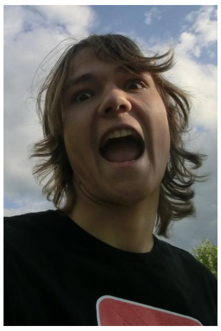
Contextproject Programming Life
Group 2 - GEVATT
Emergent Architecture Design

TU Delft



Ruben Bes
rbes
4227492

Mathijs Hoogland
mhhoogland
4237676

Jasper Denkers
jdenkers
4212584

Robbert van Staveren
rhvanstaveren
1527118

Willem Jan Glerum
wglerum
4141040

June 19, 2014

Abstract

This is the emergent architecture design for the Programming Life Contextproject, a second year course from Computer Science at TU Delft. It contains information regarding our design goals, used languages and used programs. It also explains our general composition of components and subsystems, as well as our data management and use of concurrency.

Contents

1 Introduction

This document provides information on the system that will be built during the context project programming life. We have also set up a couple of goals which will be followed as much as possible. The architecture of the product will be discussed in the form of high level components. These may be split up into subcomponents and/or subsystems.

1.1 Design goals

Our product will focus on:

- Availability

  We are building this program using scrum, a development process consisting of sprints of a predetermined length. At the end of each sprint, we plan to have a working system, with more functionality being added each sprint. Using this approach, our customer can try the product after each iteration, resulting in feasible feedback which we can use to quickly alter course.

- Manageability

  Doctors will not have to manage anything besides adding or deleting patients, as they have no direct access to the databases or any data. They will only be provided visualizations to interact with.

  As for the software manageability, we are using checkstyle and JavaDoc comments to make sure all the code is in a correct and uniform layout while everything is also well documented.

  Since our application runs on a server and a doctor can access our data with a web page, no extra programs are required. The user will log on with his credentials and can select a VCF-file to upload to our server as well as provide some data on the trio.

- Performance

  As VCF-files are quite large, the data will be sent to the server while the doctor can still access the website. The data can be analyzed quickly by the server and the results will be returned and locally visualized.

- Scalability

  Our application will be scalable in a sense that multiple users can upload files. However the databases containing information about the genetic variations (STRING, dbSNP and CADD) will continue to grow and thus the computational power required to gather the information needed will also grow. This however is not the problem of our server, but of the host of the server.

- Reliability

  The application should be very reliable, as servers are generally always on. The user only needs to connect to the web page. To make sure the software does not suddenly break, it will be tested thoroughly. In case something does go terribly wrong, we are using GitHub with branches so changes can be reverted easily.

- Secureness

  Doctors are required to log onto our system via the secure web page, thus unauthorized access is prevented. The data sent over will only be saved for a short time, long enough for it to be analyzed. After this it is deleted and the results are submitted to the user and saved into the database for quick access later.

1.2 Programming languages and programs

The system features a web application which we can separate in a frontend and backend. The Play Framework is used to realize this. The main programming language used in the backend and in Play is Java. For templating, the Play framework also uses Scala. The frontend is based on regular web languages: HTML, CSS and JavaScript. We fasten frontend develop by using both JavaScript and CSS libraries: Bootstrap for visual aspects and jQuery for faster JavaScript development. Both jQuery and Bootstrap increase cross-browser compatibility.

2 Software architecture views

This section discusses the system's architecture. The first subsection explains the high level architecture, such as the packages used and how to make parallel development possible. The program is composed of subsystems which depend on one another, this will be explained in the second subsection. The third subsection focuses on the relation between the hard- and software. Finally, the fourth subsection discusses the management of data used by the product.

2.1 High Level Architecture

The system could be divided in several parts and these could be explained by walking through the program flow. Firstly, a VCF file is uploaded and processed. This is the first part of the system and based on the outcomes the visualisations are done.
These visualizations could be separated in three steps, each of them representing a big component of the system.
The first visualization is a overview of a whole patient. It shows all mutations found in a patients VCF-file and in fact offers the highest level of browsing. Besides a big overview it shows mutations differently based on potential danger. The CADD database is used to retrieve this information.
The second visualization shows the position of a mutation on genes. The information used for viewing the position is found in the dbSNP database.
The third visualization show relations between proteins. Information is retrieved using the STRING database.
These visualization form three big parts of the architecture. Every part is dividable into two sub-parts; visualization itself and the database information retrieval. Our team members can work independently on these three parts,

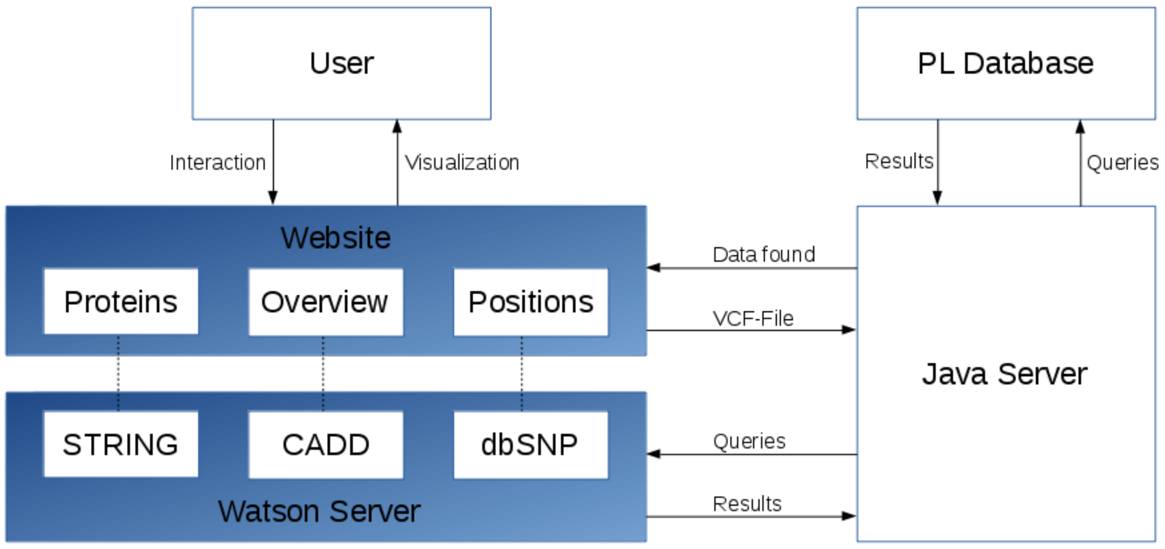


Figure 1: Subsystem decomposition of GEVATT

specifically on database or visualization.
Furthermore the system features a server to client with model view controller architecture, which is used by the Play framework. Play handles request from users.

2.2 Subsystem decomposition

- Watson server

  The Watson server owned by the TU Delft hosts the databases STRING[1], dbSNP[2] and CADD[3] and processes queries given by the Java server and returns the results.

- Java server

  The java server makes queries and sends these to Watson. It makes these based on the VCF-file given by the web page. It processes these and determines which data is to be retrieved. The data retrieved is then sent to the web page.

- PL database

  The Programming Life database contains the user and patient information. It is needed to log in and add or view patients.

- Website

  The website receives the VCF-file from the user and passes it on to the JAVA server. When it receives data back from the server it visualizes this for the user.

- The user

The user can interact with the website and pick a VCF-file to send. The web page outputs a visualization and the user can draw conclusions from this.

2.3 Hardware/software mapping

This system will only be used by one type of person, namely a doctor. This means that only one interface will have to be developed and used. A user needs to log on to a web page, after which he or she will be shown a page where VCF-files can be uploaded and analyzed. The web page can be accessed from devices connected to the Internet but is targeted at desktops.

2.4 Database Connectivity

- dbSNP

  The database dbSNP is used by the java server to get information on the SNP's found in the VCF-file supplied by the user. What is most often retrieved is the gene associated with this SNP so we can use it to search for information about this gene in the STRING database.

- STRING

  The STRING database contains information about genes and proteins and how these are connected. Using information retrieved from dbSNP, we can look at which SNP affects which gene(s) and what genes it/they affect.

- CADD

  The CADD database contains information on how damaging the mutation is. This can be used to determine whether a mutation is harmful or not, and so the doctor can see it might be the cause of a disease.

- Programming Life

  We use an extra database hosted at the webserver to store extra information for the web-application.
  The credentials of the users, the doctors, are stored in the database, the passwords are hashed and salted.
  Each user can have multiple patients which are linked through the id of the user. Some basic information is stored per user including the uploaded VCF-file. For every patient the mutations found in the VCF-file are stored as a single mutation, which is linked to each patient with the patients id.

To be able to test our application neatly we use a Repository and Service design pattern. This isolates the methods that require database access to a different class. The repository interface defines the methods and next we can implement this repository. And because of the interface we can have multiple kind of databases to store information, for example MySQL, PostgreSQL or an In-Memory database.

Next we build a service around this, so other classes can reach these methods through a service. Because of this setup we can easily mock the methods that

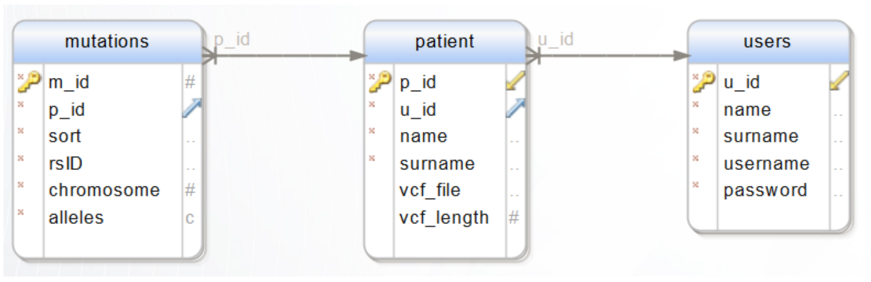


Figure 2: Programming Life database design

require database access, instead of mocking an entire database connection, which is not possible because of its static behavior.

2.5 Concurrency

At the moment we have no indication how computationally intense our calculations are, thus we cannot say for sure whether or not we will use multiple threads. Until we know more, our program will run on a single thread.

3 Glossary

- SNP

  Single Nucleotide Polymorphism, a single change in DNA compared to the reference genome.

- VCF

  Variant Call Format, a file containing all the SNP's and their ID's of a child and it's two parents.

References

[1] Andrea Franceschini, Damian Szklarczyk, Sune Frankild, Michael Kuhn, Milan Simonovic, Alexander Roth, Jianyi Lin, Pablo Minguez, Peer Bork, Christian von Mering, et al. String v9. 1: protein-protein interaction networks, with increased coverage and integration. *Nucleic acids research*, 41(D1):D808–D815, 2013.

[2] Stephen T Sherry, M-H Ward, M Kholodov, J Baker, Lon Phan, Elizabeth M Smigielski, and Karl Sirotkin. dbsnp: the ncbi database of genetic variation. *Nucleic acids research*, 29(1):308–311, 2001.

[3] Martin Kircher, Daniela M Witten, Preti Jain, Brian J O'Roak, Gregory M Cooper, and Jay Shendure. A general framework for estimating the relative pathogenicity of human genetic variants. *Nature genetics*, 46(3):310–315, 2014.