter" v="yes"
  
  Details of tagging see later
  ```

- For now: we write
  
  abc="xyz"

  instead of
  
  ```
  <tag k="abc" v="xyz" />
  ```
OSM ways

Ways:

- A sequence of nodes, referred by their ID
- Usually complete geo objects, i.e. with properties
- Three possible geometries:
  - (open) line string
  - closed line string
  - polygon
OSM ways

- Open line string: geo objects such as streets
- Closed line string: the actual object is the line string, *not* the content, example: roundabout
- Polygon: the actual object is the *content area*, not the border line. Examples: park, forest, lake
How to distinguish these geometries:

- Open line string: first and last node are not equal
- Closed line string vs. polygon (area):
  - no simple rule
  - sometimes distinguishable by their type
    - `highway` usually is closed line string
    - `building` usually is polygon
  - sometimes optional tag `area="yes"`
Further geometries

- OSM nodes and ways cannot express these geometries:

  - Multi Points
  - Multi Polygons
  - Multi Line Strings
  - Polygons with Holes
### OSM relations

**Relations:**
- Increase the ability to express complex objects and complex geometries
- Types of relations:

<table>
<thead>
<tr>
<th>Complex geometries</th>
<th>e.g. polygons with holes, multiple polygons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objects that share parts to avoid redundancy</td>
<td>e.g. shared borders of two cities</td>
</tr>
<tr>
<td>Combine objects to 'bigger' objects</td>
<td>e.g. 'Hiking trail Frankonia to Baltic Sea' that contains smaller hiking trails</td>
</tr>
<tr>
<td>Objects with piecewise defined properties</td>
<td>e.g. a highway with different speed limits (note: a way entry can only have a single set of properties)</td>
</tr>
<tr>
<td>Relationship between objects</td>
<td>e.g. a turn restriction between roads at a certain crossing</td>
</tr>
</tbody>
</table>
OSM relations – complex geometries

Multipolygon and polygons with holes:
- Relations with tag *type="multipolygon"*
- Each member must be a closed linestring
- Each member should have *role="outer"* (a shell) or *role="inner"* (a hole)
- Unfortunately: no expression which hole belongs to which shell
Borders:

- Borders (e.g. city borders) are often a collection of lines, each of it representing a part of the border.
- Reason: shared borders should only be stored once.
- The problem: neither ordering nor orientation are specified in the member description.
- Difficult to create a city area from borders.
OSM object IDs:
- nodes, ways and relations have unique IDs
- They do not change over time, i.e. can be used to identify objects between imports
- Note: they are only unique inside a type (nodes, ways, relations)
  - node with ID 1 and way with ID 1 possible
  - to have unique object IDs for all types you have to artificially distinguish types, e.g.

\[
\begin{align*}
\text{ownID} &= \text{ID} \cdot 3 + 0 \quad \text{for nodes} \\
\text{ownID} &= \text{ID} \cdot 3 + 1 \quad \text{for ways} \\
\text{ownID} &= \text{ID} \cdot 3 + 2 \quad \text{for relations}
\end{align*}
\]
Tagging objects

Tagging:

- Users can assign *arbitrary* pairs of key/value to objects
  - no limits for number of pairs
  - no obligatory keys, to superset of keys
  - all values allowed that can be expressed as string
- Only recommendations:
  - you may ask the editor for useful keys
  - no technical check, if entries are useful
- Currently, one problem for automatic classification
Tagging objects

However, there is some structure:
- usually there is a most important key/value
- unfortunately, not syntactically indicated
There are keys that define main classes

- Traffic

<table>
<thead>
<tr>
<th>Main Class</th>
<th>Subclass example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>highway=&quot;...&quot;</td>
<td>highway=&quot;motorway&quot;</td>
<td>Streets, roads, paths</td>
</tr>
<tr>
<td>railway=&quot;...&quot;</td>
<td>railway=&quot;station&quot;</td>
<td>Train stuff</td>
</tr>
<tr>
<td>cycleway=&quot;...&quot;</td>
<td>cycleway=&quot;lane&quot;</td>
<td>Cycling</td>
</tr>
<tr>
<td>waterway=&quot;...&quot;</td>
<td>waterway=&quot;river&quot;</td>
<td>Water</td>
</tr>
<tr>
<td>aeroway=&quot;...&quot;</td>
<td>aeroway=&quot;terminal&quot;</td>
<td>Flying</td>
</tr>
<tr>
<td>junction=&quot;...&quot;</td>
<td>junction=&quot;roundabout&quot;</td>
<td>Junctions</td>
</tr>
</tbody>
</table>
**OSM object classification**

- **Buildings, amenities, shops etc.**

<table>
<thead>
<tr>
<th>Main Class</th>
<th>Subclass example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amenity=&quot;...&quot;</td>
<td>amenity=&quot;restaurant&quot;</td>
<td>Amenities</td>
</tr>
<tr>
<td>shop=&quot;...&quot;</td>
<td>shop=&quot;hairdresser&quot;</td>
<td>Shops</td>
</tr>
<tr>
<td>craft=&quot;...&quot;</td>
<td>craft=&quot;plumber&quot;</td>
<td>Craft</td>
</tr>
<tr>
<td>office=&quot;...&quot;</td>
<td>office=&quot;lawyer&quot;</td>
<td>Offices</td>
</tr>
<tr>
<td>barrier=&quot;...&quot;</td>
<td>barrier=&quot;bollard&quot;</td>
<td>Barriers, walls</td>
</tr>
<tr>
<td>man_made=&quot;...&quot;</td>
<td>man_made=&quot;water_tower&quot;</td>
<td>Special buildings</td>
</tr>
<tr>
<td>power=&quot;...&quot;</td>
<td>power=&quot;generator&quot;</td>
<td>Electricity</td>
</tr>
<tr>
<td>military=&quot;...&quot;</td>
<td>military=&quot;bunker&quot;</td>
<td>Military</td>
</tr>
</tbody>
</table>
### OSM object classification

**Countryside, nature**

<table>
<thead>
<tr>
<th>Main Class</th>
<th>Subclass example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>natural=&quot;...&quot;</td>
<td>natural=&quot;rock&quot;</td>
<td>Anything natural</td>
</tr>
<tr>
<td>landuse=&quot;...&quot;</td>
<td>landuse=&quot;farm&quot;</td>
<td>Agriculture, farms</td>
</tr>
<tr>
<td>geological=&quot;...&quot;</td>
<td>geological=&quot;valley&quot;</td>
<td>Geological formations</td>
</tr>
</tbody>
</table>

**Non-physical objects**

<table>
<thead>
<tr>
<th>Main Class</th>
<th>Subclass example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>route=&quot;...&quot;</td>
<td>route=&quot;bus&quot;</td>
<td>Sequence of ways</td>
</tr>
<tr>
<td>boundary=&quot;...&quot;</td>
<td>boundary=&quot;postal_code&quot;</td>
<td>Boundaries, e.g. city</td>
</tr>
<tr>
<td>place=&quot;...&quot;</td>
<td>place=&quot;county&quot;</td>
<td>Captions, labels</td>
</tr>
</tbody>
</table>
OSM object classification

- Leisure, tourism

<table>
<thead>
<tr>
<th>Main Class</th>
<th>Subclass example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sport=&quot;...&quot;</td>
<td>sport=&quot;golf&quot;</td>
<td>Places for sport</td>
</tr>
<tr>
<td>leisure=&quot;...&quot;</td>
<td>leisure=&quot;park&quot;</td>
<td>Places for leisure</td>
</tr>
<tr>
<td>tourism=&quot;...&quot;</td>
<td>tourism=&quot;hostel&quot;</td>
<td>Related to tourism</td>
</tr>
<tr>
<td>historic=&quot;...&quot;</td>
<td>historic=&quot;monument&quot;</td>
<td>Historic objects</td>
</tr>
</tbody>
</table>

- (list is not complete)
Further classifications

- For some classes, further classes are recommended, e.g. for `amenity="restaurant"`
  a further tag
  `cuisine="..."` (e.g. `cuisine="bistro"`) is expected

- The problem:
  - it is not clear, whether a tag is meant as classification (such as 'cuisine') or as property (such as 'opening hours')
  - rule-based analyses required to map classes to objects
Ambiguity:

- Three ways to express 'bistro'
  - shop="bistro"
  - amenity="bistro"
  - amenity="restaurant", cuisine="bistro"

- Two ways to express combined foot/cycleway
  - highway="cycleway", footway="yes"
  - highway="footway", cycleway="yes"

- To quickly distinguish object types (e.g. for routing), a simpler classification scheme is required in own databases
# Naming objects

## Tags to name objects:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>name=&quot;...&quot;</td>
<td>Standard name</td>
<td>Germany</td>
</tr>
<tr>
<td>name:de=&quot;...&quot;</td>
<td>Name in different languages</td>
<td>Deutschland</td>
</tr>
<tr>
<td>*_name=&quot;...&quot;</td>
<td>(*\in{\text{int, nat, loc, reg}} ) international, national,</td>
<td>Bayern</td>
</tr>
<tr>
<td></td>
<td>local or regional name</td>
<td></td>
</tr>
<tr>
<td>short_name=&quot;...&quot;</td>
<td>Common abbreviation</td>
<td>UK</td>
</tr>
<tr>
<td>official_name=&quot;...&quot;</td>
<td>Official name</td>
<td>Principality of Andorra</td>
</tr>
</tbody>
</table>
Further tags for naming:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref=&quot;...&quot;</td>
<td>Name of motorways, bus lines etc.</td>
<td>A1, Route 66, 23</td>
</tr>
<tr>
<td>_ref=&quot;...&quot;</td>
<td>*∈{int, nat, loc, reg} international, national, local or regional name</td>
<td>A1 (without blank), A 1 (with blank)</td>
</tr>
<tr>
<td>addr: housename=&quot;...&quot;</td>
<td>House name as part of address information</td>
<td>Hotel 4 Seasons</td>
</tr>
</tbody>
</table>
# Addresses

## Defining postal addresses:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr: country=&quot;...&quot;</td>
<td>Country code as used in Internet names</td>
<td>DE</td>
</tr>
<tr>
<td>addr: city=&quot;...&quot;</td>
<td>Local city name</td>
<td>Nürnberg</td>
</tr>
<tr>
<td>addr: postcode=&quot;...&quot;</td>
<td>Postcode</td>
<td>90489</td>
</tr>
<tr>
<td>addr: street=&quot;...&quot;</td>
<td>Street</td>
<td>Kesslerplatz</td>
</tr>
<tr>
<td>addr: housenumber=&quot;...&quot;</td>
<td>House number</td>
<td>12</td>
</tr>
<tr>
<td>addr: housename=&quot;...&quot;</td>
<td>House name</td>
<td>Gebäude A</td>
</tr>
</tbody>
</table>
Where is an object located semantically?

- In which city, state, country resides an object?
- Can in principle derived by
  - border geometries: area required, geometric check is time consuming
  - postal addresses: not every relationship is encoded in the postal address (e.g. suburbs, districts in town, villages)
- Object can have explicit is_in tags, e.g. is_in="Nuremberg"
  - controversially discussed
  - not consistently used
Street topologies:
- Not actually supported by OSM
- The street network can be derived from street geometries:
  - if streets are connected, they must have a shared node
  - if crossing streets that are not connected (bridges, tunnels), they must not have a shared node
# Street topologies and route planning

## Tags that affect route planning:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>oneWay=&quot;yes&quot;</td>
<td>Driving only possible in the given direction</td>
<td></td>
</tr>
<tr>
<td>maxSpeed=&quot;...&quot;</td>
<td>Speed limit</td>
<td>50, 60 mph</td>
</tr>
<tr>
<td>highway=&quot;...&quot;</td>
<td>Street type</td>
<td>residential, secondary, motorway</td>
</tr>
</tbody>
</table>

In addition:
- Information about traffic signs, traffic lights
- Relations can identify turn restrictions
Public Transportation:

- Stops and stations are stored
- Connections are also available
  - typically as relations of streets or tracks
- The problem: no timetables available
Example tool chain – HomeRun

The *HomeRun* environment:

- Import, management, access of large amounts of geo data
- Also: foundation for spatial services, support for mobile platforms
  - *dorenda* map renderer and viewer
  - *donavio* navigation environment
HomeRun object classification

- Objects are classified by a 5 digit number
- Implicitly: a tree
HomeRun OSM import

The HomeRun import chain:

- Parse XML
- Replace all references to nodes and ways by their geometry
  - Following references during a query is too time consuming
- Classify objects (rule-based):
  - Define the classification number
  - Decide geometry (esp. closed line string vs. area)
  - Find out the appropriate (best) name
  - Distinguish tags (names, address, is-in, links, organizational)
HomeRun OSM import

- Resolve relations:
  - Build multipolys with holes
  - Build areas from line string borders

- Prepare route planning:
  - Compute the street topology from geometries
  - Retrieve routing relevant properties (oneway, maxspeed, avgspeed)

- Compute is-in relationship:
  - Detect import larger objects (city, state etc.)
  - Geometrically check if 'inside'
HomeRun geo database

How HomeRun stores geo data:

- Postgres database *without* spatial extension
  - geometries stored as *Well-known-binary (WKB)* in BLOBs
  - own spatial index (*Extended Split Index*)
- Also conceivable: SQL databases with spatial extension (e.g. PostGIS)
- HomeRun supports mobile devices
  - spatial extension to SQLite
  - spatially indexed virtual memory arrays (called *spatial hashtables*)
10 golden rules to start

How to start?

1. Enter some data into OSM – this is useful to get an idea of the process and structure.

2. Browse through the OSM Wiki to learn about object classifications.

3. For the first step, concentrate on a specific task, e.g. map painting, route planning.

4. Create an OSM XML parser (a simple one is sufficient). Be sure, the parser does not read the entire file before analyzing it.
5. Replace references to nodes by their coordinate – following references at runtime is too time consuming.

6. From the beginning, use a database (a non‐spatial often is sufficient) – using files will only work for small regions.

7. From the beginning use a geometry library. Don't reinvent the wheel – geometric computation is challenging.
8. Use an own classification schema (e.g. based on a number) – the OSM way to classify objects is too complex at runtime.

9. Think about rule-based classification and name finding (simple rule execution will be sufficient).

10. In early stages – forget relations. They are very complex to analyze, often inconsistently stored and often not useful.

Good luck 😊
Jörg Roth

Univ. of Applied Sciences Nuremberg
Joerg.Roth@Ohm-hochschule.de
http://www.wireless-earth.org