LapsoCarte

A web platform to visualize and interact with temporal and spatial referenced data

Camilo Vargas Cabrera
c.vargas124@uniandes.edu.co

Bachelors Degree's Project

Supervised by:

Jose Tiberio Hernandez PhD.
jhernand@uniandes.edu.co

Systems and Computer Engineering
Los Andes University

Bogotá D.C, Colombia
2015

1 Introduction

A common characteristic in urban planning and modeling is the temporal-spatial nature of information. Researchers of multiple disciplines use different tools and approaches to explore historical data or build and run simulation models.

Systems dynamics, cellular automata and agent based models are implemented in plain programming languages and domain specific tools. Temporal-spatial information visualization often requires to manually export and transform data. Otherwise the visualization is limited by the ones available in the used tool.

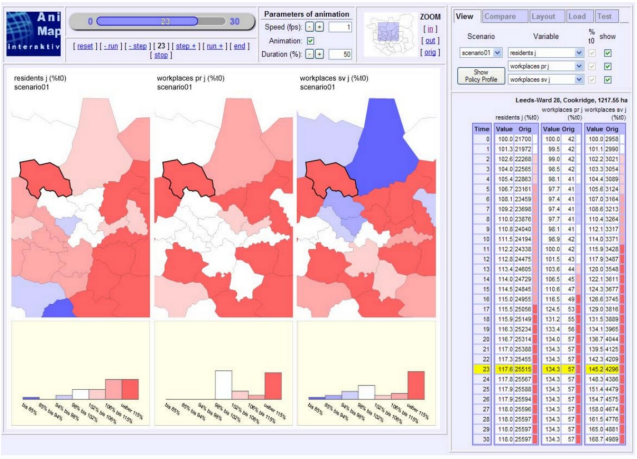
It is common to find a grow number of data specific visualizations that use what Bach et al. (2014) define as *time juxtaposing* representation for temporal-spatial information. Most visualizations rely on the target data schema, which makes them not reusable in other contexts.

Since the nature of the data and its expected representation is essentially the same, *LapsoCarte* proposes a generic data representation coupled with a case-specific data-level integration strategy. This approach provides a reusable codebase for basic temporal-spatial data analysis and exploration. In addition, the web-based nature of *LapsoCarte* provides a platform-independent access to the visualization, which allows the analyst to use the tool from any web browser.

1.1 Background

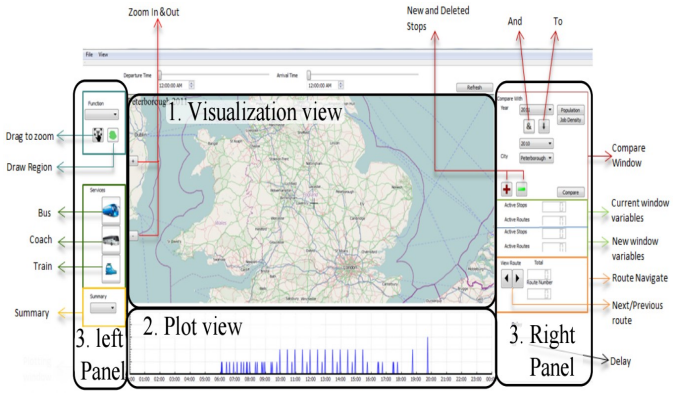
The following are some examples of time juxtaposing visualizations for temporal spatial data sets:

Emberger, Pfaffenbichler y Riedl (2010) developed a web representation for the results of the *Metropolitan Activity Relocation Simulator*. ANIMAP provides a set of maps to explore the spatial dimension and a slider to navigate in the temporal dimension. It uses a choropleth function to represent data magnitudes in the maps' polygons and graphs on bottom to support data analysis. There is a data table with a filter to inspect granular data:



ANIMAP Interface (Emberger, Pfaffenbichler y Riedl 2010)

Berenjkoub et al. (2015) proposed a web-based tool to analyze the longitudinal transportation data of Great Britain. The application processes the data of routes and frequencies published by the *Great Britain Transport Authority* and depicts it in an interactive web application. A map in the center represents the *spatial dimension* and two sliders on top allows to navigate in the *temporal dimension*. The application provides filters for the represented data and a plot view in the bottom:



Graphical User Interface (Berenjkoub et al. 2015)

Tableau is a commercial data visualization tool for business intelligence. Bracco presents a *Tableau* based data exploration tool in his 2014 blog post *Mapping the City of Houston's Electricity Usage Over Time*. The visualization has a slider on top to navigate in the temporal dimension and a map in the center to explore the spatial dimension. Data magnitudes are represented by the circles' radius and colors:

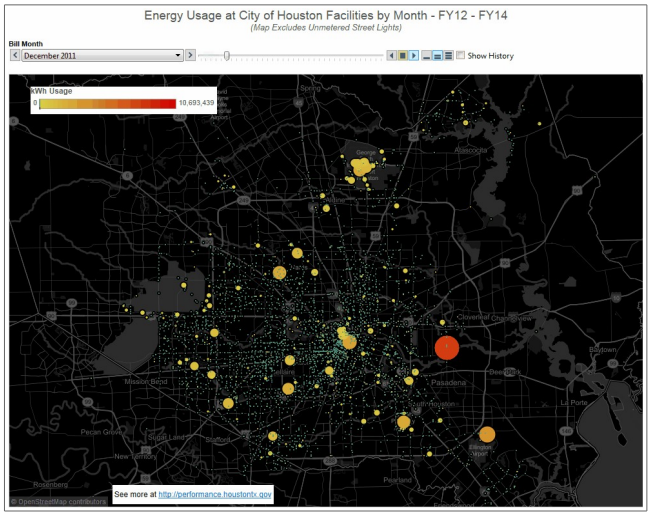
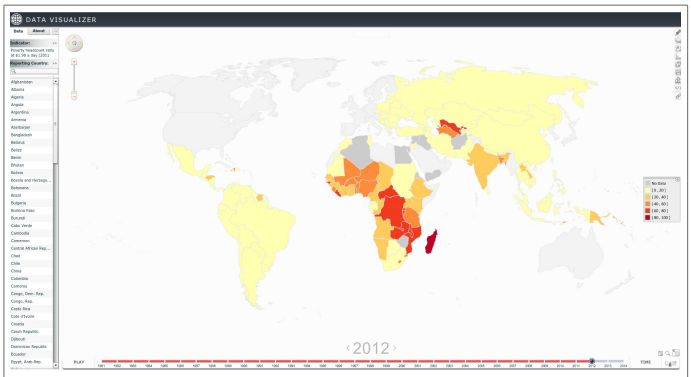


Tableau based visualization (Bracco 2014)

The World Bank's Data Visualizer tool features a choropleth map to explore the spatial dimension and an animated slider on bottom to navigate in the temporal dimension:



World Bank data visualizer

2 Dimensions

In *LapsoCarte* data is stored, indexed, retrieved and explored in a generic four dimensional schema:

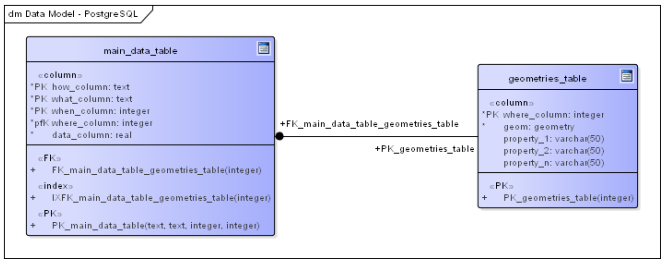
- **How**: Main grouping level to differentiate between data types or alternatives. *Hows* are represented as strings.
- **What**: Secondary grouping level to differentiate between data sets. *Whats* are represented as strings.
- **When**: Temporal dimension. *Whens* are enumerable and sortable elements, represented as integers.
- **Where**: Spatial dimension. *Wheres* are geometries represented as points, lines, polylines or polygons.

This data structure is influenced by the Spatial-temporal Data Exploration Model (MEDET) proposed by Nemocon (2015) to explore temporal-spatial referenced information.

2.1 Server side data model

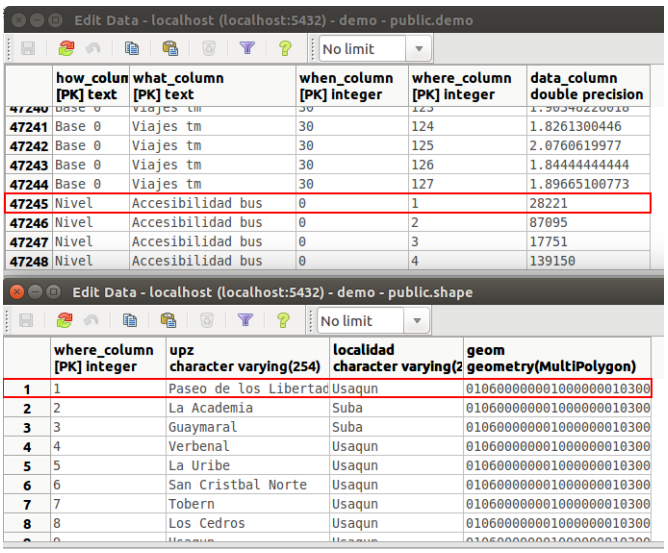
The back-end relies on a given database structure of two tables:

- **Main data table**: This table has five columns, one for each dimension plus the data itself. The primary key is a compound key of the four dimensions.
- **Geometries table**: This table is used to normalize the geometries information. It has a primary key, a geometry and an arbitrary number of additional columns for geometry's properties.



Server-side data model

By design, the database is in *third normal form* for dimensions, data and geometries. Normalization is enforced on geometries' features but not on geometries' properties.



Server-side sample data

2.2 Client side data model

In the front-end data, exploration follows a hierarchical structure: *How*, *What*, *When*, *Where*.

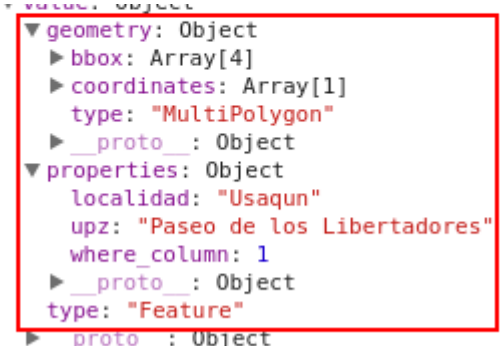
When the user loads *LapsoCarte* for the first time, the browser downloads all *data* and *geometries* in JSON format.

Data is sent by the server in the aforementioned hierarchical structure. Once it is received, the client builds a JavaScript map to efficiently browse data. For example, the code to access the element in row 47245 of the server-side sample data is:

```
let data = dataMap.get('Nivel')
                  .get('Accesibilidad bus')
                  .get(0)
                  .get(1); // data = 28221
```

Client-side data sample code

Geometries are sent in an indexed JSON object. The index is the geometries' table primary key and the element is the geoJSON representation of the geometry. The client builds a JavaScript map for geometries' *features* and *properties*:



Client-side geometries map

Each geometry exists only once on the client's *Leaflet Controller* and its style is updated on each dimension change.

2.3 Drivers

A *driver* is a bridge between data producers and *LapsoCarte's* data schema. *Drivers* are essentially small applications who systematically take a given data input and populate the *dimensions* based data structure.

In a simple implementation, a *driver* is a *parser* coupled with a *bash script* which processes a text file with a given structure. The *parser* reads the file and produces a SQL script which creates the dimensions based structure in a database. The *bash* script automates the process of retrieving the information, running the parser and populating the database.

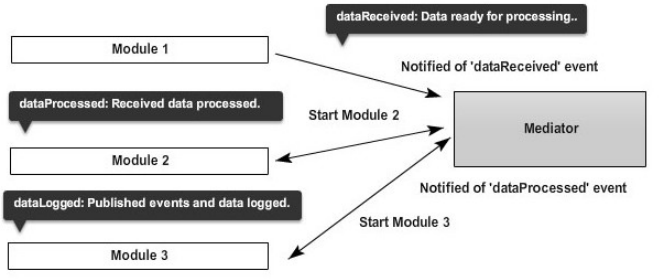
Most drivers are a one-time problem-specific implementation because they are strongly linked with the source system output.

3 Architecture

LapsoCarte architecture is based on a variation of the *Mediator Pattern.*

Osmani in its *Patterns For Large-Scale JavaScript Application Architecture* blog post explains: *Notice how at no point do any of the modules* ***directly communicate*** *with one another. It (the mediator) decouples modules by introducing an intermediary as a central point of control (...) It is typically easier to add or remove*

features to systems which are loosely coupled like this.



Mediator Pattern example (Osmani 2011)

The mediator approach works well to synchronise events between modules. However, this could be needlessly complex to handle shared objects like the GUI current state, the dimensions-indexed JavaScript's map or some library specific objects. To overcome this, *LapsoCarte* implements a global accessible *PROJECT* and *INSTANCE* objects.

3.1 Controllers and Events

In *LapsoCarte* a controller is a piece of code responsible of handle a specific task. *MainControllers* are *mediators* while *SubControllers* are *modules*. Most *SubControllers* encapsulate the logic to drive a specific JavaScript library. However, when the task is too complex, a *SubController* could act as a *mediator* between a group of child *controllers*. This structure generates a hierarchical *MainController - SubControllers* tree.

Events trigger a call to controllers' methods. Each controller have *mc_* and *sc_* methods. *mc_* methods are called by *MainControllers* while *sc_* methods are called by *SubControllers*. Mediators sync actions between its *SubControllers* on each *mc_* or *sc_* method call.

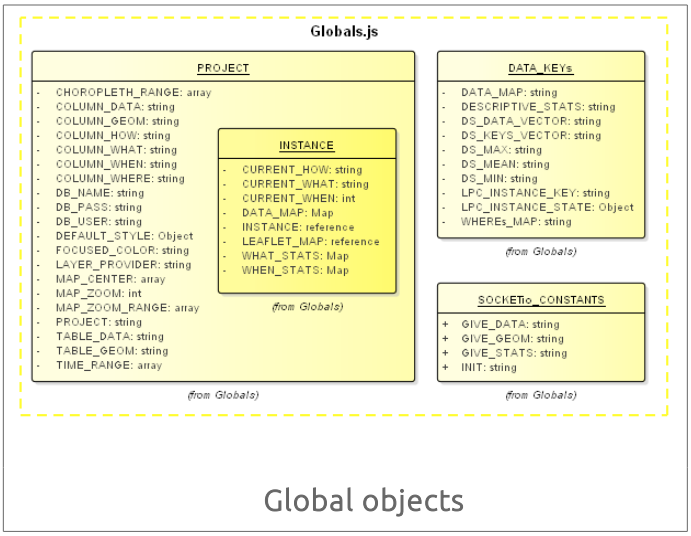
3.2 Shared Global Objects

There are some objects, such as the current when, where and data set, that are constantly changing by user interaction and are required for multiple controllers. Passing these objects back and forth between controllers could produce synchronization problems and memory leaks, besides an unnecessary overhead.

Issues arise when one controller stores a reference to an object that could be changed for another controller. The controller with the wrong reference not only misbehaves but also the old object can not be destroyed by the garbage collector.

To overcome this, *LapsoCarte* uses a simplified *blackboard pattern* for compile-time constants and run-time objects. The objects are indexed using global constants, which allows them to be reached from any controller and eases code refactoring. Unlike standard *blackboard pattern* implementations, all the controllers accessing the global objects run in the same thread, so concurrency is not a problem.

All the global constants and shared objects reside in the *Globals.js* file in the project root. This file exports three objects used in both server and client code:



Global objects

- The *PROJECT* object: This object defines all the project-specific variables used across all controllers. The *PROJECT* object acts as a single point of intervention to easily adjust *LapsoCarte* to different data sets. It is a *compile-time* object in the sense that it is defined *before* the application execution.
- The *INSTANCE* object: This is a run-time object stored inside the *PROJECT* object. The *INSTANCE* object is the *blackboard* that contains the references to the objects used by multiple controllers.
- Data keys object: This object defines the keys used to save and retrieve the elements in the *INSTANCE* object.
- Socketio constants: This object contains the strings used by server and client *SocketioControllers* to identify the messages sent through the socket.

3.3 Putting it all together

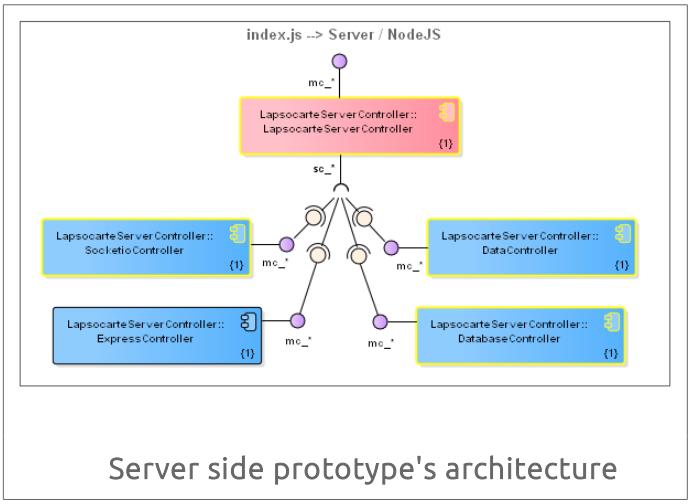
The color convention for the next graphs is:

- Controllers:
  - Red controllers are mediators.
  - Blue controllers are modules.
  - Green controllers provide support code used by multiples controllers.
  - Aqua controllers are embedded modules whose code resides in its parent controller.
- Globals:
  - Yellow-border controllers access or update elements in the _PROJECT_ or _INSTANCE_ objects.

3.3.1 Server side architecture

The server side has five controllers: four modules and one mediator. The *ExpressController* handles the Express Framework which acts as HTTP server. The *SocketioController* drives the Socket IO library for socket communication between server and client. The *DatabaseController* connects and makes the queries to the PostgreSQL database.

The *DataController* builds the hierarchical data structure and calculates descriptive statistics for each level.



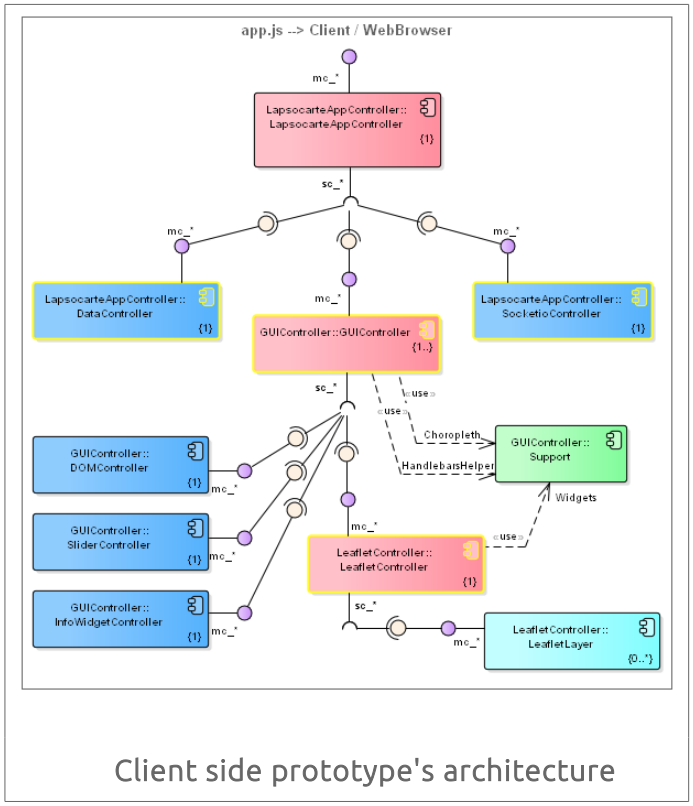
Server side prototype's architecture

The *LapsoCarteServerController* acts as mediator between the four modules. In particular, the *server controller* initializes the modules in order so the web server only accepts connections when all modules are ready.

3.3.2 Client side architecture

The client side has ten controllers: six modules, three mediators and one support controller. *LapsoCarteAppController* is the mediator for the client's backend. It syncs the client-side communication and data modules with the user interface.

The *GUIController* handles the client frontend, synchronizing the visual elements on user interaction. The next code segment shows the *GUIController* actions when the user fires a mouse over event on an element in the *Where* dimension.



Client side prototype's architecture

The next code segment shows the key elements of the prototype's architecture:

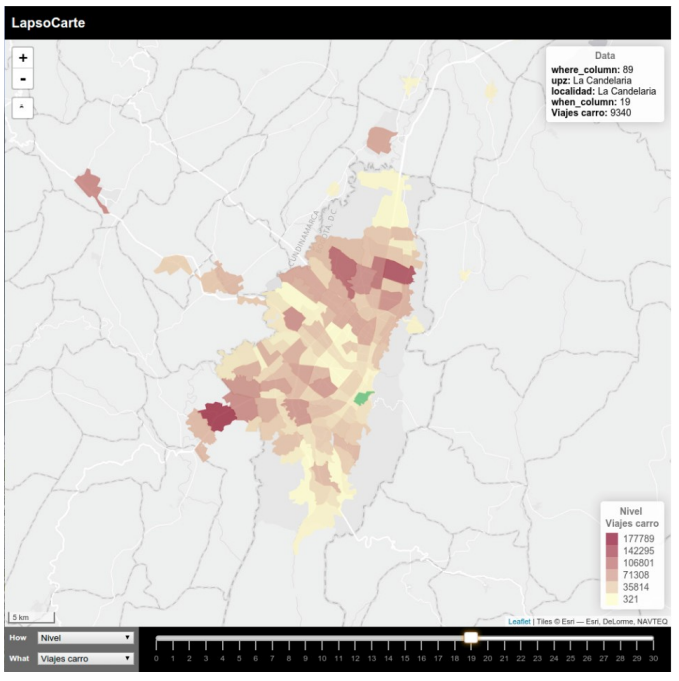
1. A controller captures an event and notifies its mediator.
2. The mediator handles the event logic:
    1. Retrieves configuration parameters from the global PROJECT object.
    2. Retrieves current state variables from the global INSTANCE object.
    3. Retrieves current state variables from the global INSTANCE object.
    4. Calls the required methods on its *SubControllers*.

```
/*
    The LeafletController calls this GUIController's method when
    an instance of its child LeafletLayerController recive a mouse over
    event. The gid parameter is the LeafletLayerController WHERE dimension id.
*/
sc_spatialObjectOver(gid) {
    // Get color for focused elements from the global PROJECT object:
    let color = glbs.PROJECT.FOCUSED_COLOR;

    // Tell LeafletController to change the color of the object:
    _leafletController.mc_colorGeometry(gid, color);

    // Get the properties for the given WHERE:
    let info = Object.assign({}, geometriesMap.get(gid)['properties']);

    // If there is data for the current state:
    if(instance[DATA_MAP]) {
        // Retrieve the current WHEN from the global INSTANCE
        // and add it to the object
        info[glbs.PROJECT.COLUMN_WHEN] = instance[CURRENT_WHEN];
        // Retrieve the data for the current WHAT in the current WHEN
        // for the selected WHERE
        info[instance[CURRENT_WHAT]]   = instance[DATA_MAP].get(gid);
    }
    // Display the information in the InfoWidgetController
    _infoWidgetController.mc_updateInfo(info);
}
```

Client side's architecture code sample

4 Graphical Interface

LapsoCarte features a simple and intuitive user interface:



LapsoCarte Graphical User Interface

The *How* and *What* dimensions selectors are located on the bottom left corner. Once the user selects a *How* and *What*, the application loads the *When* and *Where* dimensions.

The *When* dimension is represented as a slider on the bottom right of the window. The slider adjusts its scale to graphically show the available *Whens* for the selected *How* and *What* dimensions. The user can interact with the slider using the keyboard's right and left keys, the mouse or by touch if using a touch-screen.

The *Where* dimension is represented as a map in the center of the window. Each *Where* is drawn as its corresponding geometry. The color of the geometry is calculated from the spatial-temporal data using a linear choropleth function. The function minimum and maximum are the absolute min and max values for the whole *How-What* data set. The user can navigate the *Where* dimension using the standard pan and zoom actions through the keyboard, mouse or touch gestures.

5 Implementation

LapsoCarte code is written in *ECMAScript 6* and uses *Babel* for backward compatibility. *LapsoCarte* implementation is based on a set of popular open source libraries and tools. The backend relies on *PostGIS* as a database engine. *NodeJS*, *Express* and *Socket.io* are used as a server-side platform. *jStat* is used by the

DataController to calculate descriptive statistics.

The frontend uses *LeafletJS* to represent the *Where* or spatial dimension. *noUiSlider* handles the *When* or temporal dimension. *jQuery* is used to manage web browser events, *Less* to compile the CSS stylesheets and *Handlebars* to manage HTML templates.

6 Use Cases

LapsoCarte usage cycle has been tested with two different data sets:

6.1 Generic Schelling Model

The Schelling's Segregation Model was proposes by Thomas Schelling in 1971. In the model there are dwellings, individuals and populations. Each individual belongs to a population. On each iteration every individual decides if stay on his current dwelling or move to another one. The decision depends on the number of neighbors of his same population, respect to a given tolerance factor.

A basic Schelling Model was implemented using JavaScript. Dwellings were the 43225 blocks of the city of Bogotá. Each dwelling was randomly assigned to a given population. A fixed tolerance level was set for all populations. A neighbor was any individual in radius of 1 kilometer of the current dwelling. If the individual moves, he randomly selects a new dwelling from the empty ones.

Tests made with the model's results dataset revealed two performance issues. The first one was related to an early proposed single-table database structure for the server side data model. The duplicate geometries's data consumed a significant amount of storage, RAM and bandwidth. Next prototypes solved this by normalizing the geometries information on the database.

The second identified limitation is related to the number of polygons that *Leaflet* can handle simultaneously. Tests show a significant performance drop when working over 3000 polygons. There is a series of proposed alternatives to deal with this:

- Migrate to the next *Leaflet* version once available for production. *Leaflet* version 1 is a major library update, which significantly improves performance, specially when working with polygons.
- Reduce geometries complexity on the server-side using the *PostGIS*'s simplify function. This function reduces geometries's complexity using the *Douglas-Peucker* algorithm.
- Implement a semantic zoom to reduce the number of simultaneously-displayed polygons.

6.2 VENSIM MARS Simulation

This dataset is the result of a *Metropolitan Activity Relocation Simulator (MARS)* model for the city of Bogotá. The model is implemented in *VENSIM*, a general purpose systems dynamics simulator. The output is a comma-separated file with the data series for ten different variables across 30 time periods. The spatial dimension is a shape of 127 districts of Bogotá.

The driver was implemented in Python and Bash. The Python parser takes the *VENSIM*'s csv output file and generates a SQL script which recreates the *main_data_table*. The SQL script for the *gometries_table* is generated using the *PostGIS*'s *shp2pgsql* tool. The bash-script automates the parser and *shp2pgsql* execution and populates the PostgreSQL database.

The application was deployed in a local server at Los Andes University. Researches of the *Urban and Regional Sustainability* group used *LapsoCarte* to visualize the *MARS* model's results.

Users valued the tool but evidence the need of:

- Add descriptive statistics for the displayed data.
- Add contextual graphics to support data analysis.
- Implement a sync side by side view to compare two data-sets.

7 Future Work

The user tests presented above, Generic Schelling Model and VENSIM MARS Simulation, give valuable insights on the future development of *LapsoCarte*. From the user experience perspective, a descriptive statistics table and support graphics are needed in order to analyze and contextualize data. A synchronized side-by-side visualization would help experts to observe correlations evolution on multiple-variable data sets. Performance issues must be solved in order to visualize data sets involving over a thousand polygons.

It would be interesting to implement a logger module to save the analysis session, so it could be recreated. Another useful feature to add would be a MEDET query builder. Finally, the application should have an authentication layer to access the visualization.

8 References

- Benjamin Bach, Pierre Dragicevic, Daniel Archambault, Christophe Hurter, Sheelagh Carpendale. *A Review of Temporal Data Visualizations Based on Space-Time Cube Operations*. Eurographics Conference on Visualization, Jun 2014, Swansea, Wales, United Kingdom. 2014.

[https://hal.inria.fr/hal-01006140]

- Günter Emberger, Paul Pfaffenbichler & Leopold Riedl. *MARS meets ANIMAP: Interlinking the Model MARS with dynamic Internet Cartography*, Journal of Maps, 6:1, 240-249, DOI: 10.4113/jom.2010.1079. 2010. [http://dx.doi.org/10.4113/jom.2010.1079]
- Marzieh Berenjkoub, Harsha Nyshadham, Zhigang Deng, Guoning Chen. *A Visual Analytic System for Longitudinal Transportation Data of Great Britain*. Department of Computer Science, University of Houston, Houston, Texas, United States of America. 2015. [http://www.cs.uh.edu/docs/cosc/technical-reports/2015/15_01.pdf]
- Frank Bracco. *Mapping the City of Houston's Electricity Usage Over Time* (blog). The City of Houston Official Site. 2014. [http://performance.houstontx.gov/content/mapping-city-houstons-electricity-usage-over-time].
- Camilo Nemocon. *MEDET - Modelo para la exploración de datos espacio-temporales*. Master Thesis, Systems and Computing Engineering Department, Los Andes University, Bogotá, Colombia. 2015.
- Addy Osmani. *Learning JavaScript design patterns*. Sebastopol, CA: O'Reilly Media. 2012.
- Addy Osmani. *Patterns For Large-Scale JavaScript Application Architecture* (blog). Addy Osmani Blog. 2011. [http://addyosmani.com/largescalejavascript/]
- Schelling, Thomas C. 1971. "Dynamic Models of Segregation." Journal of Mathematical Sociology 1:143-186.