Architecture Design

Group PL4
17 June 2016

Arthur Breurkes	4398033
Ricardo Jongerius	4379500
Niek van der Laan	4296915
Daphne van Tetering	4375165
Niels Warnars	4372069
Ties Westerborg	4386817

Context coordinator: Dr. T. Abeel
Teaching assistants: Mr S. Huisman MSc
Mr J. Linthorst MSc
Mr J. Hommes

Table of Contents

1. Introduction

This document serves to describe the architectural design of the multi-genome visualizer built for the Programming Life context project that serves as replenishment of existing single-genome browsers.

1.1. Design goals

During the development of the DART-2N visualizer, a number of design goals have been specified. The aim was to follow the corresponding quality standards at all time. These design goals have been drafted to retain a level of quality.

1.1.1. Usability

The DART-N visualizer aims at users who are considered experts in the field of genome research. For this target audience, only a limited set of tooling is available to visualize and compare mutations in data sets consisting of hundreds of genome samples. Most tooling is only able to visualize single genomes or does not have the ability to show mutations in a graph-like setting. Therefore, it is important that the visualizer provides correct functionalities that satisfy these customer needs. A consequence of having a specific target audience entails that using the visualizer may not be as straight-forward for a layman as it is for an expert. During the development sprints continuous feedback from the customer has been used to improve the product. Customer satisfaction ensures the integrity of the created software.

1.1.2. Scalability

The first version of the visualizer was designed for a small data set of ten genome samples. This data set was used to finetune functionalities and optimize the graph processing. The final goal is to achieve adequate scalability so that datasets up to multiple hundreds of genome samples can be processed and analyzed with loading times which do not exceed the one minute mark. More specifically, data will be visualized in an efficient way, by only processing and visualizing relevant information with regard to the zoom level.

1.1.3. Maintainability

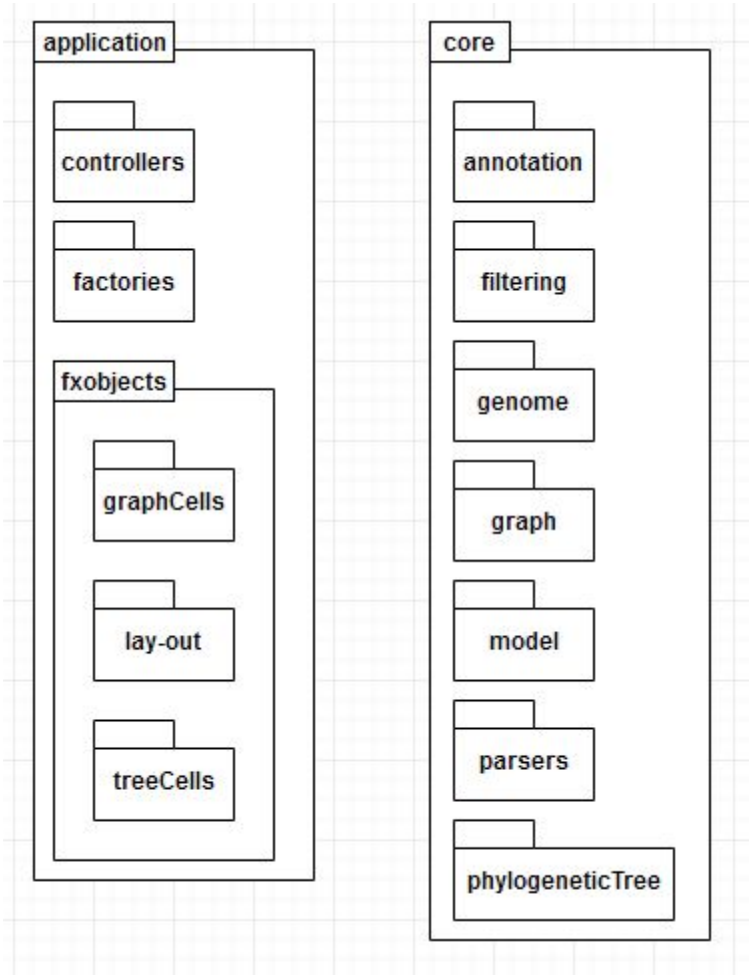
While implementing the customer wishes, a clear structure was tried to be maintained. To do so, the aim was to keep code well documented and structured in a clear way, while the individual core classes are being covered by appropriate test cases that cover all non-complex code structures. On the short-term, this approach has the advantage that bugs and defect can be located and resolved in an early stage of development. On the long-term high maintainability aids future development of the product by guaranteeing a level of code correctness.

2. Software Architecture Views

In this chapter, a sketch of the architecture of our system will be discussed. To do so, first the package decomposition is covered, followed by the core functionalities and an overview of code quality improvement tactiques.

2.1 Subsystem / package decomposition

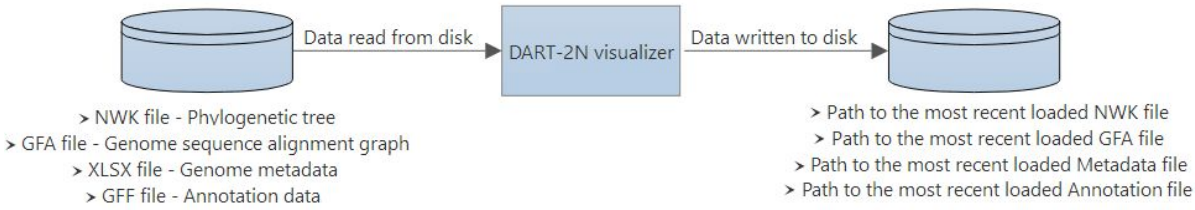
Classes are stored in packages to keep an overview. The following packaging structure has been specified:



In this set-up the application package contains all of the GUI elements and controllers and the core package all non-gui functionality.

2.2 Persistent Data Management

The program deals with several different manually loaded data files. These are used to generate node and edge data. This node and edge data is only stored in memory and is never written to disk storage. The only data that is written to disk are the names of recently loaded data files. This allows for quick file selecting when the program is executed every time, from off the first time. All read and write operations take place in the Main Controller.



2.3 Concurrency

Data redundancy
During the execution of the program, data is never stored redundantly in memory. As the underlying architecture of Java ensures that objects are created once and later only referenced to using a pointer, ensures that objects shared between classes and methods are always up to date.

2.4 Core functionalities

A number of functionalities is most important for a proper usage of the visualizer, these features include:

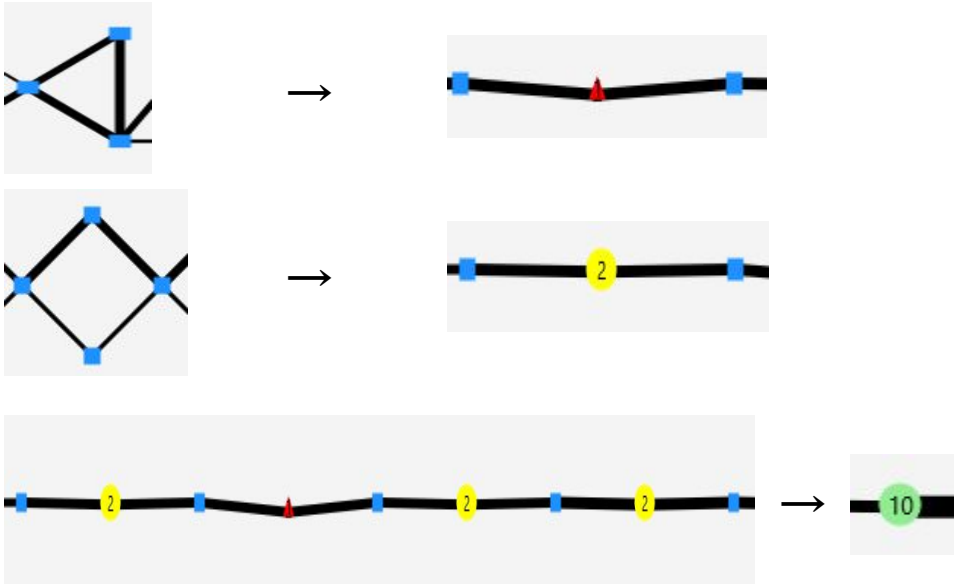
- **Parsers**
  A user has the ability to load the following files:
  - GFA file containing genome data of multiple samples.
  - NWK file containing the phylogeny of the data set.
  - GFF file containing annotation data.
  - XLSX file containing metadata of the data set.

  On selection, a file's data is read from disk and further processed.

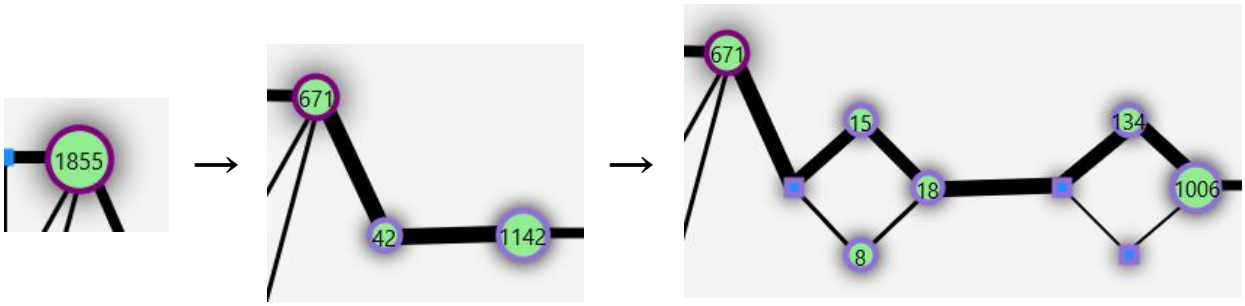
- **Graph reducer**
  Visualizing the entire graph - which can contain up to several hundred-thousands of nodes - is not feasible. Therefore, after the base graph is loaded via the Graph Parser it is collapsed in the Graph Reducer. Bubble and indel mutations are collapsed into special bubble or indel nodes and if multiple nodes are spanned horizontally they are horizontally collapsed into a collection node. By recursively collapsing mutations and bubbles multiple zoom levels are created. These zoom levels are stored in the Model. Inside of the Graph class the processed (bubble) nodes are taken from one of the zoom level maps and placed back in the Model as GUI Cells.

- **Graph and semantic zooming**
  The graph view shows multiple genomes in a node-like setting. At the top view only the cells are shown from the most collapsed level map, whereas the lowest zoom level map shows all nodes in the graph.

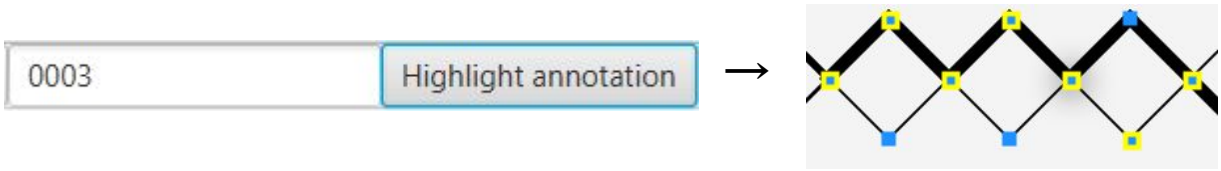
A user can scroll from to the top view down to nucleotide level. Whenever a scroll event occurs the Graph Controller switches the scene to a higher or lower zoom level by loading a new Model containing a different set of Cells. When zooming out certain node structures are no longer shown and replaced by a new symbol that represents a bubble, indel or horizontally collapsed collection of cells.



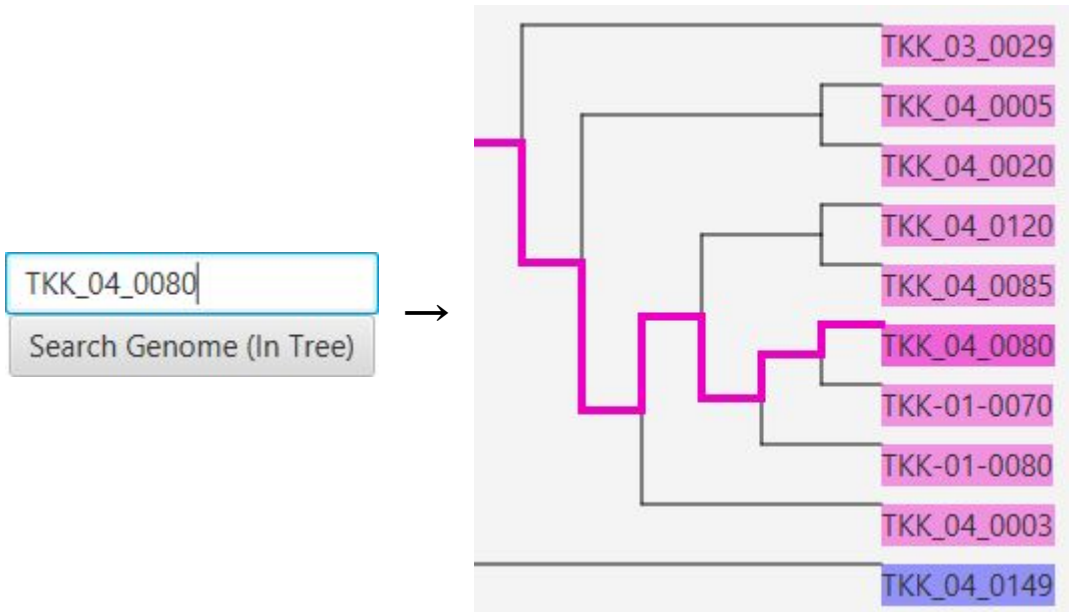
Whenever a user selects a node and zooms, focus will be kept on the nodes forming the original upper bubble.



- **Annotations**
  If an annotation (GFF) file is loaded from disk the Annotation Processor will match the annotations read from file to nodes present in the lowest zoom level. At the lowest zoom level this gives the user the ability to select an annotation specified for the reference strain. Search actions are processed by the Main Controller which will together with the Annotation Processor validate the input and highlight the Cells that represent the nodes in an annotation. A user can search on both the ID of an annotation and on its full name.



- **Phylogenetic tree with search / filter options**
  The phylogenetic tree shows the evolutionary relationship between genome samples based on single nucleotide mutations. For easy navigation, the user can select one or multiple genomes in the tree which can also be filtered in the graph. The Tree Mouse Handler in this case ensures that the path from the selected Genome up to the root of the tree will be colored. A search for a Genome is handled by the Main Controller.



2.5 Code Quality

To maintain a certain level of code quality, the following aspects have been covered:

- **Pull-Based Development**
  At all times, new features had to be developed in separate branches. Whenever a feature was considered done, the feature could be merged into the project via a pull request. Whenever a pull request was opened, code quality and correct functionality of collaborating components was checked by at least one of the team members. If the request got approved, the branch could be merged into the master. This way a stable version of the application was available at any given time.

- **Analysis**
  In addition to code reviewing on pull requests, code was written according to common coding conventions as much as possible. To enforce this method of development well known tools as CheckStyle, FindBugs and PMD have been used.

3. Glossary

Annotation

Data format indicating which nucleotides in a genome are responsible for the creation of a certain protein.

Genome sequence alignment graph

A graph representing the genome data of multiple DNA samples in such a way that overlapping pieces of DNA sequence form shared nodes. Each genome is represented by a unique path through the graph.

Metadata

Data about the tuberculosis patient and the drugresistency of the collected TBC sample.

Multi-genome browser

Piece of software that can be used to explore and compare the genome sequences of multiple DNA samples at the same time.

Phylogenetic tree

A tree showing the evolutionary relationship between multiple TBC genomes. The tree used in the project is an approximation based on single-nucleotide mutations.

Single-genome browser

Piece of software that can be used to explore the genome sequence of a single DNA sample.