Protocol to integrate databases containing geo-tagged data with PoCRA's GIS-dashboard

20th April, 2020

Prepared by Rahul Gokhale, Reviewed by Milind Sohoni

The GIS-dashboard is being developed with the aim of providing public facilities for geo-visualizing various geo-tagged and geo-referenced data-sets generated under PoCRA. Two of the basic and sought about use-cases are:

- **Mapping a location-based (geo-tagged) data-set:** Such data is linked to geographic locations (in terms of, say <latitude, longitude> of each location). The user-requirement is to be able to map these locations, or a user-specified filtered subset of them, on a web-map (the GIS-dashboard's map-view, in this case). A further requirement would be to colour each of the mapped locations based on the value of a user-specified attribute.

- **Visualizing project summaries/indicators for administrative regions:** Such data is linked to various administrative regions. Examples would be, the number of Farm-Field Schools per district, number of beneficiaries under DBT per village, etc. The user-requirement is to be able to visualize a map of the specified-level (district, taluka, etc.) of administrative regions, filtered by some query. A further requirement would be to colour each of these regions according to the value of a user-specified summary- or indicator-attribute.

The data-sets are stored in databases that may be managed by different parties, while the GIS dashboard prototype is being developed by IIT, which has no control over such databases. Consequently, a protocol has to be set that allows integrating the project's databases with PoCRA's GIS-dashboard in order to geo-visualize the geo-tagged data-sets stored in them.

A common approach for integrating database services over the internet is through web-APIs. This is a safe approach even when the involved IT-services are not collaborating towards a common task. However, it requires considerable time and efforts for design and implementation of the API. Also, if the application's data requirements are complex, the API design can be tricky and hence demanding on the API implementers.

When the involved IT services are between mutually trusted-parties, a much simpler and efficient approach is possible; namely to allow direct (although restricted) access to the database from the application (that requires the data). A specific implementation of this approach is where the database managers create database-views/-tables containing data-sets that they wish to provide to the application. This approach eliminates the need for APIs and hence saves on time and effort. Also, the complexity of data requests and its responses aptly becomes the concern of the application developers who demand/need that data.

In the following, the context in which this approach will be implemented is explained briefly. This also explains the choices made regarding the implementation. Next, the architecture of the integrated system is presented. Finally, a detailed protocol to be followed to implement and operationalize the architecture is presented.

GIS data-sets

From the technical viewpoint, there are generally three broad GIS data-types: points, lines and polygons. Every *GIS-(data-)entity* (a location, a segment of a path/network, boundary of a region) has one of these GIS data-types (point, line, polygon respectively) and has some attribute data linked to it. In this document, we refer to a GIS-entity coupled with its linked attribute data as a *GIS data-set*. For example, attributes of an FFS are primarily linked to its location; hence it forms a point-based GIS data-set. Attributes for a drainage network are linked to the lines representing the network-segments; hence it forms a line-based GIS data-set. Finally, say the attributes like cluster-assistants, cluster-level plans, etc. is linked to the cluster-boundary; hence it forms a polygon-based GIS data-set.

GIS database

GIS is a significant component of PoCRA's technical support. Most of the GIS data-sets being used in the project are stored as shapefiles and geotiffs on the hard-disks of the various workstations procured for the project-work. At the same time, a GIS-enabled database on the cloud-platform used in the project is a suitable place to store them such that, this database not only provides a common reference point for all GIS data-sets, but also enables GIS computations and selective publication(on web-GIS maps) from this same reference point.

GIS dashboard

The GIS-dashboard is meant to be the public-facing application that publishes GIS data-sets from this GIS-enabled common reference database. While it is possible, in principle, to store the complete GIS data-sets on the GIS reference database, it may reduce performance, flexibility and ease of maintenance. The attribute data from these GIS- data-sets is of significant size and generally would require independent maintenance owing to its domain-specific concerns.

The solution for geo-visualization is then to store the attribute data from these GIS data-sets in their own independent databases. For example, there is a database maintaining the attributes for the FFSs, a different database maintaining the attributes for MLP, another one for DBT, and so on. None of these need to be GIS-enabled, since geo-visualization is not a crucial component of their purpose and would require the replication of required GIS-entities (like district-boundaries, etc.) in each of them. On the other hand, the GIS-dashboard has it stored in

its GIS-database, all the GIS-entities to which the above attributes from various databases are conceptually linked. To enable geo-visualization, this conceptual linkage has to be translated into concrete data-linkage by implementing IT-coupling between the GIS-dashboard application and the project's various databases.

This IT-coupling can be implemented, as mentioned earlier, by allowing the GIS-dashboard application to access (parts of) the project's attribute databases. There are a few important implementation concerns in doing so. We describe them below.

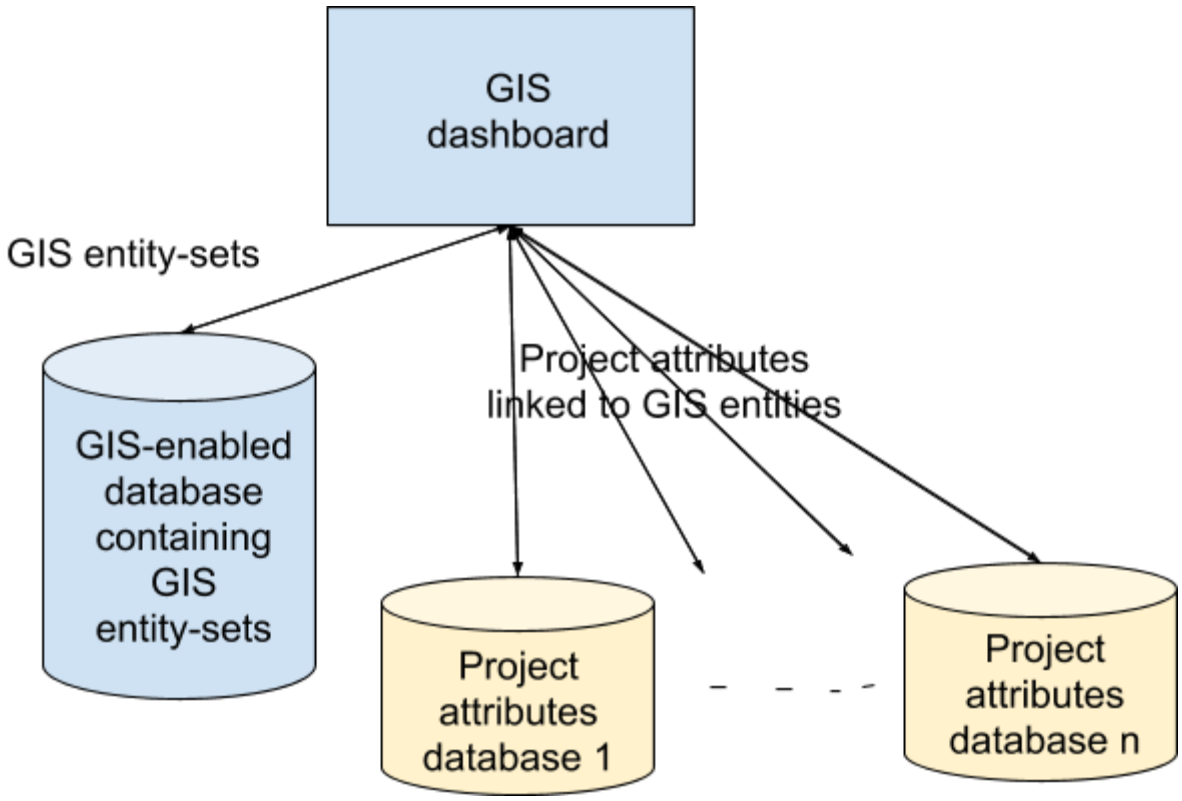
- **How frequently do the GIS-entities change?:**
  Location GIS-entities are prone to frequent changes. New locations may be added(eg: new weather stations) and also entirely new location-based data-sets may be added (eg: soil-sampling locations from some study). On the other hand, path/network and regional-boundary GIS-entities change much less frequently. This may probably be because of the time, effort and costs associated with generating such data-sets as opposed to mapping a set of few locations, for example using the common GPS-enabled mobile-phone.
- **Response-time of the GIS-dashboard application when displaying the GIS-entities:**
  As a GIS-(data-)entity, locations are very light-weight in terms of storage, computation and transferring over the internet. On the other hand, paths/networks and region-boundaries consume considerable resources and take time for communication. A typical client (web-browser on the user's computer) can map locations easily, while mapping paths and region-boundaries is resource- and time-expensive. For the latter, it is better to compute images of their maps on the server and serve those images instead. This is a job requiring specialized GIS software (like a mapping server).

Thus it is seen that the GIS-entities(points) in a location-based data-set can change frequently, have low computational resource demands and do not require a GIS-enabled database to store legibly. On the other hand, GIS-entities having GIS data-type 'line' and 'polygon' rarely change over the duration of the project(or even later), have high computational resource demands and require GIS-expertise for maintenance in a GIS-enabled database. These implementation concerns lead to two categories of GIS data-sets: location-based data-sets and others (path/network and regional boundary data-sets). The former can be maintained entirely in the project's databases while the latter must be loaded into the GIS-dashboard's database.

The architecture

The following two parts of any GIS data-set have to be considered in-tandem when designing the architecture to offer web-based geo-visualization:

1. GIS entity-set ('points', 'lines' or 'polygons')
2. attributes linked to each of these entities ('plot_id' of FFS, 'total_storage_capacity' in the village, etc.)



Accordingly, the architecture, as depicted above, consists of:

1. **GIS entity-sets:** A set of GIS entity-sets will be maintained by the GIS-dashboard developers. Each entity-set will have a specified index key to identify the GIS-entities. For example the GIS entity-set of polygons of village boundaries can be indexed by the census code.
2. **Database-views of linked attributes:** Each database that needs geo-visualization on the dashboard will maintain a set of database-views. Each such view will contain attributes linked to one of the GIS entity-sets maintained as in item 1. This link will be manifested by the view through columns referring to that GIS entity-set's index-key(mentioned in item 1). For example, a database needing geo-visualization cropping attributes per village, will have a database-view having census_code as one of its columns, while the other columns specify the actual cropping attributes.
3. **The linkage between GIS-dashboard and project's databases:** The dashboard will display the attribute-views from project's databases using the linked GIS entity-set and allow for filters on the attributes of as well as selected geographic properties of the GIS-entities. For example, the phase of implementation of each village or the area of the villages can be used to filter the set of village polygons.

Some of the GIS data-sets currently available for use on the GIS dashboard are:

Name	Type	Keys
District	Polygon	district
Taluka	Polygon	district, taluka
Cluster	Polygon	cluster_code
Village	Polygon	census_code

The protocol to operationalize the architecture:

Following are the steps to be followed when integrating GIS data-sets to the GIS-dashboard. Steps specific to a category are indicated appropriately.

1. PMU shares the data files containing the GIS-entities with the GIS-dashboard developers. Existing examples are the shapefile of all the villages in the 15 districts of PoCRA, the shapefile of drainage networks in these districts, etc.

2. The GIS entity-set obtained in step 1 is loaded into the dashboard's database.

3. The GIS-dashboard developers publish the name of the loaded GIS entity-set and also a key (represented by a tuple of database-columns) that can be used to index into the entity-set. For example, the census-code of a village may be published as the key (tuple <census_code> containing just one column) for the 'Village' entity-set. A census-code, say 532643, will refer to the village polygon having census-code 532643.

4. New point-data sets can be created as well. Create a database-table containing the following columns so that each row uniquely represents one location:

    a. **myname_id:** data-type integer(unique within this view/table); identifying a location in the data-set
    b. **latitude:** data-type real/double-precision; representing the latitude of the location
    c. **longitude:** data-type real/double-precision; representing the longitude of the location
    d. Any other geographical attribute.

    This table can be named as the database managers see fit. For example: 'FFS_locations'. However, the three column names ('FFS_id', 'latitude' and 'longitude') mentioned above should be used exactly as specified. This table will be converted by the GIS dashboard into a GIS data set and will be made available to the PMU to create views.

5. Managers of a project's database who wish to use geo-visualization, can create any number of database-views whose columns contain the actual attribute data that has to be geo-visualized. Each of the views must satisfy the following criterion:

A. It may refer to keys from several GIS datasets, however, for each view, there should be a designated geographical key. For example, a view on cropping patterns may link with the village GIS dataset and use the census-code key as an index for the view.

Example 1:
View-name: **Village_crops**

census_code	**crop_name**	**crop_area_ha**	**crop_type...**	**..**
'530764'	soyabean	50	Long Kharif	..
'531027'	cotton	75	..	..
..	..	..	..	..
..	..	..	..	..

Note: The "census_code" column of this table will be used to refer to the corresponding village's polygon in the GIS-dashboard's database.

Example 2:
View-name: **FFS_static**

id	**plot_code**	**soil_type**	**...**	**..**
1	plot_code_1	medium	..	..
2	plot_code_2	deep	..	..
..	..	..	..	..
..	..	..	..	..

Note: The "id" column of this table will be used (for its geographic location information) to refer to the row in the view/table created in step 4.

B. If a view V must have links to two GIS keys, for example FFS_id and census_code, then two copies will be created, Va and Vb where the first key is the designated geographical key and Vb where the second key is the designated key. This will enable the display of Va as points and Vb as polygons by the dashboard.

6. The project's database managers will avail the geo-visualization facilities by sharing a document with the GIS-dashboard developers that contains the following details:
    a. Name of the data-set it wishes to publish (as the user would see it). For example: Village-wise Cropping
    b. Database access details (of its own database):
        - Host-name (or IP)
        - Port
        - Database-name
        - User-name
        - Password
    c. GIS entity-set to link to: Mention database as 'GIS-database' and the name (as published by the GIS-dashboard developers) of the GIS entity-set to link to. For example: 'Database: GIS-database, Name: Village' for the example of step 3.

    d. Meaning of the data-attributes provided through the views.

        Example:

        View-name: FFS_locations GIS dataset

column_name	**column_label**(What name the user would see for this attribute)	**meaning** (What does the column of the view represent?)
FFS_id	N/A or null	GIS key
latitude	Latitude	(pre-defined as)latitude of the location
longitude	Longitude	(pre-defined as)longitude of the location
elevation	Elevation	Elevation of the location above mean sea-level
..	..	..

        View-name: FFS_static_data

column_name	**column_label**(What name the user would see for this attribute)	**meaning** (What does the column of the view represent?)

FFS_id	N/A or null	GIS key which links to FFS_locations
plot_code	Plot-Code	Unique code given to the FFS plot
soil_type	Soil-Type	Soil-type
..	..	..

View-name: crop_current_status

column_name	**column_label**(What name the user would see for this attribute)	**meaning** (What does the column of the view represent?)
FFS_id	N/A or null	GIS key which links to FFS_locations
season	Season	Name of ongoing season
cropping_system	Cropping-System	Type of the cropping-system used

One table each for any more such views.

7. The dashboard will display the attribute-data from the views as being linked to the corresponding GIS entity (village, location, etc.)