Emergent architecture design

Byzantine Generals

TU Delft

Preface

Ali Smesseim, 4386248, asmesseim
Samuel Sital, 4225139, ssital
Kamran Tadzjibov, 4280784, ktadzjibov
Ravi Autar, 4361172, raviautar
Adam el Khalki, 4348435, aelkhalki
Byzantine Generals
Delft, April 2016

Contents

1 Introduction

This document will represent the architecture of our context project Programming Life. We will update this document each sprint to keep it as recent as possible. What we try to achieve with this project is to make researchers life more easy by developing a tool to visualize different DNA sequences in order to research diseases as Tuberculosis more thoroughly.

1.1. Design goals

We need design goals in order to achieve the purpose of the application. These design goals contribute to the development of the application.

1.1.1. Availability

The product must always be working. It can be found at the master branch so that the client can always give feedback that can be used to adjust the product to the clients needs.

1.1.2. Interactivity

An efficient way has to be provided to visualize DNA sequences in an interactive way.

1.1.3. Scalability

The product must handle data of more than 6000 Genomes. It must not take too long or a reminder system has to be provided to remind the user if the data is processed.

1.1.4. Maintainability

We would like to maintain the product easily. This is realised by providing the right documentation. It should be open for extension and closed for modification. We try to follow the SOLID principles.[1]

1.1.5. Useability

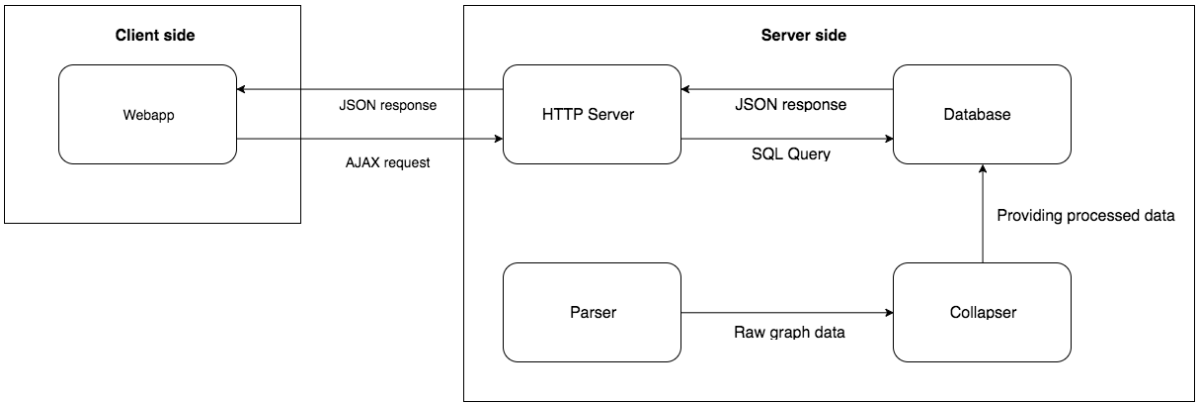
The users of the system are biologists. They sould be able to use the product easily and know how to use the product.

[1] https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)

2

Software architecture views

2.1. Overarching architecture



The primary model that is used is the client–server model. The reason this model was chosen, is because it leads to highly decoupled modules. For example, the client could be changed completely without having to adjust the server, and vice versa. Another advantage of this model is that the team could be split up into two parts. One part worked on the client, and the other part worked on the server.

The client is a web application in the browser. In the experience of this team, the rate of development of a web application is far greater than the rate of development of a standalone application. Primarily the weakly typed nature of JavaScript, and the massive amount of JavaScript libraries aid the rate of development.

The main programming paradigm used at the client is data-driven development. Two reasons dictate this choice, the first of which is that the main purpose of the client is visualizing data, for which data-driven development is a exquisite choice. The second reason is that the most popular graphing library for JavaScript, D3.js, encourages data-driven development. In fact, D3 is an abbreviation for "Data-Driven Documents".

Communication between the client and the server is interfaced using a formal API. Other than making the client and the server decoupled, this interface also allows data recovry from the server no matter from where you are calling it.

The server's main purpose is to derive information and to retrieve the information from the database. The database is populated with the information that is to be displayed, once. After the database has been populated, then the only purpose of the server is to retrieve the information from the database. Needless to say, the hardest part of this project was to derive information from the datasets.

The provided datasets define a graph, and metadata about the nodes and edges of this graph. Since these graphs can contain tens of thousands of nodes, and many more edges, it is not meaningful to display the entire graph, since meaning of the graph can easily be lost in the details. Instead, the details should be filtered away if the user wants a high level overview, and the details should be shown if the user wants more information about one specific part of the graph. To make such an algorithm that can filter away needless details was the main struggle of this project. Mainly because the structure of the graph is not formally defined, and there are many edge cases that need to be dealt with.

The server is entirely developed in Java. All group members are familiar with Java, and the preference for a statically safe typed and relatively fast language dictated this choice.

The software development process that in hindsight might have been the best option, is test-driven development. A more extensive test suite would also have helped, as regression has happened in this project. Test-driven development by nature forces the developer to write more extensive unit tests, which make up for a healthy regression test suite.

2.2. Sub-systems

There are currently 5 sub-systems. Below we will give more data about these sub-systems.

2.2.1. Server-side: web-server

We use a web-server to represent the processed data to the client. Initially the complete dataset is loaded, ofcourse with some optimization. Furthermore we use an API on the server. The API is documented on github. Actions of the user trigger calls to the server, which responds to the client with data.

Database
The database stores information about specimen and about the genomes belonging to the specimen. The database is used to store information about the graphs for each zoomlevel. The data requested from the database by the server is directly converted to JSON. There are two types of database connections. One is used for fetching data as described above. The other one is used to setup and fill the database with the data that is provided by the parser.

Parser
The parser module parses .GFA file files to the datastructure we are using. Furthermore it parses .nwk files to a datastructure that can be visualized by the webapp. Last it parses the metadata that is provided to a datastructure that can be written to the database.

Collapser
To provide the client the proper data to visualize the genomeset efficiently. The bubbles are recognized in here and the data gets ranked into a Zoomlevel. The data that is provided by the parser is first processed by detecting bubbles. Next the bubbles are feeded to the bubble collapser, which collapses the bubbles so that the data that is created represents the actual data. Last the bubbles get dispatched by the bubble dispatcher.

2.2.2. Client-side: webapp

The webapp is used for visualizing the data that is provided by the server. It calls the API that the server is based on. Furthermore it provides an interactive way of handling the application.

2.3. Hardware to software mapping

The product must be run on a workstation. The workstation should have Postgres and Java installed. It should have a minimum of 2 GB of RAM. Furthermore the webapp can run on the workstation or on any other computer that has a webbrowser. As we will try to use a database, data persistency will not be an issue.

2.4. Persistent data management

The persistent data that is used by the product mainly is the datastructure that represents a graph. The data the webapp is looking at and a reasonable amount of data around this webapp data will be saved in main memory. This is to provide fast response to the webapp.

2.5. Concurrency

As multiple users can use the product, concurrency is quite important. At this point in time we support using multiple users on one dataset. These users cant edit the data. This does not need any concurrency. In the future we would like to add the feature that multiple users can each use their one dataset and even dataset sharing.

Glossary

DNA sequences A nucleic acid sequence is a succession of letters that indicate the order of nucleotides within a DNA (using GACT) or RNA (GACU) molecule. 1

Genomes In modern molecular biology and genetics, the genome is the genetic material of an organism. It consists of DNA (or RNA in RNA viruses). 1

.GFA file This is the Graphical Fragment Assembly format, or GFA in abbreviation. Used to represent data of a set of genomes and their mutations. 3

Zoomlevel To visualize the data and to get situational awareness we use zoomlevels to represent the toplevel overview of a set of mutations. 3