Department of Management Science & Technology

Advanced Topics in Software Engineering

Team Hypergraphs:
Charalampos Tsiligiannis (3130208)
Aggelos Gickas (8100035)

Contents

1. Project Understanding

1.1. Project Presentation

The project we chose to contribute to during the semester is Gephi,an open-source project started at 2007 by Mathew Sebastian.Gephi is an interactive visualization and exploration platform for all kinds of networks and complex systems dynamics and hierarchical graphs. It's usage is of outmost importance in many applications such as: Exploratory Data Analysis, Link Analysis, Social Network Analysis and Biological Network Analysis. As far as the technology of the project is considered, Gephi is implemented in Java programming language, using OpenGl platform for the visualization part.

More specifically, Gephi supports the following features :

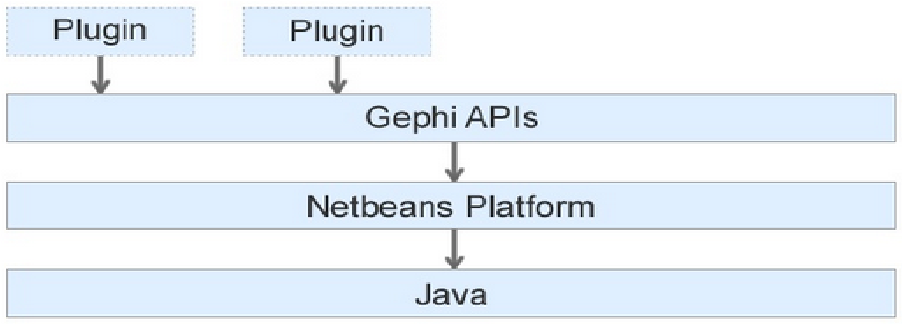
- **Ergonomic interface**: based on NetBeans UI
- **High-performance**: built-in 3D rendering engine.
- **Native file formats**: GDF (GUESS), GraphML (NodeXL), GML, NET (Pajek), GEXF and more.
- **Customizable by plugins**: layouts, metrics, data sources, statistics, manipulation tools, rendering presets and more.

1.2. Architecture

Gephi's architecture is designed and developed using the MVC (model view controller) framework and can be described modular and extensible. The architecture from an abstract view is structured in such a way that external and internal API's are integrated with the core of the application to mainly provide:

- Performance, Usability and Modularity
- An easy way to use and extend
- Easy implementation of external plugins
- Inexpensive and fast way to maintenance

The picture below displays Gephi's architecture in an abstract view



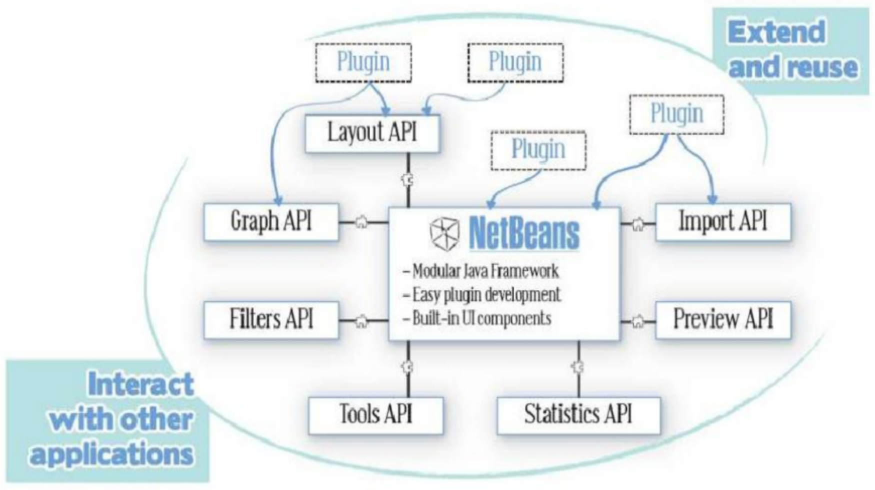
Gephi consists of multiple modules, which depend on each other through well-written APIs. Modules expose public APIs and depend on each other without circular dependencies, in that way plug-ins are allowed to reuse existing APIs, create new services or even adopt an improved implementation.

An important distinction must be made between SPI and API in order to understand Gephi's architecture better.

- <u>SPI (Service Provider Interface)</u>
  - SPIs define how Gephi can be extended.For example a plug-in extends the Layout interface to add a new layout.

- <u>API (Application Programming Interface)</u>
  - Each API has a role and a set of packages other modules can use.For instance a module call the Layout API to run a layout algorithm for 100 iterations.

The picture below displays Gephi's architecture in an integrated way



2. Contribution

2.1. Clarifications

Before proceeding to the detailed description of our contribution we would like to emphasize some aspects the project. First we noticed that there has been no activity (commit, issue or report) on the project for the past eight months. Moreover, after we decided the ways in which we would contribute, we started working on the branch of the latest experimental version of Gephi **(0.9)**. Unfortunately it turned out to be quite unstable, making it impossible to test our changes. After a discussion with the assistant supervisor of the course Mr. Marios Fragkoulis, we proposed to begin developing our own contribution in the branch of the latest stable version of Gephi **(0.8.2)**. Finally a substantial issue that impeded our endeavour was the bad design decisions of previous contributors. Consequently,we were unable to provide unit tests for our code and hence we were forced to manually test our algorithm with various sample graphs for the third feature of the project. Generally due to many problems in the architecture of Gephi we had to do lot of refactoring in order to implement our features in respect to good design standards.

2.2. Contribution Selection

After reading the idea list of the project's wiki on GitHub, we selected a wide variety of features concerning both user experience and functionality as well. Despite having the freedom to implement our own ideas, we decided to select some of the issues that had been already proposed or reported by the community. After group discussion we concluded we will mainly focus on functionality, selecting two main features and a small feature regarding user experience. We also created an implementation of Kruskal's algorithm for finding the minimum spanning tree, but unfortunately we did not manage to thoroughly complete it before the final deadline.

The following list displays the selected features accompanied by a brief description for each one:

- Color increases incrementally when painting nodes on graph
  - Description : User has the ability to paint various nodes on the graph according to his/her will.
  - Problem : Color is increasing incrementally in a way that multiple clicks are needed to fully paint the surface of node.
- Saving shortest path in attribute column in Data laboratory
  - Description : Gephi provides a data laboratory table where statistics measures and attributes considering nodes and edges are saved.User has the ability to request a shortest path from a source node to a target node.
  - Problem : Need for separate column to save the shortest path the user selected in the source's node row.
- Edge support for centrality metrics
  - Description : Gephi provides a variety of quantitative methods for centralities metrics.
  - Problem : Algorithm is implemented only for unweighted graphs.

2.3. Contribution Design & Implementation

- Color increases incrementally when painting nodes on graph

That was an easy feature to implement, it just needed to change the way RGB values are assigned when a node is pressed. More specifically we changed the method called pressingNodes in the Painter class in the ToolsPlugin module.

For the remaining two features we had to do some changes in the AlgorithmsPlugin module and more specifically we had to refactor the classes AbstractShortestPathAlgorithm, BellmanFordShortestPathAlgorithm and DijkstraShortestPathAlgorithm. In means of architecture design this was necessesary to achieve modularity and high performance. It was also essential in order to create some additional functionality through methods which were implemented and added later.

- Saving shortest path in attribute column in Data laboratory

  For this feature we created two new methods createShortestPathLabel updateShortestPathColumn in the ShortestPath class in the ToolsPlugin module. The first method creates a string with the labels of the nodes which are included in the shortest path and the second method updates the shortest path attribute column. The source node's row in the table, is where the string is inserted after computed.

- Edge support for centrality metrics

  Before proceeding to our main feature we needed to make huge refactoring changes in the class GraphDistance in the StatisticsPlugin module. After the refactoring was completed it was much easier for us to read and understand the code. We added a new method to calculate the metrics named calculateWeightedMetrics and many utility methods to complete the feature (saving metrics in attribute columns and assigning class's fields).

For all the features we have created javadoc comments in order to help future contributors to understand our code.

3. Team Cooperation

As a team we established some cooperation principles. Each change was preceded by extensive discussion. Each member presented his opinion about how the change should be designed and implemented. Disagreement was rare, but in case there was one, we would always reach a fruitful comprise by reminding each other the fundamental software design principles. Finally, acknowledging each other's strengths and weaknesses during the semester, not only enhanced our contribution quality, but also provided breeding ground for individual intellectual development throughout the activity.

4. Organizing on GitHub

To implement our changes we forked the project to our own repository on GitHub. For each feature we created a separate branch for safety and functionality reasons. During each week through phone or skype calls we planned our weekly contribution, in order to finalize and commit our changes on Sunday.

Before any code commit was pushed to a branch, we discussed its content and suitability. After completing the features we didn't make any pull requests for we are the sole active fork currently on GitHub. A detailed description of our progress during the semester is listed below For each branch created, we display the commits we pushed to the repository.

Links of our commits per branch:

- painter-tool

  Change the way the painter tool works the color is now applied immediately

- algorithms-plugin

  Minor refactoring changes

  Add method for returning the nodes of the s-p

- shortest-path

  Save shortest path label in an attribute column

- average-path-length

  Refactoring changes in the metrics calculation

  Add method to check if graph is weighted
  Update pom.xml
  Add support for betweenness centrality in weighted graphs
  Add support for closeness in weighted graphs
  Add support for eccentricity in weighted graphs
  Add method for calculating graph fields
  Add support for reports in weighted graphs
  Minor refactoring changes
  Fix closeness centrality calculation
  Add Javadoc comments
  Minor refactoring changes

- minimum-spanning-tree ***(Incomplete feature)**

  Add Kruskal algorithm class
  Add utility functions for the algorithm
  Add method to check for cycle creation
  Add execution and support method
  Add method to return MSP