Location-aware Computing
Web Mapping APIs

http://www.cs.unibo.it./difelice/

Context Aware Systems

Prof. Marco Di Felice
Department of Computer Science and Engineering
University of Bologna
APIs for Location-based services

COMPONENTS of LOCATION-BASED SERVICES ARCHITECTURE

Source: http://geoawesomeness.com/

We will consider data formats for Location based Service and Content Providers
Map & Data

It is all about data-quality!

Source: Mapping the planet and the space around it, by Francesco Mete, EBWorld Solutions
Map Projections

✧ A **geographic coordinate system** (GCS) defines locations on the earth using a three-dimensional spherical surface.

**Datum** ➔ model of the earth (spheroid/ellipsoid). It consists of a series of values defining the shape and size of the ellipsoid and its orientation in space.

Source: Melita Kennedy, Understanding Map Projections
Map Projections

Map Projections is a technique of transforming a three-dimensional surface into a flat map sheet.

Set of mathematical equations transforming angular coordinates (latitudes, longitudes) into 2D locations on a plane:

\[(\lambda, \varphi) \leadsto (x, y)\]
Map Projections

Due to the problem itself, any Map Projection introduces some kind of distortion …

Source: Melita Kennedy, Understanding Map Projections

Map Projections

✧ Some projections are designed to minimize the distortion of specific characteristics of data.

✧ **Conformal projections** ➔ preserve local shape. Graticule lines on the globe are perpendicular.

✧ **Equal area projections** ➔ preserve the area of displayed features. To do this, the other properties of shape, angle, and scale are distorted.

✧ **Equidistant maps** ➔ preserve the distances between certain points. Scale is not maintained correctly by any projection throughout an entire map.

✧ **True-direction projections** ➔ The shortest route between two points on a curved surface such as the earth is along the spherical equivalent of a straight line on a flat surface.

Source: Melita Kennedy, Understanding Map Projections
Map Projections

CONIC PROJECTION

CYLINDRICAL PROJECTION

PLANAR PROJECTION

Source: Melita Kennedy, Understanding Map Projections
Map Projections

MERCATOR PROJECTOR

✧ Cylindrical projection.

✧ Meridians are parallel to each other and equally spaced. The lines of latitude are also parallel but become farther apart toward the poles. The poles cannot be shown.

✧ Scale is true along the equator or along the secant latitudes.

✧ Any straight line drawn on this projection represents an actual compass bearing.

Source: Melita Kennedy, Understanding Map Projections
Map Data Acquisition

http://www.aerialarchives.com/Aerial-Maps-of-San-Diego.htm
Map Data Structure

ꦿ Map is divided into a sequence of square cells (tiles)

魍 All map tiles are square-shaped and equal sized (Gmaps: 256x256 pixels).

 yü Each tile at any zoom level k is replaced by 4 equal-sized tiles at zoom level k+1.

רוע Tiles might come from various servers, including different mapping providers.

 гаран Tiles can be rendered on the server rather than on the user browser.

https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system
Map Data Structure

◊ Map is divided into a sequence of square cells (tiles)

https://docs.mapbox.com/vector-tiles/reference/

RASTER MAP TILE

An image (PNG, JPEG, …)

https://www.googleapis.com/tile/v1/tiles/0/0/0?
session=session_token&key=YOUR_API_KEY

Request a specific Tile, zoom 0, position 0,0 (Gmaps)
Map Data Structure

✧ Map is divided into a sequence of square cells (tiles)

✧ Represent geospatial vector data through geometries (points, lines, polygons) and features.

✧ They can be rendered on the client browser or on the server.

✧ Vector data is smaller than raster (bitmap) image.

✧ Allow re-styling and customization on the browser side.
Map Service Architecture

✧ **Web Map Service (WMS)** → Standard from the **Open GeoSpatial Consortium (OGC)** for serving geo-referenced map images generated by a map server.

✧ **HTTP-based** interaction between the client and the map-server

✧ Three main operations:
  - `GetCapabilities` → get the metadata of the WMS service
  - `GetMap` → retrieve a Map image based on user’s inputs (e.g. location and bounding box)
  - `GetFeatureInfo` → query for additional info regarding a specific feature
Map Service Architecture

Web Map Service (WMS) \(\rightarrow\) Standard from the Open GeoSpatial Consortium (OGC) for serving geo-referenced map images generated by a map server.

Source: Free and Open Source Web Mapping Maria Antonia Brovelli, Giorgio Zamboni
Working with Maps

- **Three types of service providers:**
  - Map tile providers $\rightarrow$ map data sources, e.g. NAVTEQ, TeleAtlas, OpenStreetMaps
  - Mapping APIs $\rightarrow$ map renderer and APIs for the application deployment, e.g. Google Maps, MapBox, Bing Maps, ...
  - Meta-map providers $\rightarrow$ toolkits enabling the integration of multiple map technologies and offering map-abstraction functionalities, e.g. OpenLayers.
Working with Maps

- Mapping APIs might have different characteristics:
  - Javascript vs Flash/ActionScript support
  - 2D vs 3D data
  - View layers (map, satellite, terrain, hybrid, aerial, etc)
  - Reverse vs Direct Geocoding
  - Mobile friendly vs Desktop friendly
  - Static vs slippy maps
  - Raster map tiles vs Vector maps
  - Licensing and pricing
OpenStreetMap

https://www.openstreetmap.org/

“Wikipedia of Maps”

✧ Founded in 2004 by Steve Coast
✧ Combination of multiple sources (aerial photography, GPS tracks, etc)
✧ Crowdsourcing; maps can be added and edited from users.
OpenStreetMap

✧ Large dataset of **XML** data
Each feature belongs to a certain **basic type** (nodes, ways, polygons).
Each feature can contain **tags**, i.e. sequence of key-value couples
✧ V: one_way K: yes
✧ V: highway K: bus_stop
✧ V: highway K: pedestrian
✧ V: bridge K: yes

More about the OpenStreetMap data structure:

**Areas**:
An area is a closed way.
For example, a plaza would be an area tagged with highway/=/pedestrian and area/=/yes.

Tags that can apply to ways or areas, like highway = pedestrian, may also require an area = yes tag.
Bing Maps

https://www.bing.com/maps

- Mapping API service from Microsoft
- Satellite imagery, aerial imagery, street view imagery
- Street data provided by the HERE company
- Javascript APIs, REST APIs, Spatial Data Service, WPF Control SDK
Google Maps v3

- Web Mapping Service operated by Google
  - Project acquired by the “Where 2 Technologies” company in 2004.
  - Combination of **aerial** photography and **satellite** photography
  - Map projection based on **Mercator** technique
  - Map coverage details:
    - https://developers.google.com/maps/coverage
OpenLayers

- Open-source, **Javascript toolkit** for map integration within Web-based location-based applications.
  - Supports multiple **map tile providers** (e.g. Bing Maps, OpenStreetMaps, Here!, Stamen, ...)
  - Supports **multiple geo-data vector formats** (e.g. GeoJSON, TopoJSON, KML, GML)
  - Map rendering based on Canvas 2D, WebGL or HTML5
  - API description available at: https://openlayers.org/en/latest/apidoc/
OpenLayers

Main components of an OpenLayers application:

- **Map** ➔ container, composed of Views and Layers
- **View** ➔ determines how the map is visualized, allowing to change the zoom level, the center location, etc
- **Layers** ➔ contents displayed on the map, can be *raster layers* (images) or *vector layers* (e.g. geometries)
- **Source** ➔ defines the layer provider (e.g. map tile service)
- **Features** ➔ for vector layers, defines the formatting styles (e.g. the line color for the GEOJSON LineStrings geometry)
OpenLayers

Map Creation:

1) Create a new `ol.Map` instance
2) Define the following properties:
   - target → HTML element where the map will be displayed
   - layers → data to be shown
   - view → `ol.View` instance defining the properties of the camera

```html
<div id="map" class="map"></div>
.map {
  width: 600px;
  height: 400px;
}
```

HTML Map Container
Map Creation:

```javascript
var map = new ol.Map({
  target: 'map',
  layers: [],
  view: new ol.View({
    center: ol.proj.fromLonLat([11.3394883, 44.4938134]),
    zoom: 13
  })
});
```

Layers can be defined here, or added programmatically through the `map.addLayer` method.
OpenLayers

Setting the View:
1. Instantiate the `ol.View` object
2. Define properties: center, resolution and rotation

```javascript
var view = new ol.View({
  projection: 'EPSG:4326'
});

view.setCenter([25.5433, 31.3123]);
view.setResolution(10000);
view.setZoom(7);
```
OpenLayers

Animate the View:

Modify the view properties setting the **transition** effects

- `ol.animation.pan` → set the view’s center
- `ol.animation.rotate` → set the rotation angle
- `ol.animation.zoom` → set the zoom level
- `ol.animation.bounce` → set the resolution property (bounce effect)

TWEEN EFFECTS

- `ol.easing.easeIn` → start slow and speed up
- `ol.easing.easeOut` → start fastest and slows to a stop
- `ol.easing.inAndOut` → start slow, speed up, then slow down ...

GEODATA AND MAPPING APIs

MARCO DI FELICE
OpenLayers

Animate the View:

Modify the view properties setting the transition effects

```javascript
var pan = ol.animation.pan({
duration: 5000,
easing: ol.easing.elastic,
source: map.getView().getCenter()
});
map.beforeRender(pan);
map.getView().setCenter([20, 20]);
```
OpenLayers

- Adding a Layer:
  - Add a **Map tile** layer *(source: OpenStreetMap)*

```javascript
var osmLayer = new ol.layer.Tile({
  source: new ol.source.OSM()
});
osmLayer.setVisible(true);
map.addLayer(osmLayer);
```
Adding a Layer:

Add a **Map tile** layer *(source: Stamen, http://maps.stamen.com)*

```javascript
var osmLayer=new ol.layer.Tile({
    source: new ol.source.Stamen({
        layer: 'terrain'
    })
});
osmLayer.setVisible(true);
map.addLayer(osmLayer);
```
Adding a Layer:

Add a **Map tile** layer (Generic Tile Provider)

```javascript
var osmLayer=new TileLayer({
  source: new XYZ({
    url: 'https://api.mapbox.com/styles/v1/mapbox/streets-v11/tiles/256/\{z\}/\{x\}/\{y\}?access_token=YOUR_MAPBOX_ACCESS_TOKEN'
  })
});
osmLayer.setVisible(true);
map.addLayer(osmLayer);
```

**X,Y,Z notation** → requesting tile <X,Y> with level of zoom equal to Z
Adding a Layer:

✧ Add a Vector layer (source: GEOJSON file)

```javascript
var geojsonLayer = new ol.layer.Vector({
    title: 'My new layer',
    source: new ol.source.Vector({
        url: 'myfile.json',
        format: new ol.format.GeoJSON()
    }),
    style: myGJSONstyle
});
geojsonLayer.setVisible(true);
map.addLayer(geojsonLayer);
```
Adding a Layer:

Add a Vector layer (source: KML file)

```javascript
var kmlLayer=new ol.layer.Vector({
    title: 'Layer KML',
    source: new ol.source.Vector({
        url: 'myfile.kml',
        format: new ol.format.KML({
            extractStyles: false
        })
    }),
    style: myKMLstyle
});
kmlLayer.setVisible(true);
map.addLayer(kmlLayer);
```
Adding a Layer:

✧ Add a Vector layer *(generic WMS service)*

```javascript
new TileLayer({
  extent: [-13884991, 2870341, -7455066, 6338219],
  source: new TileWMS({
    params: {'LAYERS': 'topp:states', 'TILED': true},
    serverType: 'geoserver',
    // Countries have transparency, so do not fade tiles:
    transition: 0
  })
})
```
Adding a Layer:

Add a style to a vector layer by creating an instance of `ol.style.Style`

```javascript
var myGJSONstyle = new ol.style.Style({
  image: new ol.style.Circle({
    radius: 5,
    fill: new ol.style.Fill({color: 'rgba(0, 255, 0, 0.13)'},
    stroke: new ol.style.Stroke({color: 'green', width: 1})
  })
});
```
OpenLayers

- Adding a Layer:
  - Add a style to a vector layer by creating an instance of `ol.style.Style`

```javascript
var featureTLightData = new ol.style.Style({
  image: new ol.style.Icon({
    scale: 0.05,
    src: "image.png"
  })
});
```
Most of the OpenLayers components can generate events; proper handlers can be defined.

```javascript
map.on('pointermove', function(event) {
    var coord3857 = event.coordinate;
    var coord4326 = ol.proj.transform(coord3857, 'EPSG:3857', 'EPSG:4326');

    console.log(coord3857, coord4326);
});
```
Google Maps v3

- Main Operations
  - Create the Map
  - Customize the Map
  - Modify the map style
  - Add Markers
  - Add Shapes and Vector Layers
  - Load Data

https://developers.google.com/maps/documentation/javascript/tutorial
Google Maps v3

- **Instantiate the Map**

```javascript
Map(MapDiv:node, opts?MapOptions)
```

```html
<div id="map"></div>
```

- `var mapDiv=document.getElementById('map')`

- **Aynchronous Map Loading**

```html
<script async defer src=https://maps.googleapis.com/maps/api/js?key=xyz&callback=initMap>
</script>
```

- `center` → defines the center of the map
- `zoom` → defines the initial zoom of the map
- `mapTypeId` → defines what type of map to display
Google Maps v3

- Instantiate the Map

```javascript
var mapDiv=document.getElementById('map');
var coord=new google.maps.LatLng(32.54,-12.33);
var options={
  center: coord,
  zoom: 4,
  mapTypeId: google.maps.MapTypeId.ROADMAP
};
var map=new google.maps.Map(mapDiv, options);
```
Google Maps v3

- **Customize the Map**
  - `disableDefaultUI` → enable/disable the default User Interface
  - `mapTypeControl` → enable/disable the displaying of the Map Type
  - `mapTypeControlOption` → control the way the Map Type is displayed
    - style → appearance of the control (DEFAULT, HORIZONTAL_BAR, DROPDOWN_MENU)
    - position → position of the control (BOTTOM, LEFT, TOP, RIGHT, ...)
    - `mapTypeIds` → map types available to users (e.g. google.maps.MapTypeId.SATELLITE)
  - `navigationControl` → display or hides the navigation control
  - `navigationControlOptions` → determines the look of the navigation control
    - style → appearance of the control (DEFAULT, SMALL, ANDROID)
    - position → position of the control (BOTTOM, LEFT, TOP, RIGHT, ...)
  - `scaleControl` → determines whether the scale control will be displayed

...
Google Maps v3

- **Customize the Map**

```javascript
var mapDiv=document.getElementById('map');
var coord=new google.maps.LatLng(32.54,-12.33);
var options= {
    center: coord,
    zoom: 4,
    mapTypeId: google.maps.MapTypeId.ROADMAP,
    disableDefaultUI:true,
    navigationControl:true,
    draggable:false
};
var map=new google.maps.Map(mapDiv, options);
```
Google Maps v3

- Customize the Style

```javascript
var mapTypeControlOptions = {
    mapTypeControlOptions: {
        mapTypeIds: [google.maps.MapTypeId.ROADMAP, 'MyStyle'],
    }
};

map = new google.maps.Map(mapDiv, options);
map.setOptions(mapTypeControlOptions);
map.mapTypes.set('MyStyle', styledMapType);
```
Define a Marker

```javascript
var myMarker = new google.maps.Marker({
  position: new google.maps.LatLng(44.497471, 11.355993),
  map: map,
  title: 'Department of Computer Science and Engineering'
});
var infoWindow = new google.maps.InfoWindow({
  content: 'My Department'
});
google.maps.event.addListener(myMarker, 'click', function() {
  infoWindow.open(map, myMarker);
});
```
Google Maps v3

Define a Marker

```javascript
google.maps.Marker(opts?options)
```

- position → LatLng object corresponding to the position of the Marker
- map → reference to the GoogleMaps object
- title → title of the Marker
- icon → image to show on the map
  - [Link to example image](http://gmaps-samples.googlecode.com/svn/trunk/markers/color/markerx.png)
    - x is a number between 0 and 65
- listener → function to invoke on events on the Marker
- Infowindow → additional information relative to the event
Google Maps v3

- **Customize the Style**

1) **Selectors** specify the geographic components to which the new style will apply. Selectors can include:
   - Feature ➔ identified by featureType (e.g. administrative elements)
   - Type ➔ graphic element to be re-styled (e.g. fill color)

2) **Stylers** define color and opacity properties applied to map elements matched by the Selector field.

```javascript
google.maps.StyledMapType()
map.mapTypes.set(NAME_LAYER, NAME_STYLED_MAP);
```
Create a Vector Layer

- **Polyline**(options?PolylineOptions)
  - path → Array ofLatLng objects
  - strokeColor → Color of the line #RRGGBB
  - strokeOpacity → Opacity of the line (0.0 = 100% opaque)
  - strokeWeight → Width of the line in pixel
  - fillColor → Fill color (#RRGGBB)
  - fillOpacity → Opacity of the fill color (0.0 = 100% opaque)

- **Polygon**(options?PolygonOptions)
  - center → Center of the circle
  - radius → Radius of the circle in meters

- **Rectangle**(options?RectangleOptions)

- **Circle**(options?CircleOptions)
Create a Vector Layer

```javascript
var route = [
    new google.maps.LatLng(44.4982761,11.3473072),
    new google.maps.LatLng(44.4942761,11.3377672),
    new google.maps.LatLng(44.4940611,11.3483242)
];
var polygon = new google.maps.Polygon({
    path: route,
    strokeColor: "#00ff00",
    strokeOpacity: 0.7,
    fillColor: "#0000ff",
    fillOpacity: 0.1,
    strokeWeight: 8
});
polygon.setMap(map);
```
Google Maps v3

Create a Shape

- `GroundOverlay()`

- Image on the map that are tied to latitude/longitude coordinates. The image moves when dragging or zooming the map

- `image`: URL of the image to display on the map
- `imageBounds`: sequence of LatLng objects, providing the bounds of the image

- Custom Overlays can be added to the GoogleMap
Load Data over the Map

```javascript
var imageBounds = {
  north: 44.4982761,
  south: 44.4900611,
  west: 11.3377672,
  east: 11.3483242
};

myOverlay = new google.maps.GroundOverlay('unibo_logo.jpg', imageBounds);
myOverlay.setMap(map);
```
Google Maps v3

**Load** Data over the Map

- `map.data` → Data layer for arbitrary geospatial data, including GeoJSON
  - `loadGeoJson()` → load a geoJSON file from an arbitrary URL
  - `addGeoJson()` → add a geoJSON file from a local variable

- `map.data.setStyle()`
- `map.data.overrideStyle()` → Define the style of the geoJSON features being displayed on the map.
Google Maps v3

Load Data over the Map

```javascript
map.data.addListener('mouseover', function(event) {
    map.data.revertStyle();
    map.data.overrideStyle(event.feature, {fillColor: 'blue'});
    alert('Passing over a ZTL Point');
});

map.data.setStyle({
    fillColor: 'blue',
    strokeColor: '#00ff00',
    strokeWeight: 10
});

map.data.loadGeoJson('http://www.cs.unibo.it/difelice/sm/files/varchi.json');
```
Google Maps v3

Create HeatMaps

google.maps.visualization.HeatmapLayer

- data → Array of LatLng points, each point can be associated to a weight
- dissipating → Specifies whether heatmaps dissipate on zoom
- maxIntensity → maximum intensity of the heatmap
- radius → radius of influence for each data point

```html
<script type="text/javascript"
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&libraries=visualization">
</script>
```
Create HeatMaps

```javascript
var heatMapData = [
    {location: new google.maps.LatLng(44.5052511,11.3406772), weight: 0.5},
    {location: new google.maps.LatLng(44.50441, 11.340981), weight: 0.5},
    {location: new google.maps.LatLng(44.495342,11.322345), weight: 0.5},
    ...
];

heatmap = new google.maps.visualization.HeatmapLayer({
    data: heatMapData,
    radius: 100
});

heatmap.setMap(map);
```
Google Maps v3

Create a Cluster of Markers

EXAMPLE based on Marker-Clusterer: https://github.com/googlemaps/js-marker-clusterer

MarkerClusterer(map: Map, markers?: Array, options?: Object)

- markers → Array of Markers to Display
- gridSize → All markers within a grid are grouped together
- maxZoom → maximum zoom level at which a marker can be part of a cluster
- zoomOnClick → zoom on a cluster when clicking over it
- imagePath → path to icons to display
- style → apply different styles to the clusters
Create a Cluster of Markers

```javascript
var markers[];
for (var i=0; i<heatMapData.length; i++) {
    var marker= new google.maps.Marker({
        position:heatMapData[i].location
    });
    markers.push(marker);
}

var markerCluster=new MarkerClusterer(map, markers, {
    'gridSize': 100,
    'zoomOnClick': true,
    'imagePath': './images/m'
});
```