Graphical application for the statistical analysis of copy number variations in aCGH data

Thesis BSc Project IN3405

Final Report

version 1.0

Julian de Ruiter (1387367)
Chris Melman (1358359)
Rien Korstanje (1270389)
Stefan Dentro (1313320)
Nanne Aben (1369342)

DELFT UNIVERSITY OF TECHNOLOGY
EEMCS FACULTY
JUNE 28, 2010

Exam committee

Prof. dr. ir. Marcel Reinders
Ir. Jeroen de Ridder
Jan Bot MSc.
Drs. Peter van Nieuwenhuizen
Chris Klijn MSc.
Henne Holstege PhD.

Preface

Students majoring in Computer Sciences at the Delft University of Technology complete their bachelor course with a BSc project. In this project, groups of students autonomously execute a project of their choosing for client, in which they are to design a software product and deliver a functioning prototype.

This thesis is the result of the BSc project of the students Julian de Ruiter, Stefan Dentro, Chris Melman, Nanne Aben and Rien Korstanje. The project has been executed within the Bioinformatics research group of the Technical University of Delft for two external clients, the Dutch cancer research group NKI and the VU medical centre.

We would like to thank our clients Henne Holstege and Chris Klijn for the time and effort spent to enrich our insight into their particular field of expertise. We would also like to thank Marcel Reinders, Jeroen de Ridder, Jan Bot and Marc Hulsman for their support during the design and development of the product.

Julian de Ruiter
Stefan Dentro
Chris Melman
Nanne Aben
Rien Korstanje

Delft; June 25, 2010

Summary

This document is the final report describing the development process of a graphical, statistical analysis application of copy number variations in aCGH data. The document is intended to provide an insight in the development process used to produce the final prototype delivered by the project.

The project entailed the design and implementation of a graphical, statistical analysis application for aCGH data. The applications main functionality is the displaying of the corresponding aCGH profile, as well as the analysis of this profile in order to determine any significant copy number variations in the analysed genome. The application also identifies genes located in these significant regions and provides annotation data about these genes, which are retrieved from a variety of external databases. Other functionality included by the product are the export of the displayed data as well as the displaying of relations between the identified gene in a circular plot using the Circos application.

The above listed functionality was derived from the extensive list of requirements created during the requirements phase of the development. The creation of this list was supported well by the basic knowledge acquired during the orientation phase, which proved invaluable in the interviews with the clients.

The design of the system divides the system into a set of six subsystems, in which the database component is divided into a client-server component. The Ibidas system forms the server-side element of this component. The subsystem division is designed to keep the different components of the application compartmentalised and well-defined. The application also incorporates a multi-threaded design in order to keep the application responsive when performing lengthy tasks.

The main problems encountered during the implementation of the application was the size of the data, which required a number of optimisations even though the data structures were designed accordingly. The main performance problems were encountered in the visualisation of data, which were solved by the implementation of a customised and optimised view to draw the data efficiently. The product has been subjected to a number of automated and manual tests to verify it meets the set requirements of functionality and the expectance of the clients.

The product produced in this project meets the set requirements and has therefore been successful. The developed prototype is still a prototype demonstrating its possibilities however and leaves much room for improvement before it should be applied in actual use cases.

Contents

Chapter 1

Introduction

1.1 Purpose

This document is the final report describing the development process of a graphical, statistical analysis application of copy number variations in aCGH data. The document is intended to provide an insight in the development process used to produce the final prototype delivered by the project.

1.2 Scope

The document describes the design process of a graphical, statistical analysis application for aCGH data, codenamed Angita. The application is being developed for the NKI and the VUMC.

The application is able to import aCGH data from a local file and identify significant aberrations, so-called copy number variations, in the genomic data using various algorithms. It provides an interface in which a loaded aCGH profile can be viewed in a linear fashion. The interface also displays the various genes located in identified aberrant regions. Genes are annotated by various databases to provide more information about the gene in question. The application also provides a view which displays relations between the genes in the identified aberrant regions of the genome, allowing the type of relations to be shown can be chosen.

The application aims to allow quick analysis of the provided aCGH data by displaying a clear view of the aCGH profile, providing various algorithms to identify aberrant regions of the genome and displaying the identified regions in a clear manner. The displayed relations between genes in the identified regions provide the user with a significant amount of extra data, which can help identify non-obvious relations between the identified regions and corresponding genes.

1.3 Overview

The remaining pages of this document describe the entire development process of the afore mentioned application. The remainder of this chapter first provides a short introduction to copy number variations of which a basic understanding is required to understand the products application. Chapter 2 proceeds to introduce the clients concerned, followed by an analysis of imposed problems and a description of the required functionality to solve these problems. A

short overview of the product is given in chapter 3. Chapter 4 solidifies the required functionality by deriving a list of the requirements that should be satisfied in order to create a useful product containing the required functionality.

The design and implementation of the product is described in chapters 5 and 6 respectively. The design chapter elaborates on the chosen design for the various elements of the product as well as the decisions behind these choices. The implementation chapter proceeds to describe the actual implementation of these designs with a focus on any problems encountered in the development process and choices made to solve these problems.

The final chapters of this document analyse the entire process. Chapter 7 discusses the various test processes used, which are also defined in a separate document, and their results when applied to the delivered product. A short evaluation of the project process is given in chapter 8. The following chapter concludes the project by elaborating on the results of the entire project. Chapter 10 continues on this subject by providing recommendations for future work done with the developed product.

1.4 Copy number variation

Genomes vary from one another in various ways and it is this genetic variation that underpins the heritability of human traits. Over the past two to three years genome sequences from various individual humans have been sequenced, allowing studies to compare these sequences to each other and the base human reference genome.[12]

The studies allow estimation of the relative contributions of sequence (base substitutions) and structural variation (insertions, deletions, CNVs and inversions). The term CNV (copy number variation) is generally used to describe all quantitative variation in the genome, which includes tandem arrays of repeats as well as deletions and duplications.[8]

1.4.1 Origin

CNVs are generally either inherited or caused by de novo mutation. Such de novo mutations are generally caused by genomic rearrangements, usually occurring through deletions, duplications, inversions and translocations. Especially low copy repeats, which are basically small repeated sequences in the genome, are susceptible to such rearrangements, mainly due to their size and high similarity.[16] Another recently proposed mechanism for the cause of CNVs is fork stalling and template switching during DNA replication.[21]

1.4.2 Effects

CNVs can affect gene function in several manners. Functional loss can be caused by deletion or disruption of a gene, the effect of which depends on the CNV occurring an a recessive or a dominant allele and the cellular function of affected gene products. CNVs can also disrupt regulatory elements, generate novel products or disrupt other genes through position effects. However, some CNVs spanning multiple genes are known to have no distinguishable phenotype.

CNVs are associated with susceptibility or resistance to disease. Especially cancer cells can show relatively large numbers of CNVs as seen in non-small cell lung cancer cells, who generally display a relatively high EGFR gene copy number.[7] An example of higher resistance to diseases due

to CNVs is the CCL3L1 gene, in which higher copy numbers have been associated with a lower susceptibility to human HIV infection.[10] Copy number variations have also been associated with autism[22], schizophrenia[23] and idiopathic learning disability[15].

CNVs are also thought to be responsible for a substantial amount of variability in the human phenotype and behavioural traits, by affecting dosage sensitive genes.

1.4.3 Application

CNVs are currently being researched for various reasons, the most important being disease related. As mentioned in section 1.4.2, CNVs can have a substantial effect on a persons susceptibility or resistance to disease. Both these effects of CNVs are worth researching, as a CNV identified as a certain susceptibility can be used for medical diagnostic purposes and certain resistance CNVs may some day be able help in the prevention or curing of certain diseases.

Much research is currently aimed at identifying the effects of various CNVs. An example is the study of cancerous cells, in which generally many CNVs occur. These CNVs can be studied to identify frequently occurring CNVs in the specific cell types, as well as the impact the CNVs have on the cell and why these CNVs lead to a cancerous phenotype. This type of research is a part of the research currently being executed by the NKI client.

Another example of the use of CNV data is in the branch of medical diagnostics. An example of this use is given by the VUMC client, who analyses the genome of foetuses in order to determine its copy number variations. The found copy number variations are compared against databases of known copy number variations and their various effects, which allows the diagnostician to screen the foetus genome for known diseases. Based on the results of this process, the diagnostician can recommend a patient to terminate the pregnancy prematurely if the resulting child has a high probability of suffering a severe disease.

Chapter 2

Problem analysis

This project encompasses the development of a software product aimed at the analysis of CGH profile data in order to determine copy number variations in the sampled genome and assist the user in formulating a diagnosis based on these variations. The project consists of two related assignments, each executed for a separate client. The next section first provides a short introduction to both of the clients and their interest in the product, followed by a description of the feature set common to both clients. The third section explains any features specific to a client.

2.1 Client background

As mentioned in the introduction, the project consists of two related but distinct assignments. Each assignment is executed for a different client, the first assignment for the VUMC medical institute of Amsterdam, the second assignment for the NKI cancer research institute. This section provides a short explanation of both clients.

2.1.1 VUMC

The VU University Medical Centre (VUMC), is the university hospital affiliated with the Vrije Universiteit of Amsterdam in the Netherlands. It was created in 2001 by the merger of the Academic Hospital of the Vrije Universiteit with the medical school of the Vrije Universiteit. It is one of the largest and leading hospitals of The Netherlands. Tertiary care departments include advanced trauma care, pediatric and neonatal intensive care, cardiothoracic surgery, neurosurgery, infectious diseases and other departments. A medical emergency rescue helicopter is also affiliated with the hospital.

The VUMC client is a research institute within the VUMC, who performs CVS tests in order to determine CNVs (if present) in unborn children. The data generated by such tests is of a large magnitude, which hinders manual interpretation of the results of such a test. The client therefore requires an application which can analyse and display this data in such a manner that the results of such a test are clear for the researcher.

The assignment executed for the VUMC therefore focusses on the analysis and visualisation of data obtained from chorionic villus sampling. Chorionic villus sampling (CVS) is a form of prenatal diagnosis to determine chromosomal or genetic disorders in the foetus. It entails getting a sample of the chorionic villus (placental tissue) and analysing its genome in order to

find copy number variations. The analysis produces an aCGH profile, which is processed by the product to determine and display genomic copy number variations in a clear manner. The product should also display the various genes contained in CNV regions and provide the user with disease related annotation data, which supports the user in determining the effect of identified copy number variations.

2.1.2 NKI

The Dutch Cancer Institute (Nederlands Kanker Institituut, NKI) was founded in 1913. The NKI is a scientific research institute and specialised clinic, which focuses on the prevention and curing of cancer. The actual client is a bioinformatics researcher.

The NKI assignment involves the same analysis and visualisation procedures as the VUMC assignment (2.1.1), the main difference being the nature of the data under analysis. Data from cancer research is harder to analyse due to large amounts of mutations in the genome and differences in sampled cells (tumorous vs non-tumorous). The client uses both human and mouse genome data in his research.

The NKI assignment has an extra focus on visualising relations between genes in regions occupied by CNVs, which allows the user to identify relations between various CNVs, and often analyses multiple samples at the same time. Due to the focus being on visualisation of relations between genes, the gene annotation should also contain data concerning the functions of genes and any other properties that may relate one gene to another.

2.2 General product functionality

The most basic functionality of the product, as required by both of the clients, concerns the displaying and analysing of aCGH profiles. The product should be able to load the results of a aCGH analysis, plot the corresponding CGH profile and use a specified algorithm to determine significant CNVs in the analysed genome. Other functionality is the displaying of genes in the identified significant CNV regions and the providing of corresponding annotation data. Other required features are displaying sample and the analysis platform metadata to provide information about the loaded sample.

2.3 Client specific functionality

2.3.1 VUMC

The most important functionality specifically required by the VUMC client is displaying disease related annotation data. This annotation data is specifically required to be incorporated directly in to the application interface.

According to the client, the final product must also be very reliable, meaning it must produce consistent and correct results as the product may be involved in important decisions and recommendations concerning actual patients.

2.3.2 NKI

The most important aspect of NKI specific functionality corresponds with visualising relations between genes in regions occupied by CNVs. The product should therefore include annotation data concerning relations between genes and include a visualisation method to display identified relations clearly. The NKI client also requires the product to be able to load and analyse multiple sets of sample data simultaneously.

2.4 Possible problems

2.4.1 Heterogeneous data sources

One of the largest problems in the implementation of the product is the heterogeneity of the various data sources from which data must be imported, which concerns both the aCGH data and external annotation data sources.

2.4.1.1 aCGH data

aCGH data is provided in various data formats, which are specific to the used platform to generate the data. Inherent to the field of bioinformatics is the lack of standardisation of these processes and data interchange formats. Whilst attempts have been made to construct some sort of standardised data format, such as the MIAME specification[6], these formats are still too vague to allow construction of a single loading process. The loading process therefore has to be flexible enough to account for these differences, whilst still maintaining a single unified structure.

Examples of differences encountered between various required file formats are: XML files vs. text files, single data files vs. multiple data files, one sample per file vs. multiple samples per file.

2.4.1.2 External databases

All the required databases are structured differently. Each organisation uses its own fields and own identifiers, there seem to be no standards. Therefore problems could arise when mapping databases. Databases are going to be queried according to the output of a query of another database. Therefore a mapping that connects the databases together must exist. At the moment, it looks like it's going to work, but the validity of the results cannot be checked until the database, file loader and front-end work together.

2.4.2 Performance issues

Another problem in the implementation of the product is the large amount of data contained in the aCGH data files. A typical sample file contains upwards of 200.000 data points, for which the product must create and handle internal representations of the measured values as well as corresponding probe and platform data. Special care must therefore be taken to optimise the handling of this data, both in the loading of the data as well as the subsequent display and analysation processes.

2.5 Summary of objectives

The above analysis mentions a number of features to be supported by the product, which can be summarised in the following list of required functionality:

- Loading various types of aCGH data files.
- Display of loaded aCGH profiles.
- Analysis of loaded aCGH files to identify significant CNV regions.
- Identification of genes located in identified significant CNV regions.
- Annotation of identified genes with disease related data within the product interface.
- Annotation of identified genes with function related data and other data that may relate genes.
- Identification of related genes through annotation data and subsequent display of the gene relations.
- Loading and analysis of multiple data sets.
- Product must be reliable, due to the sensitive nature of its use.

The product should also be designed to tackle the problems mentioned in previous section, specifically the performance issues inherent to the large data sets and the heterogeneity of the various data sources.

Chapter 3

Application overview

This chapter contains a basic overview of the functions the product contains and how they are executed. As this document was completed before the final prototype was finished, minor discrepancies may exist between the actual product and this document. There are also minor differences in the graphical interface between the supported platforms (Windows, Mac OS X 10.5+ and Linux). The images in this chapter were made on the Linux platform.

3.1 Overview

Figure 3.1 shows the following view elements:

1. The overview bar, which shows an overview of entire CGH profile by plotting all the data points in the sample against the genome.
2. The main view, which shows a more detailed view of the data points that are again plotted against the genome. This view differs from the overview in navigation possibilities, which allow the user to zoom in/out and scroll through the data horizontally. The alternating black-and-white line represents the chromosomes and is labeled correspondingly. The length of an element in the line corresponds to the length of a chromosome.
3. The view that contains the genes that lie in identified significant regions. It also visualises data about commonly occurring CNVs, which is supplied by the DGV database.
4. The navigation toolbar, that enables the user to quickly zoom in on a region of interest.

View 2 and view 3 are connected, so any scrolling or zooming action in one of these views is automatically passed on to the other view. This ensures that these two views are always synchronised and the user can easily see corresponding genes and DGV-data in view 3 for the shown data points in view 2.

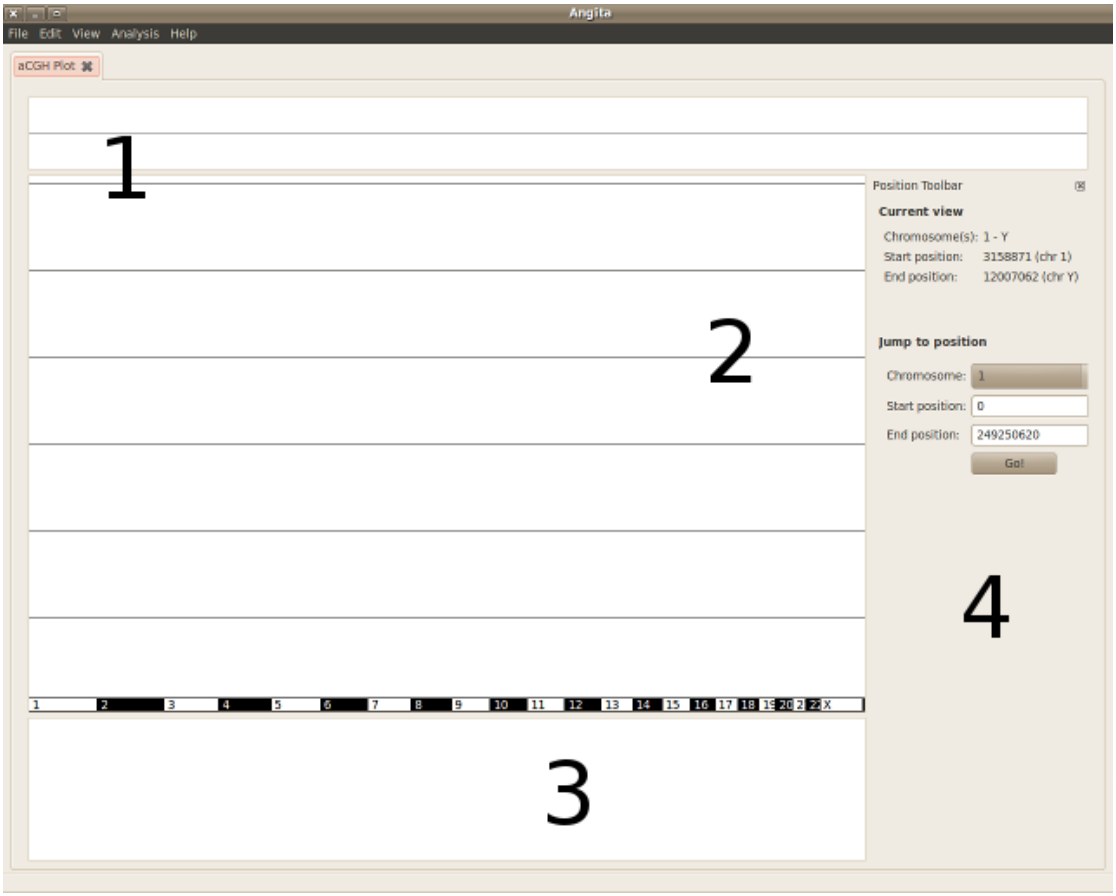


Figure 3.1: The product has just been started

3.2 Loading a sample

When the sample is first loaded, only an outline is given of the loaded data as is shown in figure 3.2. A more detailed view containing all data points is shown when the view is zoomed-in on a section of the profile, as is shown in figure 3.3. Zooming in also triggers the loading of DGV-data, which is represented by the shown green and red bars. A green bar indicates a commonly occurring gain and a red bar indicates a commonly occurring loss. The intensity of the colour corresponds to the number of gains/losses that were observed in DGV.

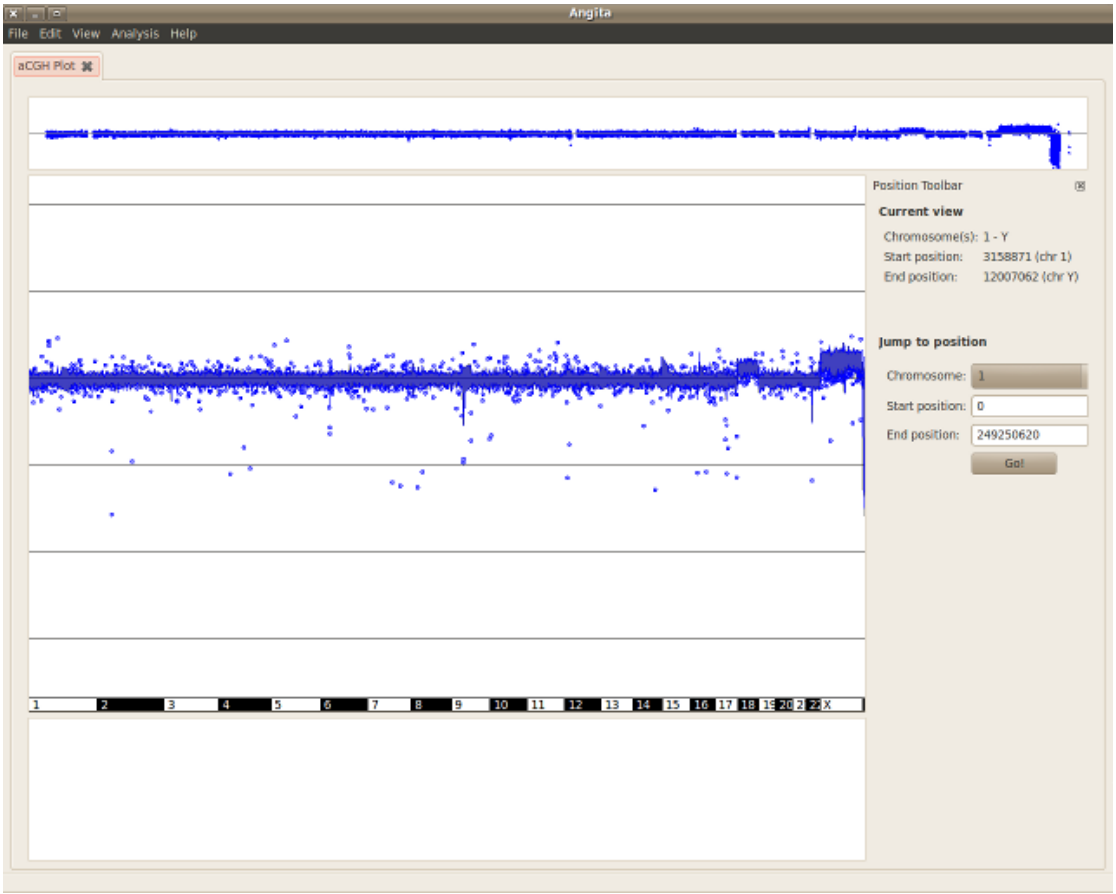


Figure 3.2: The sample has been loaded, the overview is created and an outline of the data is visible

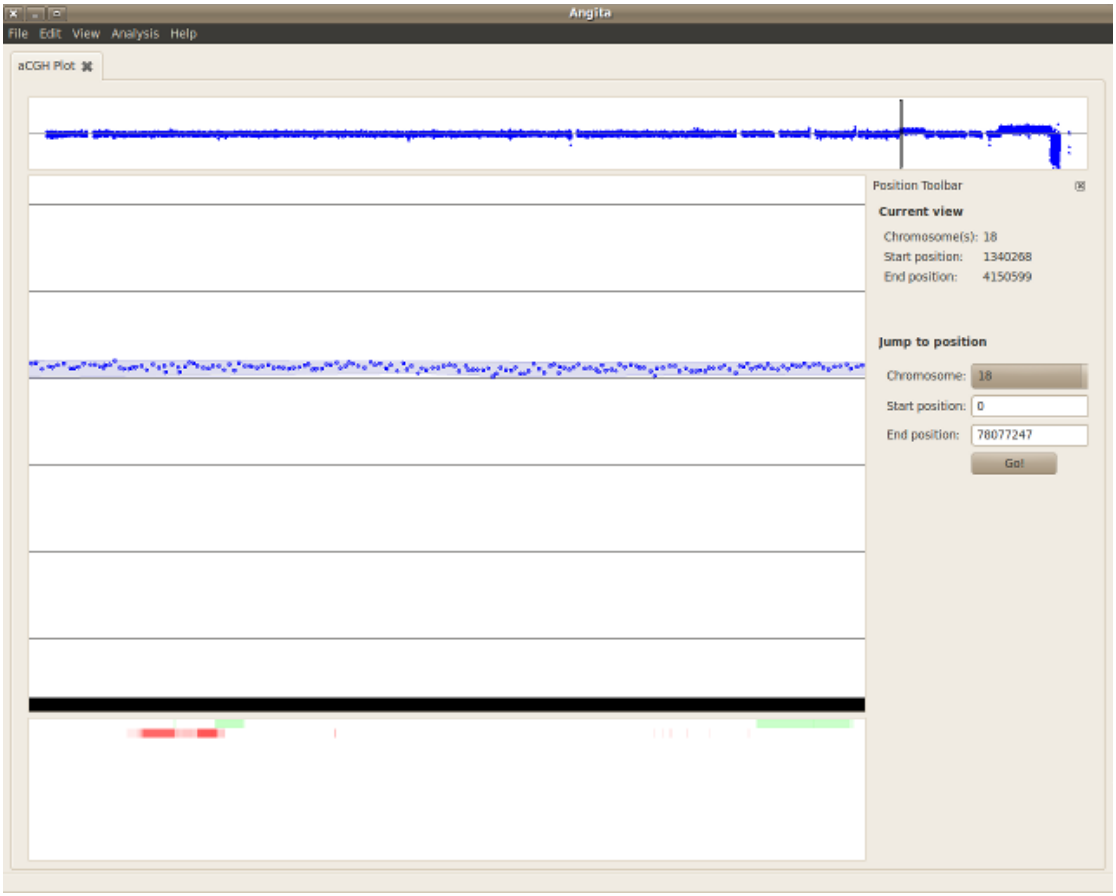


Figure 3.3: Zoomed in, showing data points and DGV data

3.3 Analysing a sample

After a sample has been loaded an analysis algorithm can be selected to identify significant regions in the profile. The product currently supplies to actual algorithm implementations: a threshold algorithm and an implementation of the KC-smart algorithm[14]. After selecting one of these algorithms, a new dialog is shown in which the user can enter parameters for the analysis. Confirming the parameter input results in the execution of the algorithm.

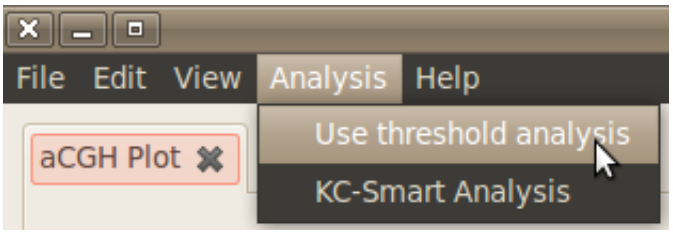


Figure 3.4: Selecting an analysis

When the algorithm has completed its execution, the CGH profile is updated to display the region identified by the algorithm, as shown in figure 3.5. Significant regions are coloured either green or red, the colours corresponding to a gain or loss respectively. The database back-end is subsequently queried for genes within these regions, which are shown in the gene view below. Zooming in on this view provides a more details on the genes, such as their names and exon/intron information. Black markings within the genes describe exons, the blank (white) regions are introns. Left-clicking on a gene result in a new tab being opened, which contains extra information about this gene (Figure 3.6). Depending on the specific gene this view contains OMIM, DECIPHER, CGC and/or DGV data.

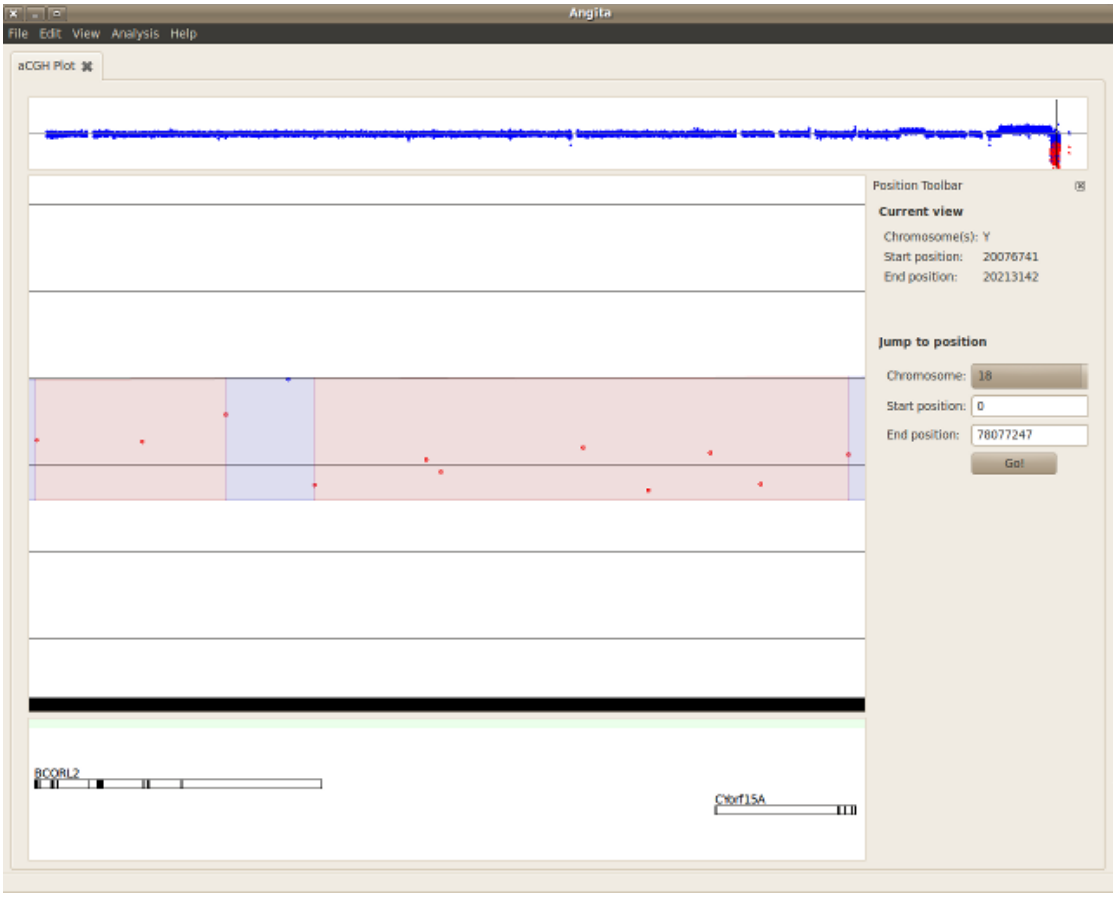


Figure 3.5: Finished analysis

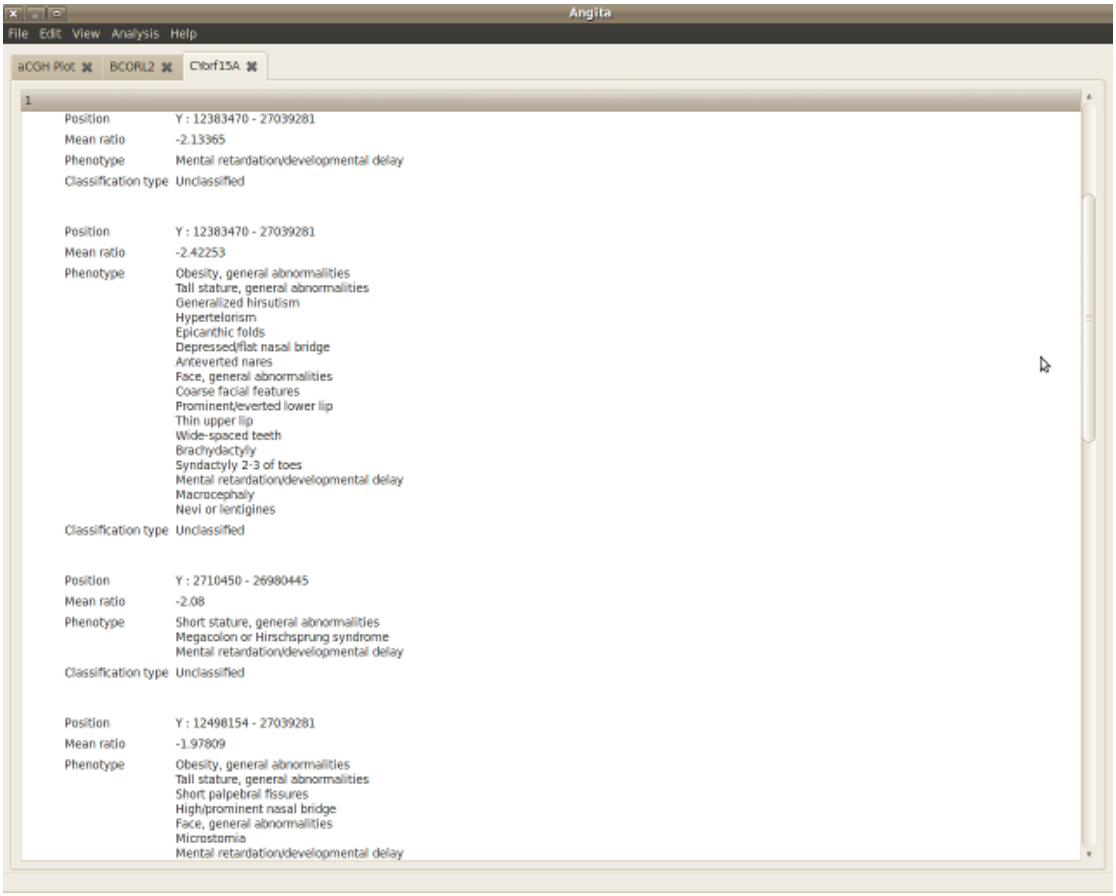


Figure 3.6: DECIPHER data for CYorf15A gene

3.4 Export to file

After the user has found a region of interest, he can export the plot and gene annotation annotation data to a PDF file. This PDF file contains the same information as the data that is viewed in the application.

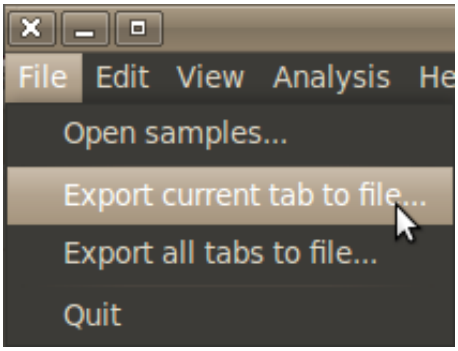


Figure 3.7: Export tab information to file

Chapter 4

Requirements

This chapter contains an overall description of the requirements of the proposed software package. These requirements have been carefully derived from the problem analysis in chapter 2 and are used to gauge the progress of the products development as well as the completeness of the final product.

The first section describes an overall perspective of the product in which a basic description of the system is given, including the various software interface on which it relies. Section 4.2 proceeds to give a short summary of the products main functions. These sections are provided to give a short but concise overview of the system.

Section 4.3 elaborates on the various constraints the product must adhere to, such as hardware limitations and safety considerations, followed by a section describing the various dependencies of the application. Section 4.5 provides a short and concise overview of the various requirements imposed on the application, which are prioritised in section 4.8 using the MoSCoW technique. Sections 4.6 and 4.7 describe various constraints imposed on the application by design constraints or software limitations.

4.1 Product perspective

The software product is an independent system but does rely on a number of external entities to perform its functions. The following external software/data sources are used by the system:

- KC-smart - identification of genomic aberrations.
- Circos - circular visualisation of gene relations.
- Ibidas - data source gene annotations, gene relations and qualifier types.
- Ensembl - data source of gene annotations.
- STRING - data source of gene relations.
- DECIPHER - data source with anonymous patient phenotype data.
- OMIM - a compendium of human genes and genetic phenotypes.
- DGV - a catalog of structural variation in the human genome.

- CGC - data source of known relations between cancer and gene variation.

4.1.1 User interface

The user interface is to be kept as clean and simple as possible (without sacrificing functionality) in order to provide a concise interface for the user to deal with. The interface should adhere to a style similar to software that is currently used by the clients to keep the learning curve of the application relatively low. A good example of such an interface is the browsing interface of the Nexus software suite, which will be used as a guideline for designing the browsing interface of plotted data.

The primary task of the user interface is presenting a plot of the loaded aCGH sample data. This plot displays the various data points of the loaded sample data and allows the user to manipulate the plot in order to view the presented data in a more convenient manner. Manipulations that the plot supports in order to facilitate these actions are: zoom functionality, side-scroll functionality and jumping between significant regions. The plot view should be accompanied by a view displaying the metadata of the loaded sample data.

The plot highlights regions of the plot that are identified as significant regions by a chosen algorithm. Gains and losses are distinguished by a difference in colour. The interface also allows a user easily navigate the genome in order to view the various found significant regions.

When focussed on a significant region of the plot, the data in the plot is accompanied by a track view which displays properties of the region. The track view must at least display the genes located in the shown region and the range of the gene in question. Selecting a gene results in the opening of a new tab displaying data concerning the gene.

The gene data view displays data concerning a specific gene as provided by the Ibidas back-end system. The view displays all acquired data in its standard configuration, but also provides a view or dialog that allows the user to show/hide certain fields.

4.1.2 Software interfaces

The product relies on the software products mentioned in this section for full functionality. KCSmart, Circos and Ibidas are directly required by the product, whilst the other software products are required by the Ibidas system in order to provide the application with sufficient annotation data.

4.1.2.1 KC-smart

Version: 1.6.0 (22 October 2008)
Source: http://www.bioconductor.org/packages/2.3/bioc/html/KCsmart.html

KC-smart uses aCGH data to identify regions that are significantly aberrant across an entire tumour set. It is based on kernel regression and accounts for the strength of a probe's signal, its local genomic environment and the signal distribution across multiple tumours.

4.1.2.2 Circos

Version: 0.52 (24 November 2009)
Source: http://mkweb.bcgsc.ca/circos/?download

Circos is a software package for visualising data and information. It visualises data in a circular layout, which makes Circos ideal for exploring relationships between objects or positions. Circos is used to visualise relations between genes, which have been identified by the product.

4.1.2.3 Ibidas

Version: 3α
Source: https://wiki.nbic.nl/index.php/Ibidas

The amount of data and the complexity of the relations between different data sets require a specialised system for efficiently storing and querying of the data. Ordinary structured query database systems are based on the assumption that data is representable as a 2D table. Combining data sources is done via Structured Query Language (SQL) using multiple tables. Biological data however requires representation of data in a larger number of dimension. Ibidas is purpose built with that idea in mind. Ibidas provides functions that are more abstract than SQL queries while maintaining the same level of flexibility and power. Ibidas is therefore the right choice for storing and accessing the various sources of biological data.

4.1.2.4 Ensembl

Version: Release 57 (3 March 2010)
Source: http://www.ensembl.org/

Ensembl is a joint scientific project between the European Bioinformatics Institute and the Wellcome Trust Sanger Institute, which was launched in 1999 in response to the imminent completion of the Human Genome Project with scientists in the United States. Its aim is to provide a centralised resource for geneticists, molecular biologists and other researchers studying the genomes of our own species and other vertebrates. Ensembl is used as a source for genomic information.

4.1.2.5 STRING

Version: 8.2
Source: http://string-db.org/

STRING (Search Tool for the Retrieval of Interacting Genes/Proteins) is a database and web resource of known and predicted protein-protein interactions. The STRING database contains information from numerous sources, including experimental data, computational prediction methods and public text collections. The database is mainly used to derive relations between genes and display data about the function of genes.

4.1.2.6 DECIPHER

Version: 4.4
Source: https://decipher.sanger.ac.uk/

The DECIPHER database of submicroscopic chromosomal imbalance collects clinical information about chromosomal micro-deletions/duplications/insertions, translocations and inversions and displays this information on the human genome map. The data in this database is used relate mutations in chromosome regions to diseases.

4.1.2.7 OMIM

Date: 4 May 2010
Source: http://www.ncbi.nlm.nih.gov/omim

OMIM is a comprehensive, authoritative, and timely compendium of human genes and genetic phenotypes. The full-text, referenced overviews in OMIM contain information on all known mendelian disorders and over 12,000 genes. OMIM focuses on the relationship between phenotype and genotype. The data in this database is used relate mutations in genes to diseases

4.1.2.8 DGV

Version: CGCh 37 (February 2009)
Source: http://projects.tcag.ca/variation/

The objective of the Database of Genomic Variants is to provide a comprehensive summary of structural variation in the human genome. The Database of Genomic Variants provides a useful catalog of control data for studies aiming to correlate genomic variation with phenotypic data. This database is used to identify variations in the processed aCGH data as CNVs.

4.1.2.9 CGC

Date: 30 March 2010
Source: http://www.sanger.ac.uk/genetics/CGP/Census/

The Cancer Gene Census is an ongoing effort to catalogue those genes for which mutations have been causally implicated in cancer. This database is used to relate CNVs to cancerous effects.

4.1.3 Communication interfaces

The product relies on a connection to Ibidas for accessing various data sources such as String and Ensembl. Ibidas functionality is provided by a server on the local network. The server provides a XML-RPC communication interface for interaction with the product. The product must ensure basic functionality without a working connection to Ibidas and will only lose functionality that is related to Ibidas, specifically: gene annotation data, gene relation identification and verification of CNVs.

4.2 Product functions

The product loads a aCGH file which adheres to a supported file format (limited by the parsers available to the system) and displays the results contained in the file in a two-dimensional plot. The plot provides the user with a manner to view the data in different scales by allowing the user to zoom-in and zoom-out on the plot as well as scroll horizontally. The product highlights certain regions of the plot (corresponding to regions of the genome) which have been identified as significant by an external algorithm and as CNVs according to data stored by Ibidas. It also displays genes located on the significant regions of the genomes in a track-like manner and provides detailed information about these genes through the data stored by Ibidas, most importantly gene function and known relations with diseases.

The product also highlights relations between genes and chromosome regions in the significant regions as identified by the data stored in Ibidas, allowing the user to reason about occurrences that often occur simultaneously. These relations are displayed in a circular plot, the types of relations included in the plot can be (de)activated by the user.

The product is also able to process a set of aCGH samples and display aggregated results in the manner as described above. The user is also able to view plots of individual results within a set.

The product allows the export of an image representation of the plot view for use in external applications. It also provides the user with a PDF export function which produces a PDF report containing important data and images.

Summary of the main functions:

- Loading and analysing of aCGH data.
- Plotting of aCGH data, highlighting significant regions as indicated by an (external) algorithm.
- Acquiring data on significant regions from gene annotation databases.
- Visualisation of relations in a circular diagram using Circos.
- Exporting of aCGH-data plots, metadata, annotations, and circular visualisations.

4.3 Constraints

4.3.1 Hardware limitations

The product must be able to run on a reasonable desktop computer with generic hardware without resulting in unreasonable waits for the user. A reasonable system is defined as having an Intel Core2Duo processor (or equivalent) and at least 2 GB of RAM. An unreasonable wait is defined as a wait of more than 10 seconds. Exceptions to this rule are made when the product is importing a new data file or running an algorithm to determine significant regions. In the case of opening a new data file a delay of up to a minute is acceptable. The delay of running an algorithm depends on the nature of the chosen algorithm and is not under direct control of the product. No limit is placed on the runtime of an algorithm, though the system should account for long delays if these are expected and remain responsive during the execution of the algorithm.

4.3.2 Reliability requirements

The analysis and corresponding visualisation must be reliable: it must produce correct results and these results must be reproducible with the same data. Errors in this process could result in interpretation of the incorrect data by the user, which may lead to incorrect conclusions.

4.3.3 Safety and security considerations

The data processed by the system and the corresponding must be kept confidential, due to the fact that data provided by the user can be traceable to a specific patient and is therefore of a confidential nature. The product must therefore not publish any data on locations other than the hardware on which it is run. The hardware systems on which the application is run are trusted to be secure in the sense that the systems are protected from unauthorised access due to the fact that they may contain sensitive data. As the hardware is considered relatively secure, the product contains no extra security or encryption measures to protect its data.

4.3.4 Database constraints

The DECIPHER data contains patient data. Because the patients privacy must be respected, DECIPHER strictly licenses usage of the data. The product can therefore not be delivered with the DECIPHER data included. Instead, the client must obtain his own license from DECIPHER. Other features of the product must still work without DECIPHER data being present.

4.4 Assumptions and dependencies

The product is designed to be cross platform and therefore functions on the most common operating systems (Windows, Linux and Mac OS X). In order to do so, it is created in the Python programming language, which is available for each of these platforms, and utilises the cross-platform graphical interface toolkit Qt. Various external software, such as Circos, is used for certain product functionality and is therefore also required for full product functionality. In summary, the product relies on the following products (and versions) to be installed on a host system in order to function properly:

- Python 2.6
- Qt4 and PyQt
- Circos

If one or more of these dependencies is not met, correct functioning of the product is not guaranteed. Where possible, the product detects missing dependencies and fails gracefully, warning the user of the missing dependencies and possibly providing a solution to the problem.

4.5 Specific requirements

4.5.1 External interface requirements

4.5.1.1 User interfaces

The users of the product have a deep understanding of life sciences but different levels of computing experience. It is therefore preferable that the application has an understandable graphical

user interface that is easy to use without too many complex options, though it may provide an option for advanced input for the more experienced user.

4.5.1.2 Hardware interfaces

The product must be able to run on a reasonable desktop computer with generic hardware without resulting in unreasonable waits for the user. A reasonable system is defined as having an Intel Core2 processor (or equivalent) and at least 2 GB of RAM.

4.5.1.3 Communications interfaces

The product relies on a connection to Ibidas for accessing various data sources such as STRING and Ensembl. The product must ensure basic functionality without a working connection, but will exhibit reduced behaviour due to the unreachable data sources.

4.5.2 Functional requirements

The product provides a graphical user interface (GUI) for the visualisation of aCGH data. In this system, the user is able to load an aCGH-file and perform an analysis on this data. The system will then plot the aCGH data, highlighting significantly amplified or deleted genes on the genome. Several databases can be queried to get annotation data for aberrant regions. The user is then able to scroll through the aCGH-data, search on subject metadata, visualise relations between genes and render documentation, as described below.

In detail, the system will adhere to the following functional requirements:

4.5.2.1 Loading and analysing of aCGH data

1. Supports loading of one or more aCGH data file from a local resource. The aCGH file must conform to one of the supported file formats. The application must at least support data files produced by the Agilent Technologies platform used by the VUMC client and the Affymetrix platform used by the NKI client.
2. Supports selection of samples that the user wants to plot from the selected aCGH files.
3. Supports transforming the green and red channel data to log2 values.
4. Supports switching green and red channels in case the labelling was erroneously done the wrong way around.
5. Supports normalisation of the log2 data.
6. Supports analysing the CGH data for significant aberrant regions, by either
   (a) finding groups of consecutive data points above a certain threshold. Both the number of consecutive data points and the threshold will be user specified.
   (b) loading the result of an external analysis algorithm. For now, the supported algorithms will be limited to KC-smart 1.6.0.
   (c) running the KC-smart 1.6.0 implementation from our system, using the loaded aCGH-data. The user must be able to specify the parameters with which the algorithm is run.

4.5.2.2 Plotting of aCGH data

1. Supports plotting one or more samples from an aCGH data-set in a single plot. In case multiple samples are plotted, they are vertically aligned on the screen.
2. Supports interactively browsing through the plot at least including the following functionality:
   (a) zooming in on the plot using the mouse scroll-wheel.
   (b) scrolling through the plot using click-and-drag with the left mouse button.
   (c) showing more detail when zoomed in. An example of such level of detail is showing gene-names, or even introns and exons, when zoomed in, but not when zoomed out.
   (d) selecting a chromosome in order to automatically zoom in on that region.
3. Supports the highlighting of regions that are identified as significantly aberrant regions in the linear aCGH plot
4. Supports plotting of a running average over the loaded aCGH data.
5. Supports plotting Cy5 data of one array against Cy3 data of another, so different subjects can be hybridised against the same control without repeating the experiment.

4.5.2.3 Acquiring data on the significant aberrant regions

1. Supports filtering of indicated gains and losses against structural variations in the human genome using data from DGV.
2. Supports mapping of genes onto identified regions, using data from Ensembl. Because only Genome build 37 is available through the Ensembl API, our prototype will be limited to build 37.
3. Supports mapping of introns and exons onto significant genes using data from Ensembl.
4. Supports acquiring data about the significant genes, using
   (a) Decipher to get common diseases associated with a significant aberrant region in a particular chromosomal location, as well as the certainty of this being the case.
   (b) Omim to get common diseases associated with a significant aberrant region in particular gene, as well as the certainty of this being the case.
   (c) Cancer Gene Census to see if a significant aberrant region in a particular gene is associated with cancer.
   (d) STRING to get functional and physical protein-protein interactions associated with a particular gene, as well as the certainty of this being the case.
5. Supports selection of a single gene in order to display annotation data, such as common diseases or protein-protein interactions associated with a particular gene.
6. Supports limiting of the databases that will be used to acquire annotation data from. For example, if the user doesn't want to use STRING, he can uncheck this in the options menu.

4.5.2.4 Relational visualisation

1. Supports listing identified genes as well as relations between these genes (the relations being diseases associated with a gene and functional and physical protein-protein relations).
2. Supports user selection of relations, allowing visualisation of only the selected relations.
3. Supports user selection of significant aberrant regions, allowing visualisation of only the genes in these regions.
4. Supports user selection of genes, allowing visualisation of only the selected genes.
5. Supports visualising the selected genes and the selected relations in a Circos-plot (a circular relational visualisation tool).

4.5.2.5 Rendering documentation

1. Supports the export of single plots as an image.
2. Supports the export of annotations for a selected gene to a tab-delimited file.
3. Supports the export of the content (plots or annotations) in all opened tabs.

4.5.3 Performance requirements

The product should respond to all data-intensive requests within ten seconds, disregarding primary analysis of supplied data. Primary analysis of the aCGH data itself may take longer, depending on the nature of the algorithm used to analyse the data. Therefore, the first analysis of a data set may take an arbitrary amount of time.

4.6 Design constraints

Annotation data and other gene related data is acquired from external data sources by the product. As this data is not under the control of the user, the product should anticipate changing data formats when interfacing with these data sources. The system will therefore work with text-dumps of the databases to prevent the system failing to function when a database changes its API. The annotation data may therefore not be the most recent data available.

4.7 Software system attributes

The product is designed for Python 2.6. Any software system attempting to run the product must therefore contain a fully functional installation of the Python 2.6 interpreter or higher. The product also requires a number of Python libraries, such as NumPy, MatPlotLib and PyQT. These packages must be installed along with our product.

4.7.1 Other requirements

4.7.1.1 Flexibility

The primary product is a working prototype of the described product and should be highly flexible in nature in order to allow customisation of the prototype to fit any unforeseen needs. Due to the fact that this product is being created for multiple clients with different needs, the software should also allow some degree of customisation.

4.7.1.2 Error handling and extreme conditions

The product must be able to handle errors that can occur in a user friendly manner, thus displaying a graphical notification of any errors that the product cannot resolve itself. The most notable situation is the lack of an internet connection, which results in the failure of acquiring annotation data. The product must not lose its basic functionality without an internet connection and must only degrade related functions.

4.7.1.3 Quality issues

In order to provide a high level of quality in the product prototype, the development process of the product adheres to the practices defined in the quality plan.

4.8 MoSCoW

MoSCoW analysis is a prioritisation technique that is used to reach a common understanding among the various stakeholders of the project. It denotes which parts are absolutely necessary (Must have); which parts are important, but can be delayed until a further release (Should have); which parts are nice to have (Could have); and which parts are not appropriate at this time (Won't have).

This chapter describes the MoSCoW analysis applied to the previously mentioned requirements. It divides the functional requirements as given in section 4.5 into Must have, Should have, Could have and Won't have categories.

4.8.1 Must have

4.8.1.1 Loading and analysing of aCGH data

1. Supports loading of one or more aCGH data file from a local resource. The aCGH file must conform to one of the supported file formats. The application must at least support data files produced by the Agilent Technologies platform used by the VUMC client and the Affymetrix platform used by the NKI client.
2. Supports selection of samples that the user wants to plot from the selected aCGH files.
3. Supports transforming the green and red channel data to log2 values.
4. Supports normalisation of the log2 data.
5. Supports analysing the CGH data for significant aberrant regions, by either
   (a) finding groups of consecutive data points above a certain threshold. Both the number of consecutive data points and the threshold are specified by the user.
   (b) loading the result of an external analysis algorithm. For now, the supported algorithms will be limited to the R-implementation of KC-smart 1.6.0.

4.8.1.2 Plotting of aCGH data

1. Supports plotting one sample from an aCGH data-set in a plot.
2. Supports interactively browsing through the plot at least including the following functionality:
   (a) zooming in on the plot using the mouse scroll-wheel.
   (b) scrolling through the plot using click-and-drag with the left mouse button.
   (c) showing more detail when zoomed in. An example of such level of detail is showing gene-names, or even introns and exons, when zoomed in but not when zoomed out.
   (d) selecting a chromosome in order to automatically zoom in on that region.
3. Supports the highlighting of regions that are identified aberrant regions in the linear aCGH plot.

4.8.1.3 Acquiring data on the significant aberrant regions

1. Supports mapping of genes onto identified regions, using data from Ensembl. Because only Genome build 37 is available through the Ensembl API, our prototype will be limited to build 37.
2. Supports acquiring data about the significant genes, using
   (a) DECIPHER to get common diseases associated with a significant aberrant region in a particular chromosomal location, as well as the certainty of this being the case.
   (b) Omim to get common diseases associated with a significant aberrant region in particular gene, as well as the certainty of this being the case.
   (c) Cancer Gene Census to see if a significant aberrant region in a particular gene is associated with cancer.
   (d) STRING to get functional and physical protein-protein interactions associated with a particular gene, as well as the certainty of this being the case.
3. Supports selecting of a single gene in order to display annotation data, such as common diseases or protein-protein interactions associated with a particular gene.

4.8.2 Should have

4.8.2.1 Plotting of aCGH data

1. Supports plotting more samples from an aCGH data-set in a single plot. These samples will be vertically aligned on the screen.
2. Supports plotting of a running average over the loaded aCGH data.

4.8.2.2 Acquiring data on the significant aberrant regions

1. Supports filtering of indicated gains and losses against structural variations in the human genome, using data from DGV.
2. Supports mapping of introns and exons onto significant genes, using data from Ensembl.

3. Supports the limiting of the databases that will be used to acquire annotation data from. For example, if the user doesn't want to use STRING, he can uncheck this in the options menu.

4.8.2.3 Rendering documentation

1. Supports the export of single plots as an image.
2. Supports the export of annotations for a selected gene to a tab-delimited file.
3. Supports the export of the content (plots or annotations) in all opened tabs.

4.8.3 Could have

4.8.3.1 Loading and analysing of aCGH data

1. Supports running the KC-smart 1.6.0 implementation from our system, using the loaded aCGH-data. The user must be able to specify the parameters with which the algorithm is run.
2. Supports switching green and red channels, in case the labelling was erroneously done the wrong way around.

4.8.3.2 Plotting aCGH data

1. Supports plotting Cy5 data from one array against Cy3 data from another, so different subjects can be hybridised against the same control without repeating the experiment.

4.8.3.3 Relational visualisation

1. Supports listing identified genes as well as relations between these genes (the relations being diseases associated with a gene and functional and physical protein-protein relations).
2. Supports user selection of relations, allowing visualisation of only the selected relations.
3. Supports user selection of significant aberrant regions, allowing visualisation of only the genes in these regions.
4. Supports user selection of genes, allowing visualisation of only the selected genes.
5. Supports visualising the selected genes and the selected relations in a Circos-plot (a circular relational visualisation tool).

4.8.4 Won't have

1. Support of linking (an abstract of) papers related to aCGH specifics.

Chapter 5

Design

This chapter describes the design of the Angita application, as well as decisions made during the design phase affecting the designs. The chapter is intended to give an overview of the design and its most important elements. More detailed information concerning the design of the application is contained in the Software Design Document[2].

The first section of this chapter discusses the basic structure of the application in which the various components and subsystems of the applications are described. The remain sections elaborate on the design of these components and subsystem, specifically the following elements: the data model (5.2), the filesystem subsystem (5.3), the user interface (5.4), the controller (5.5), the algorithm subsystem (5.6), the circos interface (5.7) and the database component (5.8).

5.1 Architectural overview

The product is a desktop application combined with a server-sided component. In a basic workflow, the product relies on a file loading subsystem which contains loaders to load aCGH files. The data from this subsystem is analysed by an algorithm subsystem and the results are matched against information contained in external databases, which is supplied by the database package through interaction with the server-sided database system. All this various information is combined into a graphical view and presented to the user. The user is able to interactively browse the various types of data in order to possibly reach conclusions about the nature of the presented case.

The server-sided component is formed by the Ibidas system. Ibidas is used as a platform for storing and querying the large amounts of data input from various external databases, thus combining these sources to a single interface for the product. The data in these databases includes basic genome information for both human and mouse, but also known gene data and associated phenotypes, relations, functions and abnormalities. The user is able to define which databases are used to collect data.

In order to present the data graphically to the user a custom plotter is created using Qt. Previously available plotters were all limited in the freedom required to present the data interactively. Even then it has been a major hassle to efficiently process all the data points and display them on screen. Additional relations between genomic regions are presented using an external link to Circos.

5.1.1 Subsystem overview

The product is composed of six separate subsystems, the first of which is the filesystem subsystem. This subsystem is responsible for transforming a specified aCGH data file into the internal model structure that can be processed and handled efficiently by the main application. It is able to parse all input files for which parsers have been supplied.

The second subsystem is formed by the user interface. The user interface is responsible for handling all user interaction and provides a number of views and configuration options.

The Controller forms the third subsystem, which combines the various graphical components presented to the user with the other systems in order to obtain the needed data for presentation to the user. This component is responsible for the communication between all of the subsystems.

The fourth subsystem is an algorithmic subsystem, which determines the significant regions of a aCGH profile using a selected algorithm. This subsystem uses various (possibly external) algorithm implementations which can be selected as required. The subsystem ensures that the supplied data is transformed into the input format required by the selected algorithm and transforms output into a generic format to return as a result.

The fifth subsystem is an interface to the Circos application, which is responsible for drawing relational diagrams to illustrate relations between various genes. The subsystem handles the generation of the configuration file required by the application to draw an image correctly and is also responsible for subsequently running Circos to generate image using the supplied configuration.

The database component of the application forms the sixth subsystem of the application. This component is used for storing and querying data from different databases. The component has been split in a client (interface) - server (Ibidas) architecture, allowing these components to communicate through a XML-RPC protocol.

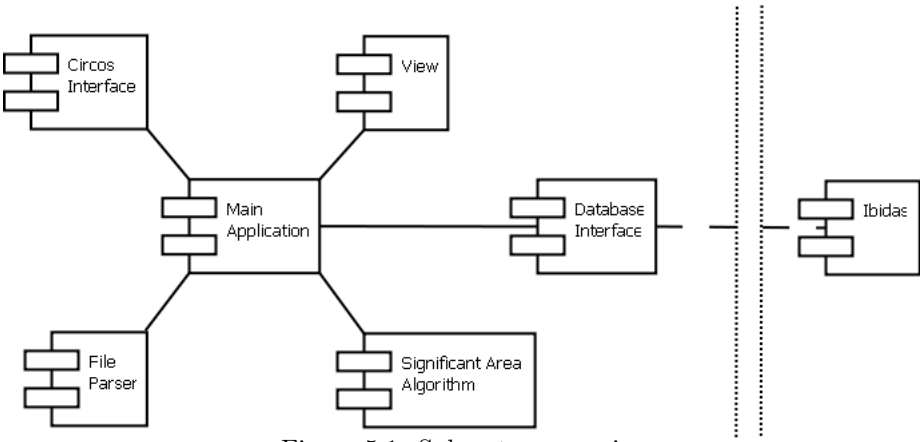


Figure 5.1: Subsystem overview

5.1.2 Data management

The product does not save any data locally. Imported aCGH data files must be stored locally, their location is specified by the user. The user is able to set a number of preferences, including database build versions and database fields. Ibidas is responsible for handling the databases and offering the data to the product.

5.1.3 Parallelisation

Due to the enormous amounts of data contained in aCGH profile files and the various external databases, the application requires extra attention to maintain responsiveness. The application has therefore been designed to run as a multi-threaded application to prevent blocking of subsystems. For example, this parallelisation allows the GUI to respond to user input while information is fetched from the database as both these tasks run in separate threads. A number of subsystems have been designed to run in their own thread, using asynchronous callbacks to communicate with the controller.

5.2 Data model

The Angita program has to process large amounts of data to perform its tasks, including both the enormous amount of data contained in the various CGH file formats and the various gene annotation data extracted from a number of external databases. In order to handle these large amounts of data efficiently, the application provides a well-defined set of model classes which model the various types of data and relations between data types in the application space.

This section first describes a general set of genome model elements that are used throughout the application, followed by an explanation of the various model elements used represent data specific to CNV samples. An overview of the general model class structure is shown in the class diagram shown in figure 5.2.

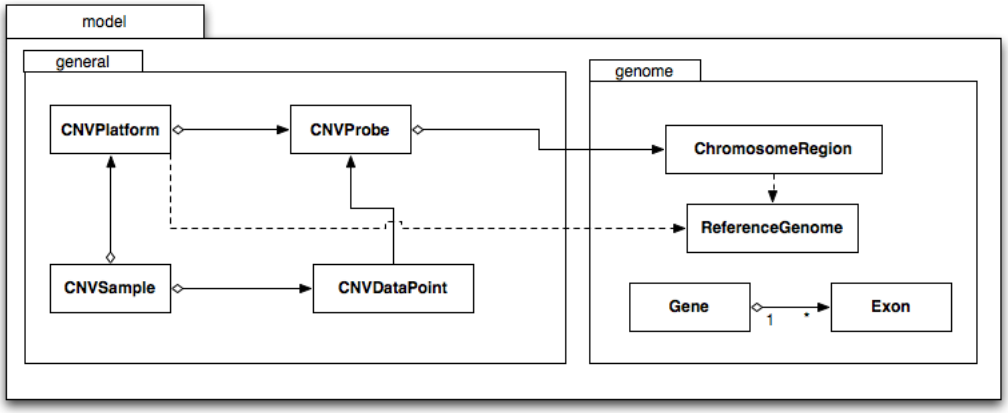


Figure 5.2: The basic structure of the Model package.

5.2.1 Genome related classes

The Angita application supplies a number of model classes that describe the most basic elements of the domain model and are used various manners throughout the application. The most prominent of these elements is the chromosome region, which is used to identify a specific genomic region on a chromosome by naming the corresponding chromosome and providing a start and an end position on the chromosome. As shown in figure 5.2, chromosome regions are mostly used to define the region of a samples CNV probe but are also used in different contexts.

The model also provides a *ReferenceGenome* class, which represents the genome on which chromosome regions are located. When a human sample is loaded in the application the human reference genome is used by the application as a reference, whilst mouse data requires the use of the mouse reference genome.

Other genome related elements are the *Gene* and *Exon* classes, which represent genes and the exons of genes respectively. A gene generally has a multiple number of exons as is depicted by figure 5.2.

5.2.2 Sample related classes

In order to handle the various formats of CNV sample (related) data uniformly, the application defines a general domain model to which all external data formats containing CNV sample data are translated for processing by the application. This model output is most often generated by the file system subsystem, which performs this translation through the parsing of the various supported file formats.

The most basic element in the sample model is the *CNVProbe* class, which represents a DNA probe used by the CNV platform to measure the copy number variations in a specific region on the genome. Each probe is generally located in a single, unique region of the genome, which is identified by the chromosome region referenced by the probe element.

A CNV platform is represented by the corresponding *CNVPlatform* class in the data model. The element holds the various probes used by this specific platform to measure copy number variations and contains metadata concerning the various properties of the represented platform.

The *CNVSample* class portrays an actual CNV sample and contains metadata concerning the sample and holds the data points measured for the sample. These data points are represented by their own element, which hold the various values measured during the aCGH analysis and references the probe for which the values were measured.

5.3 Filesystem

The filesystem subsystem is responsible for transforming a specified CGH file into its corresponding representation in internal model classes, which can be processed/handled efficiently by the main application. The subsystem is able to parse all input files that adhere to the supported data formats.

The main problem encountered in the process of designing a generalised file parsing structure is the heterogeneity of the various file formats. Inherent to the field of bioinformatics is the lack

of standardisation of processes and data interchange formats. Whilst attempts have been made to construct some sort of standardised data format, such as the MIAME specification[6], these formats are still too vague to allow construction of a single, unified parser.

Even though the current implementation of the subsystem supports the processing of only three differing file types, each of these file types differs substantially enough in its format to require an adjusted approach in the parsing of the file format. Examples of differences between these file formats are: XML files vs. text files, single data files vs. multiple data files, one sample per file vs. multiple samples per file.

In order to keep the parsing pipeline as general as possible, the parsing of a specific file type is not performed by a single element but is spread over three separate elements. The function of these three elements are shortly described below, the exact implementation of the entire hierarchy is detailed in the technical design document. A short overview of the general filesystem structure is shown in the class diagram below.

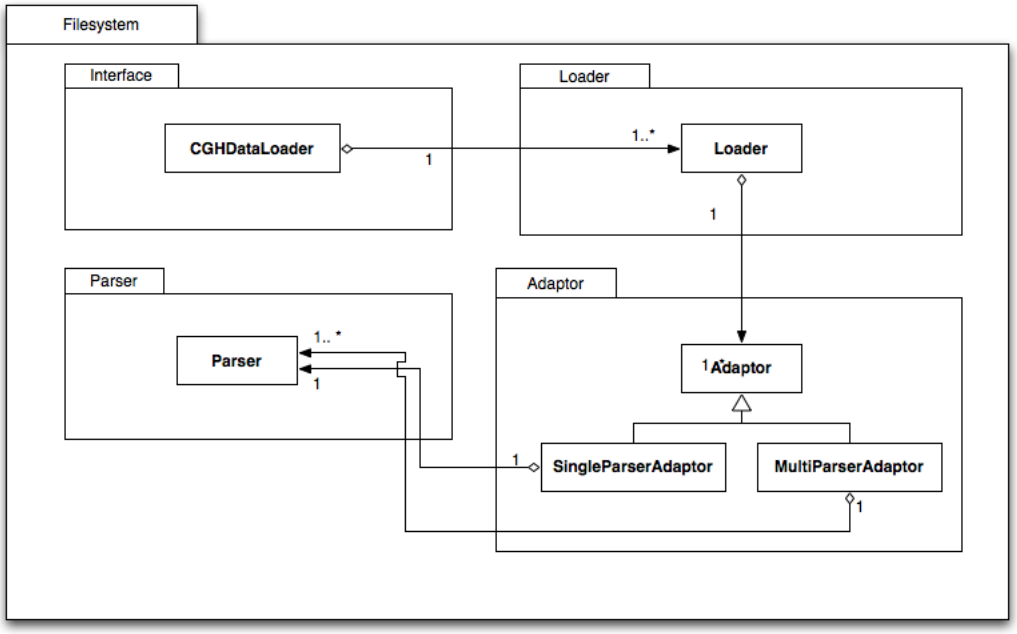


Figure 5.3: The basic structure of the Filesystem subsystem.

Parsers The parser classes are responsible for parsing the actual text file to a python representation. A parser is provided a reference to a text file and in turn provides a generator which can be used by other classes. Each call to the generator results in the parsing of a line of the given text file to a tuple of variables, which is subsequently returned to the calling code. The benefit of returning data in this tuple format is that this format does not impose any assumptions or limitations on the parsed data, leaving the interpretation of the returned data to the calling code.

Adaptors The adaptor classes are responsible for calling the various parsers and transform the parser tuple data into the format required by the application. Examples of this are the various metadata adaptors, which accept a parser tuple and return a dictionary containing the various metadata, and the various data adaptors which return actual model representations of data such as CNV data points. Depending on their type, an adaptor can have a single parser or multiple parsers in order to load the required data.

Loaders The loader classes handle the various adaptors and aggregate the data returned by these adaptors into the final model representation of loaded sample file. The loader classes are the highest level of the subsystem and can be directly called by code outside the subsystem. The classes therefore all adhere to a single public interface and return data in a uniform format. The loaders are managed by the CGHDataLoader, which forms the main interface to the subsystem through which loaders are accessed.

Subsystem Interface The interface of the filesystem subsystem is provided by the CGHLoaderInterface class. This class aggregates the available filesystem loader classes and provides methods with which code external to the subsystem can query the names of the available loaders and retrieve an instance of a specific loader for a given file name. In this manner external code can retrieve loader instances without having to reference the internal classes directly. As each loader implements the same interface, each returned instance can be handled in the same manner and therefore doesn't require any loader specific code.

5.4 User interface

The user interface classes are responsible for the entire graphical interface with which the user interacts. These classes are responsible for handling all user-actions and data to the users view. The classes support the user in all the actions he execute in the application, which include the following actions:

- Loading aCGH data
- Analysing aCGH data
- Plotting the genome, showing
  - aCGH data point
  - significant regions
  - genes
- Viewing annotation data
- Viewing sample metadata
- Rendering plots, annotation data and metadata to documentation.

Most of these actions have been implemented using existing widgets in the Qt Framework[20]. Qt is a cross-platform application and UI framework. Qt was chosen to build the Graphical User Interface as it is available on all major operating systems, has python bindings through PyQt and looks aesthetically better then any considered alternative.

The design of the specific interface widgets is described in the following sections.

5.4.1 Mainwindow

The *Mainwindow* is the view through which a user executes all his actions. It is responsible for handling the various dialogs; annotation-, metadata- and plot-tabs; and the communication with the controller. The annotation data, metadata and plots are each viewed in a separate tabbed widget. The tabbed interface is designed to allow the user to switch between annotation data and plots easily, which would be more difficult in a multi-window design.

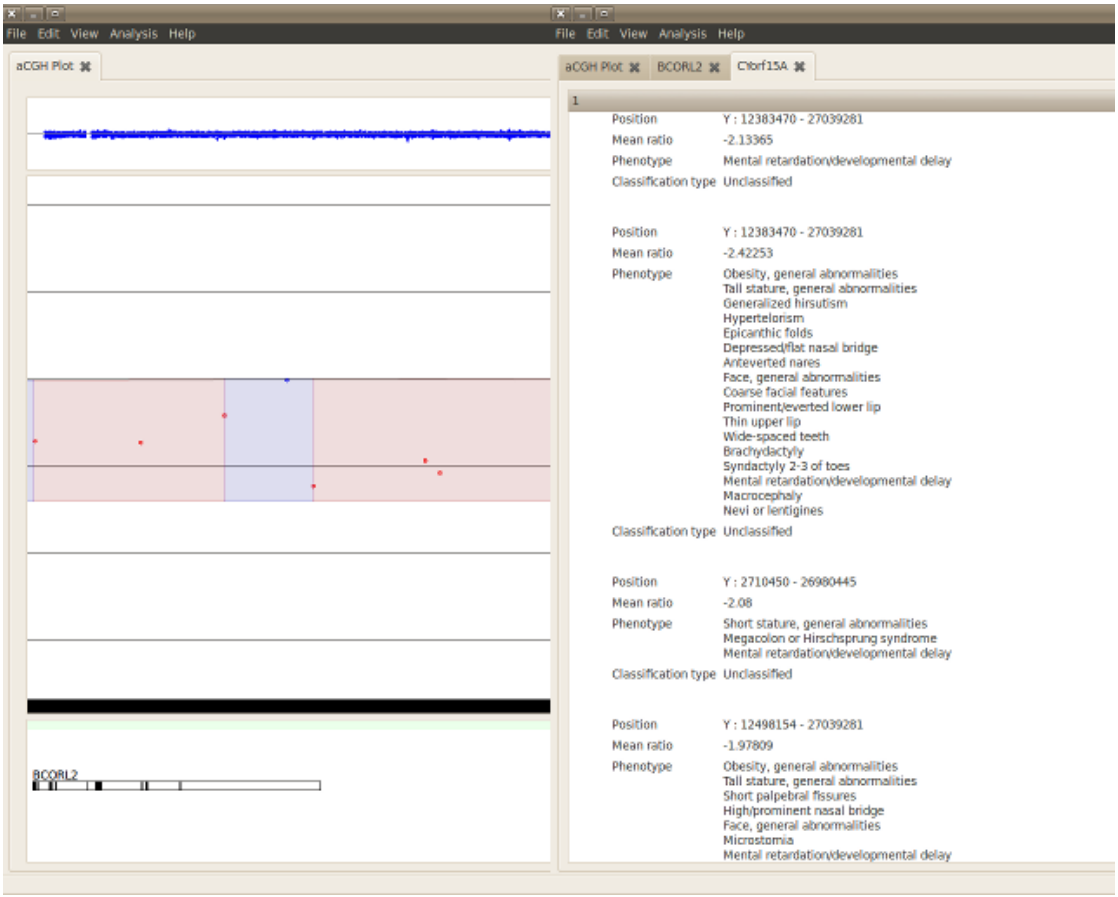


Figure 5.4: An example of a plot-tab and an annotation-tab.

5.4.2 Dialogs

The functions of loading of aCGH data and the rendering of documentation are each contained in their own window. The standard Qt class *QFileDialog* is used for both cases, as this dialog resembles the standard save-file dialog on every OS and is perfectly suited for the task. As these cases use standard Qt classes to perform their work, they do not appear in the class diagram.

5.4.3 Tabbed widgets

The *QTabWidget* contains the plot, annotation data and sample metadata. The plot is shown automatically when the application is started. Annotation and metadata tabs are only shown when the corresponding data is requested by the user.

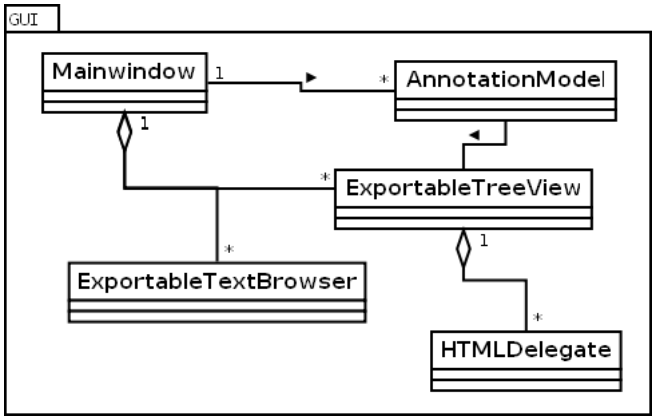


Figure 5.5: Relations between Mainwindow and the other classes involved in making tabs.

Gene annotation data tabs are constructed by the *AnnotationModel* class. This class is responsible for receiving annotation data from the controller and subsequently creating a *QWidget* that is added to the *QTabWidget* in the *Mainwindow*. A tree view is used to display annotation data, because annotation data can be substantially large in size. The view shows the corresponding gene-name along with its position and the names of the included databases when opened. The user can expand a database node in the tree view in order to show the data from extracted from the corresponding database.

5.4.4 CNVBrowser

The purpose of the CNV browser is to provide the user with a visual representation of aCGH profile and analysis data. It also shows genes found in significant regions and enables the user to explore the data set visually.

5.4.5 Overview widget

The application provides an overview widget to help the user navigate the genome using the *CNVBrowser* view. This widget displays the entire CGH profile, along with a rectangle which represents the current view of the user in the *CNVBrowser* view, thus providing an easy view of where the user is currently located in the CGH profile. It also allows the user to manipulate the contents of the *CNVBrowser* view through various mouse interactions.

As illustrated by figure 5.6 the *CNVBrowser* consists of a *CNVview* and a *GeneView*. These views are backed by *CNVScene* and *GeneScene* respectively, which are containers that manage the graphical items shown in the views. The views render the contents of the scenes using the

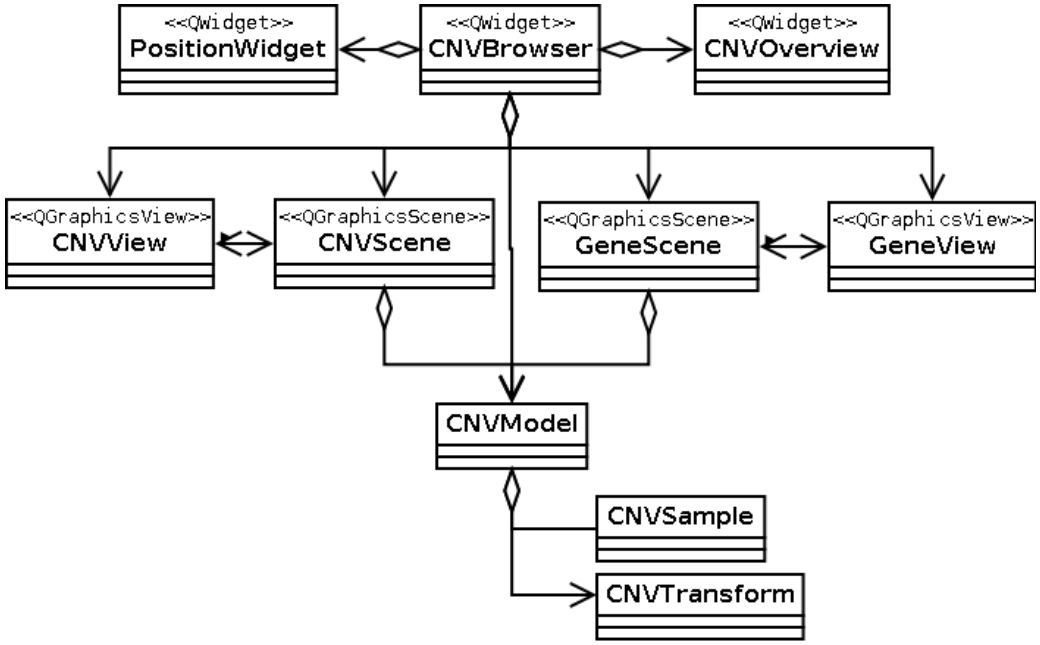


Figure 5.6: The *CNVBrowser* maintains a model that is visualised by the views.

Qt Graphics framework and allow the user to manipulate the content of the views. The actual sample and gene data are separately managed by the model.

The following sections describe the structure and implementation of the overview widget. The first section describes the *CNVOverview* class itself, whilst the second details a pixmap class used to draw the CGH profile efficiently within the overview view. The third section explains the implementation of the *OverviewPointTransform* class, a transformation class used to translate screen coordinates to base pair positions and vice versa. On overview of the overview widget structure is shown in 5.7.

5.4.5.1 CNVOverview

The *CNVOverview* class is essentially a *QGraphicsView* subclass and is responsible for handling the drawing of the overview widget and responding to the various events received by the view. The class maintains three graphical objects: a line representing the central axis of the view, a rectangle which denotes the current view of the *CNVBrowser* view and a pixel map item which displays the CGH profile in the background of the view. The overview manipulates these items as required to ensure an up-to-date view, depending on received external signals.

The class provides the following manners in which the user can manipulate the *CNVBrowser* view through interaction with the *CNVOverview* view: rubber-band selection, dragging the view rectangle, clicking on a position within the CGH profile and scrolling within the overview.

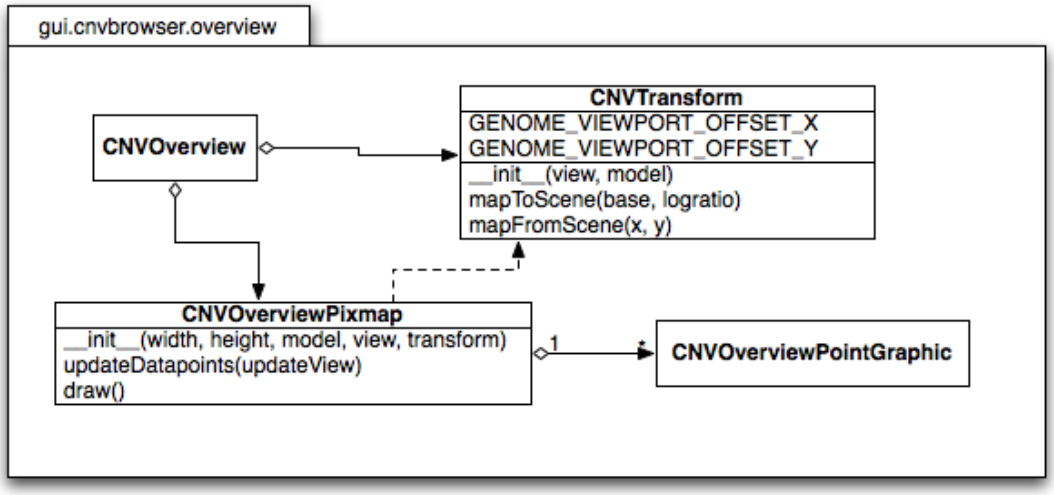


Figure 5.7: The basic structure of the overview widget design.

Rubber-band selection Rubber-band selection allows the user to select a region of the CGH profile by right clicking within the overview and dragging the cursor to create a selection area within the overview. When the right mouse button is released, the selected area is set as the new view for the CNVBrowser and the view rectangle is updated correspondingly.

Dragging the view rectangle Dragging the view rectangle allows the user to scroll the current view along the CGH profile horizontally. Note that the overview limits the range of the view rectangles motion, as the view rectangle is not allowed to leave the CGH profile area.

Clicking on the CGH profile Clicking on a position within the CGH profile repositions the current view to horizontally centre on the clicked point in the CGH profile. This allows the user to quickly reposition the current CNV browser view to a specific point on the CGH profile.

Scrolling The overview also provides scroll functionality, which responds to mouse scroll events and either enlarges the width of the view rectangle or decreases it width depending on the scroll direction of the user, adjusting the *CNVBrowser* view accordingly.

5.4.5.2 CNVOverviewPixmap

The *CNVOverviewPixmap* class is a subclass of the Qt *QPixmap* class, which is essentially a container for image data in a pixel map format. The class is responsible for drawing a static image of the CGH profile which fits within the CNVOverviews viewport dimensions. Having a static image to display the data points gives the CNVOverview a great performance enhancement as the overview only needs to draw the corresponding data points once, instead of having to redraw the data points repeatedly.

In order to draw the CGH profile correctly, the *CNVOverviewPixmap* class is provided with a number of data points and an OverviewPointTransform instance, which it uses to calculate the correct position of each data point within its image dimensions. It creates a corresponding graphic point for each data point with the calculated new coordinates and proceeds to render an image containing these points once, only requiring a redraw on a resize event of the overview or when the CGH profile changes.

5.4.5.3 OverviewPointTransform class

The *OverviewPointTransform* class provides two important utility methods for the two classes mentioned in the previous two sections, the *mapToScene* method and the *mapFromScene* method. The *mapToScene* method accepts a base pair position and a log ratio value and returns the corresponding coordinates in screen space, which are then subsequently used for drawing purposes and event handling.

The *mapFromScene* method is the inverse of the *mapToScene* method, as its name implies. This method accepts a x- and a y-coordinate in screen space and translates the coordinate pair to the corresponding base pair position and log ratio value. The method is mainly used by the various event handlers, which require mouse event coordinates to be translated to a base pair representation.

5.4.6 Position widget

The *PositionWidget* is a simple Qt widget that listens to the various signals emitted by the *CNVBrowser* and displays various data about the current view of the *CNVBrowser* view, such as the left-most and right-most base pairs currently in view as well as the currently visible chromosomes.

The widget also provides a panel containing various control inputs, which allow the user to select a chromosome and input a starting position and an end position in base pairs. After a subsequent click on the button input, the panel updates the CNVBrowser to display the CNVRegion specified specified by the chromosome name and position combination. The panel is only able to select a region on a single chromosome as the clients deemed it unnecessary to include functionality to select a region spanning multiple chromosomes in this fashion.

5.4.7 ListeningScrollView

As mentioned before both the *CNVView* and the *GeneView* enable the user to navigate the data. Any changes made to a view is communicated to the other as these changes should also be reflected by the other views.

The user can interact with the view in three ways:

- The data can be moved from side to side by dragging the view. This is done by holding down the left mouse button and moving the mouse.
- Zooming is done using the mouse wheel, resulting in the view zooming-in or zooming-out on the data in the view. This zooming is centred on the mouse cursor, meaning that the area under the mouse cursor is effectively enlarged. This allows the user to point at a gene or data point and zoom in on this point without having to drag the view to keep this point in view.

- A specific area can be zoomed in by dragging a box around the area that should be enlarged. This area is then fit into the view.

5.4.8 Report

Another function provided by the application is rendering documentation from the plot, the gene annotations and the sample metadata. The software package ReportLab [13], an open source engine for creating PDF files, is used to do so. The package *Report* defines an interface to communicate with the ReportLab package. Using this interface elements such as paragraphs, images and page breaks can be easily appended to a PDF instance from other parts of the program.

5.5 Controller

The *Controller* is responsible for the integration of the different components/subsystems. The controllers primary function is translating requests of the *Graphical User Interface* (GUI) to requests required by the designated subsystem. The *Controller* is linked to the *filesystem-*, *algorithm-* and *database*-subsystems as well as the *processing* and *job-dispatcher* packages.

The *Controller* contains a number of functions that keep track of loaded samples, modified samples and samples visible in the *GUI*, acting as the primary data storage facility. It also implements the *AbstractDatabaseInterfaceListener* interface, which contains a number of functions that must be implemented to perform its function as a database listener. Examples of such functions are the various callbacks used for asynchronous communication with the database.

An emphasis in the design of the controller is ensuring the fast relay of requests to the required subsystem, in order to keep the controller (and thus the system) responsive. In order to do so, all requests to subsystems are performed asynchronously to prevent blocking as the controller can not handle any new requests while in a blocked state. This functionality is especially important in this product due to the high probability of some processes, such as executing an analysis algorithm or loading of a file, taking a substantial amount of time. The controller uses the *Job-Dispatcher* to offload requests.

5.5.1 Job-dispatcher

The *Job-Dispatcher* was created to allow the offloading of functions and processing them inside their own thread. Python makes it possible to hand a function pointer, arguments and a callback function pointer to a thread and let that thread calculate the result. The *Job-Dispatcher* contains a thread pool that starts out with a fixed number of worker threads. When the load increases a number of workers is automatically added. Each worker is supplied with a job and reports its result to the thread pool, which handles this result appropriately.

The *JobManager* is the *Job-Dispatchers* interface to the rest of the product. It accepts a job and hands it off to the thread pool. Each job receives a unique identifier, which is used to track the job in question. The *JobManager* actively polls for jobs that are done. When a completed job is found, a callback function is called to notify the caller that the job has been completed. A special exception callback is called if the job has halted with an exception. This way the caller can respond accordingly to the results of the threads computation, examples being the repeat of a database request or displaying a simple notification in the event of an exception.

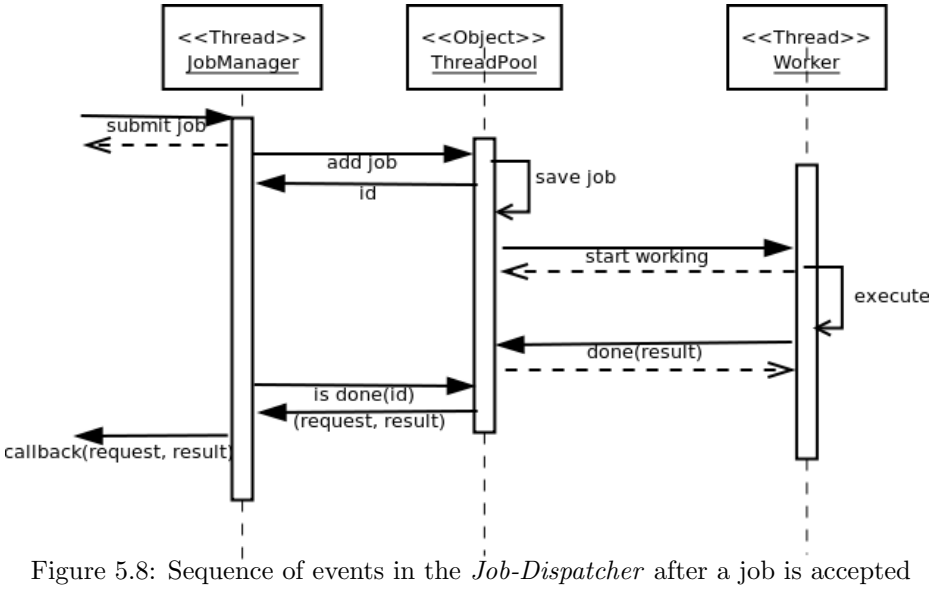


Figure 5.8: Sequence of events in the *Job-Dispatcher* after a job is accepted

5.5.2 Processing

The processing package of the application contains implementations of a number of algorithms that are run on *CNVSample* instances after these are loaded but before they are further used in the program. This includes normalisation of the data and computation of the visible outline. The following sections describe these algorithms as well as the deprecated Horizon Cluster Algorithm, which is no longer used in the application.

Normalisation To counteract systematic shifts in the data, the data is first normalised using the standard score. The μ and σ are calculated over the whole gene. The log ratio value for each data point is adjusted according to the following equation, where x denotes the original value: $x : ((x - \mu)/\sigma$.

Outline Algorithm The outline of a data set consists of a set of lines that tightly encloses the majority of the data points, providing a good low detail representation of the data set. This is used by the *CNVBrowser* to represent the bulk of the data points without having to draw each individual point.The outline is computed by dividing the data set into a number of bins and drawing an outline based on (a multiple of) the standard deviation of the bin.

Horizon Cluster Algorithm The horizon cluster algorithm was designed to cluster data quickly and is executed in a single pass over a sorted set of data points. It does however not provide a stable set of clusters and is therefore no longer used in the product.
The horizon cluster algorithm creates clusters of points, not comparing each data point against all clusters for inclusion but excluding the clusters that are too far away (e.g. over the horizon) from consideration. This is possible as the data points are assumed to be sorted on the x axis and not overlapping. As such any cluster that is too far away for a datapoint D, will also be too far away for any datapoint E that comes after it, providing a large optimisation over an implementation that considers all clusters.

5.6 Algorithm

The algorithm subsystem of the application contains and manages implementations of a number of algorithms that are run on CNVSample instances in order to identify significantly aberrant genomic regions in the corresponding genome. This section describes the design and implementation of the subsystem by first mentioning the model classes defined by the subsystem to store data, followed by an explanation of the actual algorithm structure and the interface provided by the subsystem. An overview of the algorithm subsystem implementation is shown in the class diagram below.

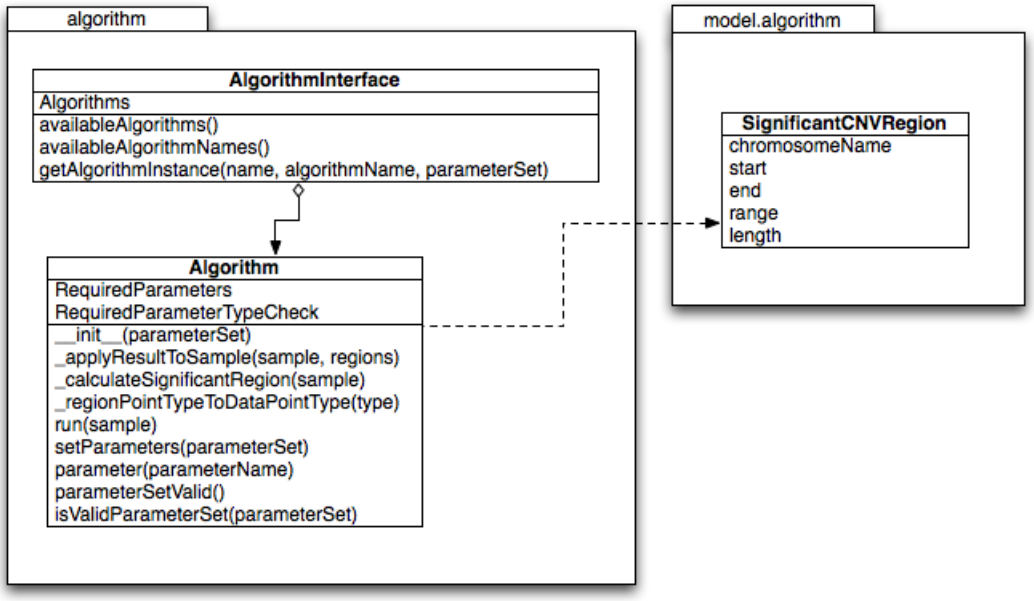


Figure 5.9: The basic class structure of the Algorithm subsystem.

5.6.1 Basic structure

The main function of the algorithm subsystem is determining the significantly aberrant regions of a subjects genome according to the supplied sample data. The *SignificantCNVRegion* model class is provided by the subsystem to identify such a region. The class references a ChromosomeRegion element which identifies the chromosome region that the identified region spans. It also contains a type field, which specifies whether the region represents a loss or a gain of genomic material.

The output of each algorithm within the algorithm subsystem is a list of *SignificantCNVRegion* instances, which represent the regions identified by the algorithm. An algorithm can also provide and return a subclass of the *SignificantCNVRegion* class in order to return any extra data concerning the identified regions.

The main interface of the subsystem is formed by the AlgorithmInterface element, which allows the querying of available algorithm implementations and the retrieval of a specific algorithm.

The basic structure of the algorithm subsystem is shown in figure 5.10. The main interface of the subsystem is formed by the AlgorithmInterface element, which allows the querying of available algorithm implementations and the retrieval of a specific algorithm.

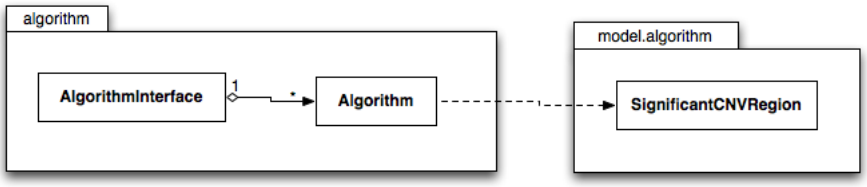


Figure 5.10: The basic structure of the Algorithm subsystem.

Algorithms are implemented as a subclass of the basic algorithm element shown in figure 5.10. This element provides the basic functionality for each algorithm, requiring specific algorithm implementations only to implement a method that accepts a given sample and returns a set of significant regions. This structure ensures that each algorithm provides the same basic functionality, including the setting and verification of a set of parameters that are passed to the algorithm when it is run, whilst requiring little actual implementation work to create (aside from the algorithm itself).

The significant regions returned by the algorithm are represented by a list of significant CNV region elements. This element specifies the start and end of the region in question, the type of the region (either being a gain or a loss), and a list of the data points contained in the region.

5.6.2 Algorithm implementations

The most important algorithms required by the application are algorithms that identify significant aberrant genomic regions in analysed CGH samples. As the resulting product is a prototype, it does not contain an entire library of such algorithms but rather implements a few algorithms which are most likely to actually be used by the clients and demonstrate the algorithm functionality of the prototype.

From the literature study and the interviews held with our clients, two specific algorithms were identified to be most valuable to the clients if implemented in the prototype: the KC-smart algorithm and a threshold algorithm. The details of these algorithms are discussed in the following sections.

Base Algorithm The base *Algorithm* class provides the basic functionality for an algorithm but lacks an actual implementation of the *calculateSignificantRegion* method, which is called upon to calculate the significant regions for a given sample. The Algorithm class can be supplied with a parameter set containing various parameters which are used by the algorithm during its execution.

KC-smart algorithm The KC-smart algorithm has been actively developed by the NKI client and is freely available in a R implementation. The description of the algorithm according to the algorithms documentation is shown below:

> KC-SMART is a method for finding recurrent gains and losses from a set of tumour samples measured on an array Comparative Genome Hybridisation (aCGH) platform. Instead of single tumour aberration calling, and subsequent minimal common region definition, KC-SMART takes an approach based on the continuous raw log2 values. KC-smart Gaussian locally weighted regression to the summed totals positive and negative log2 ratios separately. This result is corrected for uneven probe spacing as present on the given array. Peaks in the resulting normalised KC score are tested against a randomly permuted background. Regions of significant recurrent gains and losses are determined using the resulting, multiple testing corrected, significance threshold. [14]

The KC-smart algorithm has a number of interesting features, one being that the KC-smart algorithm can be run on multiple samples, providing an aggregate result in which KC-smart identifies can be used to identify correlation between different samples. Another interesting feature is the ability to perform multi-scale analysis. The algorithm can be run using different kernel widths, where small kernel widths will allow detection of small regions of aberration and vice versa large kernel widths allow large regions of aberration to be detected.

KC-smart text file algorithm Due to the fact that the KC-smart algorithm can take a substantial amount of time to complete when run with a large number iterations or on a large data set, an extra algorithm has been implemented that essentially loads and parses the text file output of a completed KC-smart session. The advantage of this method is that the KC-smart algorithm can be run wherever and whenever the user would like, requiring only its results to be supplied.The algorithm uses the same parser/adaptor structure as the filesystem subsystem. It opens the file, whose path is supplied as a parameter to the algorithm, and parses its contents to the equivalent list of SignificantCNVRegion instances, which is then returned and processed in the manner described above.

Threshold algorithm The threshold algorithm analyses a sample in order to determine groups of consecutive points on a chromosome that have an absolute log ratio larger than a given threshold. This algorithm has been implemented for the VUMC client as the client uses a similar workflow when identifying significant regions, by selecting groups of three or more consecutive data points with an absolute log ratio larger than a certain threshold.

5.7 Circos Interface

Circos [4] is an external Perl program designed for circular visualisation of (genomic) relations. The Circos interface package is designed to form an interface to circos, which transforms the internal model structures into the (complex) configuration input required by Circos and is also responsible for executing the Circos program on the supplied configuration.

The interface has only one class(*CircosInterface*). The class is supplied a set of significant regions, which is used to identify the various genes contained in the corresponding regions. The interface subsequently uses Ibidas to search for interesting relations between the identified genes or their

corresponding proteins. These relations can differ in mode and action, which is visualised by the interface in its output by using different colours for each mode and action.

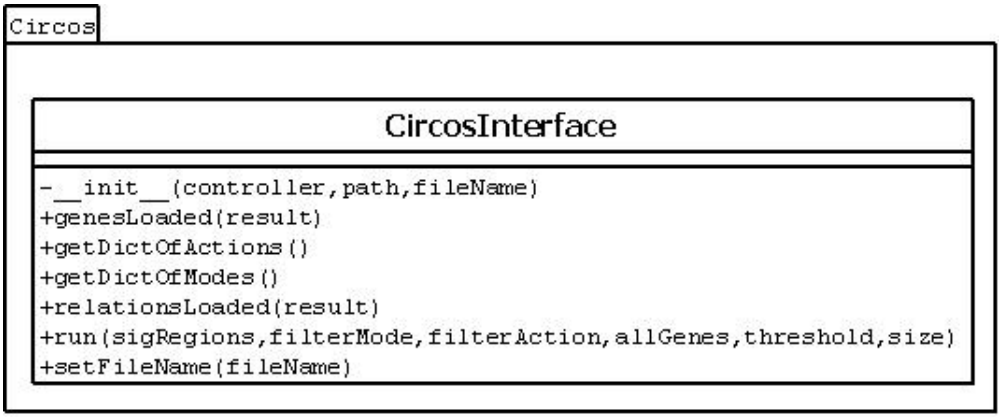


Figure 5.11: ClassDiagram circos

5.8 Database

The Database module is used for storing and requesting data from the external databases described in the requirements of the application. The module is split in a client (interface) - server (Ibidas) architecture, which communicate through XML-RPC. The server component is run on a central server server. The following sections describe the design of the database module.

5.8.1 Interface

The client-side component of the database module offers a clean interface that implements the same functions as that are available in the server component. The subsystem consists of an XML-RPC client implementation and a database interface. The XML-RPC client basically offers a single function: the sending of a request for data to the database. The XML-RPC client is implemented asynchronously to make sure the database subsystem does not block while waiting for a response.

The *Database-Interface* effectively offers database functionality to other components of the product by implementing the same basic functions as the server component. When a subsystem requests data from the database the interface creates a new job containing the request. The job retrieves the requested database from the server component and translates the format returned by the server into the required format. When the requested data is completed and has been transformed, the corresponding callback functions are called to inform the caller that the request has been completed.

A database listener, such as the controller, must implement a number of functions specified in the *AbstractDatabaseInterfaceListener* interface. The functions defined in the interface are callbacks that are called when a reply from the database is received. A special callback is defined in case an exception is encountered to inform the caller that the exception has been generated.

5.8.2 Ibidas

The large amounts of biological data stored in the various external databases must be kept available and accessible in order to supply the user with the various annotation data. The Ibidas system is a database system which provides the functionality required to store such data in a single database, allowing data to be queried in a uniform manner.

Normal databases present data in a flat 2D manner, however biological data requires a more flexible solution. Ibidas is able to maintain the data in the manner in which it is presented (potential multidimensional), instead of just combining a large number of 2D tables. Ibidas also keeps queries relatively simple as it does not require complex joins to couple the various data sources as a conventional database would.

A limitation of the Ibidas system is that it is currently only able to maintain the databases in memory, preventing storage of the data in a MySQL back-end. A startup script is supplied with the system, which ensures the system contains the required data after start up. The first start-up requires Ibidas to parse the various data formats, which are loaded into the system through text file dumps. The system can subsequently create a database dump, which can be loaded more quickly on the next start-up. This limitation does still limit the size of the database and results in high memory use.

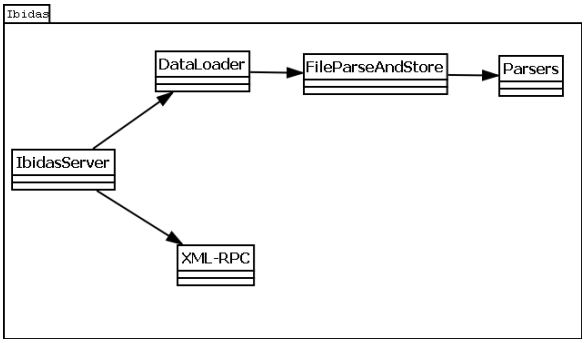


Figure 5.12: The global class structure of Ibidas.

The product offers information from the following databases:

- Ensembl (basic gene information)
- CGC (cancer gene datasource)
- OMIM (compilation of human genes and genetic phenotypes)
- DECIPHER (compilation of rare genetic phenotypes)
- DGV (compilation of known CNV variations)
- STRING (compilation of protein-protein interactions)
- UCSC (chromosome band information, among others)

A number of data parsers are constructed to be able to use data from different sources.

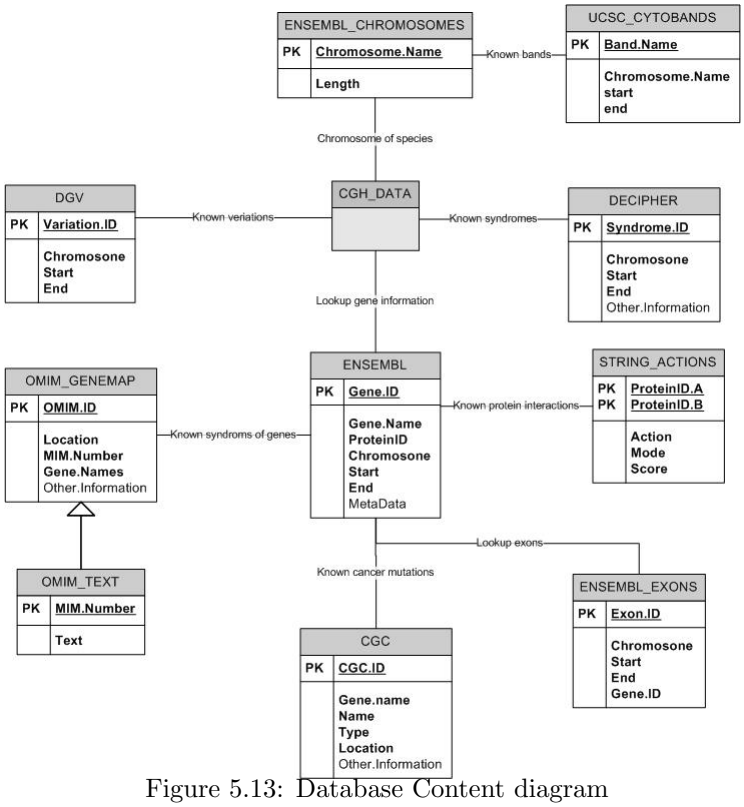


Figure 5.13: Database Content diagram

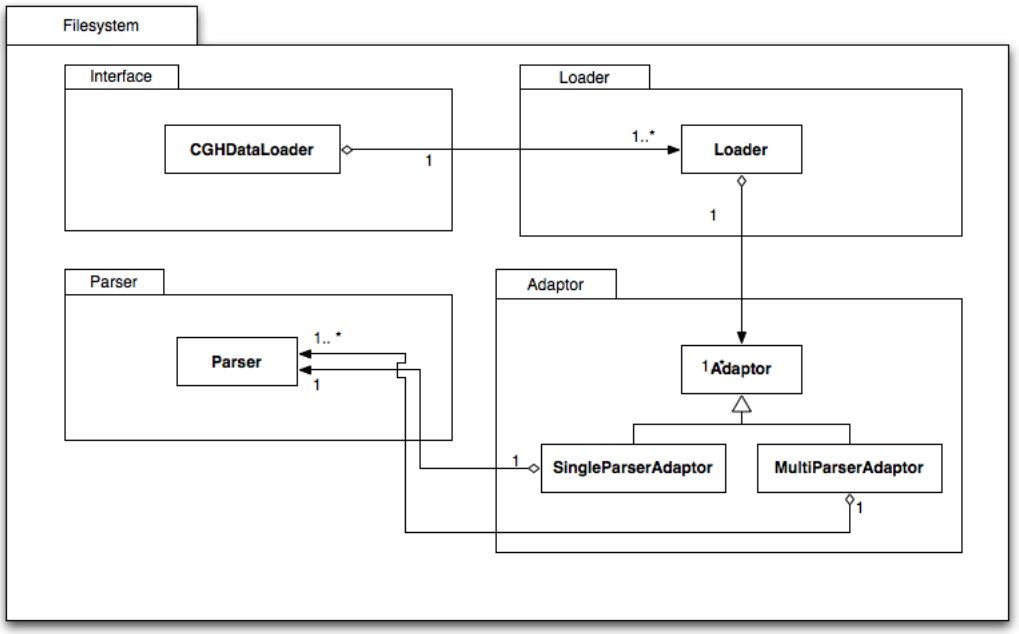


Figure 5.14: The basic structure of the Filesystem subsystem.

Chapter 6

Implementation

The Angita application is the desktop application component of the software system. The application is run on the client system and is among others responsible for the loading of CGH files, the visualisation of loaded CGH samples, running analysis algorithms on loaded CGH samples and displaying various types of related metadata.

After the design and orientation phase were concluded the implementation was started in the implementation phase. The design splits the application in six separate subsystems. Due to the iterative nature of the project and a commitment to deliver a prototype each week, the most basic functionality in each subsystem was implemented separately, before being connected to the others. More complex functionality was added later on. During the implementation several problems were encountered and decisions made to solve them.

This chapter provides a brief overview of the implementation of each component and subsystem, as well as any problems encountered and decisions made solving these problems. Each element is described in the same order as that of the previous chapter, which detailed the design of the various elements. More detailed information can be found in the Technical Design Document[3], which is included in the appendix of this document.

6.1 Data model

The Angita program has to process large amounts of data to perform its tasks, including both the enormous amount of data contained in the various CGH file formats and the various gene annotation data extracted from a number of external databases. In order to handle these large amounts of data efficiently, the application provides a well-defined set of model classes which model the various types of data and relations between data types in the application space. The design of the data model is described in section 5.2.

While the data model itself is well defined, not all aspects of its use had been considered during its design. Though the initial design accounted for the the CGH samples, it did not account for a number of derivatives of the sample, examples of which are the significant regions and outline elements. The design of the data model lacked a collection of these three elements along with an observable pattern to notify owners of a change. It was decided not to implement such a collection as a good implementation would require extra development time, whilst the current

situation, in which the data is implicitly managed by the CNV Browser, was deemed sufficient for the current application.

6.2 File System

The file parsing subsystem is responsible for transforming a specified CGH file into its corresponding representation in internal model classes, which can be processed/handled efficiently by the main application. The subsystem is able to parse all input files that adhere to the supported data formats. As described in section 5.3, the subsystem has been designed to accommodate various file formats by splitting the loading sequence into a number of separate classes and steps.

6.2.1 Supported file types

The current implementation of the subsystem supports the loading of three different data formats.

US2 The US2 data format corresponds with the data format which was supplied by the VUMC client. This data format supplies all data within a single text-based and tab-delimited file and contains the data of a single sample.

The first three lines of the file contain metadata concerning the platform used for analysis of the sample and metadata concerning the sample itself. The second set of three lines contains various statistics of the measurements made when the sample data was recorded in the same format. This data is currently ignored, but could easily be loaded as part of the general metadata.

The remainder of the file contains data concerning the probes used in the analysis of the sample and various measured values, such as the log ratio measured for each probe. Currently only lines containing a systematic probe name for the corresponding probe are parsed by the parser. Probes without a systematic name are currently ignored, as these probes generally do not correspond to a genomic position but are usually reference values used to normalise measured values.

GPL The GPL data format is the data format that was first supplied as test data for the application in its early development stages. This data format consists of multiple files, where a single XML-based file contains platform metadata and the metadata of the various samples. The platform probe data is contained in a separate text-based file, the location of which is specified in the platform metadata. The XML-file also specifies a separate text-based file for each sample, which contains the measured values of that sample per probe.

Both the platform probe data file and the sample data file are text-based and tab-delimited. The files only contain a single data type and are therefore uniform in structure. As with the US2 format, probes without a corresponding systematic name are ignored, as are measured sample values for those probes.

ETABM The ETABM data format corresponds with the data format which was supplied by the NKI client. This data format consists of two files, both of which are text-based and tab-delimited. The metadata file carries the SDRF file extension and contains metadata for each of the measured samples. Each metadata entry references a data file in which the measured sample

values are kept. This data file is row-ordered and generally contains data of multiple samples in a single file by listing values for each sample in subsequent columns.

6.2.2 Encountered problems

During the development of the product, the filesystem subsystem has been redesigned multiple times before reaching the single unified structure described in section 5.3. A number of different designs was made and implemented, one example being a simple parser structure that parsers a text file directly and returns the parsed file. This approach was however not very flexible and resulted in a large amount of duplicate functionality among the various parser classes.

Another more advanced implementation used regular expressions to parse the various text files in a similar structure as the current implementation. A drawback of this approach is that it required a complex regular expression to be designed for each file, which can be quite time consuming. An advantage of the regular expression approach is that such an approach is very robust and does not fail easily on malformed files.

Due to the large size of the files involved in the loading process, the design of the filesystem has had to focus on performance as well as flexibility. Though the various mentioned designs have varying levels of performance, the difference between the various implementations was never larger than one or two seconds (on a runtime of 15-20 seconds). This difference was not considered large enough to merit a design based solely on its speed, therefore the current design has been chosen for flexibility and not its speed per-se.

6.3 User interface

The user interface presents the application to the user and should enable the user to execute all his tasks in a quick and efficient manner. These tasks include the loading and analysis of CGH data, visually exploring the CGH data and its significant regions, looking up the genes with associated those regions and providing access to metadata associated with these genes.

The first subsection describes the basic implementation of the graphical interface with a brief walkthrough of the various graphical elements. The following subsection briefly elaborates on the problems encountered in the implementation of the user interface.

6.3.1 Basic implementation

Once the CGH data is loaded as illustrated in 6.1, it is represented using a graphical polygon outline in the centre window. This enables the application to provide the user with the impression of all 100.000 data points without actually rendering them all.

The significant regions on the chromosome in which gains and losses occur are represented using red and green colours. These are not immediately visible, but become so after the user executes an algorithm that analyses the sample. Genes are now also visible in the gene view (bottom view). However due their size, they're too small to be properly identified yet.

To investigate the significant regions found a user can zoom in to view them in more detail as illustrated by figure 6.2. Once sufficiently zoomed in, the outline polygon is faded out and the actual data points are shown. The bottom view now shows in greater detail the gene found in

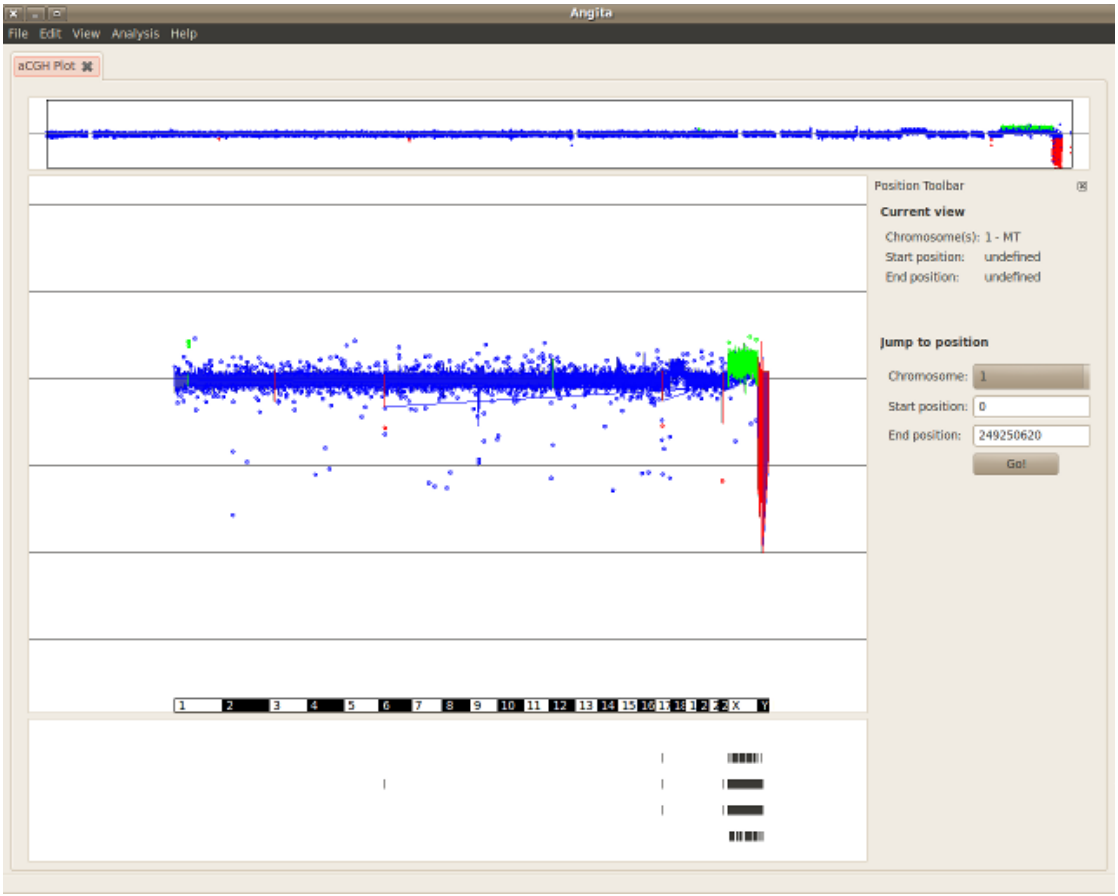


Figure 6.1: The view of the application the CGH data has been loaded and analysed for gains and losses.

this significant region as well as the DGV data.

Once a gene of interest is found, it can be selected to open a tab that displays the meta data about that particular gene as shown in 6.3. This is a sufficient improvement over Nexus which would simply refer the user to a number of external websites.

6.3.2 Encountered problems

The PyQt graphics system was chosen to alleviate the implementation of the user interface, which is a set of Python binding for the Qt interface toolkit. This proved to be a good choice. While it was not possible to render all data points using this framework, it did provide enough customisation capabilities to allow the implementation of custom widgets which could provide the needed functionality. Unfortunately due to some subtle differences in implementation across platforms OpenGL could not be used on Ubuntu, making the application somewhat slower on that platform.

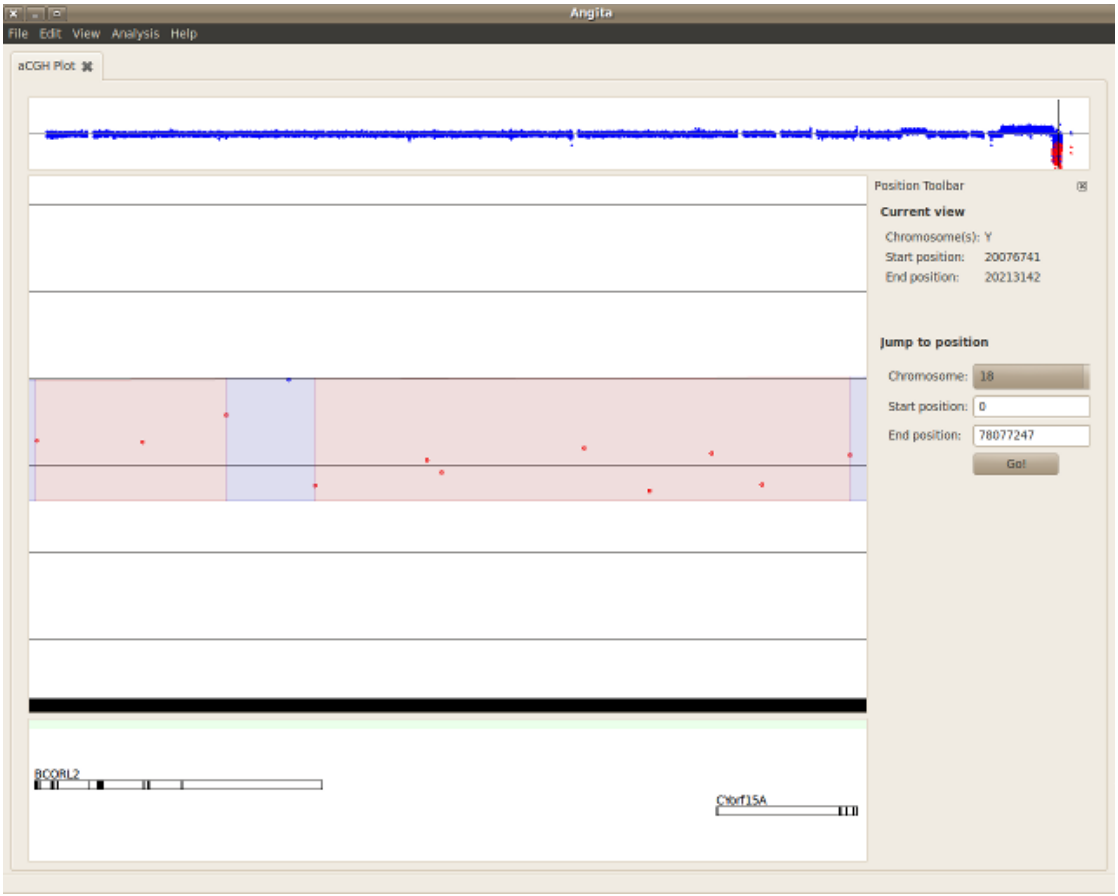


Figure 6.2: The view of the application, zoomed in on a particular gene.

The main problems encountered during the implementation of the user interface occurred when implementing the visualisation of the CGH data. Existing software could not efficiently plot all data points at once and did not provide the desired interactivity. To alleviate these problems the choice was made to write a custom visualisation that would allow the user to smoothly navigate the data.

This visualisation has undergone a number of changes during implementation which was aimed at improving the performance of the view whilst not omitting any data critical to the user. Performance problems mainly occurred when viewing a large portion of the genome, which results in a large portion of the data points being shown simultaneously. A first solution to this problem was a clustering algorithm, which aggregated similar points into clusters when zoom-out and drew only these clusters instead of the individual points. This clustering did however result in loss of visible data and was relatively slow.

An improvement on the clustering algorithm was provided by an outline algorithm, which basi-

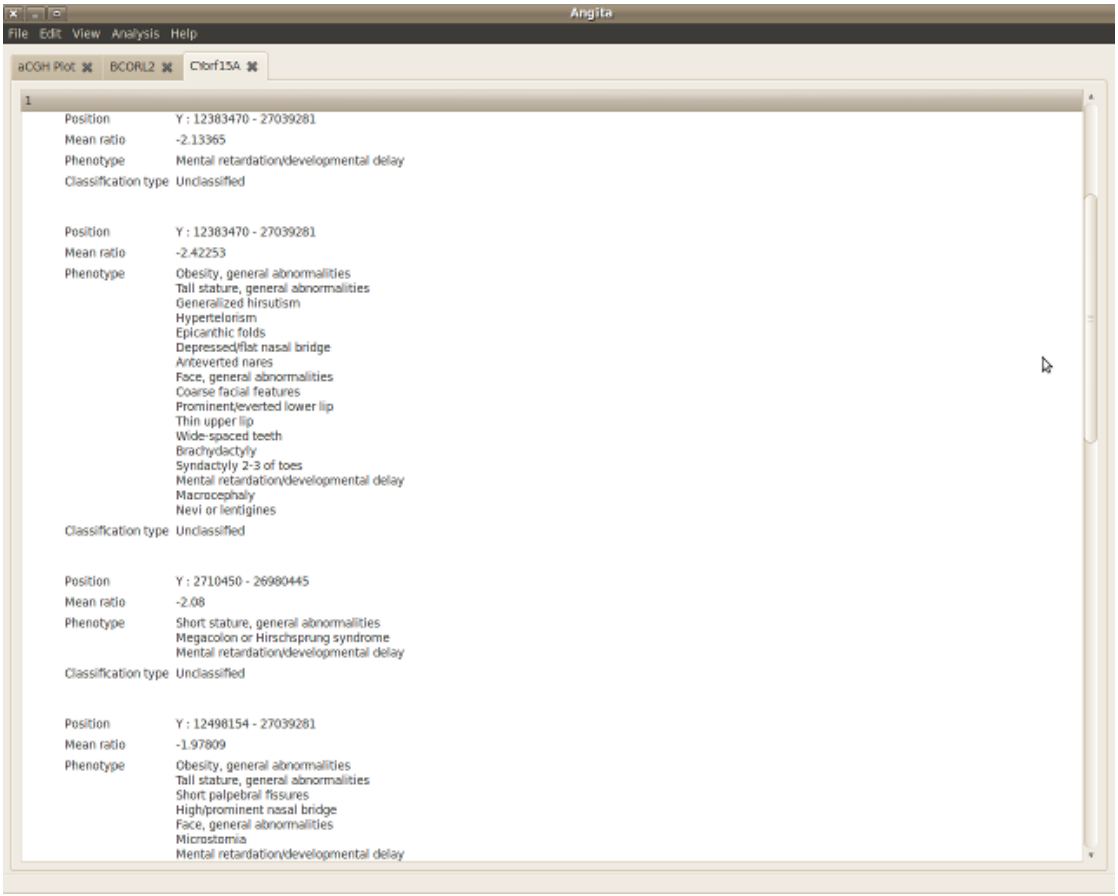


Figure 6.3: The genetab of the application showing information from the DECIPHER database.

cally divides the data into a number of bins of fixed size and calculates the standard deviation of the contained points for each bin. An outline is then plotted by drawing a line over these bins at a fixed multiple of the standard deviation on both the positive and the negative y-axis. This outline is relatively simple to plot, providing high performance, and also resembles a good approximation of the actual data set. Points outside the outlines are still drawn separately to avoid loss of these details. This algorithm is currently used to implement the level-of-detail used in the product.

Other difficulties in the implementation were encountered during the implementation of the overview widget. The first implementation of the widget used two different coordinate systems to define the location of its view rectangle, as the mouse event handlers provide positions in screen coordinates, whilst the CNV browser provides positions in base pairs (left-most and right-most). The implementation was later adjusted to use only the base pair coordinate system for setting positions allowing positions to be set in base pairs, after which the class internally translates to and from these coordinates using the provided transformation class when needed. This allowed for a uniform handling of coordinates providing a simpler and less error-prone implementation.

A currently unresolved issue in the overview widget are its performance issues on the initial draw. The first draw of the widget requires the image containing all data points to be drawn, as do resize events that affect the window. This drawing requires a substantial amount of time because the widget draws every data point in the image to achieve maximum detail. This drawing currently blocks the interface and should either be split to a separate thread, thus reducing the impact of the long draw sequence, or simplify the drawing procedure by approximating groups of points. To solve resize problems the widget could scale the drawn image for small resize adjustments, only redrawing the image for large changes in the overviews dimensions.

6.4 Controller

The primary function of the controller is to manage the data model based on input from the user. Due to the iterative nature of the project the controller wasn't implemented until other sub systems had been completed. Very few problems were encountered with the controller itself, however its current design proves to be somewhat cumbersome. Recommendations to solve this are outlined in chapter 10.

6.5 Algorithm

The algorithm subsystem provides a number of algorithm implementations which are run on a sample and return a set of determined significant CNV regions in the given sample. The design of the subsystem is discussed in section 5.6 of this document. The actual implementation of the subsystem has shown little deviation from this design. This section describes a short description of the currently implemented algorithms and an explanation of a problem encountered during the implementation of the subsystem, which required the design to be adjusted slightly.

6.5.1 Implemented algorithms

6.5.1.1 KC-smart

The KC-smart algorithm has been actively developed by the NKI client and is freely available in a R implementation. The KC-smart algorithm implementation uses the RPY2 library to interface directly with the R implementation of the KC-smart algorithm, which is provided through the bioconductor software package. The implementation therefore requires a working installation of the R and the KC-smart R implementation.

The implementation of the algorithm in the application is responsible for translating the model representation of a CNV sample into the R equivalent, which is subsequently passed to the R implementation. After running the actual KC-smart algorithm, the results are transformed back into a list of significant CNV regions and returned as normal.

6.5.1.2 KC-smart text file algorithm

Due to the fact that the KC-smart algorithm can take a substantial amount of time to complete when run with a large number iterations or on a large data set, an extra algorithm has been implemented that essentially loads and parses the text file output of a completed KC-smart session. The advantage of this method is that the KC-smart algorithm can be run wherever and

whenever the user would like, requiring only its results to be supplied. The algorithm uses the same parser/adaptor structure as the filesystem subsystem.

6.5.1.3 Threshold algorithm

The threshold algorithm analyses a sample in order to determine groups of consecutive points on a chromosome that have an absolute log ratio larger than a given threshold. This algorithm has been implemented for the VUMC client as the client uses a similar workflow when identifying significant regions, by selecting groups of three or more consecutive data points with an absolute log ratio larger than a certain threshold.

The algorithm performs its function by iterating over the sorted collections of data points contained in the CNVSample instance. If the algorithm encounters a data point that has an absolute log ratio larger than the given threshold, the algorithm flags the point and continues its iteration. If the next point is also above the given threshold, it is also flagged. As soon as the algorithm encounters a point with a log ratio below a given threshold or with a value that has a different sign (positive/negative) than the previous data points, the list of flagged points is inspected.

If the number of flagged points is at least the number of points required, the list of points is converted to a significant region and the algorithm resets its list of flagged points and continues its iteration. If the number of flagged points is less than required, it simply resets the flagged list and continues iteration, as this is not a significant region with the given parameters. This sequence of events is repeated separately for each chromosome.

6.5.2 Encountered problems

The implementation of the subsystem deviates only slightly from its original design, which shows that the initial design was sufficiently correct to implement the required functionality. A small discrepancy with the initial design is that the algorithm does not return a new sample instance on which the result has been applied. Instead, the subsystem now returns a list of significant regions and modifies the passed sample in place.

This implementation detail allows analysis to be performed without having to duplicate the sample, which takes considerable time and memory. A drawback is that a sample is tainted by analysis, which is currently dealt with by resetting a samples state before running an algorithm to avoid mixture of algorithm results.

6.6 Circos Interface

Circos is a Perl written visualisation tool, and since it has no user-friendly interface, an extended interface was needed for the application. The interface searches within provided significant regions for genes, filtering against a selected set of relations using the STRING database. Based on the found genes and identified relations, the interface generates a configuration file and uses a system call to call Circos to create the required image.

The implementation of the code used to generate the configuration file was troublesome and was done in a trial-and-error fashion, due to lack of documentation for the Circos application.

Another encountered problem was the reliability of the Circos application as the application has unexpected behaviour on certain inputs, leading to loss of settings or incorrect images.

6.7 Database

Ibidas is used as the database system in the implementation to integrate the various sorts of biological data. Its current implementation manages the full body of knowledge from the following databases: STRING, Ensembl, CGC, Omim, DECIPHER, DGV and UCSC.

The first design of the product was a client-only application, which quickly became unwieldy as the load of Ibidas increased with the number of databases loaded. The burden of the Ibidas systems was therefore the main reason to transition from a client-only to client-server architecture as this allows the load of the Ibidas system to be carried by a single central system. Ibidas is however still a heavy burden on the host of the server component and will most probably not scale well with multiple clients.

As mentioned in the previous chapter, Ibidas is unable to store data in a back-end at this time and therefore requires all loaded data to be kept in memory. This implementation effectively limits the data that can be loaded to the size of the servers memory, which prevents adding more databases into the system.

Despite the above mentioned limitations and the alpha status of Ibidas, the system has proved to be a good solution for the storage of the various data formats in a uniform manner. Its usefulness should only improve as the development of Ibidas continues.

For security reasons a password identification system was added to the server component to prevent unauthorised access to Ibidas. This was not foreseen in the design but was required due to the nature of the data contained in the database, which is confidential in a number of cases. The password identification is implemented as an extra layer in the server component and acts through the passing of extra function arguments since there are no security options built-in by the XML-RPC protocol.

Chapter 7

Test Results

The product has been subjected to automatic unit and integration testing in order to ensure a certain quality, as well as an acceptance test and a system test. Found failures were either reported to the corresponding developer or registered in the Trac system and assigned later. The acceptance test was held when the project was close to completion, in which the user interacts with the application with as little help from the developers as possible. This test is one of the most important, as problems in the intuitiveness of the user interface become apparent during this test.

7.1 Code test

7.1.1 Unit testing

Unit testing is a software verification and validation method in which a programmer tests if individual units of source code do what they are designed to. A unit is the smallest testable part of an application, an example being the basic functions of a class. Table 7.1 shows the percentage of application code that is covered by the unit tests.

It is worth mentioning that graphics code mostly cannot be tested through unit tests, because a lot of functions require user interaction. Some functionality can not or should not be tested, including scaffolding to run subsystems individually and mutually excluding states. Therefore the percentages are all between 85% to 95%. This percentage is within the desired range and the implemented tests are therefore deemed to be sufficient.

The product is shown to pass all individual unit tests. Combined with the achieved amount of coverage, this means that the product passes the set goals for unit testing.

7.1.2 Integration testing

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. In the product all individual modules are combined in the Controller. The GUI is attached to the Controller and cannot be tested automatically because it requires user interaction.

Component	Coverage %
Aggregation	86.2%
Algorithm	87.1%
Database	85.8%
Filesystem	88.8%
Job-Dispatcher	93.1%
Model	86.9%
GUI	21.0%

Table 7.1: Obtained testing coverage percentages

Before a test is run the required subsystems are setup using specific configurations that relate to the working conditions of the function under test. Afterwards the system is returned to a steady state, ready for the next test. Every function that the Controller offers publicly are tested. Input and output assertions are tested to assure improper input is caught and proper output is given.

All integration tests pass on the final product, although some tests need to be run individually as the tests generate more threads than the maximum number of threads in Python. This means that the product also passes the set goals for integration testing.

7.1.3 Ticket system

A ticket system is an application that assists programmers in tracking reported software bugs. It provides a clear overview of requested features, known bugs and allows these to be prioritised and organised by release. Trac[9] was used as the ticket system for thir project.

Due to the iterative nature of the project and frequent collaboration between the developers, Trac has not been used to track minor issues that could be solved on short notice. All issues that were not important enough to have to be solved directly and all issues that could not be solved are reported on Trac.

Currently there are 14 issues on track, of which 10 have been resolved. The 4 remaining issues are minor.

7.2 Acceptance test

An acceptance test with the VUMC client was held ten days before the end of the project. This allowed a near finalised version of the product to be shown, whilst still allowing the results of the test to be used to tweak the final version of the program. The acceptance test has been executed with only one person, the VUMC client, as no other test subjects with the correct expertise were available on short notice. The NKI client was unavailable for the acceptance test.

During the test the user was asked to use the program independently. It quickly became apparent that a small tutorial on how to navigate on the aCGH plot (zooming and scrolling) was necessary. The user indicated that this problem was likely due to the fact that the interface of the application differs from the currently used system in navigation. After a very short tutorial, the user continued without any problems.

Another issue that occurred was that the settings dialog, which defines settings such as genome build, was placed in an unexpected location of the user interface. The client had expected to set these values when loading a file, but at the moment of testing, this dialog had to be opened separately from the menu. This flaw has been adjusted in the final version of the application.

The client also noted that the threshold algorithm, which is used to identify significant regions, does not identify all the regions it is required to as it leaves some regions unidentified. The client suggested a new algorithm, which automatically sets the threshold to 2σ. This is a simple solution, which is included in the final product but may however not solve the problem entirely.

When viewing annotation data, the client remarked that the DECIPHER data looked different from what she expected and enquired if it was possible to filter all entries categorised under Copy Number Variants. The client also asked if it was possible to create an export function that renders a tab-delimited file containing a summary per identified significant region. Such a summary should contain the position of the significant region, the number of probes it contains and the names of the genes that lie within the region. These features may not be included in the final product.

Finally, the client also noted that she was very pleased with the way DGV data was visualised. In comparison with the program she currently uses, the product shows more information in just as little space by using green and red for gains and losses and colour intensities for the number of observed variations.

7.3 System test

System testing is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic. The system testing phase was evaluated by comparing the final product functionality to its requirements. Properties such as time limits on certain functionality have been measured in a program execution using realistic data.

To avoid having to reiterate every requirement formulated in chapter 4, this section only describes the requirements that have not (entirely) been satisfied by the final product. These requirements are discussed for each section of the MoSCoW analysis.

7.3.1 Must have

All requirements listed in the *Must have* section of the MoSCoW document are satisfied by the final product. This is expected as the final product would not have been in an acceptable state if these requirements had not been satisfied.

A slight difference with the specified requirements concerns the first point of section 4.8.1.3, which states that the prototype would be limited to genome build 37 due to limitations of the Ensembl API. This limitation was circumvented later in the development of the product, so the product now supports both build 36 and build 37 of the genome.

7.3.2 Should have

A requirement which is currently listed in the *Should have* section of the MoSCoW but is missing from the implementation of the current product is the first point of section 4.8.2.1. This requirement specifies that the product should support plotting of multiple samples. This requirement is currently not implemented by the product, but only lacks support in the graphical user interface of the application and is planned to be supported by the final product.

All other requirements are satisfied by the product.

7.3.3 Could have

Two requirements in the *Could have* section are not satisfied by the product. The first of these requirements is the requirement listed in section 4.8.3.2, which specifies that the product should support the plotting Cy5 data from one array against Cy3 data from another, so different subjects can be hybridised against the same control without repeating the experiment. This feature is currently not supported due to a lack of implementation time and because the feature was deemed less important than others.

The second feature not satisfied by the product is the fourth item listed in section 4.8.3.3, which mentions the visualisation of only selected genes in the relational view. This feature is currently not supported as it was found to be redundant.

7.3.4 Won't have

The single feature listed in the *Won't have* section of the MoSCoW document is not supported by the product, due to lack of development time and its low priority.

7.3.5 Conclusion

The above analysis of the system shows that all critical requirements and most of the optional requirements are satisfied by the system. The only notable exception is the single missing should have item, which is planned to be implemented in the final implementation week and should therefore also be satisfied by the product. Other missing features were defined as extra features by the MoSCoW diagram and were therefore not required to be included in the product.

As the product satisfies the basic requirements and implements a great number of the optional requirements listed in the *Could have* section of the MoSCoW, the product passes the system test.

Chapter 8

Process evaluation

This project is our final project of the bachelor Computer Sciences at the Technical University of Delft and is intended to test all skills acquired in the course of the bachelor, which not only includes Software Engineering skills but also orientation skills in a previously unknown domain of expertise. The required orientation requires among others the interviewing of clients, documentation and presentation of the product. This chapter discusses the manner in which these various elements were executed,

8.1 Project start-up

The first week of the project was mostly used by the development team to learn the basic concepts in genetics. This was done through a basic study of literature, as well as observing existing database systems and visualisation tools and creating a draft of an initial application concept. This proved to be a good basis for the interviews with the clients. It was however still clear that the team sometimes lacked knowledge of certain details, which had not yet come up in the preliminary research. This required certain research elements to continue throughout the process, even after completing the orientation phase.

The studying of literature was done on an individual basis, though certain documents were required to be read by all members of the team. The individual tools examined were divided among the various team members. A more detailed description of the orientation phase is contained in the orientation report[1], which is included in the appendix of this document.

8.2 Requirements analysis

During the requirements analysis phase, the development team regularly discussed the various components of the analysis, thus involving all team members in important design decisions. These discussions proved very useful, as there were often different opinions about design issues or a clients wishes. This has helped both the design and the specification of the requirements, which is evident from the relatively small amount of adjustments required after presenting the requirements to the clients.

8.3 Implementation

A week-to-week schedule was made for each developer during the implementation phase, based on the prioritisation of the requirements in our analysis. This allowed each developer to create his own specialisation to work on and ensured that it was clear who was responsible for certain functionality. In retrospect, it seems that the amount of work required for the front-end was underestimated though, due to the decision to make a custom implementation of a plot. To solve this planning issue, a second developer was assigned to the front-end, followed by a third person in some phases.

Other aspects of the implementation went relatively well. Most code that was checked-in showed little issues and was eventually documented and tested, which made it easy for other team members to use and adjust code. Trac was used to keep track of larger issues that could not be resolved immediately. Larger design changes, which were sometimes required due to the iterative development approach, were discussed extensively in the entire group to prevent problems on a larger scale.

8.4 Presentations

Prototype presentations to the supervisors were often problematic, due to unexpected problems with the application during the presentations. This was mainly due to hardware problems in the host system or the occurrence of a situation unforeseen by testing. The demonstrated prototypes also often lacked certain functionality that was included by the application itself but not yet incorporated in the graphical user interface, due to the lagging implementation of the user interface.

8.5 Documentation

The progress of the project has been extensively documented as required. The LaTeX software package was used to create the various documents, as to allow the documents to be managed by the version control system. The construction of each document followed the same procedure in which the structure and contents of the document were discussed in the group, after which the identified sections were divided among the team members. Each created piece of documentation was reviewed by another team member in order to keep the quality of the documents high.

8.6 Conclusion

Overall, we have gained experience on a whole range of subjects that are all relevant in ICT, by autonomously (with exception of our supervisors) creating a product for clients who work in a domain other than ICT, along with the required presentation and documentation. The group process went well and was well structured, resulting in a structured and relatively well-tested system accompanied by an extensive amount of documentation.

Chapter 9

Conclusion

This project entailed the design and implementation of a graphical, statistical analysis application for aCGH data. The applications main functionality is the displaying of the corresponding aCGH profile, as well as the analysis of this profile in order to determine any significant copy number variations in the analysed genome. The application also identifies genes located in these significant regions and provides annotation data about these genes, which are retrieved from a variety of external databases. Other functionality included by the product are the export of the displayed data as well as the displaying of relations between the identified gene in a circular plot using the Circos application.

The above listed functionality was derived from the extensive list of requirements created during the requirements phase of the development. The creation of this list was supported well by the basic knowledge acquired during the orientation phase, which proved invaluable in the interviews with the clients. The interviews and extensive discussions of the interview results proved valuable as the resulting requirements required only minor adjustments.

The initial design of the system proved sufficient as the actual implementation only differs in minor aspects. The subsystem division proved useful as this kept the different components of the system well-defined and well-separated, also allowing work on the subsystems to be performed separately. The client-server architecture of the database back-end also proved a necessary and good solution to distribute the weight of the Ibidas application to a different host. The multi-threaded design of the application also proved important as it effectively kept the application responsive when performing lengthy tasks.

The main problems encountered during the implementation of the application were the size of the data, which required a number of optimisations even though the data structures were designed accordingly. The main performance problems were encountered in the visualisation of data, which were solved by the implementation of a customised and optimised view to draw the data efficiently. The complexity of the front-end as a whole was underestimated considerably, resulting in the development of the user interface lagging behind that of the other parts of the product.

The quality of the product was tested according to the test plans drafted during the requirements phase. The unit and integration tests proved especially useful as they allowed automated testing of various subsystems and components, which ensured each component was of considerable quality before its use. The system and acceptance tests were used to assess the quality and

completeness of the entire product. As the implemented tests are sufficient according to the set test goals and the product passes all of the implemented tests, the product is deemed to be of sufficient quality for a prototype release.

The product does however not implement all the required functionality, which was shown in the analysis of the system test results. The missing functionality was rated as medium-low to low priority however and it is therefore not deemed to be a large flaw in the final product. The product has therefore passed the system test, which leaves the missing functionality to be implemented in future releases.

On the whole, it can be concluded that the product produced in this project meets the set requirements and has that the project has therefore been successful. There were no major problems in the development process, which can also be considered a success. The developed prototype is still a prototype demonstrating its possibilities however and leaves much room for improvement before it should be applied in actual use cases.

Chapter 10

Future Recommendations

As the main goal set by this project was to create a working prototype of a product, which mainly demonstrates the full capabilities of the application were it to be completed, the produced product still has much room for improvement. This chapter discusses a number of the problems and limitations of the current product and provides suggestions to solve these issues.

10.1 Licensing

Products used internally (Qt, Threadpool and Ibidas most notably) are based on different licenses. In order to release the product to the public it must be checked how to comply with these licenses and if they interfere with each other.

By replacing Threadpool (as brought forward in section 10.4) one of the licenses could be bypassed. Another potential licensing issue is that a company named webMethods holds a US software patent on XML-RPC. [11] A possible solution is presented in section 10.4.

Product	**License**
Qt	Commercial, LGPL 2.1 or GPL 3.0 (depending on license the product) [19]
Threadpool	MIT [5]
Ibidas	Unknown

Table 10.1: Products and their licenses

10.2 Graphics

Qt is developed to make simple things easy. It turns out though that more complex elements are not so easy to implement correctly in Qt. For example, adding points to a canvas is relatively easy but adding a lot of points is very slow and requires a lot of processing power. More research is needed to effectively speed up the system calls behind the addition of points. A possible solution is to re-factor the front-end to an implementation in the C programming language. The differences between Qt in Python and in C are minimal, requiring only syntax differences to be

applied throughout the current implementation of the classes that are to be converted.

Creation of the overview image in the graphical user interface (GUI) is currently slow. It is just a static image but requires all the data points of a sample to be drawn for its creation. It therefore takes a long time before this process is finished. This is the primary reason for the loading of data being slow. A solution could be to offload the creation of the overview to a new GUI thread that builds the image and returns it to the main GUI.

While the GUI is adding data to the canvas a progress bar is currently shown. The progress in the bar has no correlation to the actual progress of the tasks however. In the future the product needs a better way of showing the user that it is busy.

Although the created product is currently platform independent, differences between the GUI in either Linux, Mac or Windows do exist. These changes mainly are due to outlining differences, which results in certain inputs being placed differently. To make the GUI more uniform across the different platforms a number of small tweaks per platform need to be made. A more serious problem is the lack of OpenGL (graphics engine) support in some Linux distributions, due to a known bug in the Linux kernel. Being able to use OpenGL is not a major requirement, but it does speed up the GUI quite a lot. If the bug in Linux is not fixed a better working solution must be found.

Another small issue in the current implementation is that user input is not rigorously checked. Therefore a user is able to try to load any file into the application. Although the application is perfectly able to deal with the exceptions this could cause, the user is currently not notified neatly. More details are mentioned in section 10.5.

10.3 GUI - Controller communication

The GUI and Controller communicate through asynchronous messages, which ensures that the GUI remains responsive while the Controller is busy. A problem with the current implementation is the manner in which messages are sent between the two components, which requires the GUI to keep track of which actions it sent to the controller and their state. The current system could be simplified using the active object pattern in combination with the observer-observable pattern.

The observer-observable design pattern is used for maintaining one-way communication between one object and a set of other objects. For example, currently the Controller sends a message to the GUI that a file containing samples has been loaded. When the observer-observable pattern is used the GUI automatically is signalled that the file is loaded, therefore there is no need to actively send signals, thus reducing complexity.

The active object pattern was created to decouple the actual method invocation and the actual execution of the method. The decoupling ensures that the client thread appears to invoke a normal method. The method invocation and execution are decoupled by constructing a command object and passing it to an executor which executes it.

By combining the active object and observer-observable pattern it is possible for the controller to immediately return an active object that can be used by the GUI. While the actual operation is run in a separate thread of the active object, the graphical display can provide a default view.

When the actual operation is completed, the active object notifies it's observers (the GUI), which can then proceed to update themselves with the new information.

A short example is illustrated by he GUI instructing the controller to load a certain file. This is an operation that can take quite some time. Rather then blocking until the operation is complete or providing a callback the controller creates an active object and returns it to the GUI. The GUI tries to visualise the data but sees the load has not yet completed so it renders a temporary image, indicating the data is being loaded. Once the data is loaded the GUI is notified and can display the real data.

10.4 Database

Ibidas can handle only one request at a time and no extra threading is used to make the connection scalable. It offers a very basic XML-RPC connection that will present problems when scaling up to a large number of clients. Another problem is that exceptions that occur in server component are not properly propagated to the client. The client can only notice that something went wrong, but not what exactly went wrong.

In the future the connection could be refactored with the use of a package like Twisted. Twisted is specifically designed for efficient client-server communication and parallelisation. It is built upon basic Python sockets and threads and uses asynchronous event communication between components. Another advantage of using sockets is that the context of an exception can be captured, therefore making the client able to better deal with exceptions.

Twisted could possibly also be used as a thread pool executor, replacing the Job-Dispatcher. That makes it possible to handle exceptions better. Twisted contains a rigorously tested thread pool implementation that would be ideal.

10.5 Error handling

Exceptions are properly caught throughout the product. Obviously, this results in a stable application that is able to handle different, possibly rare, error conditions. These exceptions are all propagated back to the Controller component, but are currently not all handled correctly by the component. In future implementations the Controller should act correctly for every expected exception. Actions can simply be the passing of a message to the user notifying of the occurred exception. But it could also act on an exception, trying again for a specified number of times in certain cases. For example an exception that occurred during transmission of requests to the database could be sent again, only notifying the user if it fails a second time.

10.6 Plugins

The manner in which an algorithm or file loader is added to the product could be improved with the addition of a plugin system. The current implementation provides an interface with which such a loader can be implemented as well as supporting base classes. The interface of the subsystem also provides easy extension of the subsystem, but currently requires modification of source code as the list of available parsers is defined in the application. This list could be

generated dynamically by a plugin system, which would allow addition of extra loaders without modifying the application itself. The same can be said for the algorithm subsystem.

10.7 Caching

The product has to handle large amounts of data and currently does not cache anything, meaning that every time a profile is loaded it requires the same processing steps. In future implementations proper caching could be implemented to cut down loading time when a profile is reloaded. Graphical interface elements could possibly use the same approach. Results from database queries could also be cached to prevent overloading the server with multiple requests for the same data. These caching mechanisms could provide the application with a substantial performance increase and as such are desirable in future releases.

References

[1] N. Aben, C. Melman, S. Dentro, R. Korstanje, and J.R. de Ruiter. Angita: Orientation Report, 2010.

[2] N. Aben, C. Melman, S. Dentro, R. Korstanje, and J.R. de Ruiter. Angita: Software Design Document, 2010.

[3] N. Aben, C. Melman, S. Dentro, R. Korstanje, and J.R. de Ruiter. Angita: Technical Design Document, 2010.

[4] BC Cancer Agency. Circos website, June 2010. `http://mkweb.bcgsc.ca/circos/`.

[5] Chris Arndt. Threadpool implementation website, June 2010. `http://chrisarndt.de/projects/threadpool/`.

[6] A. Brazma, P. Hingamp, J. Quackenbush, G. Sherlock, P. Spellman, C. Stoeckert, J. Aach, W. Ansorge, C.A. Ball, H.C. Causton, et al. Minimum information about a microarray experiment (MIAME)toward standards for microarray data. *Nature genetics*, 29(4):365–371, 2001.

[7] F. Cappuzzo, F.R. Hirsch, E. Rossi, S. Bartolini, G.L. Ceresoli, L. Bemis, J. Haney, S. Witta, K. Danenberg, I. Domenichini, et al. Epidermal growth factor receptor gene and protein and gefitinib sensitivity in non-small-cell lung cancer. *JNCI Journal of the National Cancer Institute*, 97(9):643, 2005.

[8] D.F. Conrad, D. Pinto, R. Redon, L. Feuk, O. Gokcumen, Y. Zhang, J. Aerts, T.D. Andrews, C. Barnes, P. Campbell, et al. Origins and functional impact of copy number variation in the human genome. *Nature*, 2009.

[9] Edgewall. Trac website, June 2010. `http://trac.edgewall.org/`.

[10] E. Gonzalez, H. Kulkarni, H. Bolivar, A. Mangano, R. Sanchez, G. Catano, R.J. Nibbs, B.I. Freedman, M.P. Quinones, M.J. Bamshad, et al. The influence of CCL3L1 gene-containing segmental duplications on HIV-1/AIDS susceptibility. *Science*, 307(5714):1434, 2005.

[11] Google. webmethods xml-rpc patent, June 2010. `http://www.google.com/patents?id=WFV4AAAAEBAJ`.

[12] HGP. Human genome project information, June 2010. `http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml`.

[13] Reportlab inc. Reportlab website, June 2010. `http://www.reportlab.com/`.

[14] C. Klijn and J. de Ronde. KC-SMART Vignette, 2010.

[15] S.J.L. Knight, R. Regan, A. Nicod, S.W. Horsley, L. Kearney, T. Homfray, R.M. Winter, P. Bolton, and J. Flint. Subtle chromosomal rearrangements in children with unexplained mental retardation. *The Lancet*, 354(9191):1676–1681, 1999.

[16] J.A. Lee and J.R. Lupski. Genomic rearrangements and gene copy-number alterations as a cause of nervous system disorders. *Neuron*, 52(1):103–121, 2006.

[17] Little, Brown, and company. *A Dictionary of Greek and Roman biography and mythology.* Smith, William, Sir, 1867. URL: http://quod.lib.umich.edu/cgi/t/text/text-idx?c=moa;idno=ACL3129.0001.001.

[18] NBIC. Ibidas website, June 2010. `https://wiki.nbic.nl/index.php/Ibidas`.

[19] Nokia. Qt licensing overview, June 2010. `http://qt.nokia.com/products/licensing/`.

[20] Nokia. Qt website, June 2010. `http://qt.nokia.com/`.

[21] Baylor College of Medicine. Copy number variation may stem from replication misstep, January 2008. `http://www.sciencedaily.com/releases/2007/12/071227183904.htm`.

[22] J. Sebat, B. Lakshmi, D. Malhotra, J. Troge, C. Lese-Martin, T. Walsh, B. Yamrom, S. Yoon, A. Krasnitz, J. Kendall, et al. Strong association of de novo copy number mutations with autism. *Science*, 316(5823):445, 2007.

[23] D. St Clair. Copy number variation and schizophrenia. *Schizophrenia bulletin*, 2008.

Appendix A

Glossary

Aberration Modification of the normal chromosome complement due to deletion, duplication, or rearrangement of genetic material.

Angita Angita is the early Roman goddess of healing, magic and witchcraft. [17]

Array comparative genomic hybridisation (aCGH) Array comparative genomic hybridisation is a technique to detect genomic copy number variations at a higher resolution level than chromosome-based comparative genomic hybridisation (CGH).

Copy number variant (CNV) A copy number variant is a segment of DNA in which non-pathogenic copy-number differences have been found by comparison of two or more genomes. The segment may range from one kilo-base to several mega-bases in size. Humans (being diploid) ordinarily have two copies of each autosomal region, one per chromosome. This may vary for particular genetic regions due to deletion or duplication.

Genome annotation The process of identifying the locations of genes and all of the coding regions in a genome and determining what those genes do. An annotation (irrespective of the context) is a note added by way of explanation or commentary. Once a genome is sequenced, it needs to be annotated to make sense of it.

VUMC The VU University Medical Centre is an academic medical centre, concerned with patient care, higher education and medical research.

NKI The Netherlands Cancer Institute researches the cause and effects of cancer. The institutes three major areas of research are fundamental, clinical and translational cancer research.

Ibidas Ibidas is aimed to be an integration platform for various sorts of biological data. Its purpose is providing a very minimal but flexible database scheme. It is possible to store any kind of biological data while still keeping the learning curve for using the system low. [18]

SQL Structured Query Language is a database computer language designed for managing data in relational database management systems.

US2, GPL and ETABM File formats that are used for aCGH data storage.

HTML HyperText Markup Language is the predominant markup language for web pages. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items.

XML Extensible Markup Language is a set of rules for encoding documents in machine-readable form. XML's design goals emphasise simplicity, generality, and usability over the Internet. It is a textual data format, with strong support via Unicode for the languages of the world. Although XML's design focuses on documents, it is widely used for the representation of arbitrary data structures.

XML-RPC XML-RPC provides a fairly lightweight means by which one computer can execute a program on a co-operating machine across a network like the Internet. It is based on XML and is used for everything from fetching stock quotes to checking weather forecasts.

SOAP Simple Object Access Protocol is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks.

Qt Qt is a cross-platform application development framework widely used for the development of GUI programs (in which case it is known as a widget toolkit). It is produced by Nokia's Qt Development Frameworks division and is free and open source software.

Circos Circos is a software package for visualising data and information. It visualises data in a circular layout this makes Circos ideal for exploring relationships between objects or positions. [4]

Appendix B

Assignment description

This assignment encompasses the development of a software product aimed at the analysis of CGH profile data in order to determine copy number variations in the sampled genome and assist the user in formulating a diagnosis based on these variations. The project consists of two related assignments, each executed for a separate client. The first assignment is executed for the VUMC medical institute of Amsterdam, the second assignment for the NKI cancer research institute.

The VUMC client is a research institute within the VUMC, who performs CVS tests in order to determine CNVs (if present) in unborn children. The data generated by such tests is of a large magnitude, which hinders manual interpretation of the results of such a test. The client therefore requires an application which can analyse and display this data in such a manner that the results of such a test are clear for the researcher.

The assignment executed for the VUMC therefore focusses on the analysis and visualisation of data obtained from chorionic villus sampling. Chorionic villus sampling (CVS) is a form of prenatal diagnosis to determine chromosomal or genetic disorders in the foetus. It entails getting a sample of the chorionic villus (placental tissue) and analysing its genome in order to find copy number variations. The analysis produces an aCGH profile, which is processed by the product to determine and display genomic copy number variations in a clear manner. The product should also display the various genes contained in CNV regions and provide the user with disease related annotation data, which supports the user in determining the effect of identified copy number variations.

The NKI assignment involves the same analysis and visualisation procedures as the VUMC assignment (2.1.1), the main difference being the nature of the data under analysis. The NKI client researches CNV in cancer cells in order to identify mutations in the genome. Data from cancer research is harder to analyse due to large amounts of mutations in the genome and differences in sampled cells (tumorous vs non-tumorous). The client uses both human and mouse genome data in his research.

The NKI assignment has an extra focus on visualising relations between genes in regions occupied by CNVs, which allows the user to identify relations between various CNVs, and often analyses multiple samples at the same time.

Appendix C

Orientation Report

Graphical application for the statistical analysis of copy number variations in aCGH data

Thesis BSc Project IN3405

Orientation Report

version 1.1

Julian de Ruiter (1387367)
Chris Melman (1358359)
Rien Korstanje (1270389)
Stefan Dentro (1313320)
Nanne Aben (1369342)

DELFT UNIVERSITY OF TECHNOLOGY
EEMCS FACULTY
JUNE 28, 2010

Contents

Chapter 1

Introduction

1.1 Purpose

This document describes the steps undertaken in the orientation phase performed for the development of a graphical, statistical analysis application of copy number variations in aCGH data and their results.

1.2 Scope

The document describes the design process of a graphical, statistical analysis application for aCGH data, codenamed Angita. The application is being developed for the NKI and the VUMC.

The application is able to import aCGH data from a local file and identify significant aberrations in the genomic data using various algorithms. It provides an interface in which a loaded aCGH profile can be viewed in a linear fashion. The interface also displays the various genes located in identified aberrant regions. Genes are annotated by various databases to provide more information about the gene in question. The application also provides a view which displays relations between the genes in the identified aberrant regions of the genome, allowing the type of relations to be shown can be chosen.

The application aims to allow quick analysis of the provided aCGH data by displaying a clear view of the aCGH profile, providing various algorithms to identify aberrant regions of the genome and displaying the identified regions in a clear manner. The displayed relations between genes in the identified regions provide the user with a significant amount of extra data, which can help identify non-obvious relations between the identified regions and corresponding genes.

1.3 Definitions, acronyms and abbreviations

Aberration Modification of the normal chromosome complement due to deletion, duplication, or rearrangement of genetic material.

Allele One of two or more alternative forms of a gene that arise by mutation and are found at the same place on a chromosome.

Angita Angita is the early Roman goddess of healing, magic and witchcraft. [7]

Array comparative genomic hybridisation (aCGH) Array comparative genomic hybridisation is a technique to detect genomic copy number variations at a higher resolution level than chromosome-based comparative genomic hybridisation (CGH).

Circos Circos is a software package for visualising data and information. It visualises data in a circular layout this makes Circos ideal for exploring relationships between objects or positions. [1]

Copy number variant (CNV) A copy number variant is a segment of DNA in which copy-number differences have been found by comparison of two or more genomes. The segment may range from one kilo-base to several mega-bases in size. Humans (being diploid) ordinarily have two copies of each autosomal region, one per chromosome. This may vary for particular genetic regions due to deletion or duplication.

Genome annotation The process of identifying the locations of genes and all of the coding regions in a genome and determining what those genes do. An annotation (irrespective of the context) is a note added by way of explanation or commentary. Once a genome is sequenced, it needs to be annotated to make sense of it.

HTML HyperText Markup Language is the predominant markup language for web pages. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items.

Ibidas Ibidas is aimed to be an integration platform for various sorts of biological data. Its purpose is providing a very minimal but flexible database scheme. It is possible to store any kind of biological data while still keeping the learning curve for using the system low. [8]

NKI The Netherlands Cancer Institute researches the cause and effects of cancer. The institutes three major areas of research are fundamental, clinical and translational cancer research.

Qt Qt is a cross-platform application development framework widely used for the development of GUI programs (in which case it is known as a widget toolkit). It is produced by Nokia's Qt Development Frameworks division and is free and open source software.

SOAP Simple Object Access Protocol is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks.

SQL Structured Query Language is a database computer language designed for managing data in relational database management systems.

US2, GPL and ETABM File formats that are used for aCGH data storage.

VUMC The VU University Medical Centre is an academic medical centre, concerned with patient care, higher education and medical research.

XML Extensible Markup Language is a set of rules for encoding documents in machine-readable form. XML's design goals emphasise simplicity, generality, and usability over the Internet. It is a textual data format, with strong support via Unicode for the languages of the world. Although XML's design focuses on documents, it is widely used for the representation of arbitrary data structures.

XML-RPC XML-RPC provides a fairly lightweight means by which one computer can execute a program on a co-operating machine across a network like the Internet. It is based on XML and is used for everything from fetching stock quotes to checking weather forecasts.

1.4 Overview

The second chapter of this document describes the steps undertaken to gain a firm understanding of the domain in which the product is to operate. Chapter three notes a number of existing software systems and elaborates on their positive and negative aspects. The fourth chapter describes the software technologies and tools that were considered to actually implement the product, whilst chapter five describes algorithms relevant to the software product.

Chapter 2

Domain

The project itself is an example of the classic software development case, in which developers design and create a product that is going to be used by domain experts in another field. In order to be able to design a usable product, it is important that the development team acquires a concrete understanding of the fields most important terms and acquaints itself with the used practices.

This chapter describes the various steps taken to acquire this understanding. The first section describes the literature study undertaken to familiarise with the various terms used in this area of the bioinformatics field, followed by an description of the interview process undertaken to discover the practices of the concerned clients. The remaining section of this chapter shortly lists the various databases and currently existing systems that are relevant to the product under development.

2.1 Literature study

The main objective of the literature study performed for this project, was to ensure that the development team gained a firm understanding of the principles and practices concerning CGH analysis experiments. In order to do so, one must also have a good understanding of basic genomic principles, such as the structure of DNA, the various relations between DNA and gene expression, the effects of deletions or amplifications of genomic material etcetera.

2.1.1 Studied material

2.1.1.1 Books

The book most referenced by the development team for orientation purposes is Molecular Biology of the Cell [3], which is used as the primary introductory material in teaching genomics as the Delft University of Technology in the Life Science and Technology bachelor. The books excerpt is shown below:

> Molecular Biology of the Cell is the classic in-depth text reference in cell biology. By extracting fundamental concepts and meaning from this enormous and ever-growing field, the authors tell the story of cell biology, and create a coherent framework through which non-expert readers may approach the subject. Molecular Biology of

> the Cell not only sets forth the current understanding of cell biology (updated as of Fall 2001), but also explores the intriguing implications and possibilities of that which remains unknown.

Molecular Biology of the Cell contains a wealth of knowledge about DNA and the various cell processes concerning DNA and has therefore extensively been referenced by the development team to gain insight in these areas. The book also contains a section specifically about probable causes of cancer and the effects of cancer on the genome [2], which directly relates to the NKI clients wishes concerning the application and was therefore also of great use.

2.1.1.2 Papers

Though the book Molecular Biology of the Cell contains a wealth of information concerning DNA processes, it mentions little specifically about copy number variations or its effects, involved practices and applications. To learn more about the topics a few papers containing more information have also been referenced during the literature study.

The most noteworthy of the referenced papers is the paper written by Conrad et al. [4], which describes the origins and functional impact of copy number variation in a human genome. This paper contains a good description of of the term Copy Number Variation, how such CNVs are discovered and the basic effect of genomic CNVs.

The literature study was also aimed at finding papers in which various visualisation techniques are used to display properties of genomic data efficiently, due to the large amount of data provided in genomic analysis. Good examples of papers demonstrating such visualisation using are provided by Ledford H. [6] and Stephens et al[11].

In summary, the following papers have been referenced during the literature study:

- Conrad et al. Origins and functional impact of copy number variation in the human genome. Nature (2009) [4]
- Sebat et al. Large-scale copy number polymorphism in the human genome. Science (2004) [10]
- Stranger et al. Relative impact of nucleotide and copy number variation on gene expression phenotypes. Science (2007) [12]
- Ledford H. Big science: The cancer genome challenge. Nature (2010) [6]
- Stephens et al. Complex landscapes of somatic rearrangement in human breast cancer genomes. Nature (2009) [11]

2.1.2 Definition list

In order to provide some kind of universal definition list to reference during the project as well as confirm a proper understanding of the material, a definition list of the most important terms has been created during the literature study. This list is shown below. The various definitions have been derived from a combination of the definitions contained in the book Molecular Biology of the Cell and their corresponding definition on Wikipedia. Terms derived otherwise are accompanied by corresponding references.

Amplicon Amplicons are the products of polymerase chain reactions (PCR) or ligase chain reactions (LCR). They are pieces of DNA that have been synthesised using amplification techniques. Amplicons can also occur naturally through events such as unequal crossing over that occurs during meiosis between misaligned homologous chromosomes.

Amplification Gene amplification is any duplication of a region of DNA that contains a gene. Duplications arise from an event termed unequal crossing-over that occurs during meiosis between misaligned homologous chromosomes.

Bioconductor Bioconductor is an open source software project written in R for the analysis and comprehension of genomic data generated by wet lab experiments in molecular biology. It provides statistical and graphical methods for the analysis of genomic data and the inclusion of biological metadata of genomic data.

Chromosome A chromosome is an organised structure of DNA and protein that is found in cells. It is a single piece of coiled DNA containing many genes, regulatory elements and other nucleotide sequences. The DNA molecule may be circular or linear, and can be composed of 10,000 up to 1,000,000,000 base pairs.

Circos Circos is a software tool designed for visualising genomic data such as alignments, conservation, and generalised 2D data, such as line, scatter, heat-map and histogram plots. It was developed to visualise intra- and inter-chromosomal relationships within one or more genomes or between any two or more sets of objects with a corresponding distance scale. Circos is written in Perl and produces bitmap (PNG) and vector (SVG) images using plain text configuration and input files.

CGH Comparative genomic hybridisation (CGH) is a molecular-cytogenetic method for the analysis of copy number changes in DNA content and in tumour cells. CGH can only detect duplications and deletions, as translocations or inversions do not change the copy number.

Copy Number Copy numbers describe duplications and deletions of large blocks of DNA. When comparing two or more genomes, copy number variants (CNVs) can be found. Copy number variation can be discovered by cytogenetic techniques such as comparative genomic hybridisation (CGH). Humans (being diploid) ordinarily have two copies of each autosomal region, one per chromosome.This may vary for particular genetic regions due to deletion or duplication. CNVs may either be inherited or caused by de novo mutation. A recently proposed mechanism for the cause of some CNVs is fork stalling and template switching, a replication misstep.

Co-regulation Co-regulation occurs when the transcription of particular set of genes is activated and deactivated by the same factors, resulting in the genes being activated and deactivated together. Co-regulation is not essentially the same as co-expression, as any step between transcription and actual expression can still prevent expression of a gene, preventing the actual expression of the genes from being activated and deactivated together.

Cytoscape Cytoscape is an open source bioinformatics software platform for visualising molecular interaction networks and integrating with gene expression profiles and other state data.

Deletion In genetics, a deletion is a mutation in which a part of a chromosome or a sequence of DNA is missing.

Ibidas Ibidas is aimed to be an integration platform for various sorts of biological data. The data can be browsed using Cytoscape.

Immunoprecipitation Immunoprecipitation (IP) is the technique of precipitating a protein antigen out of solution using an antibody that specifically binds to that particular protein.

KC-Smart KC-Smart is an aCGH analysis package for Bioconductor that works non-discretised data. By using kernel convolution, it prevents loss of potentially valuable biological information, which would come from discretisation methods.

Microarray A DNA microarray consists of an arrayed series of thousands of microscopic spots of DNA oligonucleotides, called features, each containing picomoles of a specific DNA sequence, known as probes. Probe-target hybridisation can be detected and quantified by detection of fluorophore-labeled targets to determine relative abundance of nucleic acid sequences in the target.

Orthology Orthologs, or orthologous genes, are genes in different species that are similar to each other because they originated from a common ancestor.

Probe A probe is a fragment of DNA or RNA, which is used in DNA or RNA samples to detect the presence of nucleotide sequences (the DNA target) that are complementary to the sequence in the probe.

Promoter A promoter is a region of DNA that facilitates initiation of the transcription of a particular gene. Promoters are typically located near the genes they regulate, on the same strand and upstream (towards the 5' region of the sense strand).

SNP A single-nucleotide polymorphism (SNP, pronounced snip) is a mutation in a DNA sequence, where a single nucleotide A, T, C, or G in the genome (or other shared sequence) differs between members of a species (or between paired chromosomes in an individual). For example, two sequenced DNA fragments from different individuals, AAGCCTA to AAGCTTA, contain a difference in a single nucleotide.

Synthetic lethality Synthetic lethality arises when a combination of mutations in two or more genes leads to cell death, whereas a mutation in only one of these genes does not.

STRING STRING (Search Tool for the Retrieval of Interacting Genes/Proteins) is a database and web resource of known and predicted protein-protein interactions.

Upstream region An upstream region is a genomic region closer to the 5' end of the sense strand compared to the referenced region.

Yeast-2-Hybrid A yeast-2-hybrid system is a molecular biology technique used to discover protein-protein interactions.

2.2 Interview sessions

To discover more about the actual work performed by the clients and the practices and conventions involved, both clients were interviewed twice. All interviews were held in an informal manner, loosely following a list of specific questions which were prepared beforehand. In the first session, we have asked general questions like:

1. What would you like to visualise?
2. What program(s) do you currently use for that purpose?

In the follow-up interviews, we had delved deeper into the subject and we could ask more specific questions regarding data-formats, used databases and used algorithms. These interviews were also used to present the ideas of the development team through both early graphical user interface sketches and vocal explanation, in order to get early feedback on the first steps of the design process.

The interview sessions were of great use for the orientation phase as they greatly clarified the actual processes used by the clients and their specific requirements for and interests in an eventual application. The inherent visit to the clients also allowed the development team to experience the clients working environment and currently used software systems, which also provided some insight into existing solutions and some ideas for the eventual application design.

2.3 Existing software systems

Both the literature study and the interview turned up a few software systems that are currently used in the field of bioinformatics for modelling and visualisation purposes similar to that of the required product. This section briefly mentions a few of these systems, a detailed explanation of these systems is given in the next chapter.

One of the systems is *Cytoscape*, which is an open source bioinformatics software platform for visualising molecular interaction networks and biological pathways and integrating these networks with annotations, gene expression profiles and other state data. Cytoscape is currently used by the NKI client to model relational data.

The *Nexus* system is currently used by the VUMC client to analyse CGH profile data. It is a proprietary software product, which has therefore only briefly been inspected in a visit to the VUMC. Part of its functionality includes plotting CGH profile data, providing methods to determine significant regions in the CGH profile data and provide extra data about these regions by linking to external databases such as Ensembl.

Circos is a software package for visualising data and information. It visualises data in a circular layout, which makes Circos ideal for exploring relationships between objects or positions.

Chapter 3

Existing Systems

3.1 Nexus

Nexus is the software package currently in use at VUMC. It is used to read aCGH data profiles and presents the user with the results. The main features include plotting the data together with genetic information, such as genes, chromosomes and bands. Its biggest advantage is that it can read multiple samples at a fast pace.

It has its weaknesses though. During the work process at VUMC external databases are browsed to compile a better interpretation of the data. Since Nexus does not have a direct connection with databases like OMIM and DECIPHER (which contain phenotypes) an external view is started in a browser. The browser takes the user directly to the specified region in Nexus, but during interviews it became clear this is a source of irritation.
Another drawback is that Nexus can not dynamically visualise the probe data. When the aCGH data is browsed, it is visualised in a simplified but inaccurate representation. This works fine when the user has not yet zoomed in. However, when the user does zoom in and finds a region of interest, he has to switch to a different view, again enter this region of interest and then wait until the probes have been loaded and drawn. This view is completely also static: it is not possible to scroll through the data, nor zooming in on the data, without loading the whole visualisation again and waiting for a long time. This too was a source of irritation for the client.

3.2 Cytoscape

Cytoscape is a software package that specialises in visualising molecular interaction networks. Although it is set up to be generic it is very powerful and flexible. Therefore it could be adapted to present aCGH data to the user and provide ways of interacting with it. Another plus is that an Ibidas plugin already exists. It is written in Java and set up to be modular. Especially in the NKI case it could have been an excellent choice.

On the other hand, Cytoscape would present VUMC client with too many options. It is too generic for a task that requires a specific workflow. The extra options would provide more clutter instead of more functionality. It would also present the user with a steep learning curve when starting to work with the product. Therefore the choice has been made to build a new application that does incorporate the usability features required.

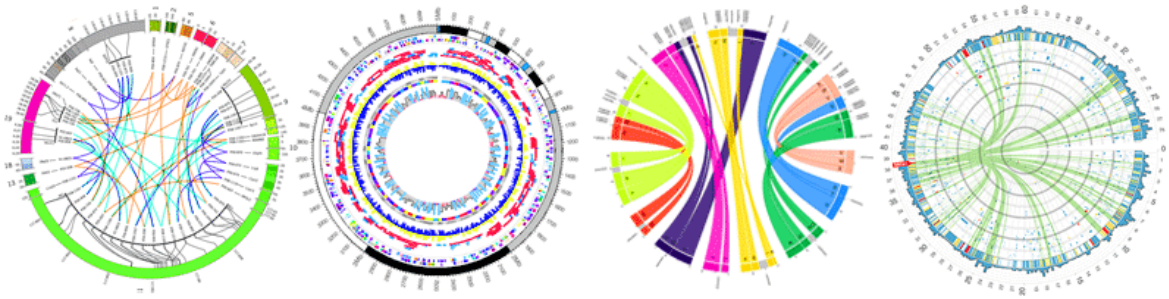


Figure 3.1: Circos presentation images [1].

3.3 Circos

Circos is becoming a major hit in genomics publications with more and more papers being enhanced with colourful images presenting research results. Circos is created with the idea that relations between entities, especially when there are a lot of them, are best presented in a circular graph. Since the required product has to be able to present relations between genomic regions Circos is an excellent choice to do so.

Internally Circos uses a configuration file with a large set of options. This is a flexible solution that is easily combined with other software packages. The only major hurdle foreseen is that Circos requires a Perl interpreter which could present problems when designing the product for a number of different types of platforms.

3.4 Ensembl

Ensembl is a database that contains basic genomic information for a number of species. Apart from that it also provides a genome browser that serves as a front-end to the database. This genome browser can be used to compare aCGH data results with a reference genome to spot differences. The gnome browser is web-based and shows data in a clean and clear manner.

Since the genome browser from Ensembl is closed and their interface changes every now and then it is not possible to use it in the product reliably. Especially the changing APIs could present a major headache in maintaining the product later on. Another limitation of Ensembl is that the data is presented as a flat image with no user interactivity possible. The choice was made to keep the visualisation techniques used by Ensembl in mind when designing the product.

Other genome browsers as Apollo, Argo, Artemis and GBench were considered. But all of them have issues with either availability of source code to extend or are created as a web application.

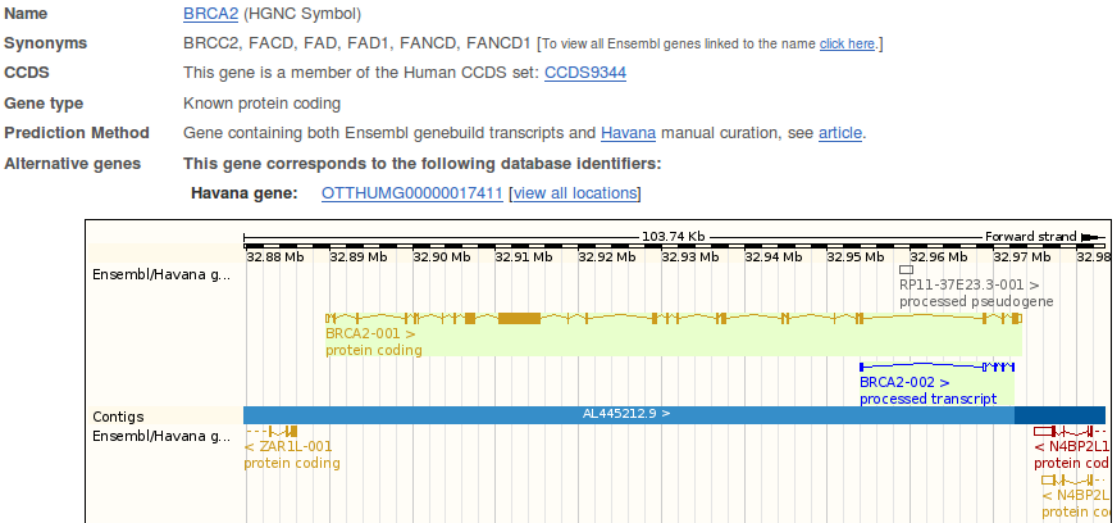


Figure 3.2: Ensembl screen showing known cancer related gene BRCA2.

Chapter 4

Technology

4.1 Programming Languages

This section discusses the programs possible languages that have been investigated for use in this project. First we discuss the requirements of this project for each programming language and we then compare the programming languages that matches these requirements.

4.1.1 Requirements

Multi-platform Due to the diversity of operating systems (Windows, MacOs) used by the client the the language of choice must be supported on multiple platforms.

Modern The language should be a modern language that supports Object Orientated programming as well as garbage collection to ensure that modern design principles can be used and minimise the time spend creating support code.

Standalone The language should be executed from within it's own interpreter or executable, rather then within another application such as a web browser.

4.1.2 Java

Java matches all the requirements discussed above. Additionally all members of the team have extensive experience in Java and there are well known and well established development tools available. As such Java is a good candidate.
Java was not considered suitable for this project. Being strongly typed and fairly strict it was assumed to seriously hamper the iterative development cycle.

4.1.3 Perl

Perl matches the requirements above and is considered to be one of the most flexible languages. However the language syntax and structure are quite complex, the installation proved to be cumbersome outside of Unix systems and the relative unfamiliarity of the development team with Perl counted against it.

4.1.4 Python

Python matches the requirement above and is considered to be a small and elegant language suitable for iterative development. Additionally members of the development team had some experience in Python. Further more, Python is considered to be a de-facto standard in Bio Informatics.

With no apparent downsides Python was selected as the preferred language for the project.

4.2 Frameworks

Elements such as the Graphical User Interface and plots are frequently used in many programs. As such frameworks have been developed that allow these to be implemented with relative ease. Python has several of such frame works, these are discussed below.

4.2.1 Plotting

There were several requirements for any plotting library. Due to the expected size of the it data it had to be able to plot a large number of data points quickly. To facilitate the clients work flow it has to allow easy navigation and be integrated with a graphical user interface framework. Finally to properly show where the user is on the genome, the view has to be able to display a customised axis.

Python has a large number of plotting Libraries in various states of development [9]. To select a plotting Library a a test case was written for the three major plotting libraries in Python, Matplotlib, Veusz and PyQwt. None could comfortably render more then 10.000 points. Neither was it possible to customise the axis sufficiently to display a genome.

With no plotting library to suit our specific needs, it was decided that it would be better to write a custom plotting utility.

4.2.2 Graphical User Interface Frameworks

The purpose of the graphical user interface is to present the application to the user. As such it has to be aesthetically pleasing to the eye. Additionally, as no suitable plotting Library could be found, the framework should provide a powerful and easily customisable graphics framework. There is an overwhelming number of GUI Frameworks for Python, most however are no longer being actively developed. The few that are such as PyGtk and PyTkinter don't provide the required graphics framework.

PyQt does provide a graphics framework capable of handling several thousand of individual graphical items [13]. The framework is also well documented and customisable.

4.2.3 Ibidas

Ibidas is a data management system in development by TU Delft. Ibidas can be used as database system and has the advantage that it can store different kind of data, so we are not bounded by 2D data. Next to that it would be easier to couple multiple databases and it could create one

stable interface instead of changeable interface for each database. The disadvantages where that the system is in alpha phase of development and almost not documented. This was partly solved since we could get support from the developers of ibidas

4.3 Development Tools

Our approach is a combination of the waterfall model and rapid prototyping. To ensure code quality we use test driven development where feasible. To support this development strategy we have investigated several development tools that we were not familiar with. These are generally accepted unless otherwise noted.

Modipyd To enable test driven development tests need to be run automatically one each change. Moipyd provides this functionality for Python.

Trac SVN is commonly used to manage the project files. Trac provides SVN in combination with a wiki and a ticketing system which allows developers to discuss issues in an integrated environment.

pep8 To ensure code quality it is possible to use a style checker. Pep8 provides style checking in for the Python standard code style. However the Python standard code style was considered to be illegible and as such this tool was not accepted for use.

Buildbot Creating a nightly build and running all unit tests on it ensures that revisions placed on SVN are. Buildbot provides such functionality integrated with Trac.

Chapter 5

Algorithms

The most important algorithms required by the application are algorithms that identify significant aberrant genomic regions in analysed CGH samples. As the resulting product is a prototype, it does not not contain an entire library of such algorithms but rather implements a few algorithms which are most likely to actually be used by the clients and demonstrate the algorithm functionality of the prototype.

From the literature study and the interviews held with our clients, two specific algorithms were identified to be most valuable to the clients if implemented in the prototype: the KC-smart algorithm and a threshold algorithm. The details of these algorithms are discussed in the following sections.

5.1 KC-smart

The KC-smart algorithm has been actively developed by the NKI client and is freely available in a R implementation. The description of the algorithm according to the algorithms documentation is shown below:

> KC-SMART is a method for finding recurrent gains and losses from a set of tumour samples measured on an array Comparative Genome Hybridisation (aCGH) platform. Instead of single tumour aberration calling, and subsequent minimal common region definition, KC-smart takes an approach based on the continuous raw log2 values. KC-smart Gaussian locally weighted regression to the summed totals positive and negative log2 ratios separately. This result is corrected for uneven probe spacing as present on the given array. Peaks in the resulting normalised KC score are tested against a randomly permuted background. Regions of significant recurrent gains and losses are determined using the resulting, multiple testing corrected, significance threshold. [5]

The KC-smart algorithm has a number of interesting features, one being that the KC-smart algorithm can be run on multiple samples, providing an aggregate result in which KC-smart identifies can be used to identify correlation between different samples. Another interesting feature is the ability to perform multi-scale analysis. The algorithm can be run using different kernel widths, where small kernel widths will allow detection of small regions of aberration and vice versa large kernel widths allow large regions of aberration to be detected.

5.2 Threshold

The interview with the VUMC client lead to the identification of a thresholding algorithm in their work process, in which a three or more consecutive data points in a CNV sample with an absolute log ratio above a fixed threshold are considered to be significant.

The implementation of such an algorithm is relatively simple and can be done by iterating over a per-chromosome-ordered collection of the data points in a CNV sample and collecting groups of consecutive points that satisfy the two requirements mentioned above. An easy improvement to the current work process is achieved by allowing the number of required consecutive points and the threshold values can be specified by the user before running the algorithm, as the client currently works with fixed values.

References

[1] BC Cancer Agency. Circos website, June 2010. `http://mkweb.bcgsc.ca/circos/`.

[2] Bruce Alberts, Alexander Johnson, and Peter Walter. *Cancer*, chapter 20, pages 1205–1268. Garland Science, 5th edition, 2008.

[3] Bruce Alberts, Alexander Johnson, and Peter Walter. *Molecular Biology of the Cell*. Garland Science, 5th edition, 2008.

[4] D.F. Conrad, D. Pinto, R. Redon, L. Feuk, O. Gokcumen, Y. Zhang, J. Aerts, T.D. Andrews, C. Barnes, P. Campbell, et al. Origins and functional impact of copy number variation in the human genome. *Nature*, 2009.

[5] C. Klijn and J. de Ronde. KC-SMART Vignette, 2010.

[6] H. Ledford. Big science: The cancer genome challenge. *Nature*, 464(7291):972, 2010.

[7] Little, Brown, and company. *A Dictionary of Greek and Roman biography and mythology*. Smith, William, Sir, 1867. URL: http://quod.lib.umich.edu/cgi/t/text/text-idx?c=moa;idno=ACL3129.0001.001.

[8] NBIC. Ibidas website, June 2010. `https://wiki.nbic.nl/index.php/Ibidas`.

[9] Python. Numeric and scientific plotting, May 2010. `http://wiki.python.org/moin/NumericAndScientific/Plotting`.

[10] J. Sebat, B. Lakshmi, J. Troge, J. Alexander, J. Young, P. Lundin, S. Maner, H. Massa, M. Walker, M. Chi, et al. Large-scale copy number polymorphism in the human genome. *Science*, 305(5683):525, 2004.

[11] P.J. Stephens, D.J. McBride, M.L. Lin, I. Varela, E.D. Pleasance, J.T. Simpson, L.A. Stebbings, C. Leroy, S. Edkins, L.J. Mudie, et al. Complex landscapes of somatic rearrangement in human breast cancer genomes. *Nature*, 462(7276):1005–1010, 2009.

[12] B.E. Stranger, M.S. Forrest, M. Dunning, C.E. Ingle, C. Beazley, N. Thorne, R. Redon, C.P. Bird, A. de Grassi, C. Lee, et al. Relative impact of nucleotide and copy number variation on gene expression phenotypes. *Science*, 315(5813):848, 2007.

[13] Troll Tech. *Troll Tech Blog*. 2010. URL: http://labs.trolltech.com/blogs/2009/03/03/performance-and-qt-45/.

Appendix D

Plan of Action

Graphical application for the statistical analysis of copy number variations in aCGH data

Thesis BSc Project IN3405

Plan of Action

version 1.3

Julian de Ruiter (1387367)
Chris Melman (1358359)
Rien Korstanje (1270389)
Stefan Dentro (1313320)
Nanne Aben (1369342)

DELFT UNIVERSITY OF TECHNOLOGY
EEMCS FACULTY
JUNE 28, 2010

Chapter 1

Introduction

1.1 Background

This project is the final project of our Bachelor in Computer Science. We were allowed to choose a project in our own field of interest and we decided to approach Mr. Reinders, Coordinator of the Bioinformatics Master Track at the Delft University of Technology (DUT). Mr. Reinders proposed a number of possible assignments. From these assignments, we have chosen one in which we have to build an application for both the VUMC and the NKI that can be used to visualise aberrant genomic data. A more detailed description of the assignment is given in chapter two.

1.2 The DUT Bioinformatics Research Group

The research group aims to better understand fundamental biophysical and biochemical processes by developing instrumentation for the analysis of bio molecules that is improved with respect to sensitivity, speed, and cost; and by providing better methodologies for the analysis and interpretation of complex bioinformatics data that describe biological processes.

We will be supervised and guided by some of the staff members and we will be able to use some of their facilities, such as their database and workspaces. A more detailed description is given in chapter four.

1.3 The plan of action

In this plan of action, we define the problem, the current situation and how we will approach this in our project. The remainder of this chapter gives a short introduction to copy number variations, which is critical for proper understanding of the product. The next chapter describes the assignment in detail, chapter three describes methodologies we will use in this project, chapter four describes practical issues such as team members and workspaces and chapter five describes the methodologies we will use to assure high product quality.

1.4 Copy number variation

Genomes vary from one another in various ways and it is this genetic variation that underpins the heritability of human traits. Over the past two to three years, genome sequences from various

individual humans have been sequenced, allowing studies to compare these sequences to each other and the base human reference genome.[4]

The studies allow estimation of the relative contributions of sequence (base substitutions) and structural variation (insertions, deletions, CNVs and inversions). The term CNV (copy number variation) is generally used to describe all quantitative variation in the genome, which includes tandem arrays of repeats as well as deletions and duplications.[2]

1.4.1 Origin

CNVs are generally either inherited or caused by de novo mutation. Such de novo mutations are generally caused by genomic rearrangements, usually occurring through deletions, duplications, inversions and translocations. Especially low copy repeats, which are basically small repeated sequences in the genome, are susceptible to such rearrangements, mainly due to their size and high similarity. [7] Another recently proposed mechanism for the cause of CNVs is fork stalling and template switching during DNA replication.[8]

1.4.2 Effects

CNVs can affect gene function in several manners. Functional loss can be caused by deletion or disruption of a gene, the effect of which depends on the CNV occurring an a recessive or a dominant allele and the cellular function of affected gene products. CNVs can also disrupt regulatory elements, generate novel products or disrupt other genes through position effects. However, some CNVs spanning multiple genes are known to have no visible phenotype.

CNVs are associated with susceptibility or resistance to disease. Especially cancer cells can show relatively large numbers of CNVs as seen in non-small cell lung cancer cells, who generally display a relatively high EGFR gene copy number.[1] An example of higher resistance to diseases due to CNVs is the CCL3L1 gene, in which higher copy numbers have been associated with a lower susceptibility to human HIV infection.[3] Copy number variations have also been associated with autism[9], schizophrenia[10] and idiopathic learning disability[6].

CNVs are also thought to be responsible for a substantial amount of variability in the human phenotype and behavioural traits, by affecting dosage sensitive genes.

1.4.3 Application

CNVs are currently being researched for various reasons, the most important being disease related. As mentioned in section 1.4.2, CNVs can have a substantial effect on a persons susceptibility or resistance to disease. Both these effects of CNVs are worth researching, as a CNV identified as a certain susceptibility can be used for medical diagnostic purposes and certain resistance CNVs may some day be able help in the prevention or curing of certain disease types.

Much research is currently aimed at identifying the effects of various CNVs. An example is the study of cancerous cells, in which generally many CNVs occur. These CNVs can be studied to identify frequently occurring CNVs in the specific cell types, as well as the impact the CNVs have on the cell and why these CNVs lead to a cancerous phenotype. This type of research is a part of the research currently being executed by the NKI client.

Another example of the use of CNV data is in the branch of medical diagnostics. An example of this use is given by the VUMC client, who analyses the genome of foetuses in order to determine its copy number variations. The found copy number variations are compared against databases of known copy number variations and their various effects, which allows the diagnostician to screen the foetus genome for known diseases. Based on the results of this process, the diagnostician can recommend a patient to terminate the pregnancy prematurely if the resulting child has a high probability of suffering a severe disease.

Chapter 2

Project Description

This project encompasses the development of a software product aimed at the analysis of CGH profile data in order to determine copy number variations in the sampled genome and assist the user in formulating a diagnosis based on these variations. The project consists of two related assignments, each executed for a separate client. The next section first provides a short introduction to both of the clients and their interest in the product, followed by a description of the feature set common to both clients. The third section explains any features specific to a client.

2.1 Client background

As mentioned in the introduction, the project consists of two related but distinct assignments. Each assignment is executed for a different client, the first assignment for the VUMC medical institute of Amsterdam, the second assignment for the NKI cancer research institute. This section provides a short explanation of both clients.

2.1.1 VUMC

The VU University Medical Centre (VUMC), is the university hospital affiliated with the Vrije Universiteit of Amsterdam in the Netherlands. It was created in 2001 by the merger of the Academic Hospital of the Vrije Universiteit with the medical school of the Vrije Universiteit. It is one of the largest and leading hospitals of The Netherlands. Tertiary care departments include advanced trauma care, pediatric and neonatal intensive care, cardiothoracic surgery, neurosurgery, infectious diseases and other departments. A medical emergency rescue helicopter is also affiliated with the hospital.

The VUMC client is a research institute within the VUMC, who performs CVS tests in order to determine CNVs (if present) in unborn children. The data generated by such tests is of a large magnitude, which hinders manual interpretation of the results of such a test. The client therefore requires an application which can analyse and display this data in such a manner that the results of such a test are clear for the researcher.

The assignment executed for the VUMC therefore focusses on the analysis and visualisation of data obtained from chorionic villus sampling. Chorionic villus sampling (CVS) is a form of prenatal diagnosis to determine chromosomal or genetic disorders in the foetus. It entails getting a sample of the chorionic villus (placental tissue) and analysing its genome in order to

find copy number variations. The analysis produces an aCGH profile, which is processed by the product to determine and display genomic copy number variations in a clear manner. The product should also display the various genes contained in CNV regions and provide the user with disease related annotation data, which supports the user in determining the effect of identified copy number variations.

2.1.2 NKI

The Dutch Cancer Institute (Nederlands Kanker Institituut, NKI) was founded in 1913. The NKI is a scientific research institute and specialised clinic, which focuses on the prevention and curing of cancer. The actual client is a bioinformatics researcher.

The NKI assignment involves the same analysis and visualisation procedures as the VUMC assignment (2.1.1), the main difference being the nature of the data under analysis. The NKI client researches CNV in cancer cells in order to identify mutations in the genome. Data from cancer research is harder to analyse due to large amounts of mutations in the genome and differences in sampled cells (tumorous vs non-tumorous). The client uses both human and mouse genome data in his research.

The NKI assignment has an extra focus on visualising relations between genes in regions occupied by CNVs, which allows the user to identify relations between various CNVs, and often analyses multiple samples at the same time. Due to the focus being on visualisation of relations between genes, the gene annotation should also contain data concerning the functions of genes and any other properties that may relate one gene to another.

2.2 General product functionality

The most basic functionality of the product, as required by both of the clients, concerns the displaying and analysing of aCGH profiles. The product should be able to load the results of a aCGH analysis, plot the corresponding CGH profile and use a specified algorithm to determine significant CNVs in the analysed genome. Other functionality is the displaying of genes in the identified significant CNV regions and the providing of corresponding annotation data. Other required features are displaying sample and the analysis platform metadata to provide information about the loaded sample.

2.3 Client specific functionality

2.3.1 VUMC

The most important functionality specifically required by the VUMC client is displaying disease related annotation data. This annotation data is specifically required to be incorporated directly in to the application interface.

According to the client, the final product must also be very reliable, meaning it must produce consistent and correct results as the product may be involved in important decisions and recommendations concerning actual patients.

2.3.2 NKI

The most important aspect of NKI specific functionality corresponds with visualising relations between genes in regions occupied by CNVs. The product should therefore include annotation data concerning relations between genes and include a visualisation method to display identified relations clearly. The NKI client also requires the product to be able to load and analyse multiple sets of sample data simultaneously.

2.4 Summary of objectives

The above analysis mentions a number of features to be supported by the product, which can be summarised in the following list of required functionality:

- Loading various types of aCGH data files.
- Display of loaded aCGH profiles.
- Analysis of loaded aCGH files to identify significant CNV regions.
- Identification of genes located in identified significant CNV regions.
- Annotation of identified genes with disease related data within the product interface.
- Annotation of identified genes with function related data and other data that may relate genes.
- Identification of related genes through annotation data and subsequent display of the gene relations.

2.5 Contacts

2.5.1 Supervision

Prof. dr. ir. Marcel Reinders (overall supervision): M.J.T.Reinders@tudelft.nl
Ir. Jeroen de Ridder (day-to-day supervison): J.deRidder@tudelft.nl
Jan Bot (database consultant): J.J.Bot@tudelft.nl
Drs. Peter van Nieuwenhuizen (bachelorproject course supervision): P.R.vanNieuwenhuizen@tudelft.nl

2.5.2 Clients

Chris Klijn (NKI): C.Klijn@nki.nl
Henne Holstege (VUMC): H.Holstege@vumc.nl

2.6 Deliverables

The following documents will be created during the development process:

- Plan of Action (this document)
- Requirements Design Document (RDD)

- Architectural Design Document (ADD)
- Technical Design Document (TDD)
- Test Document (TD)
- Quality plan (QP)
- Application documentation

The end result of the process will result in a single application, which will be used by both clients, each to their own goals.

Chapter 3

Action

3.1 Introduction

To execute this project we choose a combination of rapid prototyping and waterfall model. There for we specify the following four phases:

- **Design**
- **Implementation**
- **Test**
- **Completing**

The influence of rapid prototyping in this model is that we will proceed quick to the implementation and testing phase and will revisited the stages multiple times. This includes an incremental way of working so that we can show progress periodically. This also gives us a chance to get feedback of the software to check if we are fulfilling the wishes of the customer.

3.2 Methodology

As mentioned in the previous chapter we use a combination of rapid prototyping waterfall model including an incremental approach where want to deliver prototypes frequently. This gives the opportunity to get fast and regularly feedback. There is also a slight deviation of the waterfall model because we choose for test driven development that leads to the fact that the phases Implementation and Test will be put together in one phase.

3.3 Techniques

At the moment there is a preference to use Python as programming language because of its power, portability and easy integration. It is also the default programming language of the bioinformatics sector. There are already some useful applications written in R, Java, Perl and Python. So those techniques are considered to be used with its given development environment.

3.4 Proceedings

3.4.1 Design

The design phase is about finding out the requirements, and making design decisions and write those down in the following documents:

- Requirements Design Document(RDD)
- Architectural Design Document (ADD)
- Technical Design Document (TDD)
- Test Document (TD)
- Quality Plan(QP)

3.4.2 Implementation & Test

The implementation component will be about writing the actual code of the application. The test component will be about implementing the TD and executing the tests, of course when test fail the application will be fixed.

3.4.3 Completing

In the Completing phase the product will be finished, the documentation should be updated and the product should be delivered to the customer. Also the project should be closed with a project review document and a fitting ending including with an end presentation.

3.5 Planning

The planning is as followed:

- **Design**: weeks 16 - 18
- **Implementation & Test**: weeks 19 - 24
- **Completing**: week 25
- **Extra**: weeks 26 - 27

For a more detailed planning see appendix A.1.

Chapter 4

Project Design

4.1 Introduction

The chapter Project Design starts with information about the project members. Since all members are connected to Delft University of Technology, only names will be listed. The information section contains a list of organisational items. Finally a word about the used facilities.

4.2 Project Members

Team members :
Nanne Aben, Stefan Dentro, Rien Korstanje, Chris Melman, Julian de Ruiter

Roles:

- Meeting chairman: Nanne
- Meeting secretary: Rien
- Blog: Stefan

More roles will be assigned during the project.

4.3 Information

- We've decided to work at least four days a week at the University, starting at 9:00 and finishing after 17:00. The fifth day is left up to the team members individually, though missed time must be made up in the individuals spare time.
- Every day starts with an informal meeting about the workday.
- Every week a formal meeting with Mr. De Ridder.
- Rooms at either tenth or eleventh floor at EECMS will be used as office, depending on availability.
- Subversion will be used as versioning system.
- An overall blog will be kept about team progress.

4.4 Facilities

Delft University of Technology has many computers available for students. The planning is to either use those or make use of private computers from the team members. The tenth and eleventh floor contains enough room for the team members to set up shop. There might be a need for storage at some point. Storage will be dealt with as soon as it is needed.

Chapter 5

Quality Assurance

5.1 Introduction

This chapter discusses the proposed measures and the clients requirements for the quality of the product. The quality constraints provided by the client are used as the guidelines for the requirements. These are made explicitly in the requirements for each artefact. As such all measures to ensure quality are described for each specific item. This chapter provides a brief summary of these measures.

5.2 Quality

Quality Assurance is a systematic way to monitor and evaluate the various aspects of a project such as the documentation, code as well as the functional and non functional requirements. The aim is to create a product "fit for purpose" and to do it "right first time round". It is important to note that quality is a property determined by the intended client and users.

5.3 Documentation

The documentation consists of all written artefacts, such as this Plan of Action. Each written artefact should adhere at least to the following requirements:

- Be originally written in Latex or plain text.
- The language used should be English.
- Adhere to any specific requirements for the artefact.

To ensure these requirements are met all artefacts will be reviewed before they are handed off.

5.4 Code

The code consists of all programming code written for this project. Be it Python, Java, R, or any other language. All code should adhere to atleast these quality requirements:

- Pass the requirements of test driven development.

- Be properly documented
- Adhere to any specific requirements for the artefact.

To ensure these requirements are met, automated testing and checking will be used.

5.5 Version control

Subversion will be used to manage the body of artefacts and intermediary products generated by the project. Trac will provide any additional tools, such as ticket support and global planning.

- All commits have a meaningful comment.
- References to tickets are prefixed with #.
- All committed code is functional.

To ensure these requirements are met team members will actively address these issues when noticed.

5.6 Evaluation

To ensure that during the process the artefacts continuously improved a brief evaluation is done with each new increment of the deliverable.

References

[1] F. Cappuzzo, F.R. Hirsch, E. Rossi, S. Bartolini, G.L. Ceresoli, L. Bemis, J. Haney, S. Witta, K. Danenberg, I. Domenichini, et al. Epidermal growth factor receptor gene and protein and gefitinib sensitivity in non-small-cell lung cancer. *JNCI Journal of the National Cancer Institute*, 97(9):643, 2005.

[2] D.F. Conrad, D. Pinto, R. Redon, L. Feuk, O. Gokcumen, Y. Zhang, J. Aerts, T.D. Andrews, C. Barnes, P. Campbell, et al. Origins and functional impact of copy number variation in the human genome. *Nature*, 2009.

[3] E. Gonzalez, H. Kulkarni, H. Bolivar, A. Mangano, R. Sanchez, G. Catano, R.J. Nibbs, B.I. Freedman, M.P. Quinones, M.J. Bamshad, et al. The influence of CCL3L1 gene-containing segmental duplications on HIV-1/AIDS susceptibility. *Science*, 307(5714):1434, 2005.

[4] HGP. Human genome project information, June 2010. `http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml`.

[5] Sanger Institute. The copy number variation (cnv) project, 2010. URL: http://www.sanger.ac.uk/humgen/cnv/.

[6] S.J.L. Knight, R. Regan, A. Nicod, S.W. Horsley, L. Kearney, T. Homfray, R.M. Winter, P. Bolton, and J. Flint. Subtle chromosomal rearrangements in children with unexplained mental retardation. *The Lancet*, 354(9191):1676–1681, 1999.

[7] J.A. Lee and J.R. Lupski. Genomic rearrangements and gene copy-number alterations as a cause of nervous system disorders. *Neuron*, 52(1):103–121, 2006.

[8] Baylor College of Medicine. Copy number variation may stem from replication misstep, January 2008. `http://www.sciencedaily.com/releases/2007/12/071227183904.htm`.

[9] J. Sebat, B. Lakshmi, D. Malhotra, J. Troge, C. Lese-Martin, T. Walsh, B. Yamrom, S. Yoon, A. Krasnitz, J. Kendall, et al. Strong association of de novo copy number mutations with autism. *Science*, 316(5823):445, 2007.

[10] D. St Clair. Copy number variation and schizophrenia. *Schizophrenia bulletin*, 2008.

Appendix A

Planning

The project consists of three phases. Firstly an orientation phase in which we're going to acquire domain knowledge and design the software. Since we're in a group of five people, we've decided to start working on a first prototype. A later document will describe what each prototype will be able to do. The idea is to use the rapid prototyping technique.

During the second phase most of the bulk programming will be done. While building a prototype every week the base of the software package will be gradually improved and expanded. The last phase is all about finishing the project. Wrapping up the code for the presentation and finalising the report. There are two extra weeks scheduled, in case there is a need for more time.

	Phase	**Deliverables**
1	Orientation	Plan of Action (PoA) Requirements Design Document (RDD) Software Design Document (SDD) Test Document (TD) Quality Plan (QP)
2	Implementation & Testing	Every week a working prototype Documentation Tests
3	Completing	Demo Report

Detailed planning:

Phase	Week	Activities	Deliverables
1	16	Getting familiar with domain Start design	@April 23: PoA, bèta RDD
	17	Start prototyping Start writing documents Finalise bèta design	@April 29: Prototype A
	18	Finalise design phase	@May 7: RDD, SDD, TD & QP
2	19		@May 14: Prototype B
	20		@May 21: Prototype C
	21		@May 28: Prototype D
	22		@June 4: Prototype E
	23		@June 11: Prototype F
	24		@June 18: Prototype G
3	25		@??: End presentation & working demo
Extra	26		
	27		

Important dates:

April 19 2010	Start project
April 30 2010	Koninginnedag
May 5 2010	Liberation day
May 13 2010	Ascension
May 24 2010	Second Pentecost day

Appendix E

Software Requirements Specifications

Graphical application for the statistical analysis of copy number variations in aCGH data

Thesis BSc Project IN3405

Software Requirements Specification

version 1.1

Julian de Ruiter (1387367)
Chris Melman (1358359)
Rien Korstanje (1270389)
Stefan Dentro (1313320)
Nanne Aben (1369342)

DELFT UNIVERSITY OF TECHNOLOGY
EEMCS FACULTY
JUNE 28, 2010

Contents

Chapter 1

Introduction

1.1 Purpose

This document describes the specification of software requirements for a graphical, statistical analysis tool of copy number variations in aCGH data. The document is intended to serve as strict specification for the software developers implementing the system and provide insight into the requirements analysis for computer engineers interested in the project.

1.2 Scope

The document describes the requirements specifications imposed on a graphical, statistical analysis application for aCGH data, codenamed Angita. The tool is being developed for the NKI and the VUMC.

The application is able to import aCGH data from a local file and identify significant aberrations in the genomic data using various algorithms. It provides an interface in which a loaded aCGH profile can be viewed in a linear fashion. The interface also displays the various genes located in identified aberrant regions. Genes are annotated by various databases to provide more information about the gene in question. The application also provides a view which displays relations between the genes in the identified aberrant regions of the genome, allowing the type of relations to be shown can be chosen.

The application aims to allow quick analysis of the provided aCGH data by displaying a clear view of the aCGH profile, providing various algorithms to identify aberrant regions of the genome and displaying the identified regions in a clear manner. The displayed relations between genes in the identified regions provide the user with a significant amount of extra data, which can help identify non-obvious relations between the identified regions and corresponding genes.

1.3 Definitions, acronyms and abbreviations

Aberration Modification of the normal chromosome complement due to deletion, duplication, or rearrangement of genetic material.

Angita Angita is the early Roman goddess of healing, magic and witchcraft. [1]

Array comparative genomic hybridisation (aCGH) Array comparative genomic hybridisation is a technique to detect genomic copy number variations at a higher resolution level than chromosome-based comparative genomic hybridisation (CGH).

Copy number variant (CNV) A copy number variant is a segment of DNA in which non-pathogenic copy-number differences have been found by comparison of two or more genomes. The segment may range from one kilo-base to several mega-bases in size. Humans (being diploid) ordinarily have two copies of each autosomal region, one per chromosome. This may vary for particular genetic regions due to deletion or duplication.

Genome annotation The process of identifying the locations of genes and all of the coding regions in a genome and determining what those genes do. An annotation (irrespective of the context) is a note added by way of explanation or commentary. Once a genome is sequenced, it needs to be annotated to make sense of it.

VUMC The VU University Medical Centre is an academic medical centre, concerned with patient care, higher education and medical research. [3]

NKI The Netherlands Cancer Institute researches the cause and effects of cancer. The institutes three major areas of research are fundamental, clinical and translational cancer research. [2]

Ibidas Ibidas is aimed to be an integration platform for various sorts of biological data. Its purpose is providing a very minimal but flexible database scheme. It is possible to store any kind of biological data while still keeping the learning curve for using the system low.

1.4 References

See references section at the end of this document.

1.5 Overview

This document contains an overall description of the requirements of the proposed software package. This includes interfaces for users and external software packages. It also defines a number of product functions and characteristics of end users (chapter 2). Chapter 3 contains a detailed list of functional and non-functional requirements, including performance and design constraints. Finally Appendix A contains a number of usage scenario's that the end product must match.

Chapter 2

Overall description

2.1 Product perspective

The software product is an independent system but does rely on a number of external entities to perform its functions. The following external software/data sources are used by the system:

- KCSmart - identification of genomic aberrations.
- Circos - circular visualisation of gene relations.
- Ibidas - data source gene annotations, gene relations and qualifier types.
- Ensembl - data source of gene annotations.
- STRING - data source of gene relations.
- DECIPHER - data source with anonymous patient phenotype data.
- OMIM - a compendium of human genes and genetic phenotypes.
- DGV - a catalog of structural variation in the human genome.
- CGC - data source of known relations between cancer and gene variation.

2.1.1 System interfaces

No system interfaces are provided by the product at this time.

2.1.2 User interfaces

The user interface is to be kept as clean and simple as possible (without sacrificing functionality) in order to provide a concise interface for the user to deal with. The interface should adhere to a style similar to software that is currently used by the clients to keep the learning curve of the application relatively low. A good example of such an interface is the browsing interface of the Nexus software suite, which will be used as a guideline for designing the browsing interface of plotted data.

The primary task of the user interface is presenting a plot of the loaded aCGH sample data. This plot displays the various data points of the loaded sample data and allows the user to manipulate

the plot in order to view the presented data in a more convenient manner. Manipulations that the plot supports in order to facilitate these actions are: zoom functionality, side-scroll functionality and jumping between significant regions. The plot view should be accompanied by a view displaying the metadata of the loaded sample data.

The plot highlights regions of the plot that are identified as significant regions by a chosen algorithm. Gains and losses are distinguished by a difference in colour. The interface also allows a user easily navigate the genome in order to view the various found significant regions.

When focussed on a significant region of the plot, the data in the plot is accompanied by a track view which displays properties of the region. The track view must at least display the genes located in the shown region and the range of the gene in question. Selecting a gene results in the opening of a new tab displaying data concerning the gene.

The gene data view displays data concerning a specific gene as provided by the Ibidas back-end system. The view displays all acquired data in its standard configuration, but also provides a view or dialog that allows the user to show/hide certain fields.

2.1.3 Software interfaces

The product relies on the software products mentioned in this section for full functionality. KCSmart, Circos and Ibidas are directly required by the product, whilst the other software products are required by the Ibidas system in order to provide the application with sufficient annotation data.

KC-smart

Version: 1.6.0 (22 October 2008)
Source: http://www.bioconductor.org/packages/2.3/bioc/html/KCsmart.html

KC-smart uses non-discretised aCGH data to identify regions that are significantly aberrant across an entire tumour set. It is based on kernel regression and accounts for the strength of a probe's signal, its local genomic environment and the signal distribution across multiple tumours.

Circos

Version: 0.52 (24 November 2009)
Source: http://mkweb.bcgsc.ca/circos/?download

Circos is a software package for visualising data and information. It visualises data in a circular layout, which makes Circos ideal for exploring relationships between objects or positions. Circos is used to visualise relations between genes, which have been identified by the product.

Ibidas

Version: 3α
Source: https://wiki.nbic.nl/index.php/Ibidas

The amount of data and the complexity of the relations between different data sets require a specialised system for efficiently storing and querying of the data. Ordinary structured query database systems are based on the assumption that data is representable as a 2D table. Combining data sources is done via Structured Query Language (SQL) using multiple tables. Biological data however requires representation of data in a larger number of dimension. Ibidas is purpose built with that idea in mind. Ibidas provides functions that are more abstract than SQL queries while maintaining the same level of flexibility and power. Ibidas is therefore the right choice for storing and accessing the various sources of biological data.

Ensembl

Version: Release 57 (3 March 2010)
Source: http://www.ensembl.org/

Ensembl is a joint scientific project between the European Bioinformatics Institute and the Wellcome Trust Sanger Institute, which was launched in 1999 in response to the imminent completion of the Human Genome Project with scientists in the United States. Its aim is to provide a centralised resource for geneticists, molecular biologists and other researchers studying the genomes of our own species and other vertebrates. Ensembl is used as a source for genomic information.

STRING

Version: 8.2
Source: http://string-db.org/

STRING (Search Tool for the Retrieval of Interacting Genes/Proteins) is a database and web resource of known and predicted protein-protein interactions. The STRING database contains information from numerous sources, including experimental data, computational prediction methods and public text collections. The database is mainly used to derive relations between genes and display data about the function of genes.

DECIPHER

Version: 4.4
Source: https://decipher.sanger.ac.uk/

The DECIPHER database of submicroscopic chromosomal imbalance collects clinical information about chromosomal micro-deletions/duplications/insertions, translocations and inversions and displays this information on the human genome map. The data in this database is used relate mutations in chromosome regions to diseases.

OMIM

Date: 4 May 2010
Source: http://www.ncbi.nlm.nih.gov/omim

OMIM is a comprehensive, authoritative, and timely compendium of human genes and genetic phenotypes. The full-text, referenced overviews in OMIM contain information on all known mendelian disorders and over 12,000 genes. OMIM focuses on the relationship between phenotype and genotype. The data in this database is used relate mutations in genes to diseases

DGV

Version: CGCh 37 (February 2009)
Source: http://projects.tcag.ca/variation/

The objective of the Database of Genomic Variants is to provide a comprehensive summary of structural variation in the human genome. The Database of Genomic Variants provides a useful catalog of control data for studies aiming to correlate genomic variation with phenotypic data. This database is used to identify variations in the processed aCGH data as CNVs.

CGC

Date: 30 March 2010
Source: http://www.sanger.ac.uk/genetics/CGP/Census/

The Cancer Gene Census is an ongoing effort to catalogue those genes for which mutations have been causally implicated in cancer. This database is used to relate CNVs to cancerous effects.

2.1.4 Communication interfaces

The product relies on a connection to Ibidas for accessing various data sources such as String and Ensembl. Ibidas functionality is provided by a server on the local network. The server provides a XML-RPC communication interface for interaction with the product. The product must ensure basic functionality without a working connection to Ibidas and will only lose functionality that is related to Ibidas, specifically: gene annotation data, gene relation identification and verification of CNVs.

2.1.5 Hardware interfaces

The product does not rely on any specific hardware interfaces at this time.

2.2 Product functions

The product reads a aCGH file which adheres to a supported file format (limited by the parsers available to the system) and displays the results contained in the file in a two-dimensional plot. The plot provides the user with a manner to view the data in different scales by allowing the user to zoom-in and zoom-out on the plot as well as scroll horizontally. The product highlights certain regions of the plot (corresponding to regions of the genome) which have been identified as significant by an external algorithm and as CNVs according to data stored by Ibidas. It also displays genes located on the significant regions of the genomes in a track-like manner and provides detailed information about these genes through the data stored by Ibidas, most importantly gene

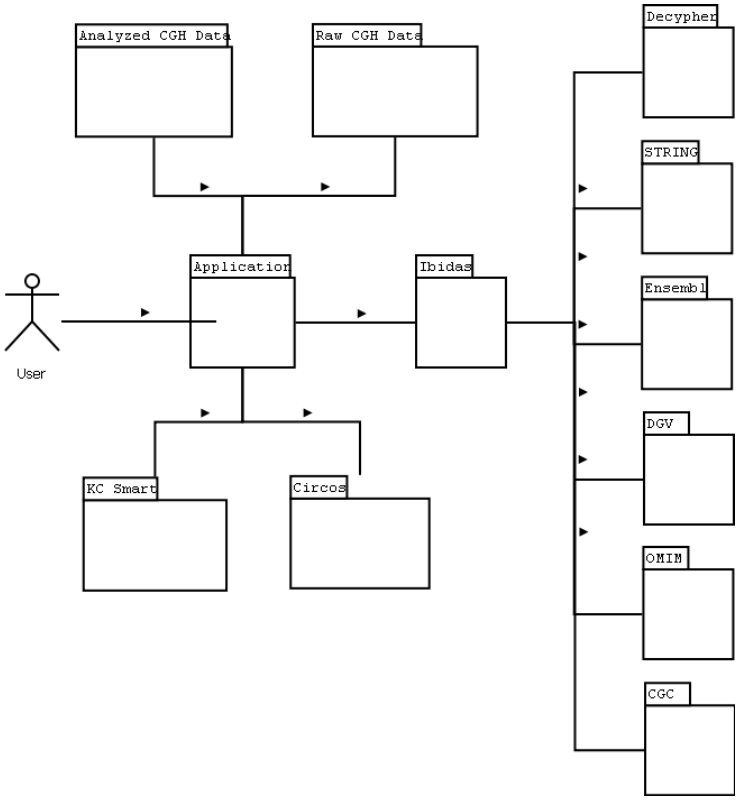


Figure 2.1: Software interface overview

function and known relations with diseases.

The product also highlights relations between genes and chromosome regions in the significant regions as identified by the data stored in Ibidas, allowing the user to reason about occurrences that often occur simultaneously. These relations are displayed in a circular plot, the types of relations included in the plot can be (de)activated by the user.

The product is also able to process a set of aCGH samples and display aggregated results in the manner as described above. The user is also able to view plots of individual results within a set.

The product allows the export of an image representation of the plot view for use in external applications. It also provides the user with a PDF export function which produces a PDF report containing important data and images (what is exactly is still to be determined).

Summary of the main functions:

- Loading and analysing of aCGH data.
- Plotting of aCGH data, highlighting significant regions as indicated by an (external) algorithm.
- Acquiring data on significant regions from gene annotation databases.

- Visualisation of relations in a circular diagram using Circos.
- Exporting of aCGH-data plots, metadata, annotations, and circular visualisations.

2.3 User characteristics

The product targets two specific audiences, the first being researchers of the VUMC, the second being cancer researchers of the NKI.

VUMC researcher This client performs work within a research institute of VUMC, concerned with performing CVS tests in order to research the effects of various forms of CNVs. The client is a geneticist, his main goal (with concern to our product) being the analysis of the results of a CVS test in order to determine whether the patient in question is healthy from a genomic point of view. The client aims to identify regions of aberrant DNA in the genome of a patient, which could underlie a phenotypic quality of that patient, such as genomic syndromes leading to mental retardation or physical malformation. In the future, the client aims to use this technique for diagnosis of syndromes in unborn children. Due to the weight of this advice, the product should be reliable (have the reproducible results for a set of data), should provide verified data (not only experimental data) and should present results in a clear and non-ambiguous manner. The researcher is not considered to be very adept with software systems, but has experience with bioinformatics systems such as Nexus.

NKI researcher This client is a bioinformatics researcher, whose primary task is researching the development of cancer for the NKI. The data provided by this client generally consists of multiple anonymised sets of aCGH data, which must be analysed statistically in order to identify significant regions within the data sets. This client is primarily concerned with analysing these large datasets efficiently, in order to identify recurring aberrant regions in sets of CGH data and any relations between these regions or genes within these regions. The client has a background in bioinformatics and is therefore considered to have a relatively large amount of experience when working with software systems.

2.4 Constraints

2.4.1 Hardware limitations

The product must be able to run on a reasonable desktop computer with generic hardware without resulting in unreasonable waits for the user. A reasonable system is defined as having an Intel Core2Duo processor (or equivalent) and at least 2 GB of RAM. An unreasonable wait is defined as a wait of more than 10 seconds. Exceptions to this rule are made when the product is importing a new data file or running an algorithm to determine significant regions. In the case of opening a new data file a delay of up to a minute is acceptable. The delay of running an algorithm depends on the nature of the chosen algorithm and is not under direct control of the product. No limit is placed on the runtime of an algorithm, though the system should account for long delays if these are expected and remain responsive during the execution of the algorithm.

2.4.2 Reliability requirements

The analysis and corresponding visualisation must be reliable: it must produce correct results and these results must be reproducible with the same data. Errors in this process could result

in interpretation of the incorrect data by the user, which may lead to incorrect conclusions.

2.4.3 Safety and security considerations

The data processed by the system and the corresponding must be kept confidential, due to the fact that data provided by the user can be traceable to a specific patient and is therefore of a confidential nature. The product must therefore not publish any data on locations other than the hardware on which it is run. The hardware systems on which the application is run are trusted to be secure in the sense that the systems are protected from unauthorised access due to the fact that they may contain sensitive data. As the hardware is considered relatively secure, the product contains no extra security or encryption measures to protect its data.

2.5 Assumptions and dependencies

The product is designed to be cross platform and therefore functions on the most common operating systems (Windows, Linux and Mac OS X). In order to do so, it is created in the Python programming language, which is available for each of these platforms, and utilises the cross-platform graphical interface toolkit Qt. Various external software, such as Circos, is used for certain product functionality and is therefore also required for full product functionality. In summary, the product relies on the following products (and versions) to be installed on a host system in order to function properly:

- Python 2.6
- Qt4 and PyQt
- Circos

If one or more of these dependencies is not met correct functioning of the product is not guaranteed. Where possible, the product detects missing dependencies and fails gracefully, warning the user of the missing dependencies and possibly providing a solution to the problem.

2.6 Data constraints

The DECIPHER data contains patient data. Because the patients privacy must be respected, DECIPHER strictly licenses usage of the data. The product can therefore not be delivered with the DECIPHER data included. Instead, the client must obtain his own license from DECIPHER. Other features of the product must still work without DECIPHER data being present.

Chapter 3

Specific requirements

3.1 External interface requirements

3.1.1 User interfaces

The users of the product have a deep understanding of life sciences but different levels of computing experience. It is therefore preferable that the application has a understandable graphical user interface, that is easy to use without too many complex options, though it may provide an option for advanced input for the more experienced user.

3.1.2 Hardware interfaces

The product must be able to run on a reasonable desktop computer with generic hardware without resulting in unreasonable waits for the user. A reasonable system is defined as having an Intel Core2 processor (or equivalent) and at least 2 GB of RAM.

3.1.3 Communications interfaces

The product relies on a connection to Ibidas for accessing various data sources such as STRING and Ensembl. The product must ensure basic functionality without a working connection, but will exhibit reduced behaviour due to the unreachable data sources.

3.2 Functional requirements

The product provides a graphical user interface (GUI) for the visualisation of aCGH data. In this system, the user is able to load an aCGH-file and perform an analysis on this data. The system will then plot the aCGH data, highlighting significantly amplified or deleted genes on the genome. Several databases can be queried to get annotation data for aberrant regions. The user is then able to scroll through the aCGH-data, search on subject metadata, visualise relations between genes and render documentation, as described below.
In detail, the system will adhere to the following functional requirements:

3.2.1 Loading and analysing of aCGH data

1. Supports loading of one or more aCGH data file from a local resource. The aCGH file must conform to one of the supported file formats. The application must at least support data files produced by the Agilent Technologies platform used by the VUMC client and the Affymetrix platform used by the NKI client.
2. Supports selection of samples that the user wants to plot from the selected aCGH files.
3. Supports transforming the green and red channel data to log2 values.
4. Supports switching green and red channels in case the labelling was erroneously done the wrong way around.
5. Supports normalisation of the log2 data.
6. Supports analysing the CGH data for significant aberrant regions, by either
   (a) finding groups of consecutive data points above a certain threshold. Both the number of consecutive data points and the threshold are specified by the user.
   (b) loading the result of an external analysis algorithm. For now, the supported algorithms will be limited to KC-smart 1.6.0.
   (c) running the KC-smart 1.6.0 implementation from our system, using the loaded aCGH-data. The user must be able to specify the parameters with which the algorithm is run.

3.2.2 Plotting of aCGH data

1. Supports plotting one or more samples from an aCGH data-set in a single plot. In case multiple samples are plotted, they are vertically aligned on the screen.
2. Supports interactively browsing through the plot at least including the following functionality:
   (a) zooming in on the plot using the mouse scroll-wheel.
   (b) scrolling through the plot using click-and-drag with the left mouse button.
   (c) showing more detail when zoomed in. An example of such level of detail is showing gene-names, or even introns and exons, when zoomed in but not when zoomed out.
   (d) selecting a chromosome in order to automatically zoom in on that region.
3. Supports the highlighting of regions that are identified as significantly aberrant regions in the linear aCGH plot
4. Supports plotting of a running average over the loaded aCGH data.
5. Supports plotting Cy5 data of one array against Cy3 data of another, so different subjects can be hybridised against the same control without repeating the experiment.

3.2.3 Acquiring data on the significant aberrant regions

1. Supports filtering of indicated gains and losses against structural variations in the human genome using data from DGV.
2. Supports mapping of genes onto identified regions, using data from Ensembl. Because only Genome build 37 is available through the Ensembl API, our prototype will be limited to build 37.
3. Supports mapping of introns and exons onto significant genes using data from Ensembl.
4. Supports acquiring data about the significant genes, using
   (a) Decipher to get common diseases associated with a significant aberrant region in a particular chromosomal location, as well as the certainty of this being the case.
   (b) Omim to get common diseases associated with a significant aberrant region in particular gene, as well as the certainty of this being the case.
   (c) Cancer Gene Census to see if a significant aberrant region in a particular gene is associated with cancer.
   (d) STRING to get functional and physical protein-protein interactions associated with a particular gene, as well as the certainty of this being the case.
5. Supports selection of a single gene in order to display annotation data, such as common diseases or protein-protein interactions associated with a particular gene.
6. Supports limiting of the databases that will be used to acquire annotation data from. For example, if the user doesn't want to use STRING, he can uncheck this in the options menu.

3.2.4 Relational visualisation

1. Supports listing identified genes as well as relations between these genes (the relations being diseases associated with a gene and functional and physical protein-protein relations).
2. Supports user selection of relations, allowing visualisation of only the selected relations.
3. Supports user selection of significant aberrant regions, allowing visualisation of only the genes in these regions.
4. Supports user selection of genes, allowing visualisation of only the selected genes.
5. Supports visualising the selected genes and the selected relations in a Circos-plot (a circular relational visualisation tool).

3.2.5 Rendering documentation

1. Supports the export of single plots as an image.
2. Supports the export of annotations for a selected gene to a tab-delimited file.
3. Supports the export of the content (plots or annotations) in all opened tabs.

3.3 Performance requirements

The product should respond to all data-intensive requests within ten seconds, disregarding primary analysis of supplied data. Primary analysis of the aCGH data itself may take longer, depending on the nature of the algorithm used to analyse the data. Therefore, the first analysis of a data set may take an arbitrary amount of time.

3.4 Design constraints

Annotation data and other gene related data is acquired from external data sources by the product. As this data is not under the control of the user, the product should anticipate changing data formats when interfacing with these data sources. The system will therefore work with text-dumps of the databases to prevent the system failing to function when a database changes its API. The annotation data may therefore not be the most recent data available.

3.5 Software system attributes

The product is designed for Python 2.6. Any software system attempting to run the product must therefore contain a fully functional installation of the Python 2.6 interpreter or higher.

The product also requires a number of Python libraries, such as NumPy, MatPlotLib and PyQT. These packages must be installed along with our product.

3.6 Other requirements

3.6.1 Flexibility

The primary product is a working prototype of the described product and should be highly flexible in nature in order to allow customisation of the prototype to fit any unforeseen needs. Due to the fact that this product is being created for multiple clients with different needs, the software should also allow some degree of customisation.

3.6.2 Error handling and extreme conditions

The product must be able to handle errors that can occur in a user friendly manner, thus displaying a graphical notification of any errors that the product cannot resolve itself. The most notable situation is the lack of an internet connection, which results in the failure of acquiring annotation data. The product must not lose its basic functionality without an internet connection and must only degrade related functions.

3.6.3 Quality issues

In order to provide a high level of quality in the product prototype, the development process of the product adheres to the practices defined in the quality plan.

References

[1] Little, Brown, and company. *A Dictionary of Greek and Roman biography and mythology*. Smith, William, Sir, 1867. URL: http://quod.lib.umich.edu/cgi/t/text/text-idx?c=moa;idno=ACL3129.0001.001.

[2] NKI. About the netherlands cancer institute, 2010. URL: http://www.nki.nl/Research/About+the+Netherlands+Cancer+Institute/.

[3] VUMC. About the vumc, 2010. URL: http://identity20.com/media/OSCON2005/.

Appendix A

Scenarios

Upload research data in application

Precondition: User has data available that is in a recognised format (e.g. CGH, KC-SMART, etc.).

Actor	Action	System response
User	Selects a menu option "import" and points to a data file.	Data is read from the file and stored internally. Views and plots are created that visualise the data. Genes are added below the corresponding area of the plots.

Analysing data relations

Precondition: Scenario *Upload research data in application* is completed.

Actor	Action	System response
User	User selects genes that are interesting and presses preview.	
System		A Circos plot is generated in the preview-section. The Circos plot shows relations between the selected genes .
User	User selects relations that are interesting and presses preview.	
System		A new Circos plot is generated in the preview-section.
User	The user is satisfied with the preview that is rendered for the current settings. He now presses generate.	
System		A new window pops up with a large Circos-plot, so the user can study the plot in detail.

Note: It should be possible to automatically update the preview when the settings change if the plot has an extremely low rendering time.

Retrieving annotations for a particular gene

Precondition: Scenario *Upload research data in application* is completed.

Actor	Action	System response
User	Selects a particular gene.	Relevant information is fetched from either external or internal data collections about the particular gene and is displayed in a view.

Filter on meta-data

Precondition: Scenario *Upload research data in application* is completed.

Actor	Action	System response
User	Fills in values for age, sex, etc.	A list of plots with data for the subjects that fall within the criteria.

Export illustration

Precondition: Scenario *Upload research data in application* is completed.

Actor	Action	System response
User	Selects a menu option “Export illustration...”.	
System		A list of available illustrations is displayed (CGH-plot, Circos-plot).
User	An illustration is selected from the list.	
System		Requests a location to save the image to and requests the save-as-type.
User	Enters a location and a save-as-type.	
System		The illustration is saved as an image.

Render report

Precondition: Scenario *Upload research data in application* is completed.

Actor	Action	System response
User	Selects a menu option “Render report...”.	
System		A list of available diagrams and information blocks is displayed.
User	A number of blocks from the list are selected.	
System		The blocks are combined in a document that is presented to the user.

Appendix B

MoSCoW

MoSCoW analysis is a prioritisation technique that is used to reach a common understanding among the various stakeholders of the project. It denotes which parts are absolutely necessary (Must have); which parts are important, but can be delayed until a further release (Should have); which parts are nice to have (Could have); and which parts are not appropriate at this time (Won't have).

This chapter describes the MoSCoW analysis. It divides the functional requirements as given in chapter 3.2 into Must have, Should have, Could have and Won't have.

B.1 Must have

B.1.1 Loading and analysing of aCGH data

1. Supports loading of one or more aCGH data file from a local resource. The aCGH data will have to be an Agilent Technologies human or mouse genome CNV microarray that is formatted as a single-table text file. The file must contain a CHROMOSOMAL_LOCATION column.
2. Supports selection of samples that the user wants to plot from the selected aCGH files.
3. Supports transforming the green and red channel data to log2 values.
4. Supports normalisation of the log2 data.
5. Supports analysing the CGH data for significant aberrant regions, by either
   (a) finding groups of consecutive data points above a certain threshold. Both the number of consecutive data points and the threshold will be user specified.
   (b) loading the result of an external analysis algorithm. For now, the supported algorithms will be limited to the R implementation of KC-smart 1.6.0.

B.1.2 Plotting of aCGH data

1. Supports plotting one sample from an aCGH data-set in a plot.
2. Supports interactively browsing through the plot at least including the following functionality:

(a) zooming in on the plot using the mouse scroll-wheel.

(b) scrolling through the plot using click-and-drag with the left mouse button.

(c) showing more detail when zoomed in. An example of such level of detail is showing gene-names, or even introns and exons, when zoomed in, but not when zoomed out.

(d) selecting a chromosome in order to automatically zoom in on that region.

3. Supports the highlighting of regions that are identified aberrant regions in the linear aCGH plot.

B.1.3 Acquiring data on the significant aberrant regions

1. Supports mapping of genes onto identified regions, using data from Ensembl. Because only Genome build 37 is available through the Ensembl API, our prototype will be limited to build 37.

2. Supports acquiring data about the significant genes, using

   (a) DECIPHER to get common diseases associated with a significant aberrant region in a particular chromosomal location, as well as the certainty of this being the case.

   (b) Omim to get common diseases associated with a significant aberrant region in particular gene, as well as the certainty of this being the case.

   (c) Cancer Gene Census to see if a significant aberrant region in a particular gene is associated with cancer.

   (d) STRING to get functional and physical protein-protein interactions associated with a particular gene, as well as the certainty of this being the case.

3. Supports selecting of a single gene in order to display annotation data, such as common diseases or protein-protein interactions associated with a particular gene.

B.2 Should have

B.2.1 Plotting of aCGH data

1. Supports plotting more samples from an aCGH data-set in a single plot. These samples will be vertically aligned on the screen.

2. Supports plotting of a running average over the loaded aCGH data.

B.2.2 Acquiring data on the significant aberrant regions

1. Supports filtering of indicated gains and losses against structural variations in the human genome, using data from DGV.

2. Supports mapping of introns and exons onto significant genes, using data from Ensembl.

3. Supports the limiting of the databases that will be used to acquire annotation data from. For example, if the user doesn't want to use STRING, he can uncheck this in the options menu.

B.2.3 Rendering documentation

1. Supports the export of single plots as an image.
2. Supports the export of annotations for a selected gene to a tab-delimited file.
3. Supports the export of the content (plots or annotations) in all opened tabs.

B.3 Could have

B.3.1 Loading and analysing of aCGH data

1. Supports running the KC-smart 1.6.0 implementation from our system, using the loaded aCGH-data. The user must be able to specify the parameters with which the algorithm is run.
2. Supports switching green and red channels, in case the labelling was erroneously done the wrong way around.

B.3.2 Plotting aCGH data

1. Supports plotting Cy5 data from one array against Cy3 data from another, so different subjects can be hybridised against the same control without repeating the experiment.

B.3.3 Relational visualisation

1. Supports listing identified genes as well as relations between these genes (the relations being diseases associated with a gene and functional and physical protein-protein relations).
2. Supports user selection of relations, allowing visualisation of only the selected relations.
3. Supports user selection of significant aberrant regions, allowing visualisation of only the genes in these regions.
4. Supports user selection of genes, allowing visualisation of only the selected genes.
5. Supports visualising the selected genes and the selected relations in a Circos-plot (a circular relational visualisation tool).

B.4 Won't have

1. Support of linking (an abstract of) papers related to aCGH specifics.

Appendix F

Software Quality Assurance Plan

Graphical application for the statistical analysis of copy number variations in aCGH data

Thesis BSc Project IN3405

Software Quality Assurance Plan

version 1.3

Julian de Ruiter (1387367)
Chris Melman (1358359)
Rien Korstanje (1270389)
Stefan Dentro (1313320)
Nanne Aben (1369342)

DELFT UNIVERSITY OF TECHNOLOGY
EEMCS FACULTY
JUNE 28, 2010

Contents

Chapter 1

Introduction

1.1 Purpose

This document describes which activities are performed in order to ensure a certain level of quality in the software development process. The purpose of this document is to provide uniform, minimum acceptable requirements for preparation and content of documents and code.

1.2 Scope

The document describes the quality measures taken to successfully build a graphical, statistical analysis application of copy number variations in aCGH data, codenamed Angita. The tool is being developed for the NKI and the VUMC.

The application is able to import aCGH data from a local file and identify significant aberrations in the genomic data using various algorithms. It provides an interface in which a loaded aCGH profile can be viewed in a linear fashion. The interface also displays the various genes located in identified aberrant regions. Genes are annotated by various databases to provide more information about the gene in question. The application also provides a view which displays relations between the genes in the identified aberrant regions of the genome, allowing the type of relations to be shown can be chosen.

The application aims to allow quick analysis of the provided aCGH data by displaying a clear view of the aCGH profile, providing various algorithms to identify aberrant regions of the genome and displaying the identified regions in a clear manner. The displayed relations between genes in the identified regions provide the user with a significant amount of extra data, which can help identify non-obvious relations between the identified regions and corresponding genes.

1.3 Definitions, acronyms and abbreviations

Aberration Modification of the normal chromosome complement due to deletion, duplication, or rearrangement of genetic material.

Angita Angita is the early Roman goddess of healing, magic and witchcraft. [1]

Array comparative genomic hybridisation (aCGH) Array comparative genomic hybridisation is a technique to detect genomic copy number variations at a higher resolution level than chromosome-based comparative genomic hybridisation (CGH).

Copy number variant (CNV) A copy number variant is a segment of DNA in which copy-number differences have been found by comparison of two or more genomes. The segment may range from one kilo-base to several mega-bases in size. Humans (being diploid) ordinarily have two copies of each autosomal region, one per chromosome. This may vary for particular genetic regions due to deletion or duplication.

VUMC The VU University Medical Centre is an academic medical centre, concerned with patient care, higher education and medical research. [3]

NKI The Netherlands Cancer Institute researches the cause and effects of cancer. The institutes three major areas of research are fundamental, clinical and translational cancer research. [2]

1.4 Overview

This document describes what documentation needs to be written, how testing will be done and how to keep track of bugs. Used tools and a number of mandatory actions (such as writing a test before writing code) are listed. Finally, a word about media control and how created records are handled.

Chapter 2

Software quality assurance plan

2.1 Management

2.1.1 Organisation structure

The image below shows the general organisational structure of the various teams involved in the production process.

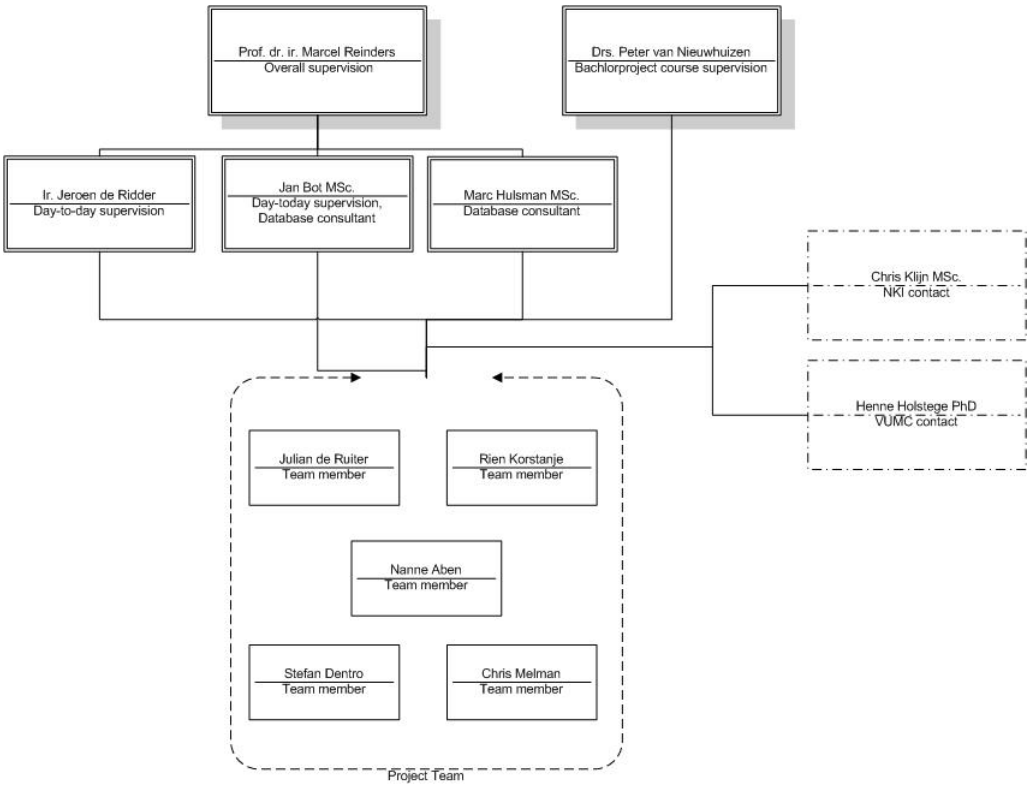


Figure 2.1: Organisation chart of the project

2.1.2 Roles and responsibilities

Supervision

Roles
Prof. dr. ir. Marcel Reinders (overall supervision)
Ir. Jeroen de Ridder (day-to-day supervision)
Jan Bot MSc (day-to-day supervision and database consultant)
Marc Hulsman MSc (database consultant)
Drs. Peter van Nieuwenhuizen (bachelor project course supervision)

Responsibilities
The supervising team is responsible for guiding the project team in their actions in order to guide the project towards a successful result. They are also responsible for providing the project team with any needed facilitations (within reason).

Clients

Roles
Chris Klijn (NKI)
Henne Holstege (VUMC)

Responsibilities
The clients are expected to provide their cooperation in order to determine their wishes toward the end product. They are also expected to provide feedback on interim products provided by the project team, as to determine the level of acceptance of the interim product.

Project team

Team members
Nanne Aben, Stefan Dentro, Rien Korstanje, Chris Melman, Julian de Ruiter

Roles

- Contact with supervisors: Julian
- Meeting chairman: Nanne
- Meeting secretary: Rien
- Blog: Stefan

Responsibilities
The project team is responsible for the production of the product. The team must create a correctly documented design of the product and deliver a corresponding product prototype.

2.2 Documentation

This project will produce a prototype that could later be expanded into a fully functional end-product. Proper documentation of the code and a proper design document is mandatory in order to ensure a certain level of maintainability. The design of the prototype is detailed in the following documents:

- Software Requirements Specification (SRS), following the IEEE 830 standard.
- Software Design Document (SDD), following the IEEE 1016 standard.
- Software and System Test Document (SSTD), following the IEEE 829 standard.

The SRS contains an overview of the basic requirements to which the prototype must adhere. The SRS is used as a guideline for the other documents. The SDD specifies the system architecture and application design, combined with a detailed description of each component in the system. The SSTD describes the various tests and testing procedures used to assert the quality of the prototype.

The contents of the end user documentation is beyond the scope of this document.

2.3 Code standards, practices, conventions and metrics

2.3.1 Documentation

Each component must be properly documented in the SDD, which must specify the design and function of each component and all its interfaces and list the dependencies and interactions of each component.

2.3.2 Design

IEEE and UML standards must be used when designing documents or models of any kind. These artefacts must be created unambiguously and in a single style across the entire project. The product itself must be designed for maintainability and flexibility and should therefore use an architecture that easily allows these qualities.

2.3.3 Source code

All source code committed to the main source tree must compile properly and result in the system passing more or an equal amount of tests than before addition of the new code (without regressions). A delivered pre-final prototype must be saved as a separate branch. Assuming that the project is created in an object-oriented style, the following overall requirements are set for any produced source code:

Components/Modules

1. A module must be self-contained.
2. A module must have a large level of coherency.
3. Modules may only communicate through well-defined and provided interfaces.

Classes

1. Classes must be foreseen of correct in-file documentation, containing at least the following:
    - Description of the class' function and general use
    - Containing module name

- Dependencies (internal/external)
- Invariants
- Author
- Known unresolved faults/incompleteness in the class implementation

2. Classes must have a large level of coherence (perform a single function).

Methods

1. All methods within a class must be foreseen of the correct in-file documentation, containing at least the following:
   - Description of the methods function and effects
   - Known unresolved faults/incompleteness in the method body
2. A method must have a large level of coherence (perform a single function).
3. A method must be less than 30 lines in length (though well-reasoned exceptions may exist).
4. Trivial lines of code in a method must not be foreseen of comments, whilst hard to understand code must be foreseen of corresponding commentary explaining its function.

2.3.4 Quality

The quality and completeness of the source code is gauged by running tests designed for the corresponding module and registering the amount of tests passed by the implementation. The exact testing procedures are described in section 2.4.

2.4 Testing practices

2.4.1 Structural testing

Unit tests are mandatory for every class, the only exceptions to this rule are graphical user interface classes. Unit tests should ensure at least branch coverage, MC/DC coverage may be required for critical sections. Extra attention must be given to component interfaces and external interfaces.

Integration testing is mandatory for interacting classes and modules in order to ensure correct interclass relations.

2.4.2 Functional testing

All classes and modules must be subjected to functional tests in order to assert that the prescribed functionality is provided by the artefact in question.

2.4.3 Overall procedure

All source code committed to the main source tree must compile properly and be able to pass the basic tests prescribed for that section of code. Failures must be registered in the Trac system and may initially be assigned to the developer responsible for the corresponding section of the system. As much detailed information as possible should be contained in the failure report, such as steps needed to reproduce the failure, the type of failure and the erroneous result. Such a report may also contain a recommendation of how to fix the fault. Resolved fault must be marked as such in Trac and steps undertaken to correct the fault must be reported.

The overall testing procedure is finished by a round of system and acceptance testing, details of which are yet to be defined.

2.5 Tools, techniques, and methodologies

2.5.1 Tools

Project management

- Subversion (SVN)
- Trac

Global programming

- Python
- Unit-test (included in Python)
- Coverage.py
- Modipyd
- Python Distribute

Visualisation

- PyQt
- Circos

2.5.2 Techniques and Methodologies

The project process is a combination of the waterfall development model and rapid prototyping. The phases of the waterfall model will be used in this process but are iterated a number of times in order allow the rapid production of new, pre-final, prototypes. Documentation must be adapted according experience gained from the built prototypes. Whilst the documentation phase focuses on the waterfall technique, development will mainly focus on the practice of rapid prototyping.

2.6 Media control

All produced documents and source code are stored in a single SVN repository. Only members of the development team have full access to the SVN repository, anonymous users only have read access through the Trac system. Access to the repository should be fully restricted should the SVN repository contain any confidential data. Every pre-final prototype must be stored as a separate branch for later use.

2.7 Records collection, maintenance, and retention

All documents are to be written in LaTeX and kept stored in the SVN repository in order to allow efficient collaboration during the development process. Through this technique the retention of files is kept as long as the SVN repository is alive. Every deliverable is versioned and stored separately in order to keep a correct version of each deliverable as it was supplied. The Trac system will be used to provide basic information about the system for the development team and other interested users. This system must be kept up-to-date with the production process.

References

[1] Little, Brown, and company. *A Dictionary of Greek and Roman biography and mythology*. Smith, William, Sir, 1867. URL: http://quod.lib.umich.edu/cgi/t/text/text-idx?c=moa;idno=ACL3129.0001.001.

[2] NKI. About the netherlands cancer institute, 2010. URL: http://www.nki.nl/Research/About+the+Netherlands+Cancer+Institute/.

[3] VUMC. About the vumc, 2010. URL: http://identity20.com/media/OSCON2005/.

Appendix G

Software Design Document

Graphical application for the statistical analysis of copy number variations in aCGH data

Thesis BSc Project IN3405

Software Design Document

version 1.1

Julian de Ruiter (1387367)
Chris Melman (1358359)
Rien Korstanje (1270389)
Stefan Dentro (1313320)
Nanne Aben (1369342)

DELFT UNIVERSITY OF TECHNOLOGY
EEMCS FACULTY
JUNE 28, 2010

Contents

Chapter 1

Introduction

1.1 Purpose

This document describes the architectural design of a statistical analysis tool for copy number variations in aCGH data. It also describes what the user interface will look like. The document is intended to serve as strict specification for the software developers implementing the system and provide insight into the component breakdown for computer engineers interested in the project.

1.2 Scope

This document describes the architectural design of a statistical analysis application for aCGH data. The tool is being developed for the NKI and the VUMC.

The application is able to import aCGH data from a local file and identify significant aberrations in the genomic data using various algorithms. It provides an interface in which a loaded aCGH profile can be viewed in a linear fashion. The interface also displays the various genes located in identified aberrant regions. Genes are annotated by various databases to provide more information about the gene in question. The application also provides a view which displays relations between the genes in the identified aberrant regions of the genome, allowing the type of relations to be shown can be chosen.

The application aims to allow quick analysis of the provided aCGH data by displaying a clear view of the aCGH profile, providing various algorithms to identify aberrant regions of the genome and displaying the identified regions in a clear manner. The displayed relations between genes in the identified regions provide the user with a significant amount of extra data, which can help identify non-obvious relations between the identified regions and corresponding genes.

1.3 Definitions, acronyms and abbreviations

Aberration Modification of the normal chromosome complement due to deletion, duplication, or rearrangement of genetic material.

Angita Angita is the early Roman goddess of healing, magic and witchcraft. [3]

Array comparative genomic hybridisation (aCGH) Array comparative genomic hybridisation is a technique to detect genomic copy number variations at a higher resolution level than chromosome-based comparative genomic hybridisation (CGH).

Copy number variant (CNV) A copy number variant is a segment of DNA in which copy-number differences have been found by comparison of two or more genomes. The segment may range from one kilo-base to several mega-bases in size. Humans (being diploid) ordinarily have two copies of each autosomal region, one per chromosome. This may vary for particular genetic regions due to deletion or duplication.

Genome annotation The process of identifying the locations of genes and all of the coding regions in a genome and determining what those genes do. An annotation (irrespective of the context) is a note added by way of explanation or commentary. Once a genome is sequenced, it needs to be annotated to make sense of it.

VUMC The VU University Medical Centre is an academic medical centre, concerned with patient care, higher education and medical research. [7]

NKI The Netherlands Cancer Institute researches the cause and effects of cancer. The institutes three major areas of research are fundamental, clinical and translational cancer research. [5]

Ibidas Ibidas is aimed to be an integration platform for various sorts of biological data. Its purpose is providing a very minimal but flexible database scheme. It is possible to store any kind of biological data while still keeping the learning curve for using the system low.

1.4 References

See bibliography section at the end of this document.

1.5 Overview

This document consists of 3 major parts. The 1st is an overview of the proposed system. It describes the chosen architecture and the internal subsystem interfaces. The 2nd part gives a more detailed description of the subsystems. Part 3 is a description of the proposed user interface.

Chapter 2

System architectural design

2.1 Overview

The product is a desktop application combined with a server-sided component. It relies on external input from local files and algorithms. In order to load aCGH profiles a number of parsers are created. The data supplied by these parsers is matched against data from the databases. This information is combined into a graphical view and is presented to the user. The user is able to interactively browse the explore the data and possibly reach conclusions about the nature of the presented case.

Ibidas is used as a platform for storing and querying large amounts of data. A number of data sets are combined and made available to the product. This includes basic genome information for both human and mouse, but also known phenotypes and abnormalities. The information is shown in a configurable manner. Users are able to define their preferences.

In order to present the data graphically to the user a custom plotter is created using Qt. Previously available plotters were all limited in the freedom required to present the data interactively. Even then it has been a major hassle to efficiently process all the data points and display them on screen. Additional relations between genomic regions are presented using an external link to Circos.

A more detailed description of the design can be found at the SDD and TDD that are included in the appendix.

2.1.1 Subsystem overview

The product is composed of six separate subsystems. The first subsystem is a file parsing subsystem, which is responsible for transforming an opened aCGH file into an internal data structure that can be processed/handled efficiently by the main application. It is able to parse all input files for which parsers have been supplied.

The second subsystem is the view. The view is responsible for handling all user interaction. It provides a number of views and configuration options.

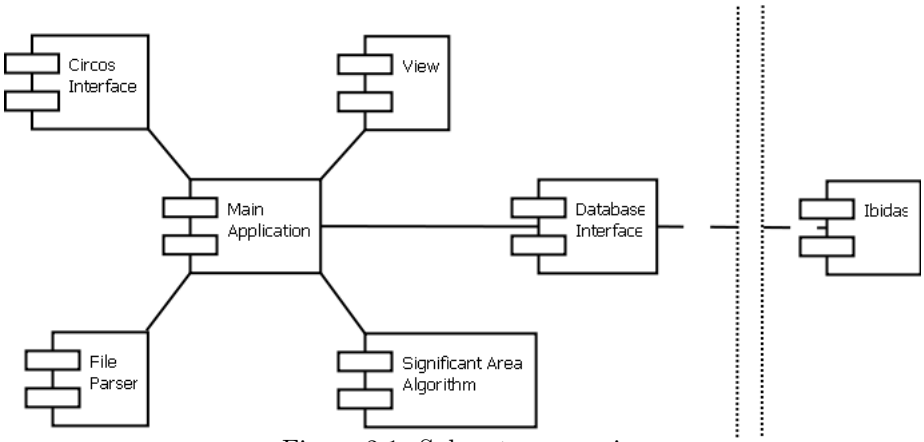


Figure 2.1: Subsystem overview

The third subsystem is the main application, which combines the various graphical components which are presented to the user with the other systems in order to obtain the needed data for presentation to the user. This component is responsible for the communication between all subsystems.

The fourth subsystem is an algorithmic subsystem, which determines the significant regions of a aCGH profile using the selected algorithm. This subsystem uses various (possibly external) algorithm implementations which can be selected as required. The subsystem ensures that the supplied data is transformed into the input format required by the selected algorithm and transforms output into a generic format to return as a result.

The fifth subsystem is an interface to Circos. It handles the generation of the configuration file and is also used for calling Circos to generate image using the given data.

The sixth and composed of a database subsystem which manages the connection to the databases and translates this data into a relatively uniform format. Ibidas is responsible for storing and handling of the data. The database subsystem provides access to Ibidas, transforming requests from the product in to Ibidas functions. It is also responsible for handling any errors that occur during this process.

2.2 Data management

The product does not save any data locally. Imported aCGH data files must be stored locally, their location is specified by the user. The user is able to set a number of preferences, including database build versions and database fields. Ibidas is responsible for handling the databases and offering the data to the product. More detail can be found in chapter 3.7.4.

2.3 Parallelisation

Great care has been taken to keep the product responsive. Due to the enormous amounts of data contained in aCGH profile files and the databases, extra attention was needed to maintain

responsiveness. A number of subsystems have been designed to run in its own thread, using asynchronous callbacks to communicate. Multi-threaded applications are basically a combination of a number of smaller processes in order to prevent blocking of subsystems. For example, parallelisation makes it possible that the GUI responds to user input while information is fetched from the database.

2.3.1 Boundary cases

Improper aCGH file

The product must be able to handle the most general errors that can be encountered when handling an external aCGH data file, most notably the following:

- No aCGH data in the file.
- Invalid aCGH data.
- Illegal aCGH data format.
- aCGH dataset too large.

In each of these cases the system displays a message to the user, which informs the user of its inability to handle the supplied file.

Starting the product

The product is started by a single (or double) click on application icon. While the application is loading, a progress bar is shown. The application updates its database information in this loading phase and loads the users preferences (if they were saved in a previous session). When the application has finished loading, the user is able to point to a datafile containing sample information. The sample information is loaded, analysed and the results are displayed by the product.

Stopping the product

When the product is shutdown, the current user preferences in the interface are saved to a text-based file. These preferences are loaded next time the product is started.

External database unavailable

The product relies on data supplied by external databases. Although the product is able to show sample information supplied by the user, referenced data and annotations will not be available. The product is still able to start, but has limited functionality (displaying only locally available data) and supplies the user with a message about the unavailability of the databases and its implications.

No internet connection available

The product is not dependent on an internet connection as the database subsystem is available locally. It must continue to work properly when no internet access is possible.

No algorithm module available

Without an algorithm module, the product has no manner to identify regions of interest in the loaded aCGH data. The application is still be able to load aCGH data, but does not identify any regions of interest for the user. The user must therefore search for such regions manually.

2.4 Discussion of alternative designs

An alternative design for the product was considered during the design phase, in which the product was also considered as a plugin for Cytoscape. Cytoscape is an open source bioinformatics software platform for visualising molecular interaction networks and integrating with gene expression profiles and other state data. Benefits of this design is the interface system of Cytoscape, which can the be utilised to display relational data and the interfaces provided by Cytoscape to access external databases. Usage of this available functionality would allow the development team to spend more time on the visual aspects of the product.

A drawback of the plugin design is that the product on which it is built, Cytoscape, is mainly used for identifying and visualising relations, something that not both clients are interested in. The supplied visualisation functionality of Cytoscape is therefore not highly valued, whilst the system may limit the product in its capability to display other, high-valued data in a different manner. Another drawback is that one client has never used Cytoscape, meaning that the use of such a plugin would require learning to use Cytoscape as well.

2.5 System interface description

2.5.1 Controller -View

The backend is represented by a special facade named Controller. The controller implements all functions needed by the GUI. These functions supply the GUI with a single point of request for information. The controller takes care of fetching the data, calling the appropriate subsystems and returning the result. Having a controller decouples the GUI from the backend. In figure 2.2 the relations of subsystems is displayed with specified interfaces for the controller to communicate with the other subsystems.

2.5.2 Controller - Algorithm

An algorithm takes a parameter containing CNV data and returns a list of interesting regions in a format defined in the Model.

2.5.3 Controller - FileReader

The FileReader takes a file location, reads the file and creates objects according to the file contents.

2.5.4 Controller - Circos

The Circos Interface takes a filename and list of significant regions (next to that some extra options). With this it creates all files needed for Circos and runs the application, with as output a picture.

2.5.5 Controller - Database

A database provides functions for connecting and disconnecting to a database. It also supplies the product with needed functions for retrieving data. A function for retrieving data takes a number of parameters to identify which types of data are of interest. It returns a container of Model objects complying with the given input parameters.

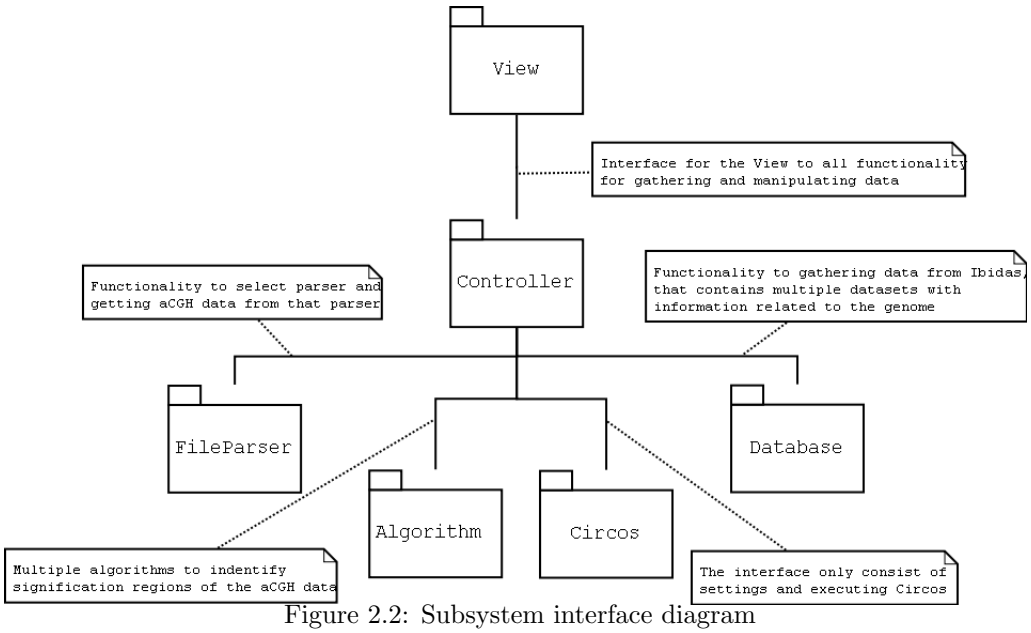


Figure 2.2: Subsystem interface diagram

Chapter 3

Design description

3.1 Shared

3.1.1 Data model

In order to handle the various types and the large amounts of data contained in CNV samples and other related data items efficiently, the application provides a well-defined set of model classes which model the various types of data and relations between data types in the application space.

This section first describes a general set of model classes that are used throughout the application, followed by an explanation of the various model classes used represent data types specific to CNV samples. An overview of the general model class structure is shown in the class diagram below.

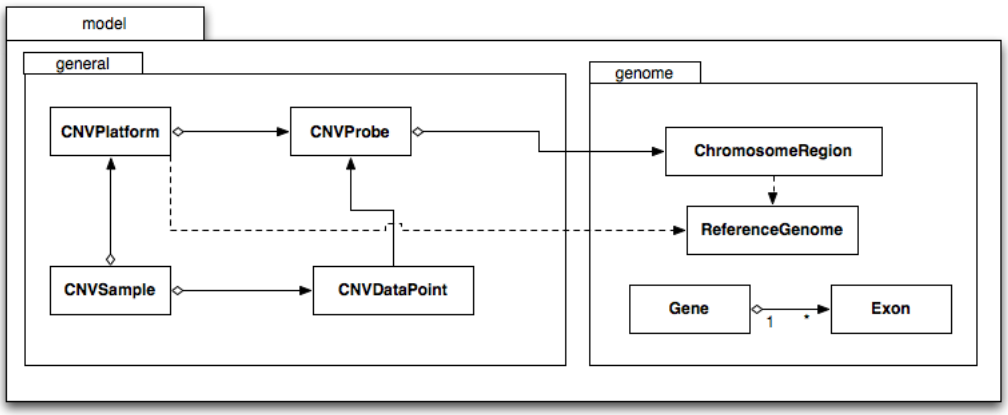


Figure 3.1: The basic class structure of the Model package.

Genome related classes

The Angita application supplies a number of model classes that describe the most basic elements of the domain model and are used various manners throughout the application. The following

paragraphs describe the basic implementation of these classes and their relations.

ChromosomeRegion The *ChromosomeRegion* class is used to identify a specific genomic region on a chromosome. In order to do so, such a region is given the name of the chromosome on which it is located, a start/end position in base pairs (zero-based) and a flag indicating if the region is located on the forward strand or on the reverse strand. The class also provides a number of convenience attributes and methods allowing simple querying of its location in other formats and other properties such as the length of the region.

ReferenceGenome The *ReferenceGenome* class provides a number of fields and methods concerned with identifying various properties of the genome of a specific animal species. The most important functionality provided by a ReferenceGenome instance is the listing of the names of the various chromosomes in the genome and the length of these chromosomes. It also provides a number of convenience methods which allows this data to be queried in various manners and allows the mapping of genomic positions, which are defined in the length of the entire chromosome instead of a specific chromosome, to chromosome positions and vice versa.

A *ReferenceGenome* instance is generally referenced by a *CNVPlatform* class (see section 3.1.1) in order to determine the species for which the corresponding platform is designed and the various characteristics of that genome. The instance is mainly used for analysation and visualisation purposes when analysing/visualising sample data for which analysis was performed with the platform in question.

Gene The Gene class represents a single gene in the system domain model. The class defines the gene id (as noted in Ensembl), the chromosome region in which the gene is located and maintains a list of the exons of the corresponding gene. The Gene class essentially contains data used to visualise genes in the application view.

Exon The Exon class represents an exon of a certain gene in the system domain model. The class defines an exon id, the chromosome in which the exon is located and the name of the gene to which the exon belongs. The Exon class is used to visualise exons in the visualisation of genes in the application view.

CNV sample related classes

In order to handle the various formats of CNV sample (related) data uniformly, the application defines a general domain model to which all external data formats containing CNV sample data are translated for processing by the application. This model output is most often generated by the file system subsystem, which performs this translation through the parsing of the various supported file formats. The most important classes in the sample related model are described below.

CNVProbe The *CNVProbe* class is the basic representation of a probe used in an array CGH analysis. The class defines only the most generic and required properties of a CNV probe, due to the fact that the various data sources each include different data concerning probes. The class defines a probe number and a chromosome region. The region of the probe is represented in the model by a *ChromosomeRegion* (described in section 3.1.1) instance. In order to allow easy referencing of the various data probes, the class also contains an identification number used to

track a probe instance within a sample context.

The *CNVProbe* class is mainly used by the *CNVPlatform* class and the *CNVDataPoint* class (both described below). The platform class tracks the entire collection of probes that are specific to that platform. The CNVDataPoint class references a specific probe instance, which represents the probe for which the data points measurement was made.

CNVPlatform The *CNVPlatform* class represents the platform used in an array CGH analysis. The most important properties of such a platform are the probes it contains, which are usually characteristic for the platform, and the various metadata describing the platforms properties. The class therefore provides a dictionary containing the platforms metadata and maintains a list of all the platforms probes, which are represented by the *CNVProbe* class. The class also provides a reference to a *ReferenceGenome* class, which represents the genome of a specific animal species and thus defines the type of genome the platform is designed for. The class also provides convenience methods allowing retrieval of specific probes by referencing their probe number.

CNVDataPoint The *CNVDataPoint* class represents a measured data point in the results of an array CGH analysis for a specific sample. Such a result generally has one data point entry per probe of the platform used to perform the analysis. The point class stores the two most important results of such an analysis, which are the measured log ratio and the p value of the measurement, which indicates the probability of the measurement being incorrect. The class also references the probe class with which it corresponds and the point type, which indicates whether a given point has been indicated as a gain, a loss or a normal point. This type field is generally set by an algorithm and is used by the visualisation of the point.

CNVSample The *CNVSample* class is the model representation of the result of an array CGH analysis. The class references an instance of the *CNVPlatform* class, which identifies the platform used to perform the analysis, holds a dictionary containing metadata concerning the sample and maintains a list of data points, which represents the measured data values for every probe in the afore mentioned platform.

Due to the fact that some samples are measured with a 5-to-3 ratio whilst others are measured with a 3-to-5 ratio and the fact that these ratios are not always clearly defined in the external data sources, the *CNVSample* class provides the *setRatio* method which allows the sample to easily switch between the two ratio types. This method effectively inverts all the data point measurement values by multiplying their values with minus one, as this is the only transformation required to switch between the two representations. This allows external code to quickly switch the ratio type of a sample without having to perform the necessary calculations itself.

The *CNVSample* class' function is representing the result of an array CGH analysis and contains all the data concerned with the loaded sample. As such, it forms the main representation of a sample in the application and is therefore a data type that is shared between the various subsystems. Examples of this practice are the file system subsystem, which is described in section 3.2 and provides a *CNVSample* instance as its output, and the algorithm subsystem, which requires a *CNVSample* instance as an input parameter on which the corresponding algorithm is subsequently run.

3.2 Filesystem subsystem

3.2.1 Overview

The file parsing subsystem is responsible for transforming a specified CGH file into its corresponding representation in internal model classes, which can be processed/handled efficiently by the main application. The subsystem is able to parse all input files that adhere to the supported data formats.

3.2.2 Subsystem structure

In order to keep the parsing pipeline as general as possible, the parsing of a specific file type is not performed by a single class but is spread over three separate class types. These function of these three class types are shortly described below, the exact implementation of the class hierarchy concerning them is detailed in further sections. A short overview of the general filesystem structure is shown in figure 3.2.

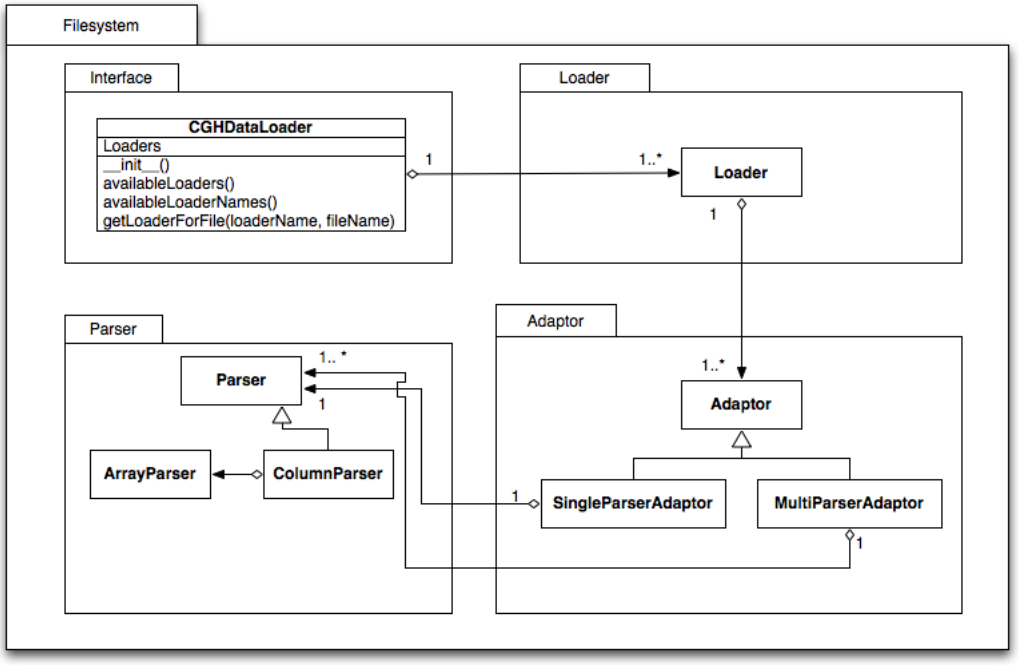


Figure 3.2: The basic class structure of the Filesystem subsystem.

Parsers The parser classes are responsible for parsing the actual text file to a python representation. A parser is provided a reference to a text file and in turn provides a generator which can be used by other classes. Each call to the generator results in the parsing of a line of the given text file to a tuple of variables, which is subsequently returned to the calling code. The benefit of returning data in this tuple format is that this format does not impose any assumptions or limitations on the parsed data, leaving the interpretation of the returned data to the calling code.

Adaptors The adaptor classes are responsible for calling the various parsers and transform the parser tuple data into the format required by the application. Examples of this are the various metadata adaptors, which accept a parser tuple and return a dictionary containing the various metadata, and the various data adaptors which return actual model representations of data such as CNV data points. Depending on their type, an adaptor can have a single parser or multiple parsers in order to load the required data.

Loaders The loader classes handle the various adaptors and aggregate the data returned by these adaptors into the final model representation of loaded sample file. The loader classes are the highest level classes of the subsystem and can be directly called by code outside the subsystem. The classes therefore all adhere to a single public interface and return data in a uniform format.

3.2.3 Design issues

The above described design of the filesystem package is not the first revision of its design as the original design was limited in its functionality. The original design is shown in figure 3.3 and contains two classes. The Loader class is the external interface for the subsystem which presents the available parsers to external applications and provides methods which can be called with a specific file path (and possibly parser id). These methods return (on success) a uniform representation of the CGH data of the specified file, which is represented by a structure of model classes of the data model.

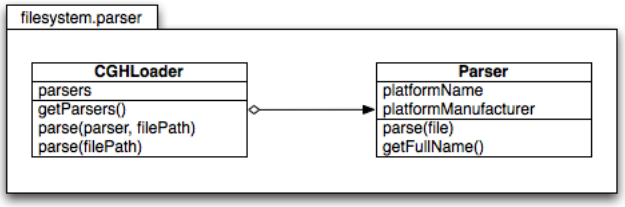


Figure 3.3: Old filesystem design

The second (set of) classes are Parser classes. These classes specify a standard interface to which all parsers must adhere and includes a few parsers which are packaged with the application by default. Each parser is able to parse a specific file format and returns the same uniform representation as defined by the subsystem (which is also returned to the application).

The limitations of this design is that it requires each file to define a new parser for each file, which must provide all functionality itself as even reuse of other parsers is limited due to their output already being model classes. The new design is much more flexible in this aspect by providing a staged loading pipeline and basic set of parser, loader and adaptor classes.

3.3 Graphical User Application

The user interface classes are responsible for the entire graphical interface with which the user interacts. These classes are responsible for handling all user-actions and data to the users view. The classes support the user in all the actions he execute in the application, which include the following actions:

- Loading aCGH data
- Analysing aCGH data
- Plotting the genome, showing
  - aCGH data point
  - significant regions
  - genes
- Viewing annotation data
- Viewing sample metadata
- Rendering plots, annotation data and metadata to documentation.

Most of these actions have been implemented using existing widgets in the Qt Framework[6]. Qt is a cross-platform application and UI framework. Qt was chosen to build the Graphical User Interface as it is available on all major operating systems, has python bindings through PyQt and looks aesthetically better then any considered alternative.

The design of the specific interface widgets is described in the following sections.

3.3.1 Mainwindow

The *Mainwindow* is the view through which a user executes all his actions. It is responsible for handling the various dialogs; annotation-, metadata- and plot-tabs; and the communication with the controller. The annotation data, metadata and plots are each viewed in a separate tabbed widget. The tabbed interface is designed to allow the user to switch between annotation data and plots easily, which would be more difficult in a multi-window design.

3.3.2 Dialogs

The functions of loading of aCGH data and the rendering of documentation are each contained in their own window. The standard Qt class *QFileDialog* is used for both cases, as this dialog resembles the standard save-file dialog on every OS and is perfectly suited for the task. As these cases use standard Qt classes to perform their work, they do not appear in the class diagram.

3.3.3 Tabbed widgets

The *QTabWidget* contains the plot, annotation data and sample metadata. The plot is shown automatically when the application is started. Annotation and metadata tabs are only shown when the corresponding data is requested by the user.

Gene annotation data tabs are constructed by the *AnnotationModel* class. This class is responsible for receiving annotation data from the controller and subsequently creating a *QWidget* that is added to the *QTabWidget* in the *Mainwindow*. A tree view is used to display annotation data, because annotation data can be substantially large in size. The view shows the corresponding gene-name along with its position and the names of the included databases when opened. The

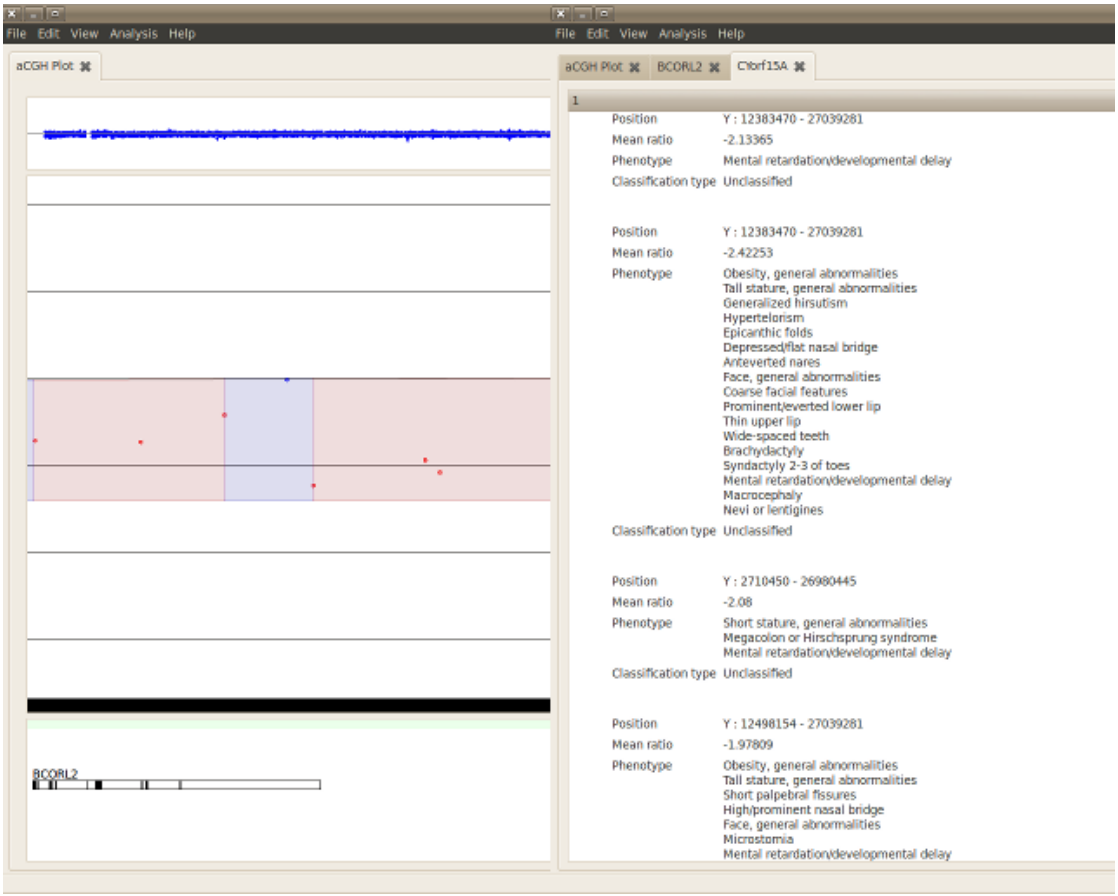


Figure 3.4: An example of a plot-tab and an annotation-tab.

user can expand a database node in the tree view in order to show the data from extracted from the corresponding database.

3.3.4 CNVBrowser

The purpose of the CNV browser is to provide the user with a visual representation of aCGH profile and analysis data. It also shows genes found in significant regions and enables the user to explore the data set visually.

3.3.5 Overview widget

The application provides an overview widget to help the user navigate the genome using the *CNVBrowser* view. This widget displays the entire CGH profile, along with a rectangle which represents the current view of the user in the *CNVBrowser* view, thus providing an easy view of

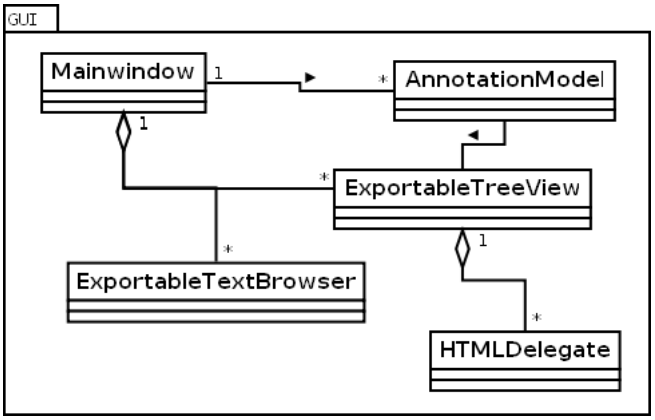


Figure 3.5: Relations between Mainwindow and the other classes involved in making tabs.

where the user is currently located in the CGH profile. It also allows the user to manipulate the contents of the *CNVBrowser* view through various mouse interactions.

As illustrated by figure 3.6 the *CNVBrowser* consists of a *CNVview* and a *GeneView*. These views are backed by *CNVScene* and *GeneScene* respectively, which are containers that manage the graphical items shown in the views. The views render the contents of the scenes using the Qt Graphics framework and allow the user to manipulate the content of the views. The actual sample and gene data are separately managed by the model.

The following sections describe the structure and implementation of the overview widget. The first section describes the *CNVOverview* class itself, whilst the second details a pixmap class used to draw the CGH profile efficiently within the overview view. The third section explains the implementation of the *OverviewPointTransform* class, a transformation class used to translate screen coordinates to base pair positions and vice versa. On overview of the overview widget structure is shown in 3.7.

CNVOverview

The *CNVOverview* class is essentially a *QGraphicsView* subclass and is responsible for handling the drawing of the overview widget and responding to the various events received by the view. The class maintains three graphical objects: a line representing the central axis of the view, a rectangle which denotes the current view of the *CNVBrowser* view and a pixel map item which displays the CGH profile in the background of the view. The overview manipulates these items as required to ensure an up-to-date view, depending on received external signals.

The class provides the following manners in which the user can manipulate the *CNVBrowser* view through interaction with the *CNVOverview* view: rubber-band selection, dragging the view rectangle, clicking on a position within the CGH profile and scrolling within the overview.

Rubber-band selection Rubber-band selection allows the user to select a region of the CGH profile by right clicking within the overview and dragging the cursor to create a selection area

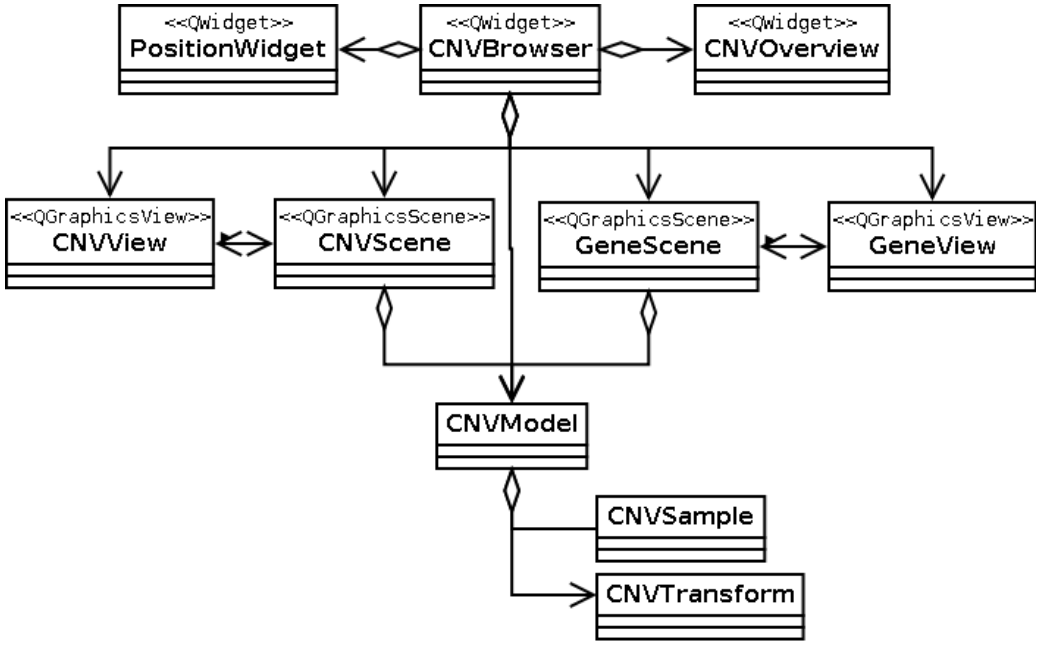


Figure 3.6: The *CNVBrowser* maintains a model that is visualised by the views.

within the overview. When the right mouse button is released, the selected area is set as the new view for the CNVBrowser and the view rectangle is updated correspondingly.

Dragging the view rectangle Dragging the view rectangle allows the user to scroll the current view along the CGH profile horizontally. Note that the overview limits the range of the view rectangles motion, as the view rectangle is not allowed to leave the CGH profile area.

Clicking on the CGH profile Clicking on a position within the CGH profile repositions the current view to horizontally centre on the clicked point in the CGH profile. This allows the user to quickly reposition the current CNV browser view to a specific point on the CGH profile.

Scrolling The overview also provides scroll functionality, which responds to mouse scroll events and either enlarges the width of the view rectangle or decreases it width depending on the scroll direction of the user, adjusting the *CNVBrowser* view accordingly.

CNVOverviewPixmap

The *CNVOverviewPixmap* class is a subclass of the Qt *QPixmap* class, which is essentially a container for image data in a pixel map format. The class is responsible for drawing a static image of the CGH profile which fits within the CNVOverviews viewport dimensions. Having a static image to display the data points gives the CNVOverview a great performance enhancement as the overview only needs to draw the corresponding data points once, instead of having to redraw the data points repeatedly.

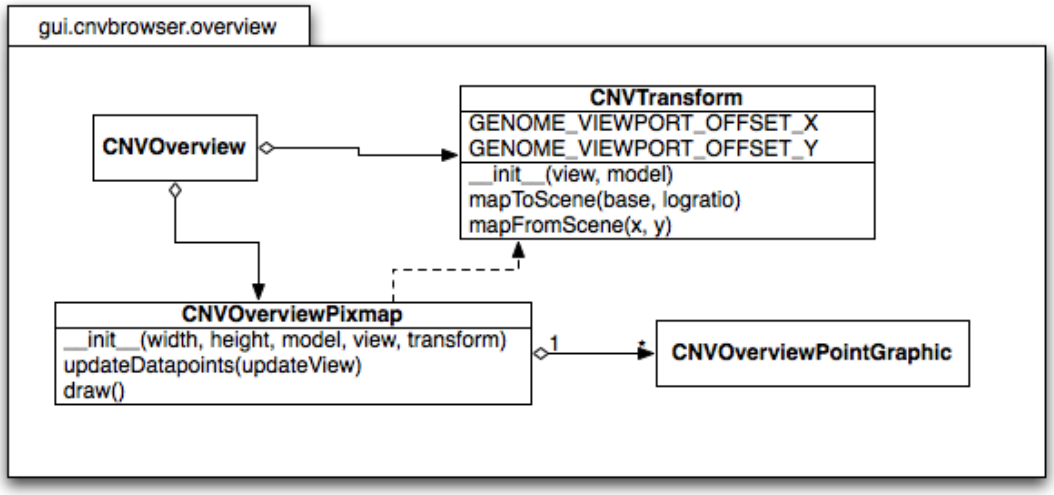


Figure 3.7: The basic structure of the overview widget design.

In order to draw the CGH profile correctly, the *CNVOverviewPixmap* class is provided with a number of data points and an OverviewPointTransform instance, which it uses to calculate the correct position of each data point within its image dimensions. It creates a corresponding graphic point for each data point with the calculated new coordinates and proceeds to render an image containing these points once, only requiring a redraw on a resize event of the overview or when the CGH profile changes.

OverviewPointTransform class

The *OverviewPointTransform* class provides two important utility methods for the two classes mentioned in the previous two sections, the *mapToScene* method and the *mapFromScene* method. The *mapToScene* method accepts a base pair position and a log ratio value and returns the corresponding coordinates in screen space, which are then subsequently used for drawing purposes and event handling.

The *mapFromScene* method is the inverse of the *mapToScene* method, as its name implies. This method accepts a x- and a y-coordinate in screen space and translates the coordinate pair to the corresponding base pair position and log ratio value. The method is mainly used by the various event handlers, which require mouse event coordinates to be translated to a base pair representation.

3.3.6 Position widget

The *PositionWidget* is a simple Qt widget that listens to the various signals emitted by the *CNVBrowser* and displays various data about the current view of the *CNVBrowser* view, such as the left-most and right-most base pairs currently in view as well as the currently visible chromosomes.

The widget also provides a panel containing various control inputs, which allow the user to select a chromosome and input a starting position and an end position in base pairs. After a subsequent click on the button input, the panel updates the CNVBrowser to display the CNVRegion specified specified by the chromosome name and position combination. The panel is only able to select a region on a single chromosome as the clients deemed it unnecessary to include functionality to select a region spanning multiple chromosomes in this fashion.

3.3.7 ListeningScrollView

As mentioned before both the *CNVView* and the *GeneView* enable the user to navigate the data. Any changes made to a view is communicated to the other as these changes should also be reflected by the other views.

The user can interact with the view in three ways:

- The data can be moved from side to side by dragging the view. This is done by holding down the left mouse button and moving the mouse.
- Zooming is done using the mouse wheel, resulting in the view zooming-in or zooming-out on the data in the view. This zooming is centred on the mouse cursor, meaning that the area under the mouse cursor is effectively enlarged. This allows the user to point at a gene or data point and zoom in on this point without having to drag the view to keep this point in view.
- A specific area can be zoomed in by dragging a box around the area that should be enlarged. This area is then fit into the view.

3.4 Controller

3.4.1 Overview

The controller subsystem manages the interaction between the model and the view. It provides an interface for the GUI and applies the operations to the model. It contains a number of functions that keep track of loaded samples, modified samples and visible samples at the GUI. It is the primary data storage facility. Because operation that take longer then a few milliseconds will end up blocking the GUI it is crucial to offload requests as fast as possible and be ready for the next request. A job dispatcher is used to offload requests.

3.4.2 Job Dispatcher

To be able to offload work to another thread, the job dispatcher takes a job and a callback. Once a job is accepted, it will be put into it's own thread which then executes the job. The call back will be used to notify the caller that his job has been finished. To check if a job is finished the job dispatcher will regularly check all jobs for completion. Once the job is completed, the call back will be called with the result.

3.5 Algorithm subsystem

The algorithm subsystem of the application contains and manages implementations of a number of algorithms that are run on CNVSample instances in order to identify significantly aberrant genomic regions in the corresponding subjects genome. This section describes the design and implementation of the subsystem by first mentioning the model classes defined by the subsystem to store data, followed by an explanation of the actual algorithm structure and the interface provided by the subsystem. An overview of the algorithm subsystem implementation is shown in the class diagram below.

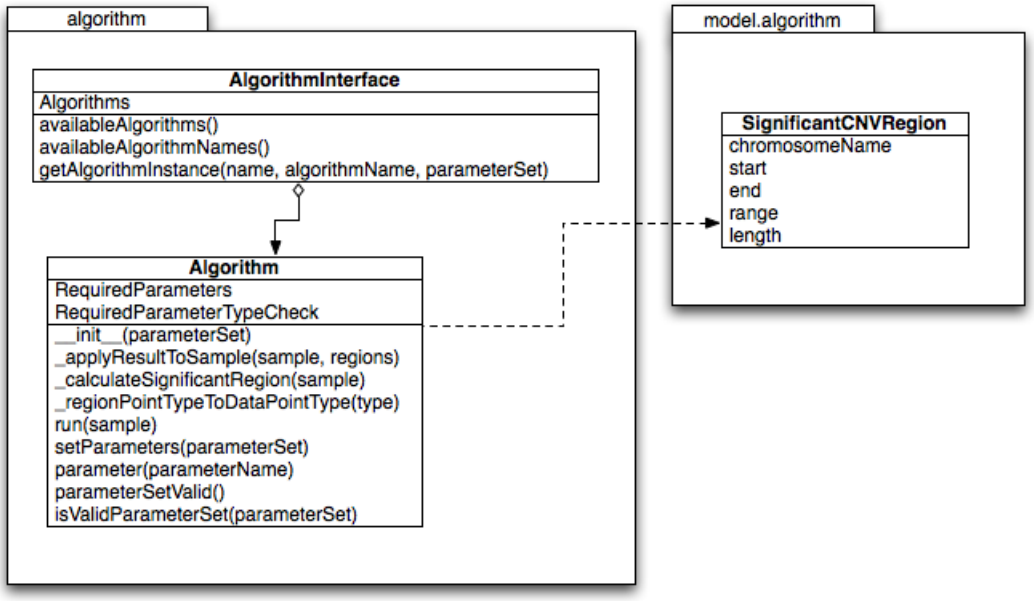


Figure 3.8: The basic class structure of the Algorithm subsystem.

3.5.1 Algorithm model classes

The main function of the algorithm subsystem is determining the significantly aberrant regions of a subjects genome according to the supplied sample data. The *SignificantCNVRegion* model class is provided by the subsystem to identify such a region. The class references a ChromosomeRegion class which identifies the chromosome region that the identified region spans. It also contains a type field, which specifies whether the region represents a loss or a gain of genomic material.

The output of each algorithm within the algorithm subsystem is a list of *SignificantCNVRegion* instances, which represent the regions identified by the algorithm. An algorithm can also provide and return a subclass of the *SignificantCNVRegion* class in order to return any extra data concerning the identified regions.

3.5.2 Algorithm implementation

The algorithm subsystem contains a number of algorithm implementations, each extending from the base Algorithm class. The base class provides a basic implementation of functionality deemed to be required by every algorithm implementation, thus providing a solid base for actual algorithm implementations. This section first discusses the implementation of the base class, followed by a description of the specific algorithm implementations currently provided in the application and their implementation.

Base Algorithm

The base *Algorithm* class provides the basic functionality for an algorithm but lacks an actual algorithm implementation, which is called upon to calculate the significant regions for a given sample. The Algorithm class can be supplied with a parameter set containing various parameters which are used by the algorithm during its execution.

An Algorithm can specify required fields by including their parameter names in the *RequiredParameters* field. The class also includes a *RequiredParameterCheck* field, which consists of function map that allows an *Algorithm* to specify functions to validate the supplied parameters.

The *run* method of the class effectively executes the algorithm and is called by external code, passing in the sample on which the algorithm must be run as an argument. The method is responsible for checking any given parameters and running the algorithm on the supplied sample. It returns a list of significant regions represented by instances of the *SignificantCNVRegion* class.

Implemented algorithms

KC-smart algorithm The KC-smart algorithm has been actively developed by the NKI client and is freely available in a R implementation. The description of the algorithm according to the algorithms documentation is shown below:

> KC-SMART is a method for finding recurrent gains and losses from a set of tumour samples measured on an array Comparative Genome Hybridisation (aCGH) platform. Instead of single tumour aberration calling, and subsequent minimal common region definition, KC-SMART takes an approach based on the continuous raw log2 values. KC-smart Gaussian locally weighted regression to the summed totals positive and negative log2 ratios separately. This result is corrected for uneven probe spacing as present on the given array. Peaks in the resulting normalised KC score are tested against a randomly permuted background. Regions of significant recurrent gains and losses are determined using the resulting, multiple testing corrected, significance threshold. [2]

The KC-smart algorithm has a number of interesting features, one being that the KC-smart algorithm can be run on multiple samples, providing an aggregate result in which KC-smart identifies can be used to identify correlation between different samples. Another interesting feature is the ability to perform multi-scale analysis. The algorithm can be run using different kernel widths, where small kernel widths will allow detection of small regions of aberration and vice versa large kernel widths allow large regions of aberration to be detected.

KC-smart text file algorithm Due to the fact that the KC-smart algorithm can take a substantial amount of time to complete when run with a large number iterations or on a large data set, an extra algorithm has been implemented that essentially loads and parses the text file output of a completed KC-smart session. The advantage of this method is that the KC-smart algorithm can be run wherever and whenever the user would like, requiring only its results to be supplied.The algorithm uses the same parser/adaptor structure as the filesystem subsystem. It opens the file, whose path is supplied as a parameter to the algorithm, and parses its contents to the equivalent list of SignificantCNVRegion instances, which is then returned and processed in the manner described above.

Threshold algorithm The threshold algorithm analyses a sample in order to determine groups of consecutive points on a chromosome that have an absolute log ratio larger than a given threshold. This algorithm has been implemented for the VUMC client as the client uses a similar workflow when identifying significant regions, by selecting groups of three or more consecutive data points with an absolute log ratio larger than a certain threshold.

The algorithm takes two parameters, one specifying the threshold which is used to test the data points and one specifying the number of consecutive data points required for the region to be considered significant.

3.5.3 Subsystem interface

The interface of the algorithm subsystem is provided by the AlgorithmInterface class. This class aggregates the available algorithm implementations and provides methods with which code external to the subsystem can query the names of the available algorithms and retrieve an instance of a specific algorithm. In this manner external code can retrieve algorithm instances without having to reference the internal algorithm classes directly.

As each algorithm implements the same interface, each returned algorithm instance can be handled in the same manner and therefore doesn't require any algorithm specific code. The only exception to this uniformity is the difference in required parameters between the various algorithm implementations, however this problem is reduced by the fact that required parameters can also be inferred in runtime from the algorithm through the *RequiredParameter* fields.

3.6 Circos Interface

Circos [1] is an external Perl program that is made for visualisation of the genome and relations to the genome. This package is made as an interface to circos and is needed since ciscos is advanced program that need a lot of customisation to get wanted output.

The interface has only one class(*CircosInterface*) With the given significant regions it searches for al genes in these regions. These genes are used by Ibidas to search for interesting relations between genes (proteins). The relations can differ in mode and action, to make this visible by using different colours for mode, and different intensity for mode. The legend of those can be accessed through the two getters for the dictionaries.

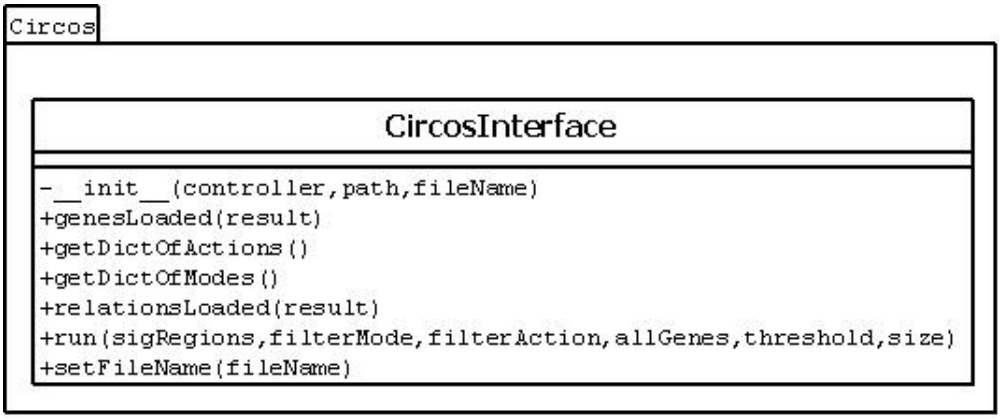


Figure 3.9: ClassDiagram circos

3.7 Database Handler

3.7.1 Overview

The database subsystem manages the connections to external databases and translates this data into a relatively uniform format. The subsystem is responsible for tracking availability of the various data sources, retrieving any queried data from the specified data source, transforming retrieved data into a uniform format and handling any errors that occur during this process.

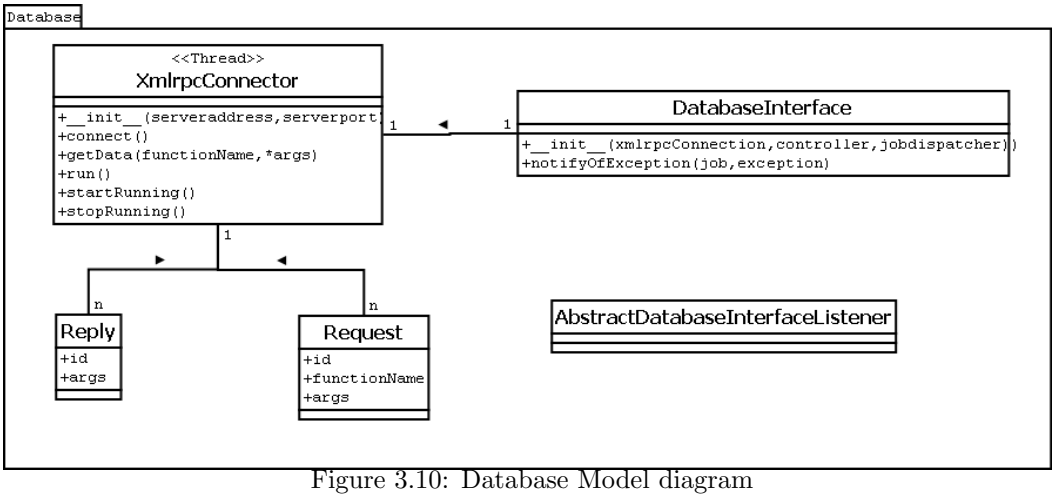


Figure 3.10: Database Model diagram

3.7.2 Overall subsystem structure

The subsystem is divided into sub packages: a model package and a Database handler package. The model package is a sub package of the main model package of the overall system, as this contains the classes used to communicate with the rest of the system and must therefore be shared with the rest of the system. The Database handler package contains the external interface of the subsystem delivering a connection to data for the application.

3.7.3 XML-RPC

Ibidas offers an XML-RPC connection, as well as a SOAP network interface. At first the product was designed as a client-only application. But this became unworkable as the load of Ibidas increased with the number of databases loaded. Therefore the choice was made to go from client-only to client-server. The XML-RPC implementation was choosen as it offers a clean and simple interface. In the long run, when the product evolves and the number of clients increases, it is probably worth while to change this to a threaded implementation with sockets. That way the server is able to handle more clients in less time.

3.7.4 Ibidas

Ibidas [4] is described as: "Ibidas is aimed to be an integration platform for various sorts of biological data. Its purpose is providing a very minimal but flexible database scheme. It is possible to store any kind of biological data while still keeping the learning curve for using the system low."

Normal databases present data in a flat 2D manner, however biological data requires a more flexible solution. Ibidas is able to maintain the data in the manner in which it is presented (potential multidimensional), instead of just combining a large number of 2D tables. Ibidas also keeps queries relatively simple as it does not require complex joins to couple the various data sources as a conventional database would.

A limitation of the Ibidas system is that it is currently only able to maintain the databases in memory, preventing storage of the data in a MySQL back-end. A startup script is supplied with the system, which ensures the system contains the required data after start up. The first start-up requires Ibidas to parse the various data formats, which are loaded into the system through text file dumps. The system can subsequently create a database dump, which can be loaded more quickly on the next start-up. The limitation does still limit the size of the database and results in high memory use.

Appendix A

GUI Sequence Diagrams

A.1 Load CGH Samples

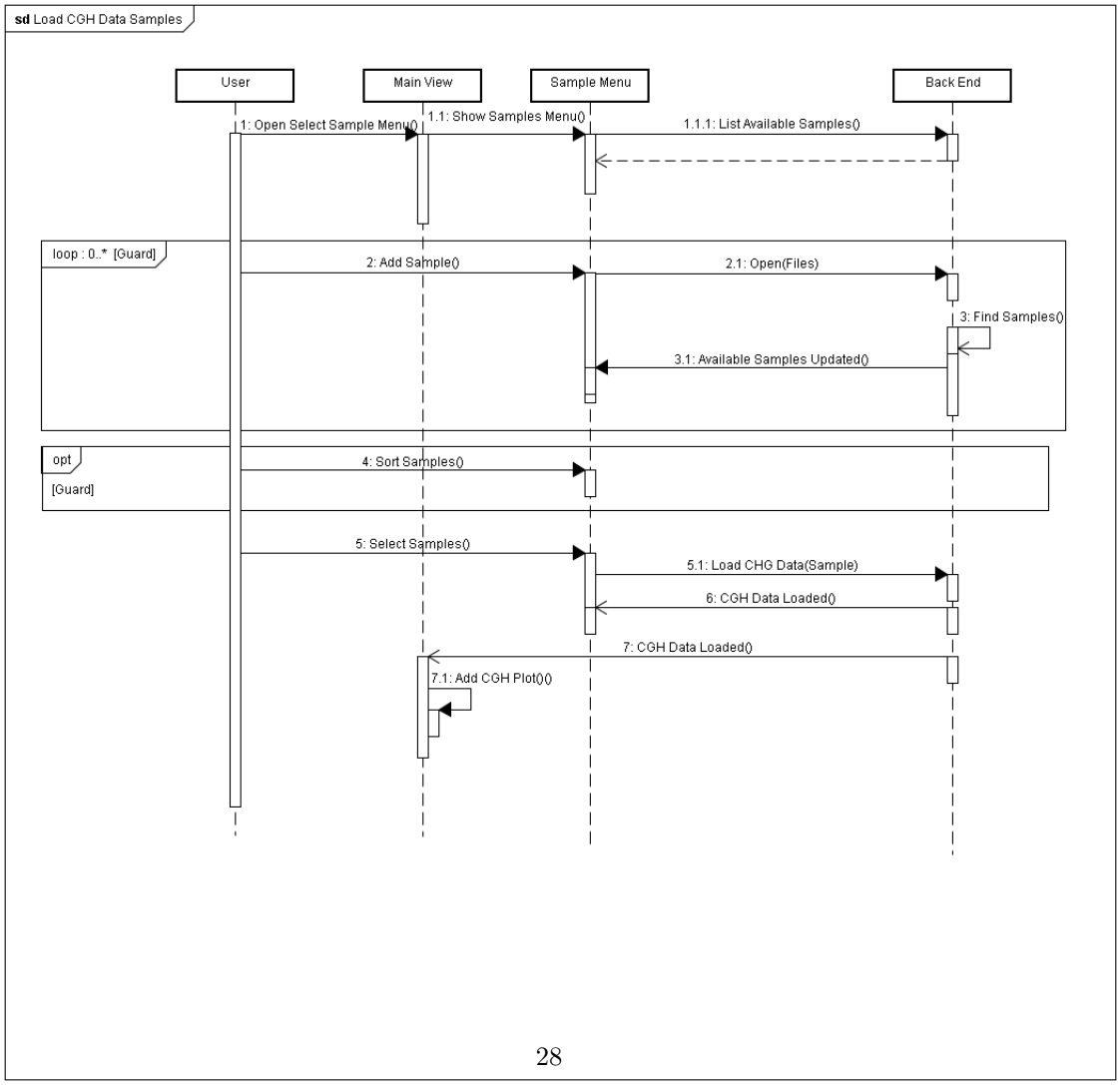


Figure A.1: Load CGH Samples

Typically CGH files contain the data of several CGH samples as well as meta data about the sampling platform. Because processing the CGH samples is relatively time consuming (approx. 10 seconds per sample) initially only the meta-data will be parsed and displayed.

This sequence details how the data will be loaded by the user without rendering the user interface unresponsive.

1. The user opens the Analysis menu.

1.1 The menu queries the back-end for a list of available samples that have been previously loaded. Since this data already exists in memory this is assumed to fast enough to be done directly. If reactivity does proof to be an issue, a list model can be used.

2. The user want to add a sample to the available samples. He selects a file with sample data.

2.1 The Sample Menu requests the back-end to open files and process the sample data.

3. When ever the back end is done processing the sample data is replies back with a event that the sample data has been processed.

4. The user can order samples by some of their characteristics.

5. The user then selects the samples and loads them.

5.1 The Sample Menu requests the back-end loads the selected samples.

6. Once the samples are loaded, the back end notifies to the GUI.

7. Once the samples are loaded, the back end notifies the main view.

7.1 The main view then adds a new graph with the sample data to the CGH plot.

A.2 Analysis of the data

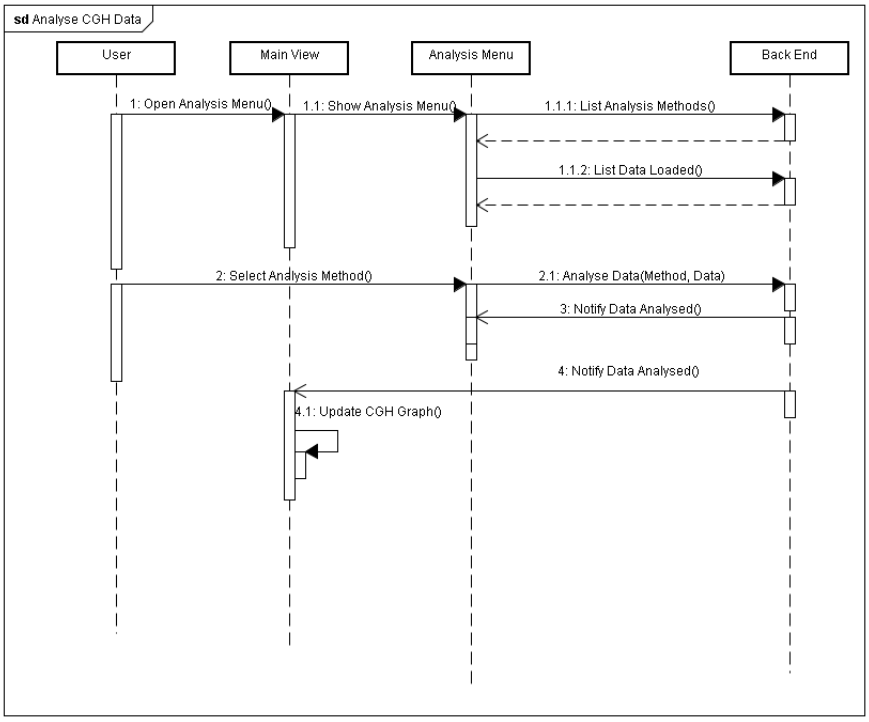


Figure A.2: Analysis of the data

The user will be able to automatically render a report of his findings. This will be done by exporting the data on open tabs. Of the plots the current view will be exported (i.e. when zoomed in, only that region will be exported). The annotation data will be exported completely.

The user will open the Render Documentation screen, he'll select a place to save the file and he'll press render. The back end will then handle the rendering. Since this is done in a different thread, the answer is given by an asynchronous process. The Render Documentation screen will be closed.

A.3 Lookup gene annotations

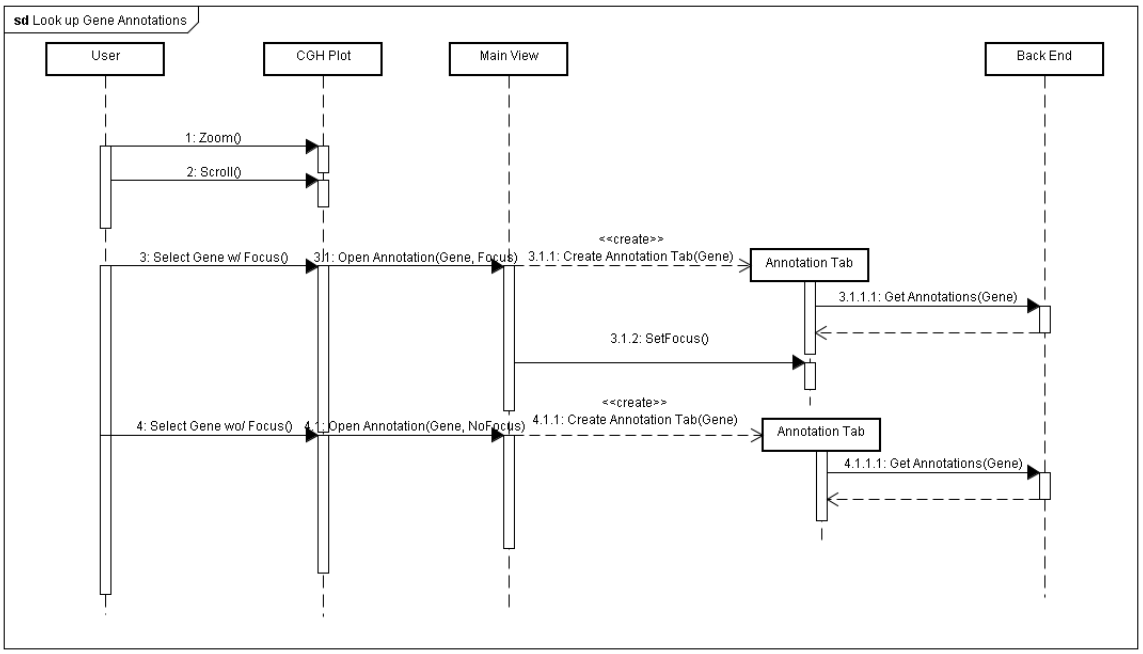


Figure A.3: Lookup gene annotations

The relevance of additions/deletions is determined by their annotations. Because annotations can be long pages full of text these can't be displayed in the gene-browser itself.

The next sequence demonstrates some navigation around the CGH graph as well as two ways to select a gene and view it's annotations.

1. The user zooms in to view a point close up.

2. The user scrolls to the side to follow a curve.

3. The user selects a gene with focus (LMB).

3.1 The CGH plot tells the main window to open a new Annotation tab. Which in turn ask the back end to load the relevant data. The load is assumed to be fast enough.

3.2 The Main view focuses on the newly created tab.

4. The user select a gene w/o focus (CMB).

4.1 as 3.1

A.4 Close CGH plot

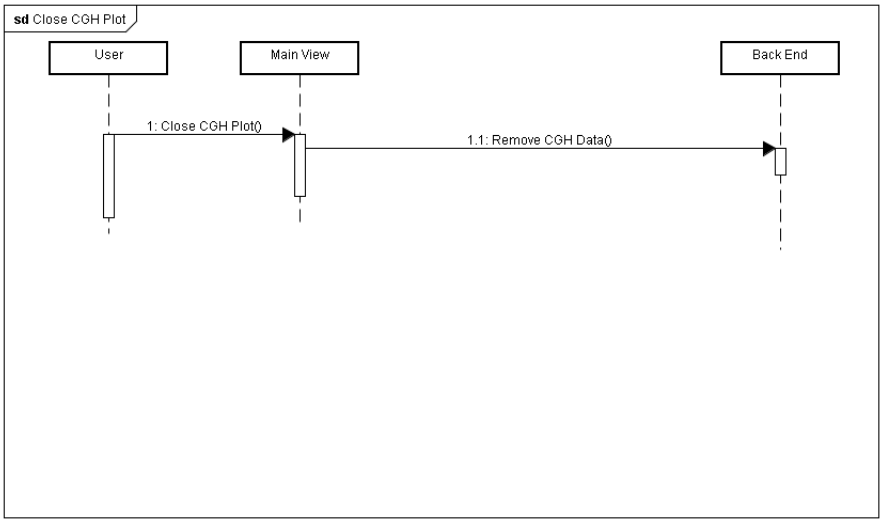


Figure A.4: Close CGH plot

Since it is possible to load multiple aCGH samples as well as adding multiple samples later on, it should also be possible close some of these samples again. This will benefit both available screen capacity and free up memory, causing the application to run faster.

The user can close an aCGH plot by clicking the 'exit'-button, much like closing a window. The plot will then be removed from the main window and the aCGH data will be cleared from the memory by the back end.

A.5 Plot with Circos

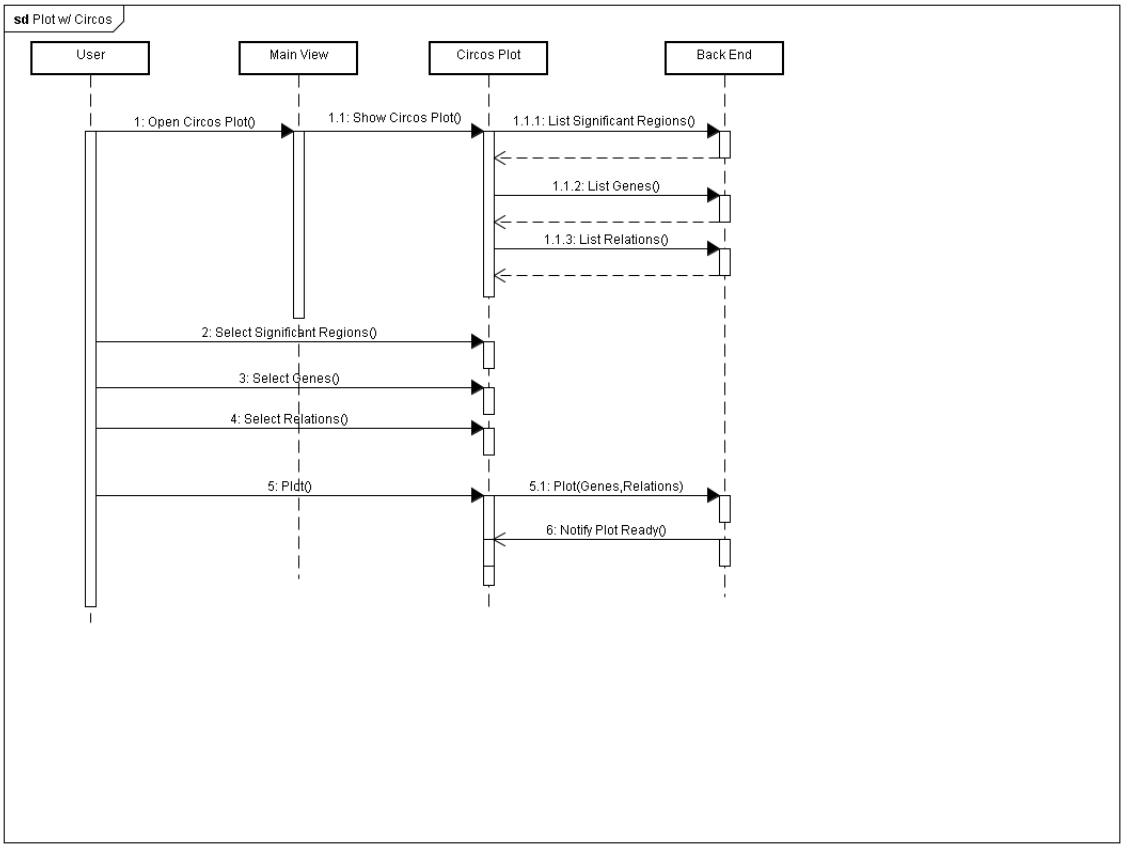


Figure A.5: Plot with Circos sequence diagram

It will be possible to view relations between various genes using the relation visualisation of Circos.

In order to do so, the user will open the Circos Plot View. This will be a new tab, where the aberrant regions are listed for each sample, as acquired from the back end. From all of these regions, the genes will be acquired from the back end too. Finally, the back end lists all types of relations that are known (e.g. functional or protein-protein interactions).

The user will then be able to choose two or more significant regions. Automatically, all genes from this region will be selected, but the user may choose to further specify which genes he wants to view in the plot. The user may also select what types of relations he wishes to plot.

Finally, the user will tell the system to plot this. This will be handled by the back-end (which hands it over to Circos). Since this is done in a different thread, the answer is given by an asynchronous process.

A.6 Render Documentation

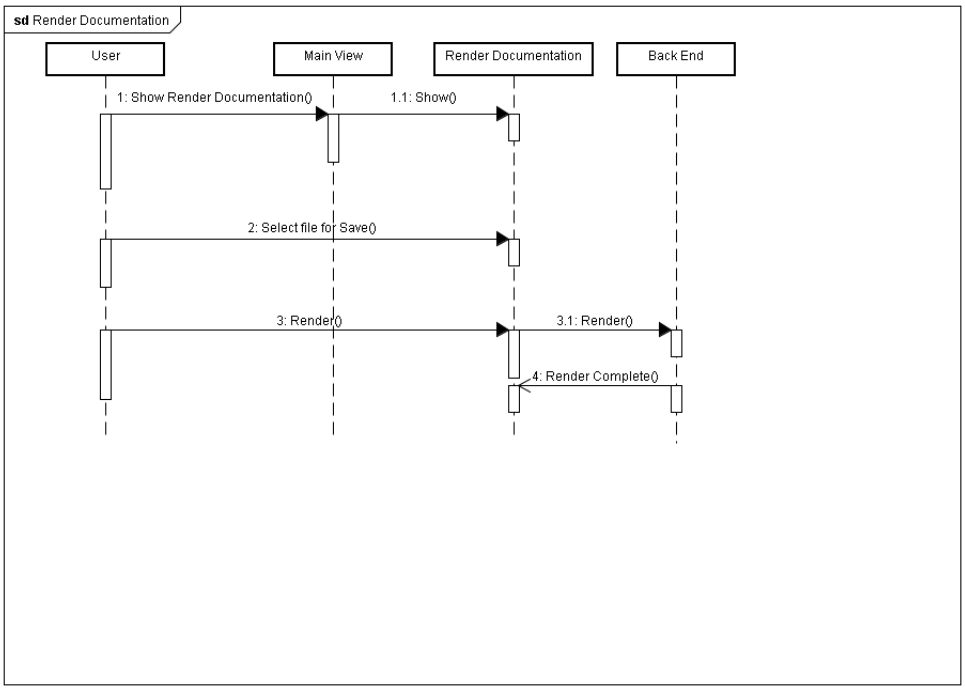


Figure A.6: Render documentation sequence diagram

The user will be able to automatically render a report of his findings. This will be done by exporting the data on open tabs. Of the plots the current view will be exported (i.e. when zoomed in, only that region will be exported). The annotation data will be exported completely.

The user will open the Render Documentation screen, he'll select a place to save the file and he'll press render. The back end will then handle the rendering. Since this is done in a different thread, the answer is given by an asynchronous process. The Render Documentation screen will be closed.

Bibliography

[1] BC Cancer Agency. Circos website, June 2010. `http://mkweb.bcgsc.ca/circos/`.

[2] C. Klijn and J. de Ronde. KC-SMART Vignette, 2010.

[3] Little, Brown, and company. *A Dictionary of Greek and Roman biography and mythology*. Smith, William, Sir, 1867. URL: http://quod.lib.umich.edu/cgi/t/text/text-idx?c=moa;idno=ACL3129.0001.001.

[4] NBIC. Ibidas website, June 2010. `https://wiki.nbic.nl/index.php/Ibidas`.

[5] NKI. About the netherlands cancer institute, 2010. URL: http://www.nki.nl/Research/About+the+Netherlands+Cancer+Institute/.

[6] Nokia. Qt website, June 2010. `http://qt.nokia.com/`.

[7] VUMC. About the vumc, 2010. URL: http://identity20.com/media/OSCON2005/.

Appendix H

Software and System Test Documentation

Graphical application for the statistical analysis of copy number variations in aCGH data

Thesis BSc Project IN3405

Software and System Test Documentation

version 1.2

Julian de Ruiter (1387367)
Chris Melman (1358359)
Rien Korstanje (1270389)
Stefan Dentro (1313320)
Nanne Aben (1369342)

DELFT UNIVERSITY OF TECHNOLOGY
EEMCS FACULTY
JUNE 28, 2010

Contents

Chapter 1

Introduction

1.1 Purpose

This document describes the test plan of a graphical, statistical analysis application of copy number variations in aCGH data. The document is intended to serve as strict specification for the software developers implementing tests and provide insight into the quality insurance of the code for computer engineers interested in the project.

1.2 Scope

The document describes the test plan used in the development of a graphical, statistical analysis application for aCGH data, codenamed Angita. The application is being developed for the NKI and the VUMC.

The application is able to import aCGH data from a local file and identify significant aberrations in the genomic data using various algorithms. It provides an interface in which a loaded aCGH profile can be viewed in a linear fashion. The interface also displays the various genes located in identified aberrant regions. Genes are annotated by various databases to provide more information about the gene in question. The application also provides a view which displays relations between the genes in the identified aberrant regions of the genome, allowing the type of relations to be shown can be chosen.

The application aims to allow quick analysis of the provided aCGH data by displaying a clear view of the aCGH profile, providing various algorithms to identify aberrant regions of the genome and displaying the identified regions in a clear manner. The displayed relations between genes in the identified regions provide the user with a significant amount of extra data, which can help identify non-obvious relations between the identified regions and corresponding genes.

The document specifies a plans for the testing of the application. It provides a global specification and therefore not contain total specification for every test that will be executed. The priority of the tests are also considered to provide a guideline for the minimal amount of testing for various parts of the application.

1.3 Test objectives

- Validate that the system satisfied the requirements for its intended use and user needs
- Validate that the right problem is solved (e.g., implement business rules and use the proper system assumptions)
- Facilitate early detection and correction of software and system anomalies
- Provide an early assessment of software and system performance
- Provide factual information to management for determining the business risk of releasing the product in its current state
- Determine that the software interfaces work correctly with other software components of the system or other systems in accordance with the requirements, and that errors are not propagated among software components.

1.4 Definitions, acronyms and abbreviations

Angita Angita is the early Roman goddess of healing, magic and witchcraft. [1]

Array comparative genomic hybridisation (aCGH) Array comparative genomic hybridisation is a technique to detect genomic copy number variations at a higher resolution level than chromosome-based comparative genomic hybridisation (CGH).

Ibidas Ibidas is aimed to be an integration platform for various sorts of biological data. Its purpose is providing a very minimal but flexible database scheme. It is possible to store any kind of biological data while still keeping the learning curve for using the system low.

1.5 References

See references section at the end of this document.

1.6 Overview

This document consists of two sections, the first being an explanation of the testing process to be used in the project. The second section provides detailed plans for each level that is tested and an overall plan how these test plans are combined.

Chapter 2

Software and system integrity levels

A scheme of integrity levels provides structured means for setting the breath and depth of testing. Every requirement of the product will be assigned a level, where a higher level requires more in depth testing compared to a lower level.

In the current context there are 5 different levels.

Description	**Level**
Software must execute correctly or grave consequences will occur. No mitigation is possible.	4
Software must execute correctly or the intended use of product will not be realised causing serious consequences. Partial-to-complete mitigation is possible.	3
Software must execute correctly or an intended function will not be realised causing minor consequences. Complete mitigation possible.	2
Software must execute correctly or intended function will not be realised causing negligible consequences. Mitigation not required.	1
No consequences.	0

Table 2.1: Overview of integrity levels

These integrity levels are assigned to components of the product. If a component is included in another component, the level of the parent component must be met.

The requirements as specified in the MoSCoW document are assigned a level. Both must and should have requirements are level 4. Could have is level 1, while won't have is level 0.

Chapter 3

Test processes

3.1 Test process

The test process consists of three levels (extracted from the V-model). The highest level is acceptance testing, the middle level is integration testing, finally the lowest level is unit testing. Every function that is not contained in the front-end must be accompanied by a test case.

3.1.1 Acceptance testing

Acceptance testing is black-box testing. It consists of two parts in order to test that the product meets the specified requirements. One is a test suite, specified according to the requirements, the second is a user test. This part is sometimes referred to as user acceptance testing. These tests will be conducted by an end-user.

The acceptance test levels are specified according to the requirement importance, as noted in the MoSCoW document. Must and should have requirements are assigned level 4. Could have requirements are assigned level 1, while won't have are assigned level 0.

3.1.2 Integration testing

Integration testing is the phase in which individual modules are combined and tested as a group. Its main purpose is to test whether the different modules are functional and reliable when combined. The test cases are constructed to test that all components within modules interact properly. Therefore branch coverage of 100% between modules is required, barring mutually excluding branches.

3.1.3 Unit testing

Unit testing is the process of testing each and every function that is used by another component. The purpose of unit tests is to test individual units of code. As a result, it affords several benefits. Unit tests find problems early in the development cycle. Unit tests must be specified such that every branch is covered, as measured with Coverage.py. This must be done by triggering if statements in all possible ways, baring the flow of events is possible to happen. MC/DC coverage is required for critical sections. Also, every exception caught must be tested.

Chapter 4

Test items

4.1 Product components

The product consists of a number of major parts. Each will need its own specific form of testing.

4.1.1 Model

The model is the heart of the product. It is essential to test it thoroughly. The functioning of the model is essential to all other modules. Therefore every function that is called by another class needs to be unit tested. Every branch in every such function must be tested.

At integration level all functions that are used by other modules must be tested, including branch coverage. When a new or modified class is checked in to SVN it must be accompanied by a (possibly updated) test suite.

4.1.2 Filesystem subsystem

The main idea behind the filesystem subsystem is that it is modular and provides a basic structure for implementing new file loaders. The module itself implements a strict scheme which plugins must adhere to.

The product is accompanied by at least one aCGH file loader. This loader must be tested according to the specified border cases (see section 7.4). Every function must be tested with an aCGH data file supplied by the clients.

4.1.3 Algorithm

The algorithm subsystem is like the filesystem subsystem designed to be modular and contains at least an implementation of the KC-smart algorithm and a threshold algorithm.

The supplied algorithms must be tested with data supplied by the clients, running them on basic representative data sets, identifying interesting regions. The results are compared to a representative result, which is either estimated manually or identified by an external algorithm.

4.1.4 Database

The database module consists of two parts, one is responsible for setting up the database connection, whilst the other is responsible for translating product requests to database queries.

Connection tests must include establishing connection, closing connection, disappearing connection, failing to connect due to database not available, failing to connect due to an error and illegal function calls.

Every function that implements a product query into a database query must be tested with a live database connection and a database containing required data. Functions must be tested with an empty query, a query returning one result and a query returning multiple results.

4.1.5 Graphical User Interface

No automated tests will be done. Graphical User Interface acceptance is tested in the User Acceptance Test described in section 4.3.

4.1.6 Circos module

The Circos module is the product interface to Circos. It must contain functions to specify which items and which relations to plot. It must also have functions to group items and/or relations and specify colours. Considering a configuration file is generated and the resulting image can only be checked manually, no tests are required.

4.1.7 Report module

The report module is responsible for creating a PDF file containing user specified elements. The module is a only an interface to an external module that is responsible for generating the document. Therefore no tests are required.

4.2 Module integration

When combining modules into a group, different functionalities blend and become one subsystem. This new subsystem has different responsibilities. Therefore the combination of modules needs to be tested.

The following combinations are to be tested:

- Model and filesystem subsystem (reading data in to the model).
- Model and database (accessing the database through the model and matching model data with data from the database).
- Model, filesystem and algorithm subsystem (reading data from a file, searching for interesting regions in the data. The data must be created in a way that the interesting regions are known beforehand, such that the results returned by the algorithm can be checked).
- Model, filesystem subsystem and Circos (reading data from a file and showing interesting regions in a circos plot).

- Model, filesystem subsystem and database (reading data from a file and matching it with data from the database).

As a result the inter-module communication is known to be correct. The product is ready for acceptance testing when all integration tests are passed and the graphical user interface is done.

4.3 User acceptance test

During the user acceptance test the clients (in name of the end-users) will conduct a test session. A developer will sit down with the client and go over the list of agreed requirements. These requirements represent the main functions of the product. The client must be able to perform the functionalities as specified. When this is the case the test will be marked as passed.

The user acceptance test is also the point at which the client accepts the layout of the graphical user interface (GUI). The client obviously is presented with a prototype of the GUI beforehand, marking the acceptance test as the last possible moment to request changes.

Chapter 5

What to test

5.1 Features to be tested

This section describes the items that are to be tested and assigns a risk to each of them. Its purpose is to supply management with an overview of the business risk when releasing the product at a certain stage. Three risks are possible: High (**H**), Medium (**M**) and Low (**L**).

- Loading aCGH data and plotting the normalised results (**H**).
- Linking genes to the correct corresponding region of the plot (**H**).
- Showing referenced information about a selected gene from specified databases (**H**).
- The possibility to select which data sources are used (**M**).
- Support of analysis of data through supported internal algorithms (**M**).
- Generation of a plot showing relations between genes (**M**).
- Generation of a report containing selected items (**L**).

5.2 Features not to be tested

- Support of analysis of data through various external software algorithms.
- Support of linking (an abstract of) papers related to aCGH specifics.

Chapter 6

Test criteria

6.1 Unit test

The unit test pass criteria are met when:

1. All coding is done.
2. Every function that is available to other modules has at least one test.
3. All branches in these functions are tested.
4. All tests pass.
5. Branch coverage is 100%, only not covering mutually excluding branches.

6.2 Integration test

Integration pass criteria are met when:

1. All coding is done.
2. Every combined function of modules is tested by at least one test.
3. Inter-module branch coverage is 100%, only not covering mutually excluding branches.
4. All tests pass.

6.3 Acceptance test

Acceptance testing is finished when:

1. All coding is done.
2. The clients have accepted the requirements after conducting the User Acceptance Test.

Chapter 7

General

7.1 Test deliverables

A small document with an overview of the acceptance tests and the remarks of the clients must be delivered. The document contains a per requirement description. It must also contain a number of statistics, including startup time, loading time of fetching view information from the database and processing time of a specified aCGH input file.

7.2 Environmental needs

- Specific aCGH data files coming directly from the clients are needed to test whether the product is able to handle them.
- Ibidas is required to test the database interfaces. It does not matter whether access is local or through XML-RPC.
- A dump of all databases mentioned in the Software Design Document must be obtained.

7.3 Responsibilities

Every class must be accompanied by a unit test file in which every function that is available to other modules is tested. It is the responsibility of the developer to make sure that tests are valid and up to date. Sometimes developer A has written a specific function, but it is edited by developer B, rendering the test invalid. It is the responsibility of developer B to adapt the tests to the changed function. All developers are required to notify the responsible person of any errors or wrongdoing.

Integration tests are conducted when modules are finished. Responsibilities are assigned later-on during the project. Acceptance testing is conducted when all modules are finished. All developers are responsible for meeting the requirements. Specific test responsibilities are assigned at a later stage.

During the project each developer, or group of developers, is responsible for a module or group of modules. It is their duty to make sure the module does what it is supposed to do, according to the

Software Design Document. This responsibility does not exclude other developers of conducting work in other modules.

7.4 Border cases

This section lists criteria that must be met when testing functions.

- If statement:
  - all values True
  - all values False
  - combinations of the above
- Functions: All paths through the function, including while, for and if statements
- When testing lists, sets or dictionaries, tests with the following criteria are mandatory:
  - 0 and 1 items
  - at least one test in range (10 - 1000) items
  - at least one test in range (100.000 - 1.000.000) items
  - one test of 1.000.000 items
- One and zero algorithms available
- One and zero file parsers available

References

[1] Little, Brown, and company. *A Dictionary of Greek and Roman biography and mythology*. Smith, William, Sir, 1867. URL: http://quod.lib.umich.edu/cgi/t/text/text-idx?c=moa;idno=ACL3129.0001.001.

Appendix A

Acceptance test

An acceptance test with the VUMC client was held ten days before the end of the project. This allowed a near finalised version of the product to be shown, whilst still allowing the results of the test to be used to tweak the final version of the program. The acceptance test has been executed with only one person, the VUMC client, as no other test subjects with the correct expertise were available on short notice. The NKI client was unavailable for the acceptance test.

During the test the user was asked to use the program independently. It quickly became apparent that a small tutorial on how to navigate on the aCGH plot (zooming and scrolling) was necessary. The user indicated that this problem was likely due to the fact that the interface of the application differs from the currently used system in navigation. After a very short tutorial, the user continued without any problems.

Another issue that occurred was that the settings dialog, which defines settings such as genome build, was placed in an unexpected location of the user interface. The client had expected to set these values when loading a file, but at the moment of testing, this dialog had to be opened separately from the menu. This flaw has been adjusted in the final version of the application.

The client also noted that the threshold algorithm, which is used to identify significant regions, does not identify all the regions it is required to as it leaves some regions unidentified. The client suggested a new algorithm, which automatically sets the threshold to 2σ. This is a simple solution, which is included in the final product but may however not solve the problem entirely.

When viewing annotation data, the client remarked that the DECIPHER data looked different from what she expected and enquired if it was possible to filter all entries categorised under Copy Number Variants. The client also asked if it was possible to create an export function that renders a tab-delimited file containing a summary per identified significant region. Such a summary should contain the position of the significant region, the number of probes it contains and the names of the genes that lie within the region. These features may not be included in the final product.

Finally, the client also noted that she was very pleased with the way DGV data was visualised. In comparison with the program she currently uses, the product shows more information in just as little space by using green and red for gains and losses and colour intensities for the number of observed variations.

Appendix I

Technical Design Document

Graphical application for the statistical analysis of copy number variations in aCGH data

Thesis BSc Project IN3405

Technical Design Document

version 1.1

Julian de Ruiter (1387367)
Chris Melman (1358359)
Rien Korstanje (1270389)
Stefan Dentro (1313320)
Nanne Aben (1369342)

DELFT UNIVERSITY OF TECHNOLOGY
EEMCS FACULTY
JUNE 28, 2010

Contents

Chapter 1

Introduction

1.1 Purpose

This document describes the technical design of a graphical, statistical analysis tool of copy number variations in aCGH data. The document is intended to serve as strict specification for the software developers implementing the system and provide insight into the requirements analysis for computer engineers interested in the project.

1.2 Scope

The document describes the technical design of a graphical, statistical analysis application for aCGH data, codenamed Angita. The tool is being developed for the NKI and the VUMC.

The application is able to import aCGH data from a local file and identify significant aberrations in the genomic data using various algorithms. It provides an interface in which a loaded aCGH profile can be viewed in a linear fashion. The interface also displays the various genes located in identified aberrant regions. Genes are annotated by various databases to provide more information about the gene in question. The application also provides a view which displays relations between the genes in the identified aberrant regions of the genome, allowing the type of relations to be shown can be chosen.

The application aims to allow quick analysis of the provided aCGH data by displaying a clear view of the aCGH profile, providing various algorithms to identify aberrant regions of the genome and displaying the identified regions in a clear manner. The displayed relations between genes in the identified regions provide the user with a significant amount of extra data, which can help identify non-obvious relations between the identified regions and corresponding genes.

1.3 Definitions, acronyms and abbreviations

Aberration Modification of the normal chromosome complement due to deletion, duplication, or rearrangement of genetic material.

Angita Angita is the early Roman goddess of healing, magic and witchcraft [7].

Array comparative genomic hybridisation (aCGH) Array comparative genomic hybridisation is a technique to detect genomic copy number variations at a higher resolution level than chromosome-based comparative genomic hybridisation (CGH).

Circos Circos is a software package for visualising data and information. It visualises data in a circular layout this makes Circos ideal for exploring relationships between objects or positions. [1]

Copy number variant (CNV) A copy number variant is a segment of DNA in which non-pathogenic copy-number differences have been found by comparison of two or more genomes. The segment may range from one kilo-base to several mega-bases in size. Humans (being diploid) ordinarily have two copies of each autosomal region, one per chromosome. This may vary for particular genetic regions due to deletion or duplication.

Genome annotation The process of identifying the locations of genes and all of the coding regions in a genome and determining what those genes do. An annotation (irrespective of the context) is a note added by way of explanation or commentary. Once a genome is sequenced, it needs to be annotated to make sense of it.

HTML HyperText Markup Language is the predominant markup language for web pages. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items.

Ibidas Ibidas is aimed to be an integration platform for various sorts of biological data. Its purpose is providing a very minimal but flexible database scheme. It is possible to store any kind of biological data while still keeping the learning curve for using the system low. [8]

NKI The Netherlands Cancer Institute researches the cause and effects of cancer. The institutes three major areas of research are fundamental, clinical and translational cancer research.

Qt Qt is a cross-platform application development framework widely used for the development of GUI programs (in which case it is known as a widget toolkit). It is produced by Nokia's Qt Development Frameworks division and is free and open source software.

SOAP Simple Object Access Protocol is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks.

SQL Structured Query Language is a database computer language designed for managing data in relational database management systems.

US2, GPL and ETABM File formats that are used for aCGH data storage.

VUMC The VU University Medical Centre is an academic medical centre, concerned with patient care, higher education and medical research.

XML Extensible Markup Language is a set of rules for encoding documents in machine-readable form. XML's design goals emphasise simplicity, generality, and usability over the Internet. It is a textual data format, with strong support via Unicode for the languages of the world. Although XML's design focuses on documents, it is widely used for the representation of arbitrary data structures.

XML-RPC XML-RPC provides a fairly lightweight means by which one computer can execute a program on a co-operating machine across a network like the Internet. It is based on XML and is used for everything from fetching stock quotes to checking weather forecasts.

1.4 Overview

Chapter two provides an overview of the design of the product. Chapter three describes the created product (Angita). It consists of a number of subsystems: Data handling, Graphical User Interface, Controller, Algorithm, Report and Circos. Chapter four is about Ibidas. It starts with an overview, followed by a description of available data parsers and the XML-RPC connection. Finally chapter five contains appended documents.

Chapter 2

Overall design

2.1 Overview

The product is a desktop application combined with a server-sided component. In a basic workflow, the product relies on a file loading subsystem which contains loaders to load aCGH files. The data from this subsystem is analysed by an algorithm subsystem and the results are matched against information contained in external databases, which is supplied by the database package through interaction with the server-sided database system. All this various information is combined into a graphical view and presented to the user. The user is able to interactively browse the various types of data in order to possibly reach conclusions about the nature of the presented case.

The server-sided component is formed by the Ibidas system. Ibidas is used as a platform for storing and querying the large amounts of data input from various external databases, thus combining these sources to a single interface for the product. The data in these databases includes basic genome information for both human and mouse, but also known gene data and associated phenotypes, relations, functions and abnormalities. The user is able to define which databases are used to collect data.

In order to present the data graphically to the user a custom plotter is created using Qt. Previously available plotters were all limited in the freedom required to present the data interactively. Even then it has been a major hassle to efficiently process all the data points and display them on screen. Additional relations between genomic regions are presented using an external link to Circos.

2.1.1 Subsystem overview

The product is composed of six separate subsystems, the first of which is the filesystem subsystem. This subsystem is responsible for transforming a specified aCGH data file into the internal model structure that can be processed and handled efficiently by the main application. It is able to parse all input files for which parsers have been supplied.

The second subsystem is formed by the user interface. The user interface is responsible for handling all user interaction and provides a number of views and configuration options.

The Controller forms the third subsystem, which combines the various graphical components

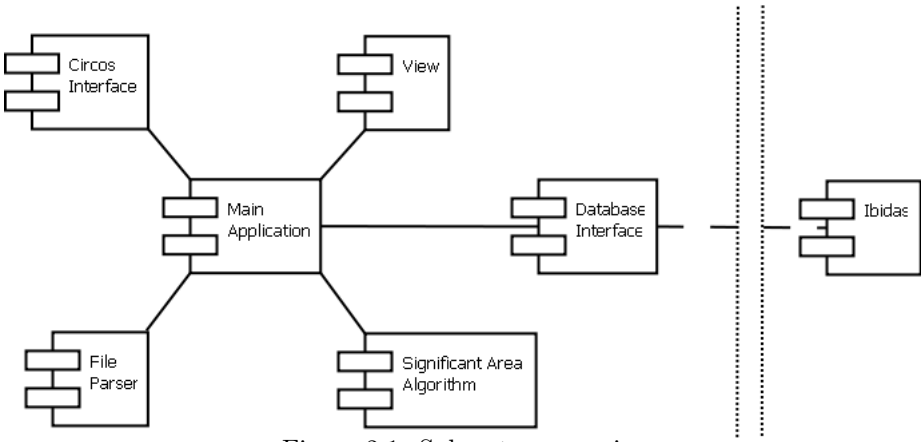


Figure 2.1: Subsystem overview

presented to the user with the other systems in order to obtain the needed data for presentation to the user. This component is responsible for the communication between all of the subsystems.

The fourth subsystem is an algorithmic subsystem, which determines the significant regions of a aCGH profile using a selected algorithm. This subsystem uses various (possibly external) algorithm implementations which can be selected as required. The subsystem ensures that the supplied data is transformed into the input format required by the selected algorithm and transforms output into a generic format to return as a result.

The fifth subsystem is an interface to the Circos application, which is responsible for drawing relational diagrams to illustrate relations between various genes. The subsystem handles the generation of the configuration file required by the application to draw an image correctly and is also responsible for subsequently running Circos to generate image using the supplied configuration.

The database component of the application forms the sixth subsystem of the application. This component is used for storing and querying data from different databases. The component has been split in a client (interface) - server (Ibidas) architecture, allowing these components to communicate through a XML-RPC protocol.

2.2 Data management

The product does not save any data locally. Imported aCGH data files must be stored locally, their location is specified by the user. The user is able to set a number of preferences, including database build versions and database fields. Ibidas is responsible for handling the databases and offering the data to the product. More detail can be found in chapter 4.

2.3 Parallelisation

Great care has been taken to keep the product responsive. Due to the enormous amounts of data contained in aCGH profile files and the databases, extra attention was needed to maintain responsiveness. A number of subsystems have been designed to run in its own thread, using

asynchronous callbacks to communicate. Multi-threaded applications are basically a combination of a number of smaller processes in order to prevent blocking of subsystems. For example, parallelisation makes it possible that the GUI responds to user input while information is fetched from the database.

Chapter 3

Angita

The Angita application is the desktop application component of the software system. The application is run on the client system and is among others responsible for the loading of CGH files, the visualisation of loaded CGH samples, running analysation algorithms on loaded CGH samples and displaying various types of related metadata.

In order to perform its tasks, the application has been divided into a number of subsystems as described in section 2.1.1. This chapter describes the implementation of these various subsystems as well as the shared domain model implementation.

The first section of this chapter describes the modules used directly in the loading of the basic CGH data, such as the domain model class structure which is used to store loaded data. Other modules described in the section are the filesystem subsystem, which is translates externally provided data files to their corresponding domain model representation, and the aggregation subsystem, which allows sample data from multiple samples to be aggregated into a single sample instance.

The second and third sections describe the graphical user interface structure and the main controller implementation respectively, followed by an explanation of the algorithm subsystem in the fourth section. The chapter is concluded by a discussion of the report module and the circos subsystem in the fifth and sixth section respectively.

3.1 Data handling

3.1.1 Data model

In order to handle the various types and the large amounts of data contained in CNV samples and other related data items efficiently, the application provides a well-defined set of model classes which model the various types of data and relations between data types in the application space.

This section first describes a general set of model classes that are used throughout the application, followed by an explanation of the various model classes used represent data types specific to CNV samples. An overview of the general model class structure is shown in the class diagram below.

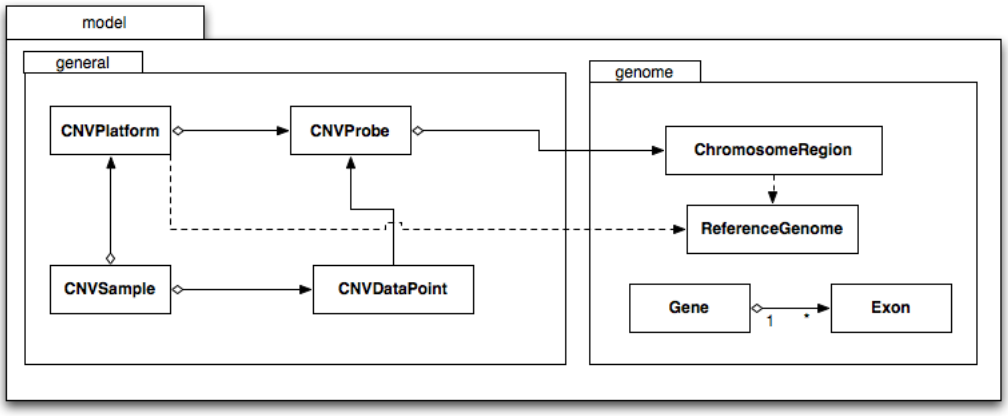


Figure 3.1: The basic class structure of the Model package.

3.1.1.1 Genome related classes

The Angita application supplies a number of model classes that describe the most basic elements of the domain model and are used various manners throughout the application. The following paragraphs describe the basic implementation of these classes and their relations.

ChromosomeRegion The *ChromosomeRegion* class is used to identify a specific genomic region on a chromosome. In order to do so, such a region is given the name of the chromosome on which it is located, a start/end position in base pairs (zero-based) and a flag indicating if the region is located on the forward strand or on the reverse strand. The class also provides a number of convenience attributes and methods allowing simple querying of its location in other formats and other properties such as the length of the region.

The *ChromosomeRegion* class is used in a number of different contexts, but its main function in the domain model is identifying the position of CNV probes on the genome.

ReferenceGenome The *ReferenceGenome* class provides a number of fields and methods concerned with identifying various properties of the genome of a specific animal species. The most important functionality provided by a ReferenceGenome instance is the listing of the names of the various chromosomes in the genome and the length of these chromosomes. It also provides a number of convenience methods which allows this data to be queried in various manners and allows the mapping of genomic positions, which are defined in the length of the entire chromosome instead of a specific chromosome, to chromosome positions and vice versa.

The application currently contains two specific instances of the *ReferenceGenome* class, one for the Homo Sapiens species and another for the Mus Musculus species, which are currently the only two species supported by the application. These instances are retrieved through a factory method in the class.

A *ReferenceGenome* instance is generally referenced by a *CNVPlatform* class (see section 3.1.1.2) in order to determine the species for which the corresponding platform is designed and the various

characteristics of that genome. The instance is mainly used for analysation and visualisation purposes when analysing/visualising sample data for which analysis was performed with the platform in question.

Gene The Gene class represents a single gene in the system domain model. The class defines the gene id (as noted in Ensembl), the chromosome region in which the gene is located and maintains a list of the exons of the corresponding gene. The Gene class is used to store gene data retrieved from Ensembl, which is subsequently used to visualise genes in the application view.

Exon The Exon class represents an exon of a certain gene in the system domain model. The class defines an exon id (as noted in Ensembl), the chromosome in which the exon is located and the name of the gene to which the exon belongs. The Exon class is used to visualise exons in the visualisation of genes in the application view.

3.1.1.2 CNV sample related classes

In order to handle the various formats of CNV sample (related) data uniformly, the application defines a general domain model to which all external data formats containing CNV sample data are translated for processing by the application. This model output is most often generated by the file system subsystem, which performs this translation through the parsing of the various supported file formats. The most important classes in the sample related model are described below.

CNVProbe The *CNVProbe* class is the basic representation of a probe used in an array CGH analysis. The class defines only the most generic and required properties of a CNV probe, due to the fact that the various data sources each include different data concerning probes. The class defines a probe number and a chromosome region as every source defines this data. The region of the probe is represented in the model by a *ChromosomeRegion* (described in section 3.1.1.1) instance. In order to allow easy referencing of the various data probes, the class also contains an identification number used to track a probe instance within a sample context. These identification numbers are generated if the data source does not contain probe identification numbers.

The *CNVProbe* class is mainly used by the *CNVPlatform* class and the *CNVDataPoint* class (both described below). The platform class tracks the entire collection of probes that are specific to that platform. The CNVDataPoint class references a specific probe instance, which represents the probe for which the data points measurement was made and as such defines various properties of the data point, an example being the genomic region in which the measured point is effectively located.

CNVPlatform The *CNVPlatform* class represents the platform used in an array CGH analysis. The most important properties of such a platform are the probes it contains, which are usually characteristic for the platform, and the various metadata describing the platforms properties. The class therefore provides a dictionary containing the platforms metadata and maintains a list of all the platforms probes, which are represented by the *CNVProbe* class. The class also provides a reference to a *ReferenceGenome* class, which represents the genome of a specific animal species and thus defines the type of genome the platform is designed for. The class also provides convenience methods allowing retrieval of specific probes by referencing their probe number.

The *CNVPlatform* class is used by the *CNVSample* class (described below) to store and provide the above mentioned data about the samples platform. The metadata of the platform may, for example, contain important information concerning the samples data measurements. The stored probe data is necessary for determining various properties of a samples data points, such as its region on the chromosome, which corresponds directly to the region of the probe for which the measurement was made.

CNVDataPoint The *CNVDataPoint* class represents a measured data point in the results of an array CGH analysis for a specific sample. Such a result generally has one data point entry per probe of the platform used to perform the analysis. The point class stores the two most important results of such an analysis, which are the measured log ratio and the p value of the measurement, which indicates the probability of the measurement being incorrect. The class also references the probe class with which it corresponds and the point type, which indicates whether a given point has been indicated as a gain, a loss or a normal point. This type field is generally set by an algorithm and is used by the visualisation of the point.

CNVSample The *CNVSample* class is the model representation of the result of an array CGH analysis. The class references an instance of the *CNVPlatform* class, which identifies the platform used to perform the analysis, holds a dictionary containing metadata concerning the sample and maintains a list of data points, which represents the measured data values for every probe in the afore mentioned platform.

The class provides two methods to access a samples point data, the first method allows external code to access a simple linear list of all data points, whilst the second allows access to a dictionary in which data points are stored in separate lists for each chromosome. Such a list contains all data points of a single chromosome and is sorted on the middle base position of the region of each data point on the chromosome (as specified by its probe) . This sorted list is used in various circumstances, most notably being various algorithms which utilise the point data, in which having sorted data points is either required or of a great advantage.

Due to the fact that some samples are measured with a 5-to-3 ratio whilst others are measured with a 3-to-5 ratio and the fact that these ratios are not always clearly defined in the external data sources, the *CNVSample* class provides the *setRatio* method which allows the sample to easily switch between the two ratio types. This method effectively inverts all the data point measurement values by multiplying their values with minus one, as this is the only transformation required to switch between the two representations. This allows external code to quickly switch the ratio type of a sample without having to perform the necessary calculations itself.

The *CNVSample* class' function is representing the result of an array CGH analysis and contains all the data concerned with the loaded sample. As such, it forms the main representation of a sample in the application and is therefore a data type that is shared between the various subsystems. Examples of this practice are the file system subsystem, which is described in section 3.1.2 and provides a *CNVSample* instance as its output, and the algorithm subsystem, which requires a *CNVSample* instance as an input parameter on which the corresponding algorithm is subsequently run.

3.1.2 File system

The file parsing subsystem is responsible for transforming a specified CGH file into its corresponding representation in internal model classes, which can be processed/handled efficiently by the main application. The subsystem is able to parse all input files that adhere to the supported data formats.

3.1.2.1 Main structure

The main problem encountered in the process of designing a generalised file parsing structure is the heterogeneity of the various file formats. Inherent to the field of bioinformatics is the lack of standardisation of processes and data interchange formats. Whilst attempts have been made to construct some sort of standardised data format, such as the MIAME specification [2], these formats are still too vague to allow construction of a single, unified parser. Whilst the current implementation of the subsystem supports the processing of only three differing file types, each of these file types differs substantially enough in its format to require an adjusted approach in the parsing of the file format. Examples of differences between these file formats are: XML files vs. text files, single data files vs. multiple data files, one sample per file vs. multiple samples per file.

In order to keep the parsing pipeline as general as possible, the parsing of a specific file type is not performed by a single class but is spread over three separate class types. The function of these three class types are shortly described below, the exact implementation of the class hierarchy concerning them is detailed in further sections. A short overview of the general filesystem structure is shown in the class diagram below.

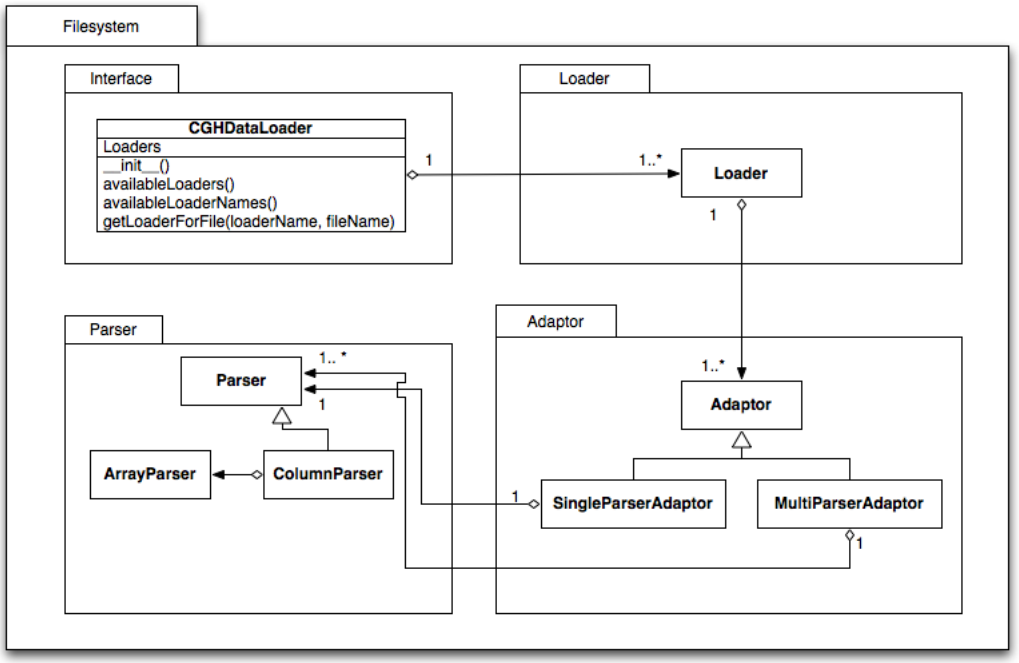


Figure 3.2: The basic class structure of the Filesystem subsystem.

Parsers The parser classes are responsible for parsing the actual text file to a python representation. A parser is provided a reference to a text file and in turn provides a generator which can be used by other classes. Each call to the generator results in the parsing of a line of the given text file to a tuple of variables, which is subsequently returned to the calling code. The benefit of returning data in this tuple format is that this format does not impose any assumptions or limitations on the parsed data, leaving the interpretation of the returned data to the calling code.

Adaptors The adaptor classes are responsible for calling the various parsers and transform the parser tuple data into the format required by the application. Examples of this are the various metadata adaptors, which accept a parser tuple and return a dictionary containing the various metadata, and the various data adaptors which return actual model representations of data such as CNV data points. Depending on their type, an adaptor can have a single parser or multiple parsers in order to load the required data.

Loaders The loader classes handle the various adaptors and aggregate the data returned by these adaptors into the final model representation of loaded sample file. The loader classes are the highest level classes of the subsystem and can be directly called by code outside the subsystem. The classes therefore all adhere to a single public interface and return data in a uniform format.

3.1.2.2 Supported file types

The current implementation of the subsystem supports the loading of three different data formats.

US2 The US2 data format corresponds with the data format which was supplied by the VUMC client. This data format supplies all data within a single text-based and tab-delimited file and contains the data of a single sample.

The first three lines of the file contain metadata concerning the platform used for analysis of the sample and metadata concerning the sample itself. The second set of three lines contains various statistics of the measurements made when the sample data was recorded in the same format. This data is currently ignored, but could easily be loaded as part of the general metadata.

The remainder of the file contains data concerning the probes used in the analysis of the sample and various measured values, such as the log ratio measured for each probe. Currently only lines containing a systematic probe name for the corresponding probe are parsed by the parser. Probes without a systematic name are currently ignored, as these probes generally do not correspond to a genomic position but are usually reference values used to normalise measured values.

GPL The GPL data format is the data format that was first supplied as test data for the application in its early development stages. This data format consists of multiple files, where a single XML-based file contains platform metadata and the metadata of the various samples. The platform probe data is contained in a separate text-based file, the location of which is specified in the platform metadata. The XML-file also specifies a separate text-based file for each sample, which contains the measured values of that sample per probe.

Both the platform probe data file and the sample data file are text-based and tab-delimited. The files only contain a single data type and are therefore uniform in structure. As with the US2 format, probes without a corresponding systematic name are ignored, as are measured sample values for those probes.

ETABM The ETABM data format corresponds with the data format which was supplied by the NKI client. This data format consists of two files, both of which are text-based and tab-delimited. The metadata file carries the SDRF file extension and contains metadata for each of the measured samples. Each metadata entry references a data file in which the measured sample values are kept. This data file is row-ordered and generally contains data of multiple samples in a single file by listing values for each sample in subsequent columns.

3.1.2.3 Parsers

As described in section 3.1.2.1, the parser classes are responsible for parsing the actual text file to a python representation. The main parser class structure is formed by two base classes, the *ArrayParser* class and the *ColumnParser* class.

ArrayParser class The *ArrayParser* class provides a simple interface for mapping a string, which is usually a single line from a file, to a tuple of values. The class is provided with a number of field names, a delimiter and an optional dictionary mapping specified functions to field names. The class then provides a parse function which uses the supplied delimiter to split a given string and links each split segment to the given list of field names. The split segments are inserted into the result tuple, unless the optional function dictionary contains an entry for a certain field field. In this case the corresponding split segment is passed to the mapped function and its return value is placed in the tuple instead. The result tuple is returned to the calling code.

In this manner the class effectively provides an easy method for parsing a given string, as file specific parsers only need to provide a delimiter and a list of column names. The column names can be used to identify entries in the tuple, whilst the function map provides a method for the calling code to transform raw data before it is inserted into the tuple. A much used example of such a transformation is the simple conversion of a numeric string value, which is the raw output of a split line, to an actual python float type. Such automatic transformations ensures that the calling code does not need to perform such transformations itself on each returned tuple.

ColumnParser class The *ColumnParser* class is the base class from which other file specific parsers inherit. The class is provided with a file name upon instantiation and in turn provides the generator *parsedItemGenerator*, which returns the data contained in the given file in the before described tuple format. It uses the *ArrayParser* class to parse individual lines in the specified values to tuples. The generator must be supplied with a delimiter, a list of column names and a function mapping dictionary, all of which are passed to the underlying *ArrayParser*. These variables are usually supplied in a default implementation in file specific subclasses, thus providing file specific parsing and no longer requiring the arguments to be filled in.

The default column parser behaviour is to iterate over and parse every line contained in the given file, but the class provides a number of methods with can be overridden in subclasses to control this behaviour. These methods allow skipping of lines in a file, validating lines, checking a tuple result for validity, post processing of a tuple result and breaking off the iteration on a given condition.

File specific classes Almost all current implementations of the file specific classes are derived from the *ColumnParser* class in the manner described in the previous section. There is one exception to this rule, which is the *GPLMetadataParser* class. This class is currently a lone

class, due to the fact that the *ColumnParser* hierarchy only supports simple delimited text-based files, whilst the *GPLMetadataParser* class must load its data from a XML file. The XML specific functionality required to do so is provided by the *XMLFileParseUtil* utility class, which is described in section XXI.

3.1.2.4 Adaptors

The adaptor classes are responsible for calling the various parsers and transform the parser tuple data into the format required by the application, as described earlier in section 3.1.2.1 The Adaptor class hierarchy is built around a number of classes, which provide various layers of functionality in order to allow for flexibility in the implementation of file specific adaptors.

The base Adaptor classes The most basic Adaptor is the *Adaptor* class itself, which is forms the base class of the entire hierarchy. The *Adaptor* class is an abstract class which does little other than define the interface to which all subclasses must adhere. The class defines the *load* method, which is the method actually responsible for fetching parser data and returning this data in the correct format. This method must be overridden by any subclass for correct functionality, as the base class provides only an empty method. It also provides two convenience methods for subclasses, which allows subclasses to reference a certain value in a tuple by a defining string name, instead of an index. This allows subclasses to access values in a given tuple without requiring indexes to be hardcoded into the subclass. This may be necessary for file types that do not have a specific order in which their data columns are contained in the data file, but whose order can be derived on loading the file.

The *Adaptor* class is subclassed by two separate base classes, the *MultipleParserAdaptor* and the *SingleParserAdaptor* class. As implied by the class names, the first of these classes is the base class for adaptors which use more than one parser, whilst the second provides a base for classes that require only a single parser. Both these subclasses define their own constructor and provide a convenience method for setting parsers from external code and an internal method for retrieving a parser instance.

The *SingleParserAdaptor* class is further extended by the *BasicMetadataAdaptor* and the *MappedMetadataAdaptor* classes. The *BasicMetadataAdaptor* class provides a default implementation of the *load* method, which should be sufficient for most metadata adaptors. This method calls the internal method *processColumn*, which can be overridden by subclasses in order to modify the behaviour of the default implementation when required. The *MappedMetadataAdaptor* class overrides this method for example in order to allow certain tuple fields to be passed through functions defined in a corresponding function mapping dictionary, much like the system used in the *ArrayParser* class. Such function maps can be used to aggregate columns or to provide default values if required.

3.1.2.5 Loaders

As mentioned before in section 3.1.2.1, the loader classes handle various adaptors and aggregate the data returned by these adaptors into the final model representation of loaded sample file. The loader classes are also the external interface through which code external to the subsystem can directly load data from files and therefore implement a uniform interface so that each class can be expected to behave in exactly the same manner.

A *Loader* class provided with a file name upon construction and is then bound to this file. The base *Loader* class provides the attribute *availableSampleNames* to allow identification of samples contained in the provided file by their name and provides the methods *sampleMetadataForSample* and *platformMetadataForSample* to fetch the various types of metadata for a specific sample. The *loadSample* method actually loads an entire sample, based on the sample name passed to the method, and returns the corresponding representation of the sample in model classes. Note that due to the sheer size of some data files, loading an entire sample may take up to 30 seconds on large files.

These methods are not filled in by the base class, due to the sheer heterogeneity of the various data formats. This results in each data format requiring a slightly different approach to load the corresponding data correctly. Each data format therefore has its own loader subclass, which overrides the required methods in the correct manner and is responsible for setting up its internal adaptors and parsers.

3.1.2.6 Subsystem Interface

The interface of the filesystem subsystem is provided by the CGHLoaderInterface class. This class aggregates the available filesystem loader classes and provides methods with which code external to the subsystem can query the names of the available loaders and retrieve an instance of a specific loader for a given file name. In this manner external code can retrieve loader instances without having to reference the internal classes directly. As each loader implements the same interface, each returned instance can be handled in the same manner and therefore doesn't require any loader specific code.

3.1.3 Sample aggregation

The aggregation package provides a number of aggregator classes, which aggregate multiple CNVSample instances into a single instance. This allows to the application to display multiple samples in a single aggregated plot, which then represents a certain relation between the different samples, depending on the used aggregator. Another use of an aggregated instance is in the algorithm subsystem of the application, which allows the application to run an algorithm on the aggregated sample instance, which can allow algorithms to identify significant regions shared among multiple samples.

The package currently provides a base *Aggregator* class, which is described in the first part of this section, and a single actual implementation of the *Aggregator* class, the *MeanAggregator* class, which is described in the second part of this section.

3.1.3.1 The base Aggregator class

The package provides a basic Aggregator class, which is effectively an abstract class that only provides basic functionality common to all aggregator instances. The main function of this abstract class is providing the basic methods which allows manipulation of the samples associated with the Aggregator instance, through use of the *addSample*, *removeSample*, *setSamples* and *clearSamples* methods. The class also provides methods which check the sanity of the associated class set, as various properties such as the platforms associated with the samples must be the same for each associated platform.

The aggregation itself is initiated by calling the *run* method of the Aggregator class. This methods basically calls the internal *aggregate* method after performing the above mentioned sanity checks. The *aggregate* method performs the actual aggregation and must be implemented by every concrete subclass.

3.1.3.2 The MeanAggregator class

The MeanAggregator is currently the only concrete implementation of an aggregator. The MeanAggregator aggregates its samples by calculating the mean log ratio value of every data point and using this mean value as the log ratio value for the corresponding data point in the aggregated sample. In doing so, the aggregator calculates the mean sample of all the corresponding data set, which allows subsequent analysis and visualisation to identify losses or gains of genomic material that are common within the given sample set. As mentioned in the previous section, the MeanAggregator simply overrides the default implementation of the *aggregate* method to perform the required calculations.

3.2 Graphical User Interface

The GUI classes are responsible for the entire front-end. They handle all user-actions and view relevant data to the user, and should facilitate the user in all the actions he wishes to do in the program. Of these user actions, the following has been identified:

- Loading aCGH data
- Analysing aCGH data
- Plotting the genome, showing
  - aCGH data point
  - significant regions
  - genes
- Viewing annotation data
- Viewing sample metadata
- Rendering plots, annotation data and metadata to documentation.

Most of these actions have been implemented using existing widgets in the Qt Framework [9]. Qt is a cross-platform application and UI framework. Qt was chosen to build the Graphical User Interface as it works cross platform, has python bindings through PyQt and looked aesthetically better then any of the alternatives. Additionally the Qt Graphics Framework provides a powerful way to render a large number of items efficiently.

The implementation of specific interface widgets and the motivation is described in section 3.2.2. For the plotting of data however, the existing widgets had to be extensively adapted. Therefore these classes are described separately in section 3.2.5.

3.2.1 Mainwindow

The *Mainwindow* is the portal through which the users executes all his actions. It is responsible for the handling of dialogs; annotation-, metadata- and plot-tabs; and the communication with the back-end.

All our dialogs are opened through the menu. The *Mainwindow* references to them, so they are not collected by the garbage collector, and it connects the dialog with itself through a signal and slot, so it can receive the result from the dialog. The dialogs themselves are described in section 3.2.2.

The annotations, metadata and plots are viewed in a tabbed widget. This decision was made, because Angita has to show a lot of information. Besides the plot, which contains thousands of data points, there was little room on the main screen for sample metadata and annotation data. Instead of opening a new window every time, they are shown in tabs, like most modern browsers show web pages in different tabs. This makes it easier for the user to switch between annotations and the plot, and it is easier for him to handle many different tabs than many different windows.

Figure 3.3: An example of a plot-tab and an annotation-tab.

Often the front-end needs data from the back-end. To do so the relevant function in the controller is called with a function pointer as parameter. This pointer points to a function in the front-end, to which the controller should return its answer. However, because the back-end is threaded (that way the GUI does not freeze while the back-end is working), the answer arrives asynchronous and Qt cannot handle this appropriately. This was overcome by using a queued signal and slot inside the front-end. This way, all Qt code remains within the front end and the asynchronous call is queued into the GUI thread.

3.2.2 Dialogs

The loading of aCGH data, the rendering of documentation, the analysing of data and the changing of settings are all handled in their own dialog. For the loading of aCGH data and the rendering of documentation the standard Qt class *QFileDialog* is used. This dialog resembles the standard save-file dialog on every OS and is perfectly suited for the task. Therefore these actions do not appear in the class diagram.

For the analysis of data and the settings-dialog, custom dialogs have been made. Instead of using *QDialogs* however, *QMainWindow* was subclassed, because the *QDialogs* had some lay-out problems on Mac OS and Linux. After removing the status-bar and the menu and adding an 'OK' and 'Cancel' button, the *QMainWindow* works great as a dialog.

Besides offering a form for the user to enter settings, the settings dialog also stores them. At the moment they are not saved to the hard drive, so any changes will be reset when the program restarts.

Finally the *ErrorMessage* class offers an easy way to show errors in a *QMessageBox*. It replaces four lines that are originally used to create the message with a single function call.

3.2.3 Tabbed widgets

The *QTabWidget* contains the plot, annotation data and sample metadata. The plot is shown automatically when the application is started. Annotation and metadata tabs are only shown when the corresponding data is requested by the user.

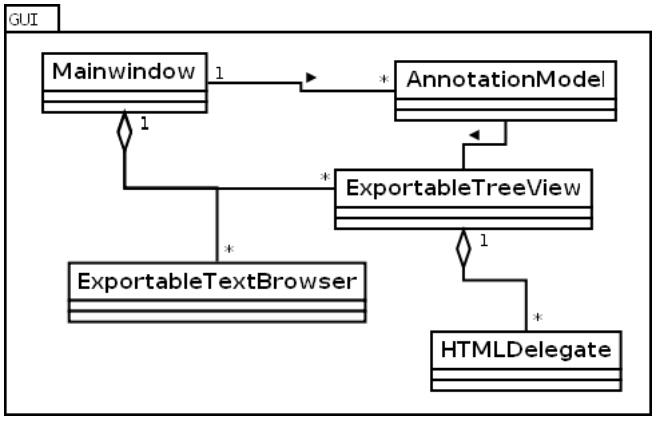


Figure 3.4: Relations between Mainwindow and the other classes involved in making tabs.

Gene annotation data tabs are constructed by the *AnnotationModel* class. This class is responsible for receiving annotation data from the controller and subsequently creating a *QWidget* that is added to the *QTabWidget* in the *Mainwindow*. A tree view is used to display annotation data, because annotation data can be substantially large in size. The view shows the corresponding gene-name along with its position and the names of the included databases when opened. The user can expand a database node in the tree view in order to show the data from extracted from

the corresponding database.

The tree view is implemented with an *ExportableTreeView*. This is simply a subclass of a *QTreeView*, but with an added function renderContents(). This makes it possible to export the contents of the tree view to plain text, so it can be used in the rendering of documentation. The tree view uses *HTMLDelegate*, so it can display HTML content. The *HTMLDelegate* adds several *QTextDocuments* to the nodes of the tree view for this purpose.

Metadata tabs are created with an *ExportableTextBrowser*. This is a subclass of a *QTextbrowser*, which also features a function renderContents(), once again to support the rendering of documentation.

3.2.4 CNVBrowser

The purpose of the CNV browser is to provide the user with a visual representation of aCGH profile and analysis data. It also shows genes found in significant regions and enables the user to explore the data set visually.

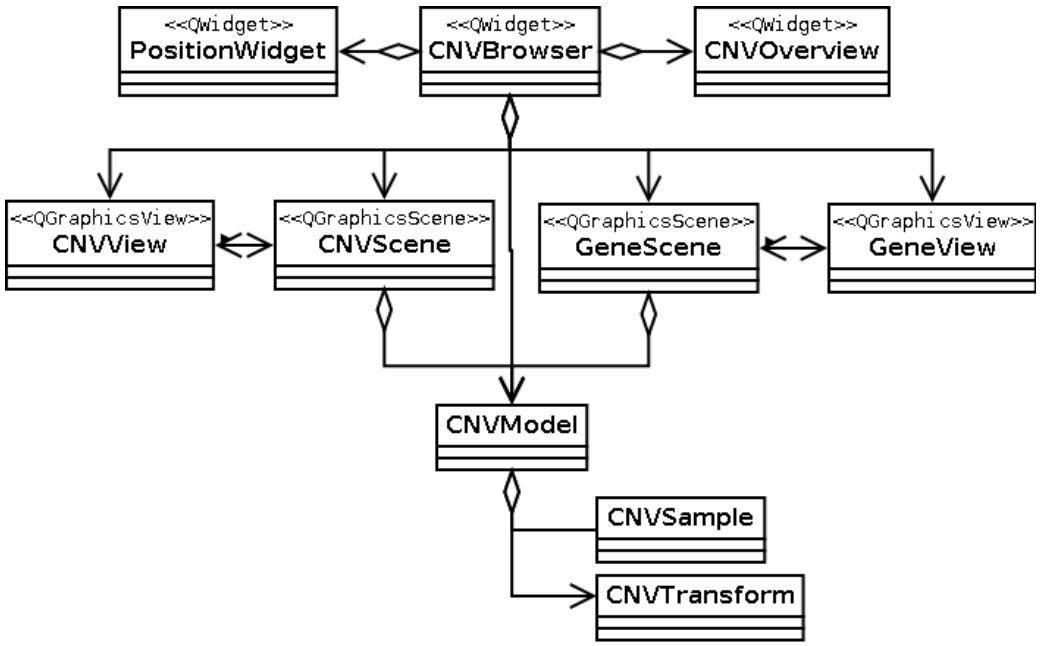


Figure 3.5: The *CNVBrowser* maintains a model that is visualised by the views

As illustrated by figure 3.5 the *CNVBrowser* consists of a *CNVview* and a *GeneView*. These views are backed by *CNVScene* and *GeneScene* respectively, which are containers that manage the graphical items shown in the views. The views render the contents of the scenes using the Qt Graphics framework and allow the user to manipulate the content of the views. The actual sample and gene data are separately managed by the model.
The purpose of the *CNVView* is to visualise the CNV data and allow the user to change which data points are visible. Because a typical CNV data sets consist of 100.000 to 200.000 points

several measures had to be taken to ensure that the view remains responsive and draws the data fluently.
The data is stored in a *CNVModel* which communicates changes to the scenes. The CNVTransform describes how data points are mapped into scene coordinates, allowing the data to be scaled in view coordinates. Additionally the model also provides a convenient reference to the *Reference Genome* and *CNVOutline*.

Level of detail A CNV data sets can consist of some 100.000 to 200.000 data points. As such it is both impractical and unnecessary to render all data points individually unless zoomed in properly.

This is shown by a simple example with a mere 10.000 points, as in this case a screen with a horizontal resolution of 1024 pixels would need to render about (100.000 / 1024) 100 points on each pixel. Considering that rendering 100 points on a single pixel would provide no useful information to the user, it is better to hide these data points until a sufficient level of detail can be observed.

Two methods were devised to draw a representative view of the entire data set without having to draw all the points in the data set: clustering data points that overlap into a smaller number of clusters or drawing an outline of the data set.

Clustering The general idea of clustering data is that when many data points overlap, only a few need to be drawn to give the same impression. Once sufficiently zoomed in, the clusters can be replaced with the individual data points.
For example, assuming that given 100.000 data points are displayed on a screen of 1000 pixels wide and 100 pixels high using dots of 10 pixels, between 10 and 1000 data points will overlap. Considering that most data points lay close to zero-line, the expected overlap was between 500 and 1000 points.

To cluster data the Quality Threshold clustering [3] algorithm was considered but it did not suffice as its run time on a data set of 100.000 points fell well outside the maximum allowable time. Instead a horizon clustering algorithm was written which is described in 3.5.3.

The horizon cluster algorithm did not prove to be satisfactory after its implementation though. While it was faster than standard Qt clustering, it was still too slow. The clustering also resulted in graphical distortion, which was caused by the graphics window allowing the user to zoom on only the x-axis, resulting in clusters being enlarged horizontally but not vertically. As such this approach was abandoned all together.

Outline When observing the complete data set there are two distinct regions: the region that contains the bulk of the data points, forming a dense region around the zero axis and outliers that lay outside this region. To speed up drawing, the region that contains the bulk of the data points can be approximated by a polygon, while drawing the outliers individually. Once zoomed in sufficiently the polygon can be replaced with the original data points.

The algorithm used to render the data points is described in 3.5.2.

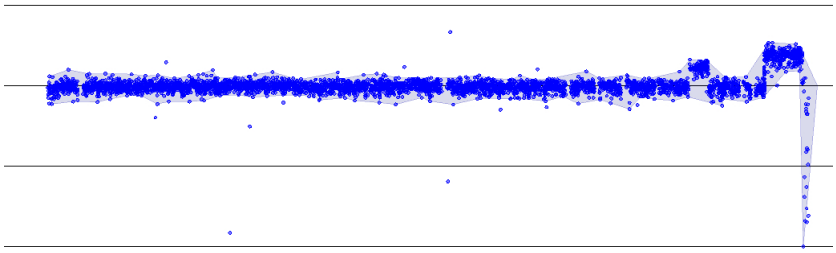


Figure 3.6: Plot of all 10.000 data points from a small set. Most individual points already can't be distinguished.

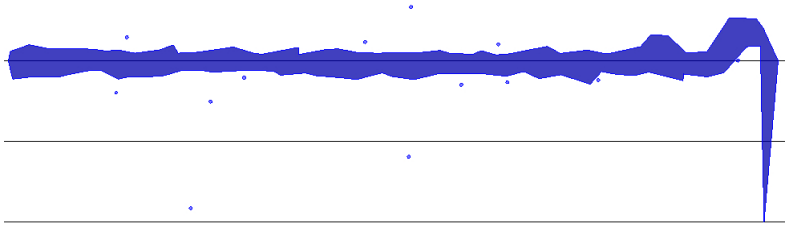


Figure 3.7: Plot of the outline of a a small set with 10.000 points. The outline approximates the data set.

Gradual loading of data Due to the size of the dataset it is only possible to load a fraction of all data points without losing responsiveness from the GUI. To ensure that this does not happen, small amounts of data are loaded in short intervals and added to the view.

Fading When dealing with level of detail transitions, it is important that the transition is smooth. An abrupt transition between levels of detail distracts the user. To ensure this does not happen, all elements fade in and out slowly during such a transition.

3.2.5 CNV Overview

The application provides an overview widget to help the user navigate the genome using the *CNVBrowser* view. This widget displays the entire CGH profile, along with a rectangle which represents the current view of the user in the *CNVBrowser* view, thus providing an easy view of where the user is currently located in the CGH profile. It also allows the user to manipulate the contents of the *CNVBrowser* view through various mouse interactions.

The following sections describe the structure and implementation of the overview widget. The first section describes the *CNVOverview* class itself, whilst the second details a pixmap class used to draw the CGH profile efficiently within the overview view. The third section explains the implementation of the *OverviewPointTransform* class, a transformation class used to translate screen coordinates to base pair positions and vice versa. On overview of the overview widget structure is shown in 3.8.

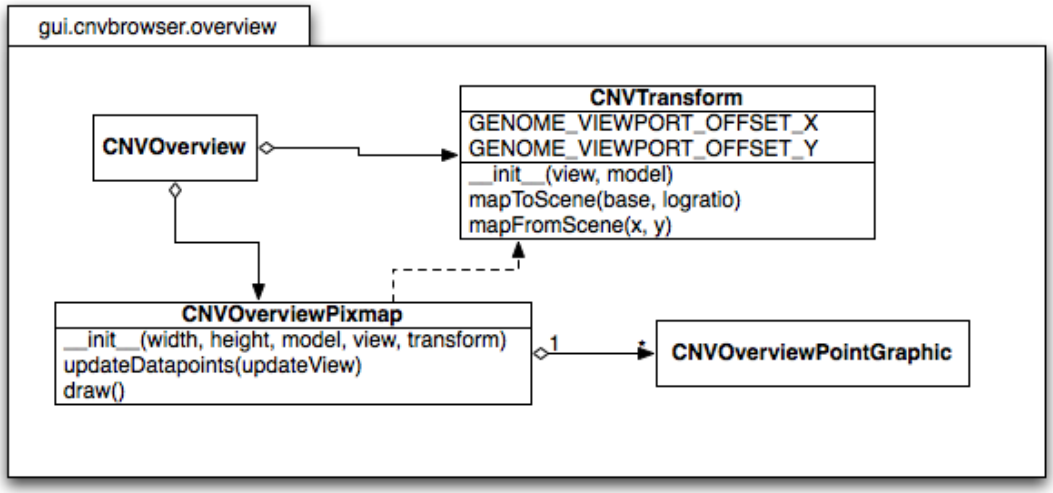


Figure 3.8: The basic structure of the overview widget design.

3.2.5.1 CNVOverview class

The *CNVOverview* class is essentially a *QGraphicsView* subclass and is responsible for handling the drawing of the overview widget and responding to the various events received by the view. The class maintains three graphical objects: a line representing the central axis of the view, a rectangle which denotes the current view of the *CNVBrowser* view and a pixel map item which displays the CGH profile in the background of the view. The overview manipulates these items as required to ensure an up-to-date view, depending on received external signals.

The axis line is added on construction of the widget, as the axis does not change during the lifetime of the application. The view rectangle is updated constantly in response to external signals, either mouse events or signals sent by the CNVBrowser view, in order to keep its position up to date with the current view.

An interesting note is that the view rectangle coordinates are not handled in screen coordinates, but in base pair positions. The benefit of this is that it allows the view rectangles position and the currently viewed region of the CGH profile to be handled in a single unit type. This allows each external signal to simply provide or calculate the left-most and right-most shown base pair positions and allow the view to automatically update itself correspondingly. The OverviewPoint-Transform class (see section 3.2.5.3) provides methods to allow easy convert screen coordinates to base pairs and vice versa.

The class provides the following manners in which the user can manipulate the *CNVBrowser* view through interaction with the *CNVOverview* view: rubber-band selection, dragging the view rectangle, clicking on a position within the CGH profile and scrolling within the overview.

Rubber-band selection Rubber-band selection allows the user to select a region of the CGH profile by right clicking within the overview and dragging the cursor to create a selection area within the overview. When the right mouse button is released, the selected area is set as the new view for the CNVBrowser and the view rectangle is updated correspondingly.

Dragging the view rectangle Dragging the view rectangle allows the user to scroll the current view along the CGH profile horizontally. Note that the overview limits the range of the view rectangles motion, as the view rectangle is not allowed to leave the CGH profile area.

Clicking on the CGH profile Clicking on a position within the CGH profile repositions the current view to horizontally centre on the clicked point in the CGH profile. This allows the user to quickly reposition the current CNV browser view to a specific point on the CGH profile.

Scrolling The overview also provides scroll functionality, which responds to mouse scroll events and either enlarges the width of the view rectangle or decreases it width depending on the scroll direction of the user, adjusting the *CNVBrowser* view accordingly.

3.2.5.2 CNVOverviewPixmap

The *CNVOverviewPixmap* class is a subclass of the Qt *QPixmap* class, which is essentially a container for image data in a pixel map format. The class is responsible for drawing a static image of the CGH profile which fits within the CNVOverviews viewport dimensions. Having a static image to display the data points gives the CNVOverview a great performance enhancement as the overview only needs to draw the corresponding data points once, instead of having to redraw the data points repeatedly.

In order to draw the CGH profile correctly, the *CNVOverviewPixmap* class is provided with a number of data points and a OverviewPointTransform instance, from which it calculates the correct position of each data point within its image dimensions. It creates a corresponding graphic point for each data point with the calculated new coordinates and proceeds to render an image containing these points once, only requiring a redraw on a resize event of the overview or when the CGH profile changes.

Due to the fact that the class draws all data points within its overview to ensure the highest level of detail possible, the initial draw and redraw of the image can take seconds to complete, depending on the size of the loaded CGH profile. This performance issue could be addressed in future releases by simply scaling the image in certain circumstances and only redrawing the image when it is absolutely necessary.

3.2.5.3 OverviewPointTransform class

The *OverviewPointTransform* class provides two important utility methods for the two classes mentioned in the previous two sections, the *mapToScene* method and the *mapFromScene* method. The class is provided a viewport and a reference genome upon construction, for which it calculates the various scale values used internally by these two methods.

The *mapToScene* method accepts a base pair position and a log ratio value and returns the corresponding coordinates in screen space, which are then subsequently used for drawing purposes

and event handling as explained in the previous section.

The *mapFromScene* method is the inverse of the *mapToScene* method, as its name implies. This method accepts a x- and a y-coordinate in screen space and translates the coordinate pair to the corresponding base pair position and log ratio value. The method is mainly used by the various event handlers, which require mouse event coordinates to be translated to a base pair representation.

3.2.6 Position widget

The *PositionWidget* is a simple Qt *Widget* that listens to the various signals emitted by the *CNVBrowser* and displays various data about the current view of the *CNVBrowser* view, such as the left-most and right-most base pairs currently in view as well as the currently visible chromosomes.

The widget also provides a panel containing various control inputs, which allow the user to select a chromosome and input a starting position and an end position in base pairs. After a subsequent click on the button input, the panel updates the CNVBrowser to display the CNVRegion specified specified by the chromosome name and position combination. The panel is only able to select a region on a single chromosome as the clients deemed it unnecessary to include functionality to select a region spanning multiple chromosomes in this fashion.

3.2.7 ListeningScrollView

As mentioned before both the *CNVView* and the *GeneView* enable the user to navigate the data. To keep this behaviour consistent across different views, the *ListeningScrollView* implements all functionality required to navigate the data.

Because the user has three different views and and changes to each view should be reflected by the other views, all views communicate changes to the area that should be visible via signals and slots. The ListeningScrollView provides *setVisibleRegion* and *transformVisibleRegion* slots that can be connected to *visibleRegionSet* and *visibleRegionTransformed* signals respectively from the *CNVBrowser*. Likewise to notify the *CNVBrowser* that the view has changed, the *ListeningScrollView* emits *visibleRegionSet* and *visibleRegionTransformed* signals.

The user can interact with the view in three ways:

- The data can be moved from side to side by dragging the view. This is done by holding down the left mouse button and moving the mouse.
- Zooming is done using the mouse wheel. When used the view either zooms in or out. Of special note is that the point in the scene under the mouse is kept under the mouse. This allows the user to point at a gene or data point and zoom in on this point without having to drag the view to keep this point in view.
- A specific area can be zoomed in on by dragging a box around the area that should be enlarged. This area is then fit into the view.

3.2.7.1 CNVView

The *CNVView* is a subclass of the *ListeningScrollView* that customises the drawing mode of the *ListeningScrollView* and ensures that the *CNVScene* has a reference to the *CNVView*.

3.2.7.2 CNVScene

The *CNVScene* is a container for *GraphicsItems*, specifically *CNVDataPoint*, *CNVOutline*. Due to the size of the data set some constraints had to be placed on the number of data points that could be placed in the scene at the same time and the number of points that could be added at a given interval.

The *CNVScene* limits the number of data points that can be visualised to *MAX_DATA_POINTS*. When these data points can't fill the view, a *CNVOutline* is presented instead. To keep track of which range should be visualised, the *loadDataInRange* slot is connected to the *cnvRegionchanged* signal. To ensure that not to many data points are loaded at once the scene loads *MAX_GROWTH* data points every *GROWTH_INTERVAL*.

Once the zoom has gone far enough to allow the data points to fill the screen from left to right, these are loaded and a flag is set to indicate to the graphics that outline can disappear as illustrated in 3.9 and 3.10.

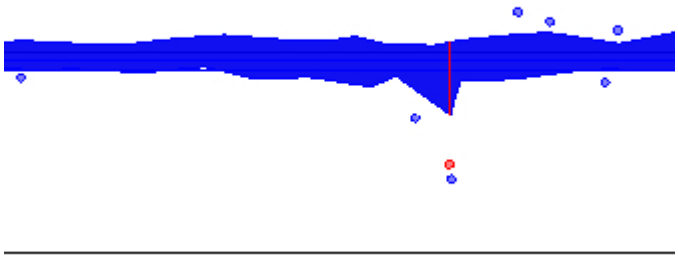


Figure 3.9: A zoomed in view just before the level of detail is sufficient to allow all data points to be shown.

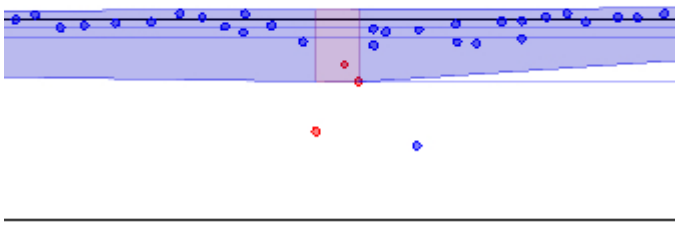


Figure 3.10: A zoomed in view just after the level of detail has become sufficient to show all data points.

3.2.7.3 GeneScene

The *GeneScene* handles the genes, the DGV data and the chromosome axis. When the genes are added to the scene, they are sorted by the strand they are on and their position on the genome. *GeneGraphics* containing *QRects* are subsequently created and drawn to visualise the genes. To make sure the genes don't overlap, different y-coordinates are set for the genes on each strand. Genes that still overlap though as they are on the same strand are displaced to avoid being drawn over each other.

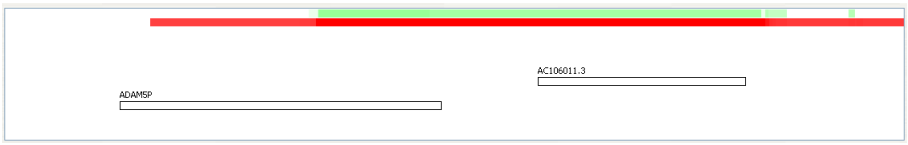


Figure 3.11: An example of the geneScene.

The *GeneGraphic* is used for visualisation of the genes. When the application view is zoomed in far enough, it also requests introns from the database and has them plotted. These introns are cached so they only have to be loaded once.

DGV contains the number of known variations in a specified region. In order to visualise all the DGV information in as little space as possible, two bars were added in the GeneView: a green bar for gains and red bar for losses. The colour-intensity of the bar indicates how many gains or losses have been observed. The colour-intensity is increased if two DGV-regions overlap.
The DGV data is only plotted when the program is zoomed in far enough, because the regions would be far to small when viewing the entire genome. Because the DGV regions are loaded from the database, they can now also be loaded on the fly. When the program is zoomed in far enough to show DGV data, the data is acquired from the database for the chromosome that is being viewed. This saves processing time and does not burden the database connection unnecessarily.

3.2.7.4 CNVModel

The *CNVModel* is a container that holds a *CNVSample*, *CNVOutline*, *CNVTransform*. Changes to any of these are communicated via signals to its observers, the *CNVScene*, *GeneScene* and *CNVOverview*.

3.2.7.5 CNVTransform

The *CNVTransform* class provides two utility methods for both scenes, the *mapToScene* method and the *mapFromScene* method. These map the position and log value of *CNVDatapoint* to scene coordinates.

3.3 Controller

The *Controller* is responsible for combining all the different components into one system. It must translate requests from the *Graphical User Interface* (GUI) to a request to the designated sub-

system. The *Controller* contains links to the *Data-Parsers* (3.3.2), *Algorithms* (3.3.3), *Database* (3.3.4), *Data-Processors* (3.3.5) and the *Job-Dispatcher* (3.3.6).

3.3.1 Overview

The *Controller* contains a number of functions that keep track of loaded samples, modified samples and visible samples at the *GUI*. It is the primary data storage facility. Furthermore it implements the *AbstractDatabaseInterfaceListener*, that contains a range of functions that must be implemented as database listener. It is crucial to offload requests as fast as possible and be ready for the next request. The *Job-Dispatcher* is used to offload requests, which is described in section 3.3.6.

Furthermore the *Controller* is designed to do as little as possible. It takes a request and hands it off to a specific subsystem. All requests to subsystems are made asynchronous to prevent blocking. When in a blocked state, the *Controller* cannot handle any requests from any subsystem. Most of the functionality is designed this way because of I/O waiting times or processes that take some time (for example, the loading of sample data). Therefore it must be ready to handle a callback as fast as possible to keep the system responsive. The longer a function or callback has to wait, the longer requests take, the slower the product feels. This was taken into account when the *Controller* was designed.

3.3.2 Data-Parser

Data-Parser functions are calls to the *Data-Parser* package. Each implemented function consists of three sub-functions. One is the external interface that can be called by another module (see 1 in figure 3.12). The second is the function that describes the steps that must be taken to complete the task (see 2 in figure 3.12). The third is the callback function (see 3 in figure 3.12). The second function is offloaded to the *Job-Dispatcher* using Python function pointers. This is done to ensure that the *Controller* blocks only as short as possible. The interface calls the middle function, the callback stores the result and calls the callback function that was specified when the interface sub-function was called.

Samples are automatically normalised after they are read (normalising is done by the processing subsystem). When normalising is finished the outline of the data is calculated and stored inside the sample. Afterwards the data is ready to be displayed on screen. This is a series of offloaded functions from which the results are stored inside the *Controller*.

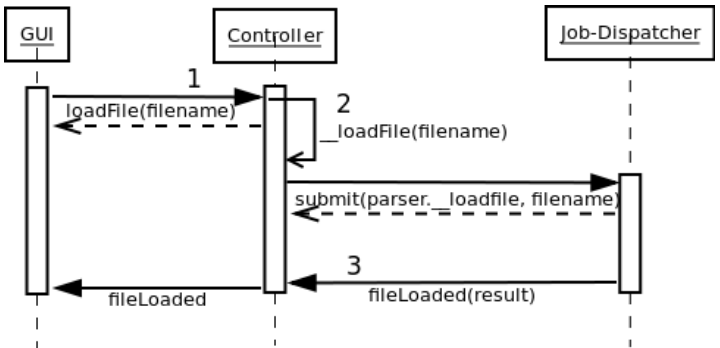


Figure 3.12: A *Data-Parser* request is offloaded to the *Job-Dispatcher*

3.3.3 Algorithm

Algorithm functions are build up in the same manner as data-parser functions. Each consists of three sub-functions, an external interface, a function that describes the steps needed to complete the task and a callback. The middle is again offloaded to the *Job-Dispatcher* to prevent blocking of the *Controller*. The callback sub-function stores the result and executes the callback function that was specified when the interface sub-function was called.

3.3.4 Database

Requests send to the *Database* are direct. The *Database* package itself is setup to be asynchronous to prevent blocking, therefore there is no need to add another level of abstraction. Each database function consists of two sub-functions. One provides the interface to other subsystems. The other is the callback function that the database calls when the query is done. The *Database* package provides functionality that notifies the listeners when an exception has occurred. This *exceptionOccurred* function is specified in the *AbstractDatabaseInterfaceListener*.

There is one exception. The function *getGenesAtRange* takes a list of regions. It makes all the calls to the database and counts the number of replies received. When all are in the callback function is called to notify the listener.

3.3.5 Data-Processor

The *Data-Processor* contains various calculation functions. Again each supplied function at the interface consists of three sub-functions. Results are stored inside the *Controller*.

3.3.6 Job-Dispatcher

To be able to offload functions and process them inside their own thread, the *Job-Dispatcher* was created. Python makes it possible to hand a function pointer, arguments and a callback function pointer to a thread and let that thread calculate the result. The *Job-Dispatcher* contains a *Threadpool* that starts out with a fixed number of *Workerthreads*. When the load increases a number of workers is automatically added. Each worker is supplied with a job and reports the to the *Threadpool*. It doesn't matter if the result is valid, or an exception occurred. There is always a result.

The *JobManager* is the interface to the rest of the product. It accepts a job and hands it off to the *Threadpool*. Each job receives a unique identifier which is used to track the job. The *JobManager* actively polls for jobs that are done. When such a job is found the callback function is called to notify the caller that the job has finished. In the case of an exception a special exception callback is called. This way, the caller can act to the events that have happened. A request for data from the database can be repeated for example or a message can be send to the user.

3.4 Algorithm

The algorithm subsystem of the application contains and manages implementations of a number of algorithms that are run on CNVSample instances in order to identify significantly aberrant genomic regions in the corresponding subjects genome. This section describes the design and implementation of the subsystem by first mentioning the model classes defined by the subsystem

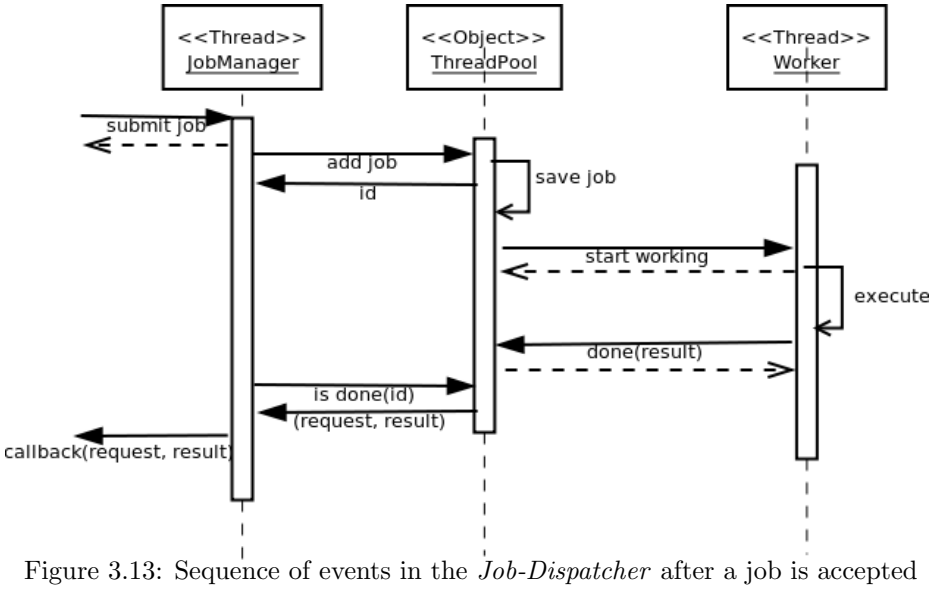


Figure 3.13: Sequence of events in the *Job-Dispatcher* after a job is accepted

to store data, followed by an explanation of the actual algorithm structure and the interface provided by the subsystem. An overview of the algorithm subsystem implementation is shown in the class diagram below.

3.4.1 Algorithm model classes

The main function of the algorithm subsystem is determining the significantly aberrant regions of a subjects genome according to the supplied sample data. The *SignificantCNVRegion* model class is provided by the subsystem to identify such a region. The class references a ChromosomeRegion class which identifies the chromosome region that the identified region spans. It also contains a type field, which specifies whether the region represents a loss or a gain of genomic material.

The output of each algorithm within the algorithm subsystem is a list of *SignificantCNVRegion* instances, which represent the regions identified by the algorithm. An algorithm can also provide and return a subclass of the *SignificantCNVRegion* class in order to return any extra data concerning the identified regions.

3.4.2 Algorithm implementations

The algorithm subsystem contains a number of algorithm implementations, each extending from the base Algorithm class. The base class provides a basic implementation of functionality deemed to be required by every algorithm implementation, thus providing a solid base for actual algorithm implementations. This section first discusses the implementation of the base class, followed by a description of the specific algorithm implementations currently provided in the application and their implementation.

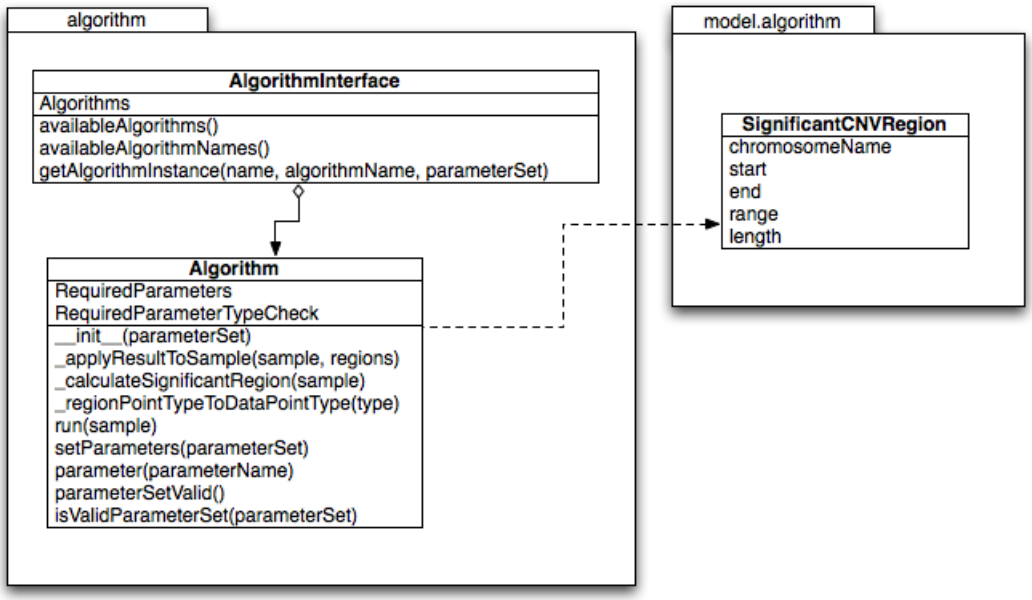


Figure 3.14: The basic class structure of the Algorithm subsystem.

3.4.2.1 Base Algorithm

The base *Algorithm* class provides the basic functionality for an algorithm but lacks an actual implementation of the *calculateSignificantRegion* method, which is called upon to calculate the significant regions for a given sample. The Algorithm class can be supplied with a parameter set containing various parameters which are used by the algorithm during its execution.

An Algorithm can specify required fields by including their parameter names in the *RequiredParameters* field. The class also includes a *RequiredParameterCheck* field, which consists of function map implemented as a dictionary, that allows an *Algorithm* to specify functions to validate the supplied parameters. The default behaviour of the class is to check a supplied parameter set using these two fields by calling the *parameterSetValid* method before executing its calculation.

The *run* method of the class effectively executes the algorithm and is called by external code, passing in the sample on which the algorithm must be run as an argument. The method first checks the current parameter set in the manner described above and then calls the *calculateSignificantRegion* method, which is supplied the given sample and returns a list of significant regions represented by instances of the *SignificantCNVRegion* class. This list of regions is used by the *run* method to flag the CNV data points in the sample data set as gains or losses, depending on the type of the identified region, and is subsequently returned as the result of the method.

Note that the *calculateSignificantRegion* method is responsible for actually calculating the significant regions and is therefore the method that is overridden by subclasses to perform their specific calculation.

3.4.2.2 KC-smart algorithm

The KC-smart algorithm implementation uses an R implementation of the KC-smart algorithm to perform its calculation. A short description of this algorithm is provided below:

> KC-SMART is a method for finding recurrent gains and losses from a set of tumour samples measured on an array Comparative Genome Hybridisation (aCGH) platform. Instead of single tumour aberration calling, and subsequent minimal common region definition, KC-SMART takes an approach based on the continuous raw log2 values. KC-smart Gaussian locally weighted regression to the summed totals positive and negative log2 ratios separately. This result is corrected for uneven probe spacing as present on the given array. Peaks in the resulting normalised KC score are tested against a randomly permuted background. Regions of significant recurrent gains and losses are determined using the resulting, multiple testing corrected, significance threshold. [6]

The implementation of the algorithm uses the RPY2 python package to interface directly with an R implementation of KC-smart. The current implementation requires a functioning installation of the KC-smart R implementation which can be installed using the bioconductor software package. It also requires the DNAcopy library to be installed, which can also be installed using bioconductor, as this package contains the data format used to supply data to the KC-smart algorithm.

When the algorithm is run, it instructs R to load the required libraries and translates the passed sample into its DNAcopy equivalent form. The resulting DNAcopy object is used in a series of calls to the KC-smart R implementation, which results in a list of significant regions identified by the KC-smart algorithm. This result is extracted from R and translated into an equivalent list of *SignificantCNVRegion* instances, which is then returned and processed in the manner described above.

The KC-smart algorithm requires the following parameters to be set: the significance cut-off value (also called the Target False Discovery Rate), the kernel width and the number of permutations the algorithm should use. For more information about these parameters and their values, please reference the KC-smart reference manual. [5]

3.4.2.3 KC-smart text file algorithm

Due to the fact that the KC-smart algorithm can take a substantial amount of time to complete when run with a large number iterations or on a large data set, an extra algorithm has been implemented that essentially loads and parses the text file output of a completed KC-smart session. The advantage of this method is that the KC-smart algorithm can be run wherever and whenever the user would like, requiring only its results to be supplied.

The algorithm uses the same parser/adaptor structure as the filesystem subsystem. It opens the file, whose path is supplied as a parameter to the algorithm, and parses its contents to the equivalent list of SignificantCNVRegion instances, which is then returned and processed in the manner described above.

3.4.2.4 Threshold algorithm

The threshold algorithm analyses a sample in order to determine groups of consecutive points on a chromosome that have an absolute log ratio larger than a given threshold. This algorithm has been implemented for the VUMC client as the client uses a similar workflow when identifying significant regions, by selecting groups of three or more consecutive data points with an absolute log ratio larger than a certain threshold.

The algorithm performs its function by iterating over the sorted collections of data points contained in the CNVSample instance. If the algorithm encounters a data point that has an absolute log ratio larger than the given threshold, the algorithm flags the point and continues its iteration. If the next point is also above the given threshold, it is also flagged. As soon as the algorithm encounters a point with a log ratio below a given threshold or with a value that has a different sign (positive/negative) than the previous data points, the list of flagged points is inspected.

If the number of flagged points is at least the number of points required, the list of points is converted to a significant region and the algorithm resets its list of flagged points and continues its iteration. If the number of flagged points is less than required, it simply resets the flagged list and continues iteration, as this is not a significant region with the given parameters. This sequence of events is repeated separately for each chromosome.

The algorithm takes two parameters, one specifying the threshold which is used to test the data points and one specifying the number of consecutive data points required for the region to be considered significant.

3.4.3 Subsystem interface

The interface of the algorithm subsystem is provided by the AlgorithmInterface class. This class aggregates the available algorithm implementations and provides methods with which code external to the subsystem can query the names of the available algorithms and retrieve an instance of a specific algorithm. In this manner external code can retrieve algorithm instances without having to reference the internal algorithm classes directly.

As each algorithm implements the same interface, each returned algorithm instance can be handled in the same manner and therefore doesn't require any algorithm specific code. The only exception to this uniformity is the difference in required parameters between the various algorithm implementations, however this problem is reduced by the fact that required parameters can also be inferred in runtime from the algorithm through the *RequiredParameter* fields.

3.5 Processing

The processing package of the application contains implementations of a number of algorithms that are run on *CNVSample* instances after these are loaded but before they are further used in the program. This includes normalisation of the data, and computing the outline. Additionally the depreciated Horizon Cluster Algorithm is described.

3.5.1 Normalisation

To counteract systematic shifts in the data, the data is first normalised using the standard score. The mu and sigma are calculated over the whole gene. Then for each data point the log value becomes x : ((x - mu) / sigma.

3.5.2 Outline Algorithm

The outline of a data set consists of a set of lines that tightly encloses the majority of the data points and thus provides a good low detail representation of the data set. This is used in the *CNVBrowser* to represent the bulk of the data points without drawing each individual point.

The outline is computed by dividing the data set into a number of bins and taking for each bin the maximum that falls within n standard deviations from the average of that bin.

The advantage of this method is that it provides a good outline using only a minimum of points to draw as can be seen in figures 3.7 and 3.6. However the outline itself is not as meaning full as other more accepted representations, such as a box plot.

3.5.3 Horizon Cluster Algorithm

The horizon cluster algorithm was written to cluster data quickly and does so in a single pass over a sorted set of data points. It does however not provide nice and stable clusters.

```
G, a set of datapoints  Dx,y sorted on by x ascending.
d, the size of a cluster in the x and y direction

begin  HorizonCluster(G, d)

    H a list of clusters that fall within the horizon
    A a list of all clusters

    for data point P in G:

        for each cluster C in H:
            if distance(C, P) > d:
                H = H \ C

        for each cluster C in H:
            if diameter(C + P) < d
                C = C + P
                continue

        C = Cluster(P)
        H = H + C
         A = A + C

    return A
```

Rather then comparing each data point against all clusters, the clusters that are too far away (eg. over the horizon) are no longer considered. This holds because the data points are assumed to be sorted on the x axis and not overlapping. As such any cluster that is to far away for a

datapoint D, will also be too far away for any datapoint E that comes after it.

The runtime is O(|G| * |H|), where in the worst case |H| equals |G|. So the worst runtime is O(|G| ^{2}). However with the given dataset no probe overlaps in the x direction and most probes are located between 1 and -1 on the y axis. As such |H| can be approximated by a constant C = 2 / d. As such the runtime will approximate O(|G| * C).

3.6 Report

In Angita it is possible to render documentation from the plot, the gene annotations and the sample metadata. In order to do so, we use ReportLab [4], an open source engine for creating PDF files. In the package *Report*, an interface for using ReportLab is defined. Using this interface, things like paragraphs, images and page breaks can be easily added from other parts of the program.

3.7 Circos

Circos [1] is an external Perl program that is made for visualisation of the genome and relations to the genome. This package is made as an interface to circos and is needed since ciscos is advanced program that need a lot of customisation to get wanted output.

The interface has only one class(*CircosInterface*) needs a full path(must be a existing path to save the output of circos), significant regions, picture name(only png extension is supported because, the SVG output of circos is not stable yet) and size. The other parameters are used for filtering the results(specified in the paragraph filter) With the given significant regions it searches for al genes in these regions. These genes are used by ibidas to search for interesting relations between genes (proteins). The relations can differ in mode and action, to make this visible by using different colours for mode, and different intensity for mode. The legend of those can be accessed through the two getters for the dictionaries. The other two functions are made for asynchronies callback from the controller for loading genes and relations

filter The genes and relations that are found can be filtered on three aspects:

- By setting the flag allGenes on true it will plot all genes that are found in the significant regions. Otherwise only the genes that have a not filtered relation are plotted.
- The two parameters filterMode and filterAction, are two list where in all relations that are wanted are specified. By default all relations are active.
- The threshold specifies a border for all relations with a score lower than the threshold are filtered out.

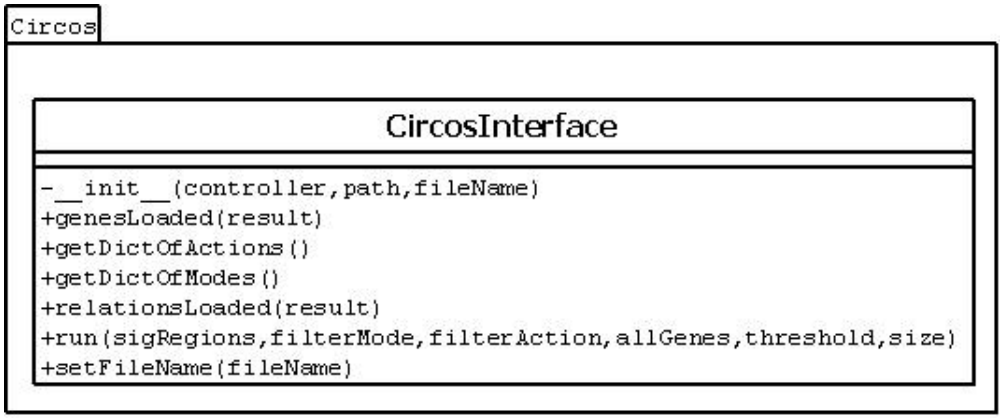


Figure 3.15: ClassDiagram circos

Chapter 4

Ibidas

4.1 Overview

Ibidas [8] is described as: "Ibidas is aimed to be an integration platform for various sorts of biological data. Its purpose is providing a very minimal but flexible database scheme. It is possible to store any kind of biological data while still keeping the learning curve for using the system low."

The large amounts of biological data stored in the various external databases must be kept available and accessible in order to supply the user with the various annotation data. The Ibidas system is a database system which provides the functionality required to store such data in a single database, allowing data to be queried in a uniform manner.

Normal databases present data in a flat 2D manner, however biological data requires a more flexible solution. Ibidas is able to maintain the data in the manner in which it is presented (potential multidimensional), instead of just combining a large number of 2D tables. Ibidas also keeps queries relatively simple as it does not require complex joins to couple the various data sources as a conventional database would.

Currently Ibidas only offers functionality to maintain the databases in memory. A startup script is created in order to have the data available when a client connects. A memory dump is created after a database is first parsed. Next time the databases are loaded through the dump loader offered by Ibidas. The data is parsed again when a dump fails to load.

A number of data parsers are constructed to be able to use data from different sources (as described in section 4.2).

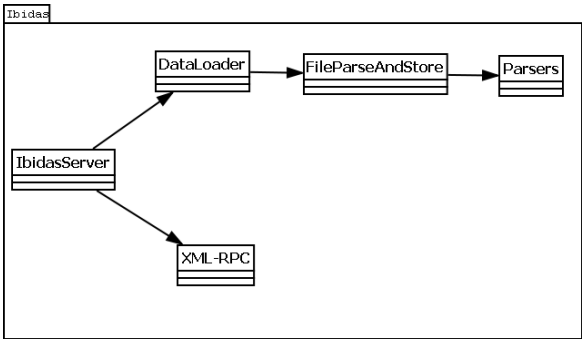


Figure 4.1: The global class structure of Ibidas.

4.2 Parsers

During the orientation phase of the project a number of databases of interest were defined. The layout of these databases is extensively researched in order to find links between them (see the System Design Document). Most of the databases use a tab separated column layout. Each different layout could be parsed, after the basic column parser was adapted. By means of inheritance large parts of the parsers could be reused.

There are two exceptions:

- **OMIM:** The OMIM extended file contains large blocks of text. Not all entries contain the same number of blocks and it is sometimes unclear when a section ends. Therefore a custom pre-parser was created that parses the extended file to a standard tab separated format. The pre-parsed output is then handed to the standard data parser.
- **STRING:** The STRING database is so vast that it cannot realistically be downloaded and loaded into the memory of the server. To be able to use data from STRING a number of filter queries have been executed to strip off unneeded fields. Afterwards the size of data collection was significantly trimmed and could be parsed. The STRING data is, just as all other databases, loaded into memory at the server. Due to speed problems the choice was made to keep the data in memory, instead of exporting it to SQL format.

The product offers information from the following databases:

- Ensembl (basic gene information)
- CGC (cancer gene datasource)
- OMIM (compilation of human genes and genetic phenotypes)
- DECIPHER (compilation of rare genetic phenotypes)
- DGV (compilation of known CNV variations)
- STRING (compilation of protein-protein interactions)
- UCSC (chromosome band information, among others)

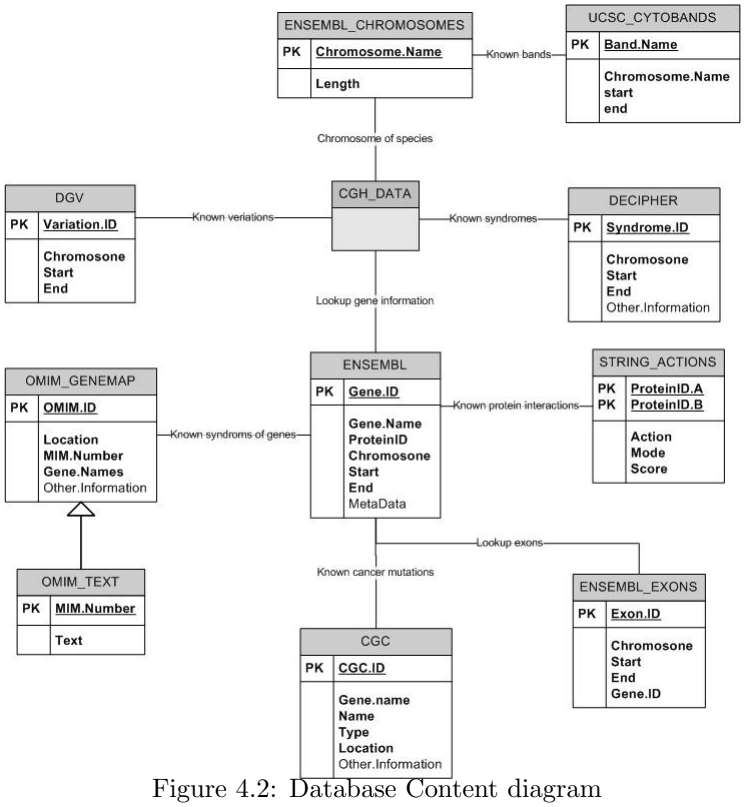


Figure 4.2: Database Content diagram

Ensembl builds v36 and v37 are both supported. When there is an option to either load v36 or v37 data the database function is fitted with a version selection parameter. The same goes for species. Human and Mouse are supported.
Due to unclear naming conventions and specifications in both Ensembl, OMIM and STRING no guarantees are made that all mappings between databases are proper. All conducted tests resulted in valid results, but clearly, not all combinations could be tested.
There are some restrictions to the databases in the system: DECIPHER is only one available for genome build v36 and STRING is at the moment only supported for Human proteins.

4.3 XML-RPC

Ibidas offers an XML-RPC connection, as well as a SOAP network interface. At first the product was designed as a client-only application. But this became unworkable as the load of Ibidas increased with the number of databases loaded. Therefore the choice was made to go from client-only to client-server. The XML-RPC implementation was chosen as it offers a clean and simple interface. In the long run, when the product evolves and the number of clients increases, it is probably worth while to change this to a threaded implementation with sockets. That way the server is able to handle more clients in less time.

4.3.1 Client-side

The client-side implementation offers a clean interface that implements exactly the same functions as are available server-sided. The subsystem consist of an XML-RPC client implementation and a database interface. The XML-RPC client basically offers a single function: Send a request for data to the database. It is responsible for handling security and exceptions. All database communication is synchronous since XML-RPC is only able to handle one request at a time from a single client. The XML-RPC client function offered to the product is implemented asynchronously make sure the database subsystem does not block. Internally a queue is used to temporarily store requests in.

The *Database-Interface* offers functionality to other components of the product. It implements the functions that the server-side offers. When a subsystem requests for data from the database the interface creates a new job containing the request. Using Pythons function pointers a request is send to the *Job-Dispatcher* (described in section 3.3.6). The dispatcher sends the request to the XML-RPC connector and awaits the result. When a reply is received the callback function at database interface is called.The database interface transforms the list of list into list of dictionaries. Dictionaries make it easier for other modules to handle the data. When data is turned into the right format, the listeners will be informed that the request is finished

A database listener (as the *Controller* is) must implement a number of functions specified in the *AbstractDatabaseInterfaceListener*. These functions are the callbacks that are called when a reply from the database is received. These functions also supply in a way of communication occurred exceptions. When an exception occurs during transmission it is caught and a special function is called to notify the listeners. Supplied with the request and the exception, a listener can act to either roll-back, notify the user or resend the request.

Due to security reasons (especially because of the confidential data) a password was introduced to make sure no unauthorised access to Ibidas is possible, this is done manually by assert statements and parameter of the functions since there was no real security options building by XML-RPC.

4.3.2 Server-side

The server-side implementation builds on the existing Ibidas XML-RPC service. Ibidas offers direct access to databases in memory through XML-RPC. Additional security measures were taken to prevent unauthorised access to the system (described in more detail in sections 4.1 and 4.3).

Chapter 5

Appendix

5.1 Model class diagrams

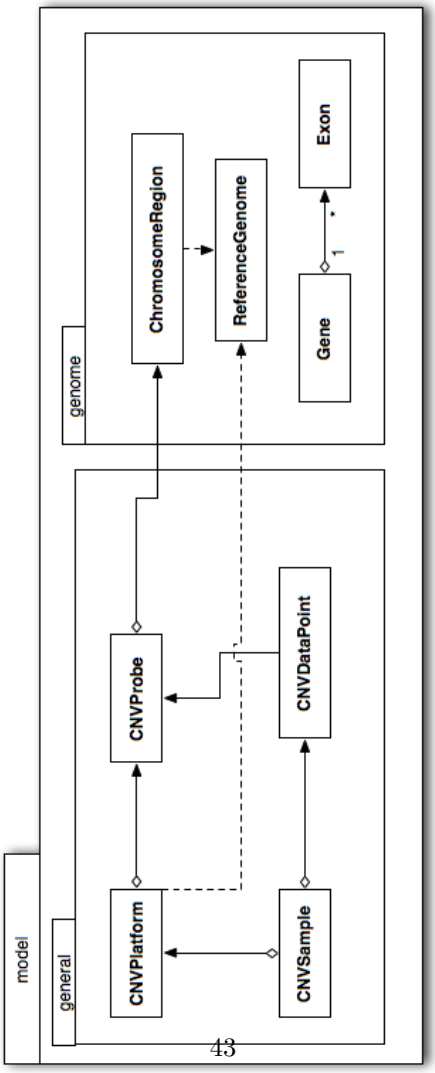


Figure 5.1: The basic class structure of the Model package.

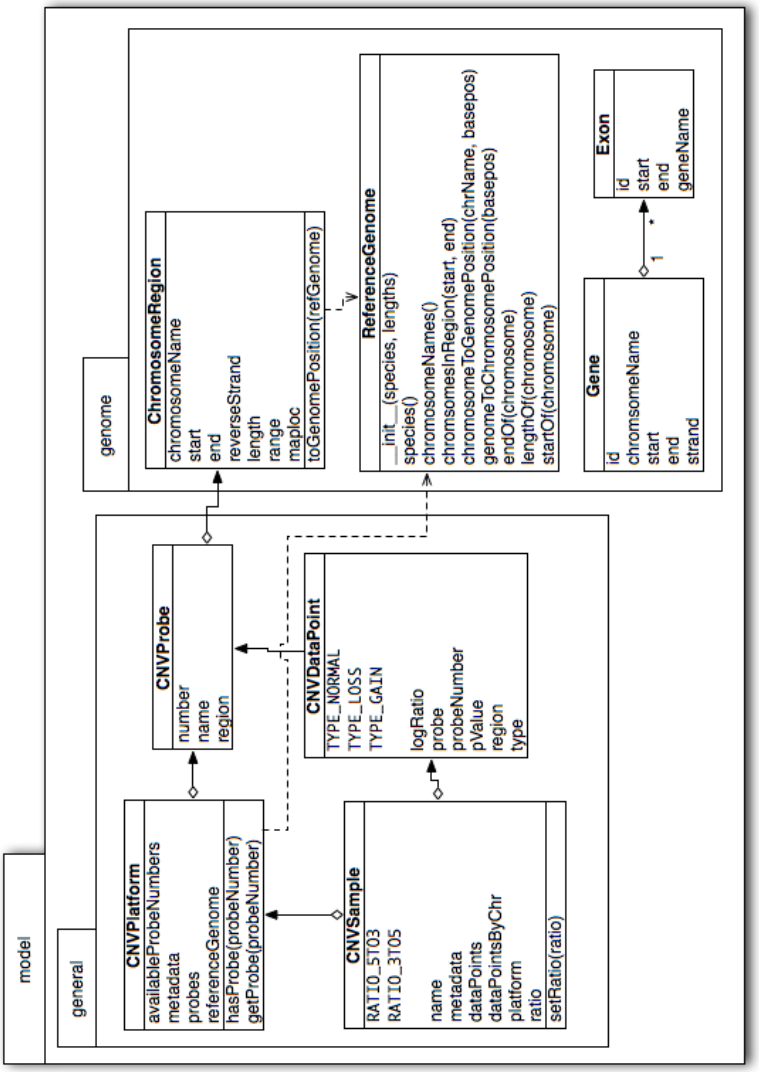


Figure 5.2: The fully specified class structure of the Model package.

5.2 Graphical user interface class diagrams

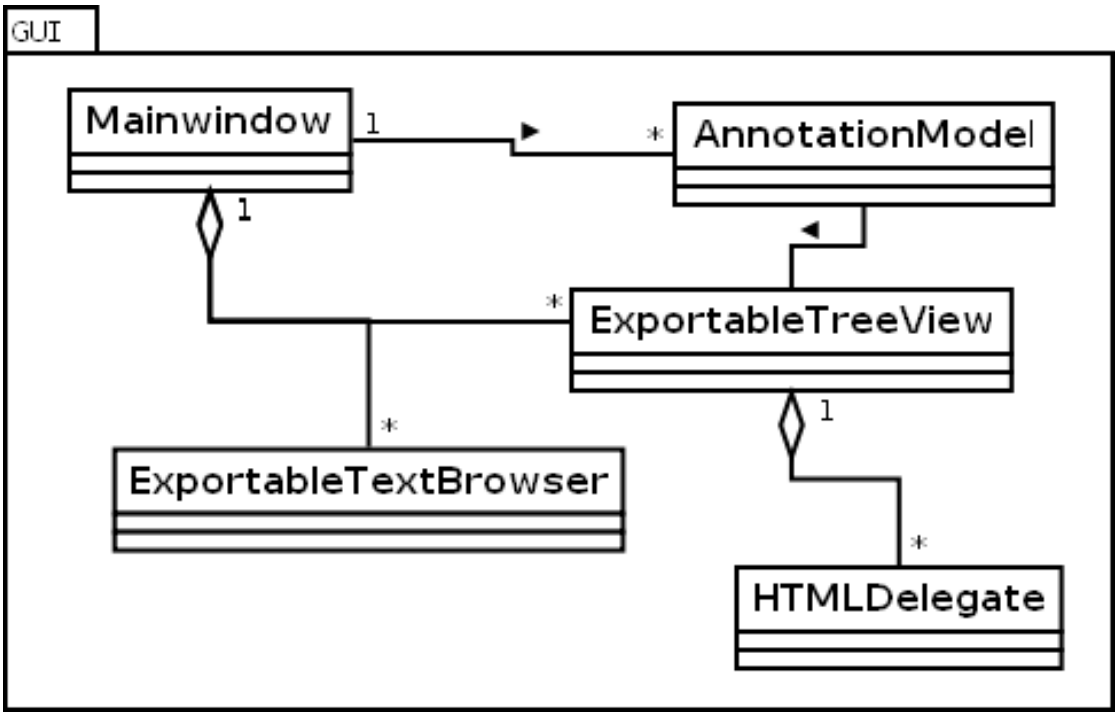


Figure 5.3: Relations between Mainwindow and the other classes involved in making tabs.

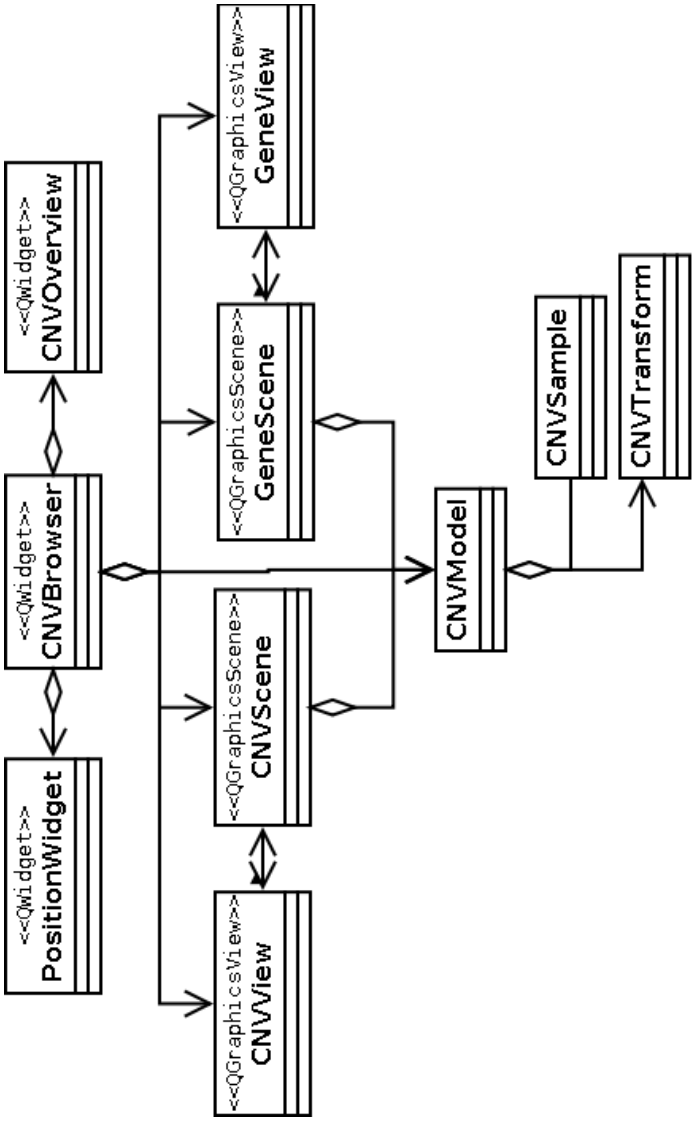


Figure 5.4: The *CNVBrowser* maintains a model that is visualised by the views

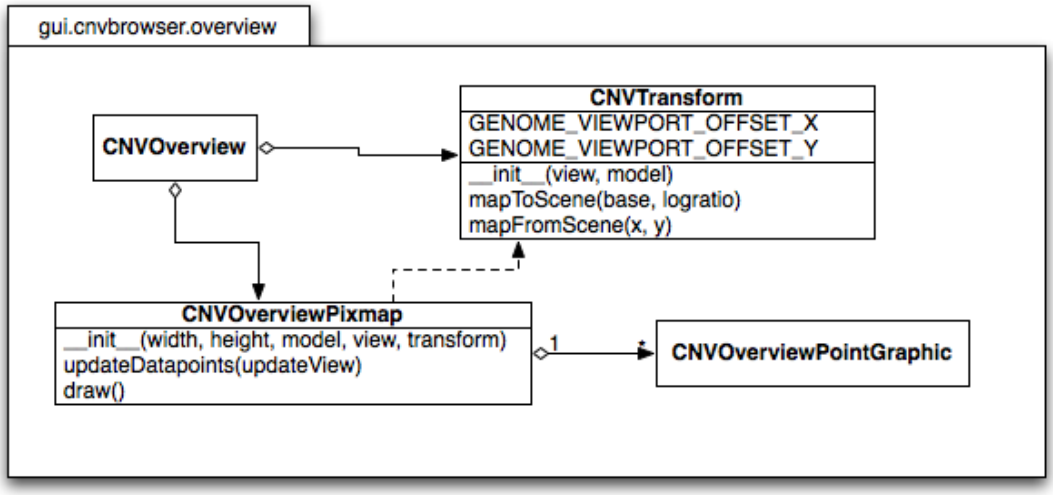


Figure 5.5: The basic structure of the overview widget design.

5.3 Filesystem class diagrams

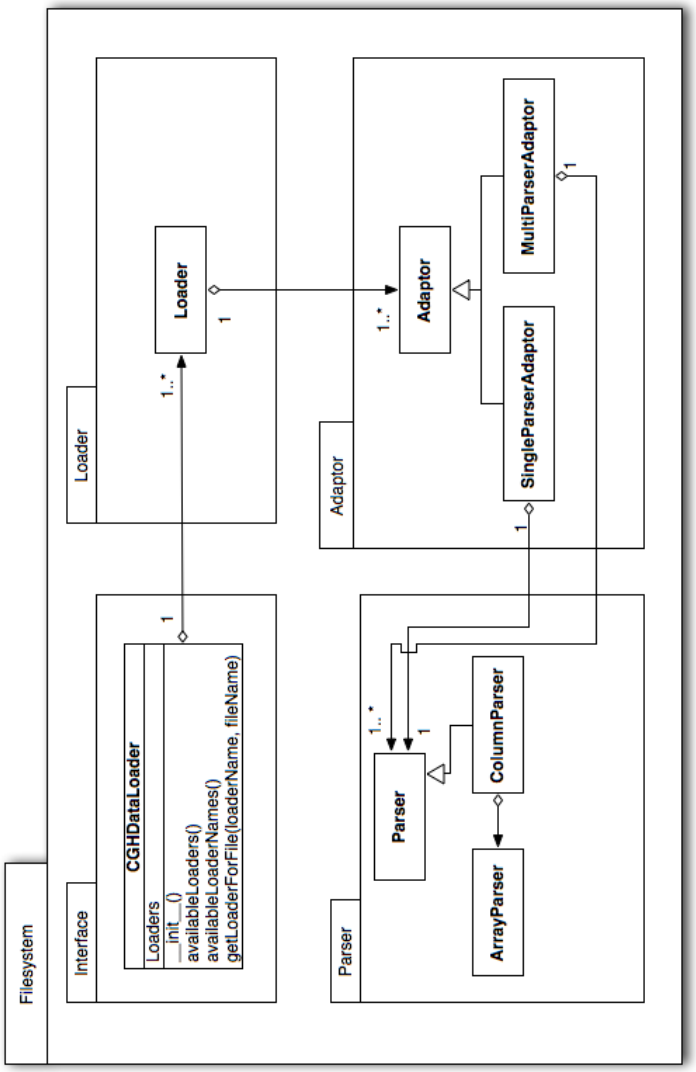


Figure 5.6: The basic class structure of the Filesystem subsystem.

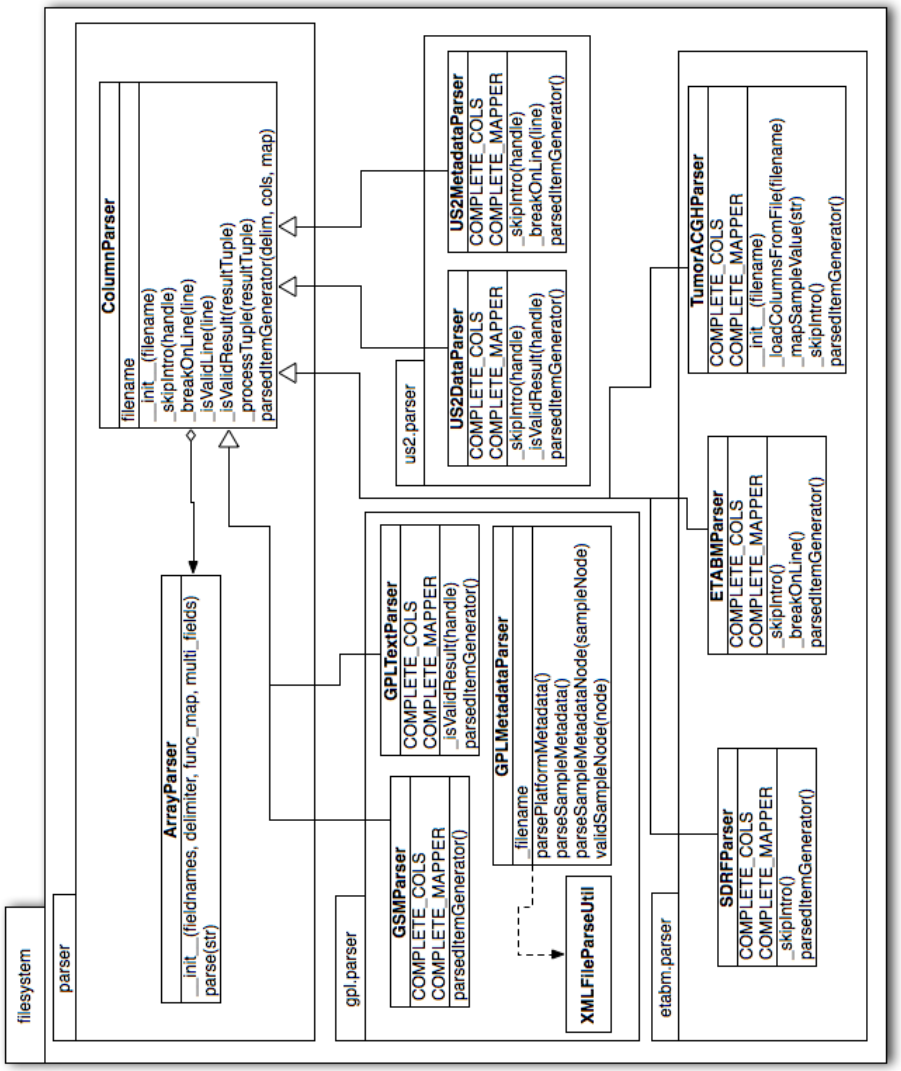


Figure 5.7: The class structure of the parsers in the Filesystem subsystem.

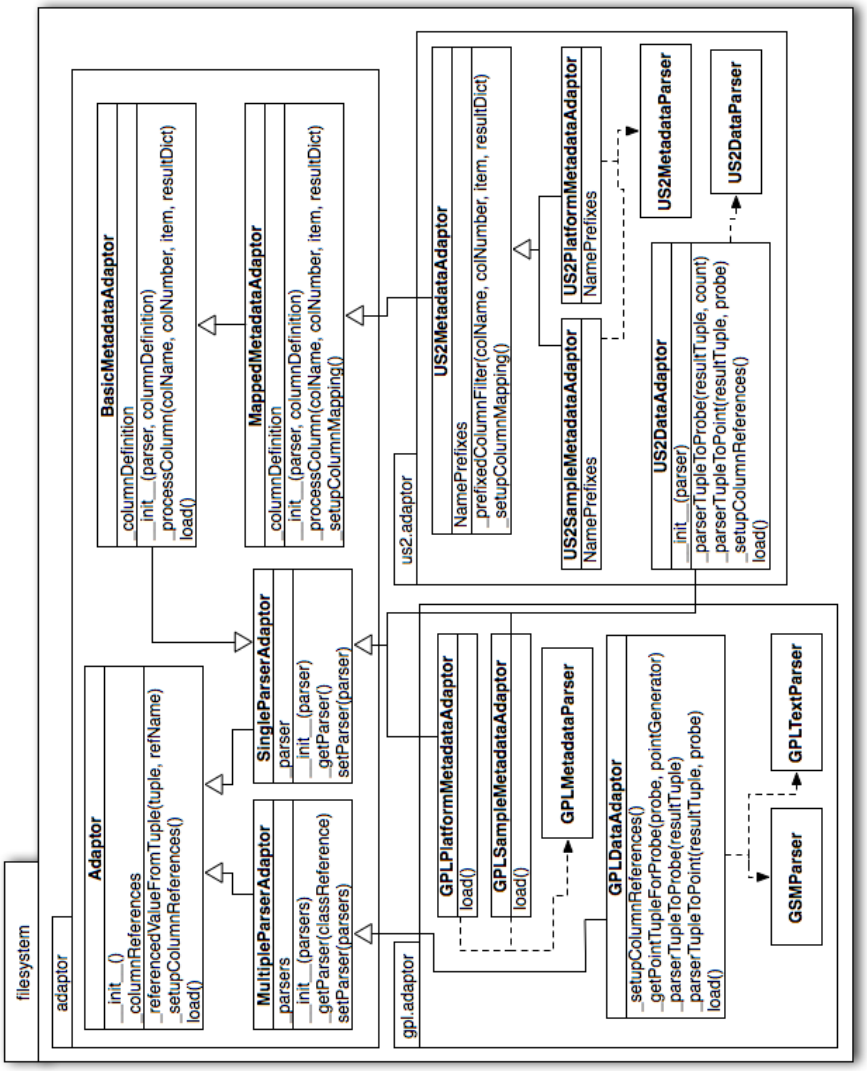


Figure 5.8: The class structure of the adaptors in the Filesystem subsystem.

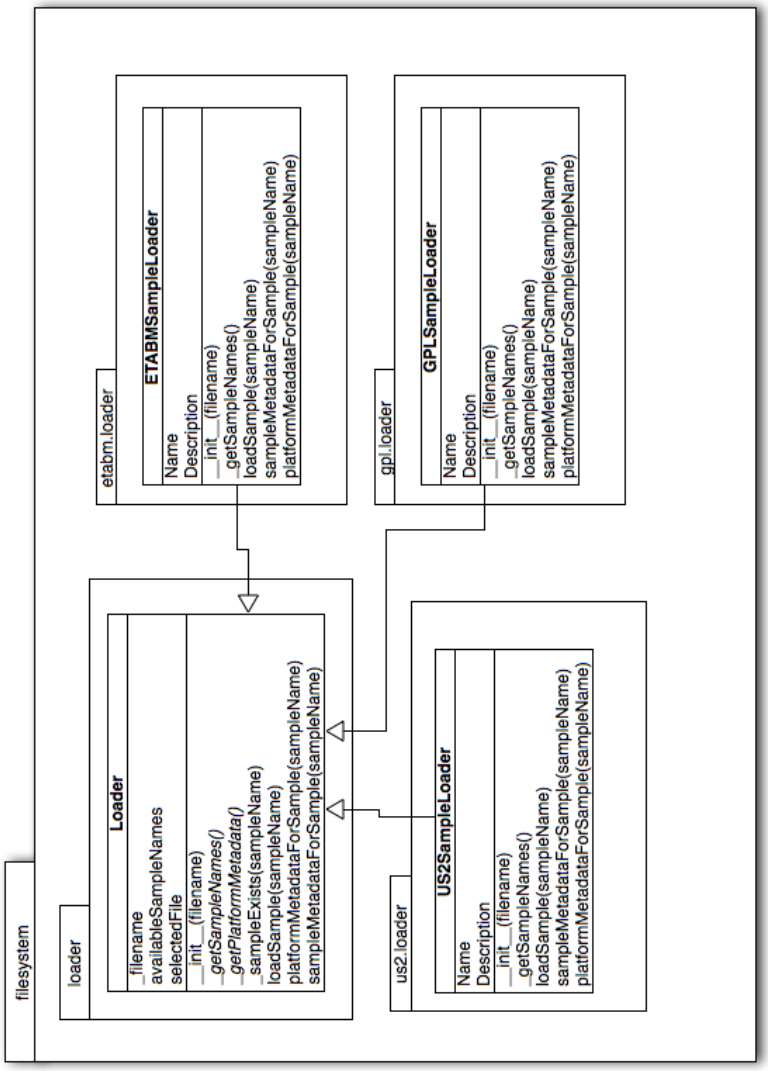


Figure 5.9: The class structure of the loaders in the Filesystem subsystem.

5.4 Algorithm class diagrams

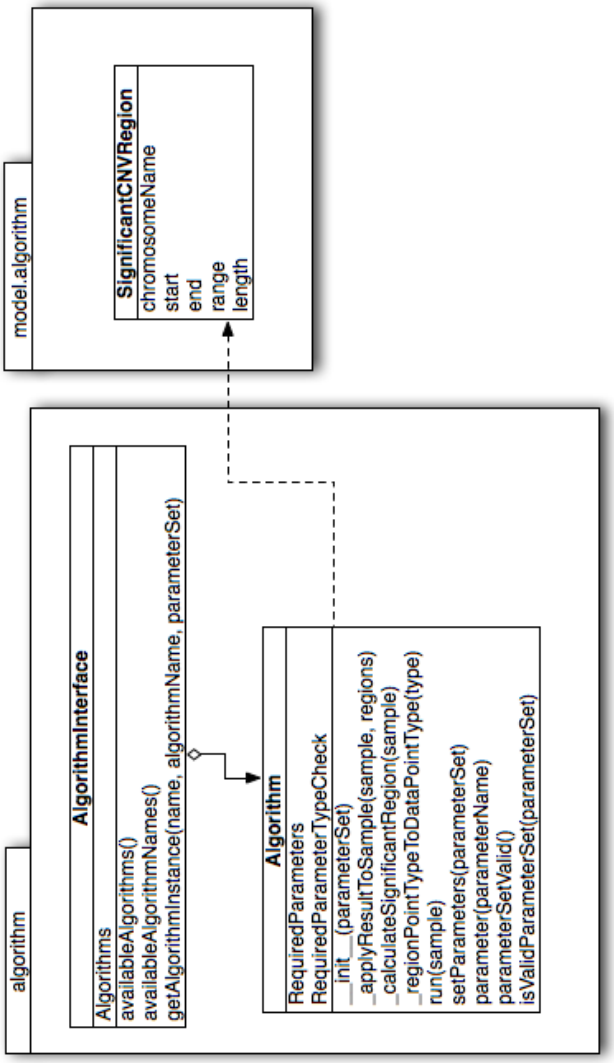


Figure 5.10: The basic class structure of the Algorithm subsystem.

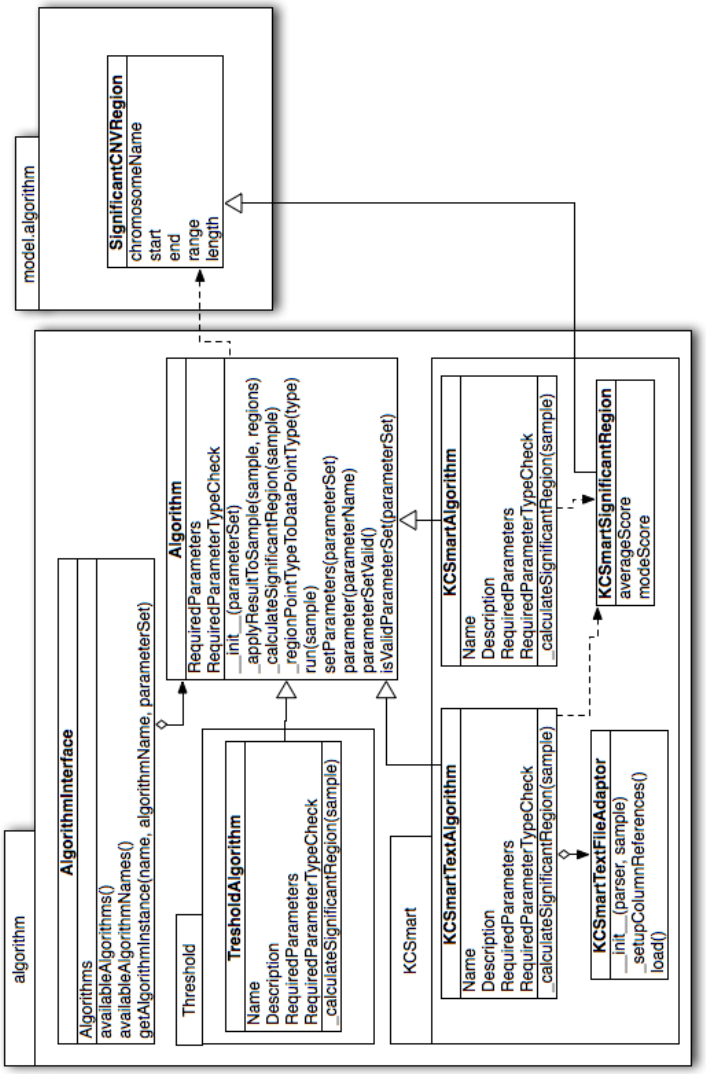


Figure 5.11: The fully specified class structure of the Algorithm subsystem.

5.5 Controller class diagrams

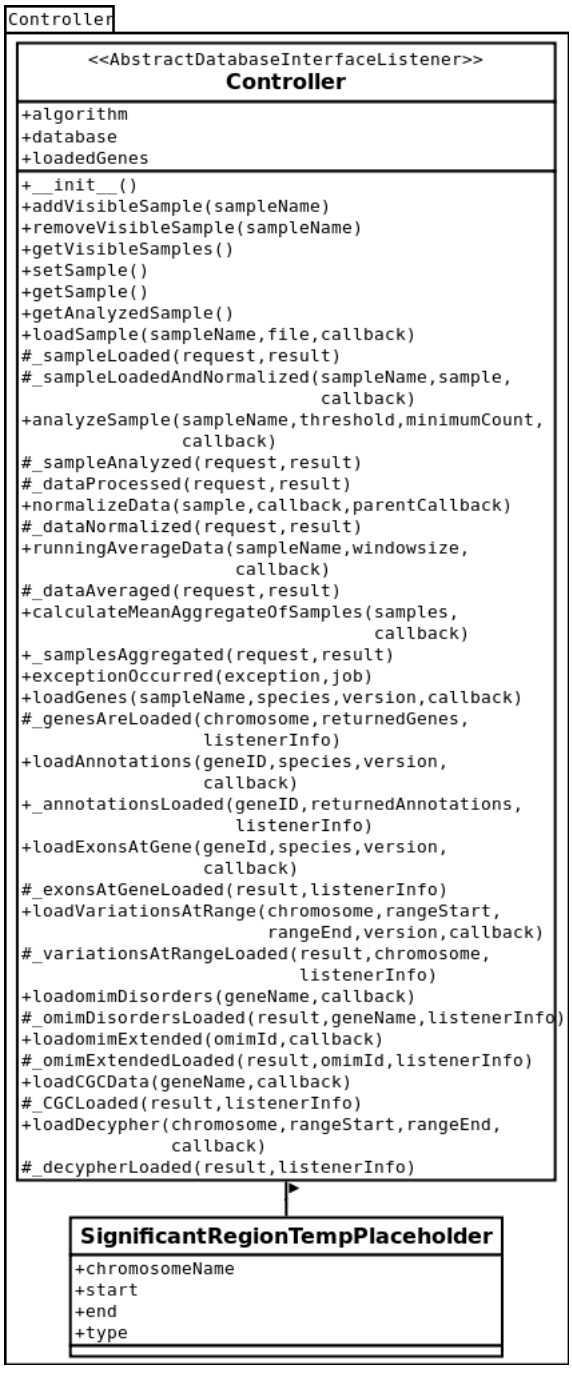


Figure 5.12: The class structure of the Controller.

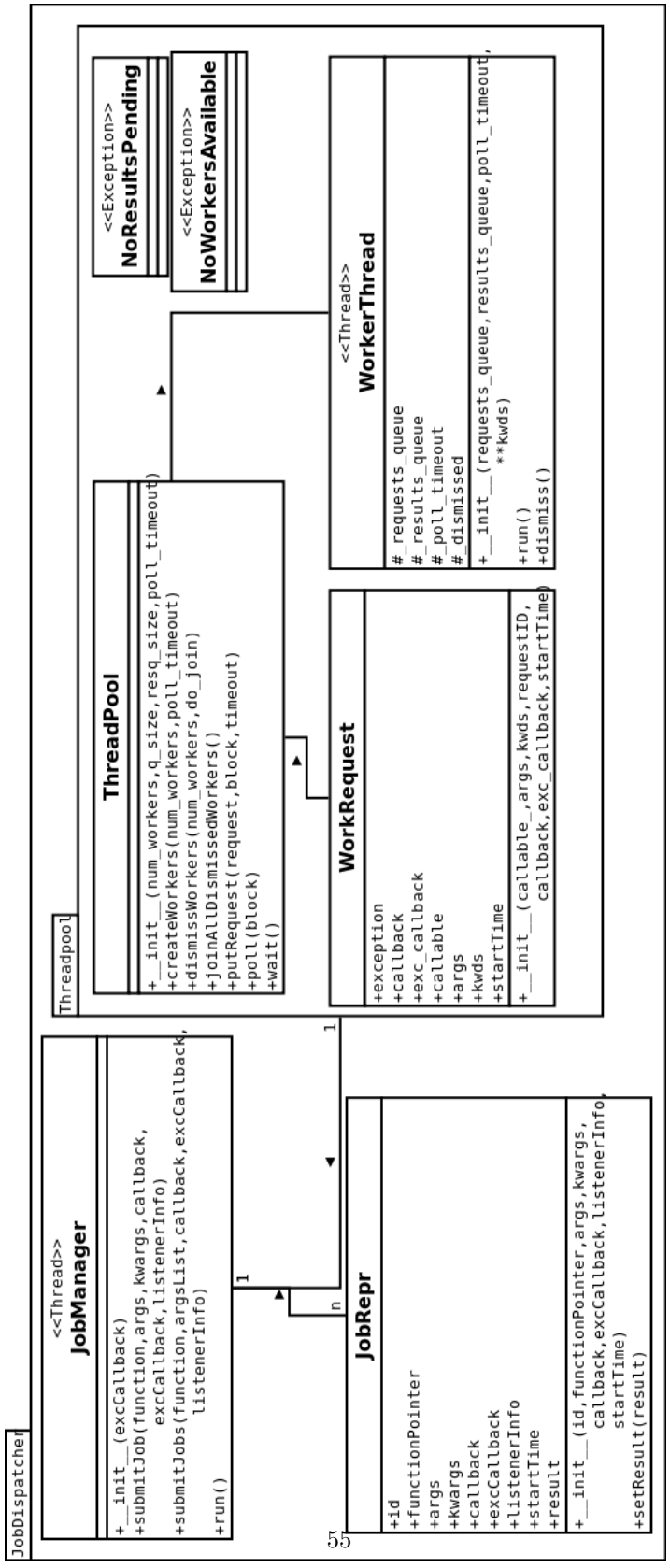


Figure 5.13: The class structure of the JobDispatcher.

5.6 Database class diagrams

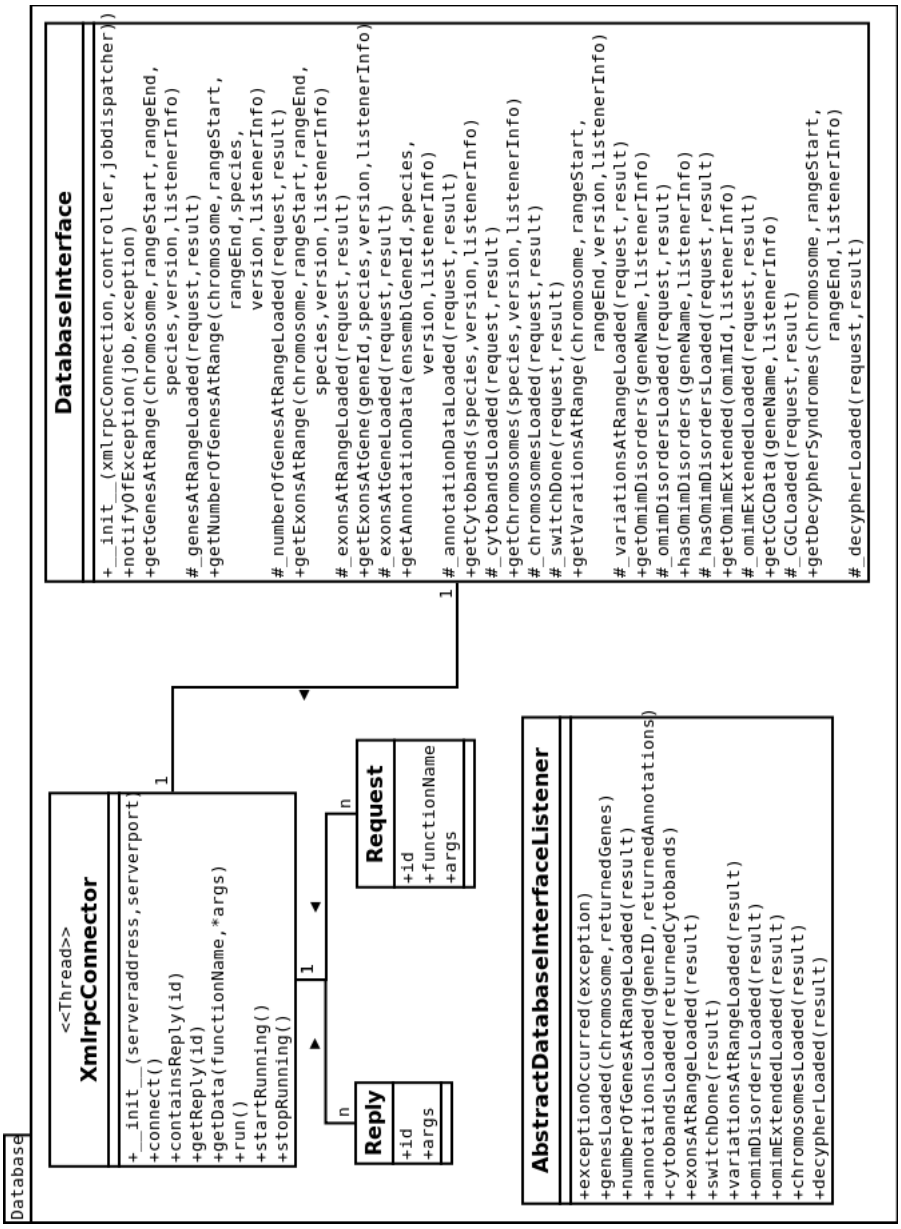


Figure 5.14: The class structure of the client side of the database.

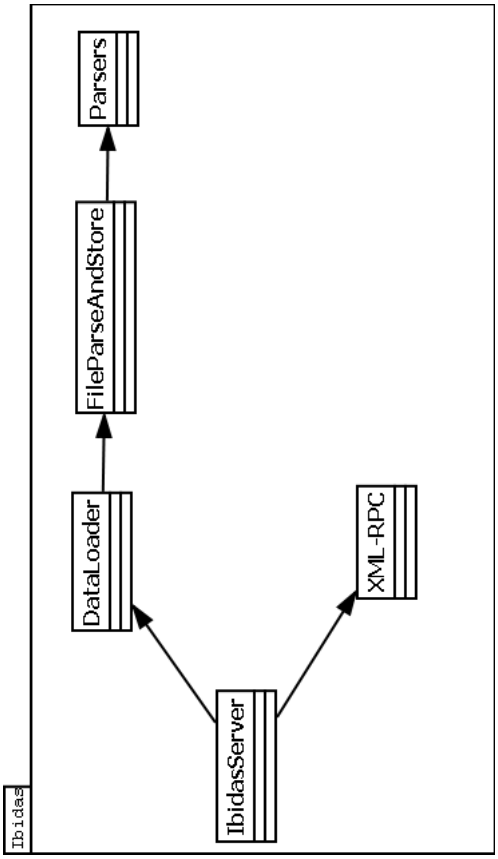


Figure 5.15: The class structure of the server side of the database.

5.7 Circos class diagrams

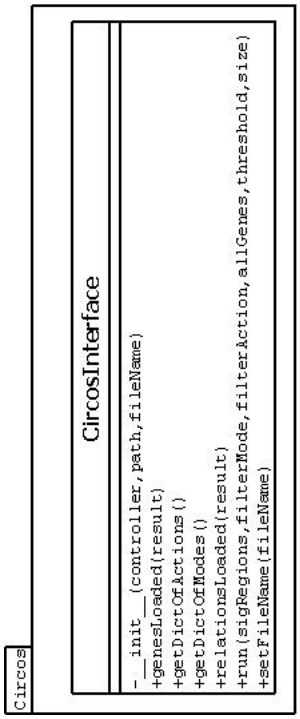


Figure 5.16: The class structure of the client side of the database.

5.8 Report class diagrams

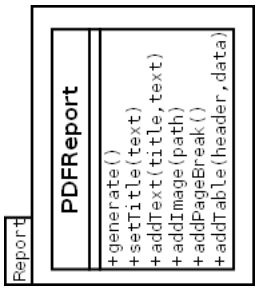


Figure 5.17: The class structure of the client side of the database.

References

[1] BC Cancer Agency. Circos website, june 2010. URL: http://mkweb.bcgsc.ca/circos/.

[2] A. Brazma, P. Hingamp, J. Quackenbush, G. Sherlock, P. Spellman, C. Stoeckert, J. Aach, W. Ansorge, C.A. Ball, H.C. Causton, et al. Minimum information about a microarray experiment (MIAME)toward standards for microarray data. *Nature genetics*, 29(4):365–371, 2001.

[3] L.J. Heyer, S. Kruglyak, and S. Yooseph. Exploring expression data: identification and analysis of coexpressed genes. *Genome research*, 9(11):1106, 1999.

[4] Reportlab inc. Reportlab website, june 2010. URL: http://www.reportlab.com/.

[5] C. Klijn and J. de Ronde. Kc-smart reference manual, april 2009. URL: http://watson.nci.nih.gov/bioc_mirror/packages/2.3/bioc/manuals/KCsmart/man/KCsmart.pdf.

[6] C. Klijn and J. de Ronde. KC-SMART Vignette, 2010.

[7] Little, Brown, and company. *A Dictionary of Greek and Roman biography and mythology*. Smith, William, Sir, 1867. URL: http://quod.lib.umich.edu/cgi/t/text/text-idx?c=moa;idno=ACL3129.0001.001.

[8] NBIC. Ibidas website, june 2010. URL: https://wiki.nbic.nl/index.php/Ibidas.

[9] Nokia. Qt website, june 2010. URL: http://qt.nokia.com/.

Appendix J

End User Document

Graphical application for the statistical analysis of copy number variations in aCGH data

Thesis BSc Project IN3405

End User Documentation

version 0.9

Julian de Ruiter (1387367)
Chris Melman (1358359)
Rien Korstanje (1270389)
Stefan Dentro (1313320)
Nanne Aben (1369342)

DELFT UNIVERSITY OF TECHNOLOGY
EEMCS FACULTY
JUNE 28, 2010

Contents

Chapter 1

Introduction

1.1 Purpose

This document contains a basic overview of how to utilise the functions that the product contains. The functions are explained through both text and images. As this document had to be completed before completion of the final prototype, minor discrepancies between the actual prototype and this document can exist. There are also minor differences in the graphical interface between different platforms (Windows, Mac and Linux). The images in this document were made on the Linux platform.

1.2 Scope

The document describes how to use a graphical, statistical analysis application of copy number variations in aCGH data, codenamed Angita. The tool is being developed for the NKI and the VUMC.

The application is able to import aCGH data from a local file and identify significant aberrations in the genomic data using various algorithms. It provides an interface in which a loaded aCGH profile can be viewed in a linear fashion. The interface also displays the various genes located in identified aberrant regions. Genes are annotated by various databases to provide more information about the gene in question. The application also provides a view which displays relations between the genes in the identified aberrant regions of the genome, allowing the type of relations to be shown can be chosen.

The application aims to allow quick analysis of the provided aCGH data by displaying a clear view of the aCGH profile, providing various algorithms to identify aberrant regions of the genome and displaying the identified regions in a clear manner. The displayed relations between genes in the identified regions provide the user with a significant amount of extra data, which can help identify non-obvious relations between the identified regions and corresponding genes.

1.3 Definitions, acronyms and abbreviations

Aberration Modification of the normal chromosome complement due to deletion, duplication, or rearrangement of genetic material.

Angita Angita is the early Roman goddess of healing, magic and witchcraft. [1]

Array comparative genomic hybridisation (aCGH) Array comparative genomic hybridisation is a technique to detect genomic copy number variations at a higher resolution level than chromosome-based comparative genomic hybridisation (CGH).

Copy number variant (CNV) A copy number variant is a segment of DNA in which copy-number differences have been found by comparison of two or more genomes. The segment may range from one kilo-base to several mega-bases in size. Humans (being diploid) ordinarily have two copies of each autosomal region, one per chromosome. This may vary for particular genetic regions due to deletion or duplication.

VUMC The VU University Medical Centre is an academic medical centre, concerned with patient care, higher education and medical research. [3]

NKI The Netherlands Cancer Institute researches the cause and effects of cancer. The institutes three major areas of research are fundamental, clinical and translational cancer research. [2]

1.4 Overview

The second chapter contains a list of software packages that are required to run the product. Chapter three describes how to load a sample into the application. How to navigate through the loaded sample data is described in chapter four. Chapter five elaborates on the analysis of sample data, whilst chapter six describes how to generate a report of significant regions within the data. Appendix A contains a sample export generated by the product.

Chapter 2

Prerequisites

2.1 Client

A number of dependencies need to be installed before starting the product. The following list of dependencies are mandatory:

- Python 2.6.x
- PyQt 4.7.2
- Perl 5.10
- RPy2 2.0.x (also known as python-rpy)
- OpenGL 3.x (also known as python-opengl)

Afterwards the product can be started by running the command *python main.py* from the commandline or a shortcut can be created.

2.2 Server

The product depends on a running Ibidas server. Although the client can be run independently, it will be unable to load meta data in such a situation. A number of (modified) files need to be copied into a basic Ibidas installation to get the basic server component working.

- IbidasXMLRPCWithoutDB.py → Ibidas root
- IbidasDataParser.py → Ibidas root
- IbidasAtomicServicesTemp2.py → Ibidas root/web
- basicparser.py → Ibidas root/parsers
- biomart.py → Ibidas root/parsers
- CGC.py → Ibidas root/parsers
- cytoband.py → Ibidas root/parsers

- decypher.py → Ibidas root/parsers
- dgv.py → Ibidas root/parsers
- omim.py → Ibidas root/parsers
- omimcompletetext.py → Ibidas root/parsers

When the flat files containing the database information are put in Ibidas root/human and Ibidas root/mouse Ibidas can be started using the *python ibidas2.py* command.

Chapter 3

Loading a sample

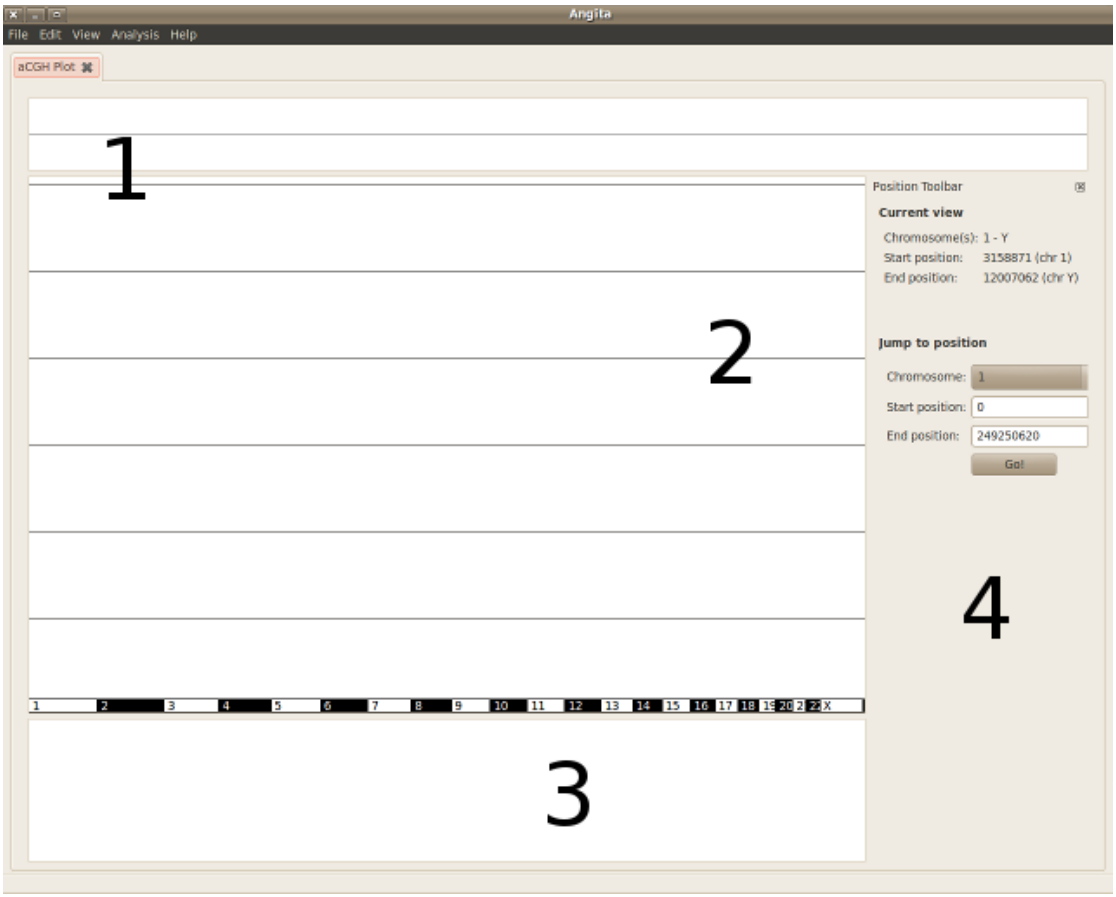


Figure 3.1: Mainwindow just after starting

3.1 Mainwindow overview

The product can be started after the dependencies mentioned in chapter 2 have been installed. Figure 3.1 shows the main window, which is shown after the product has been started. The list below denotes the function of the numbered elements in the figure.

1. Is the overview bar, which is where a full overview of the genome will be drawn.
2. The main view that contains the data points. The alternating black-and-white line represents the various chromosomes of the genome, where each block represents a single chromosome. The length of an element in the line corresponds to the length of a chromosome.
3. View that contains meta data related to the data shown in view 2.
4. Navigation between different regions within the data.

3.2 Open file

To load an aCGH file containing a sample go to *File* → *Open samples...* US2, GPL and ETABM filetypes are supported.

The aCGH data is automatically normalised. Afterwards the overview is created and an outline of the data is shown. The outline is calculated as two times the variance of the data. Data points that are outside the outline are plotted individually. Figures 3.2, 3.3 and 3.4 provide an overview of these steps.

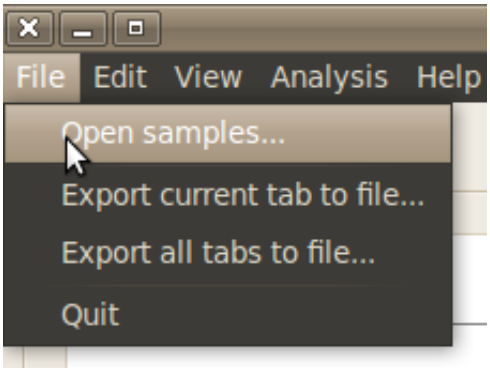


Figure 3.2: Opening a file containing a sample

Figure 3.3: The progress bar, visible while loading

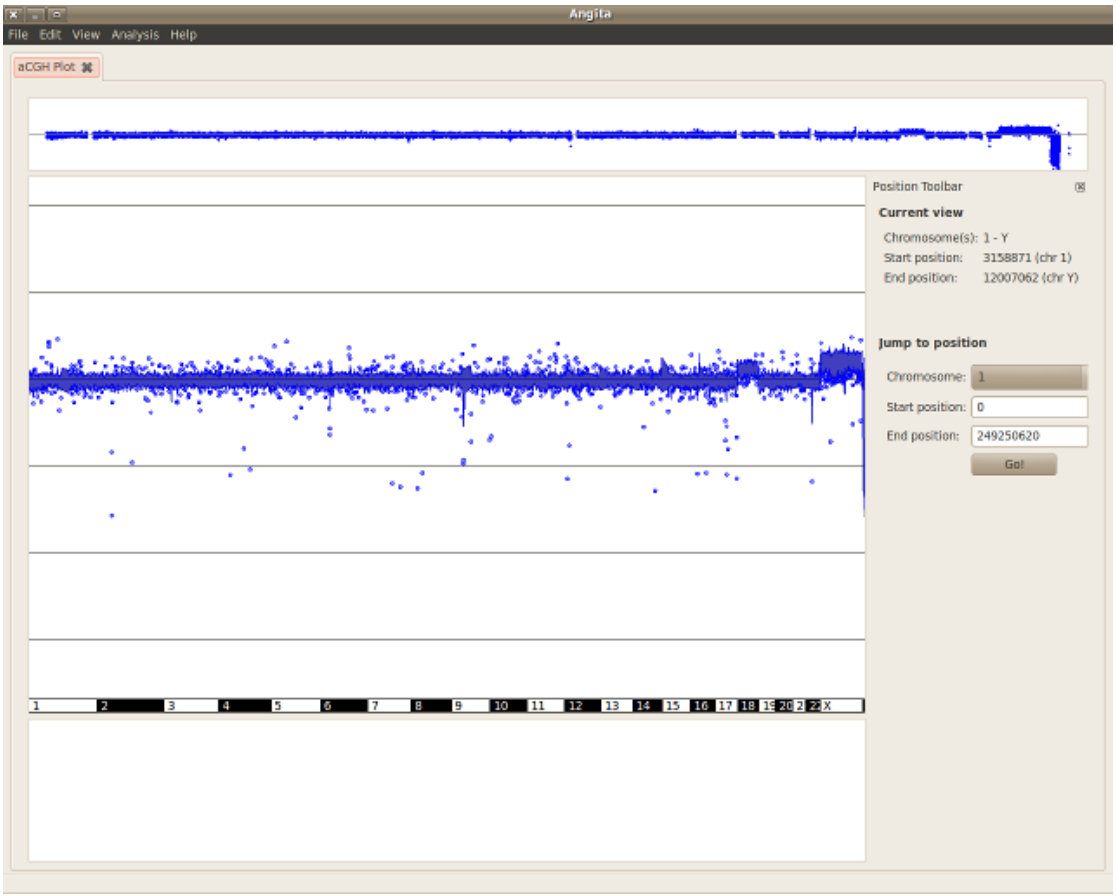


Figure 3.4: Mainwindow showing the overview and loaded data.

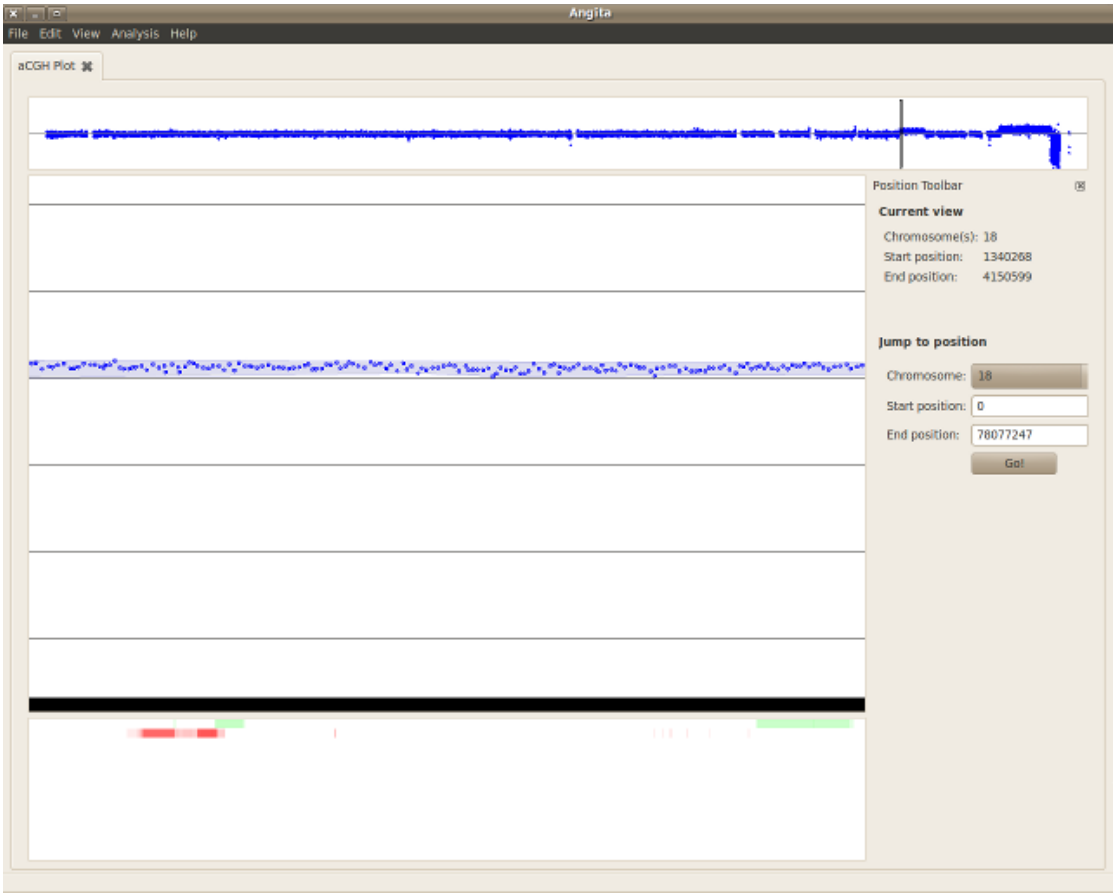


Figure 3.5: Zoomed in, data points and DGV information becomes visible

Actual data points are visible after zooming in (Figure 3.5) on the main view. See section 4.1 for more details on zooming in the views. The meta data view shows DGV data, which is automatically loaded. The DGV data is shown in coloured bars, in which green represents a gain and red represents a loss. The intensity of a bar corresponds to the number of gains or losses found in that region.

Chapter 4

Navigation

4.1 Zooming

Zooming actions are required to vary the scale of the data shown in the various views. Zooming can be done in a number of ways. The most simple manner is simply putting the mouse above an interesting area and scrolling using the mouse scroll wheel. A different zoom manner of zooming is done by dragging a box over the overview or data view whilst holding the right mouse button, after which the views will show the selected region. The position toolbar allows direct and precise manipulation of the view by entering the required view data in its input fields. The position toolbar is located at the right-hand side of the screen.

To zoom in within the overview bar, drag a box over the overview while holding the right mouse-button down (Figure 4.1).

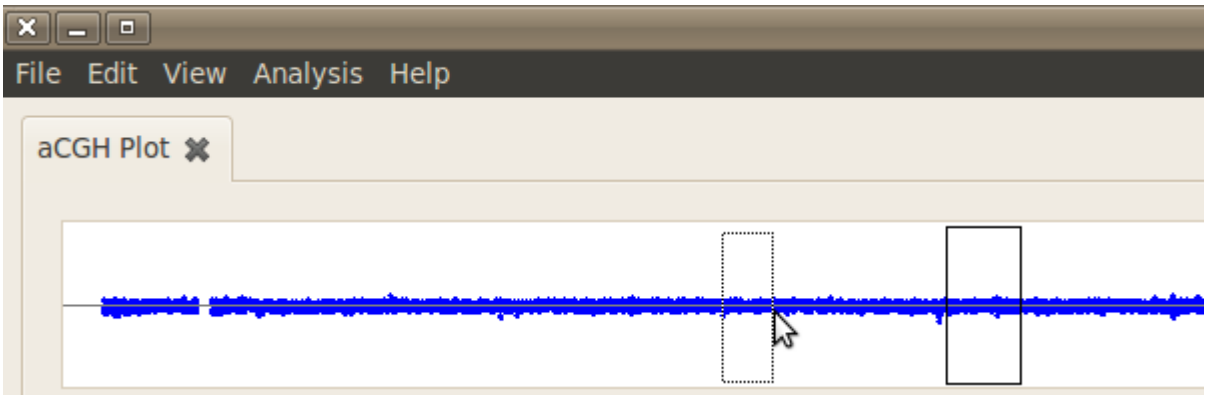


Figure 4.1: Zooming in the overview bar

To zoom within the data view, drag a box over the data while holding the right mouse-button down (Figure 4.2).

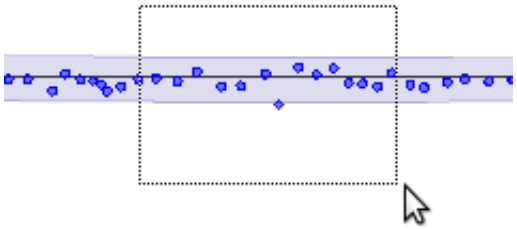


Figure 4.2: Zooming in the data view

4.2 Dragging

Navigating through the data by dragging is possible in all three views. The box in the overview can be dragged to an interesting position, whilst the other two views allow dragging of the canvas by holding the left mouse button.

Chapter 5

Analysing a sample

5.1 Select analysis

After a sample is loaded an analysis can be selected to identify the significant regions of the loaded CGH profile. Currently a simple threshold procedure and a KC-Smart analysis are provided by the product. To analyse the data go to the menu item *Analysis* and press (as an example) *Use threshold analysis*. (Figure 5.1)

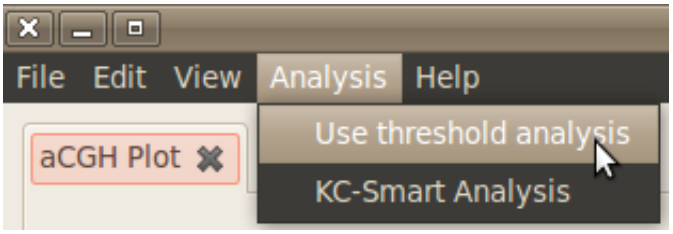


Figure 5.1: Selecting an analysis

5.2 Set analysis parameters

The product will now ask for the threshold level and how many consecutive data points must be above the threshold (Figure 5.2). A sequence of data points that fits the criteria will be marked as a significant region when the analysis has been run.

When the analysis is finished the product will look similar to figure 5.3. Significant regions are coloured either green or red and genes found in the identified regions are shown in the gene view. Black markings within the genes describe exons, no markings are introns.

Left-clicking on a gene results in the opening of a new tab containing information about the corresponding gene (Figure 5.4). Depending on the gene this could be OMIM, DECIPHER, CGC and/or DGV data.

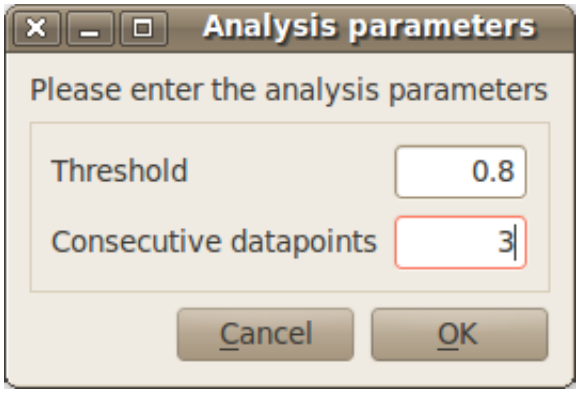


Figure 5.2: Window asking for analysis parameters

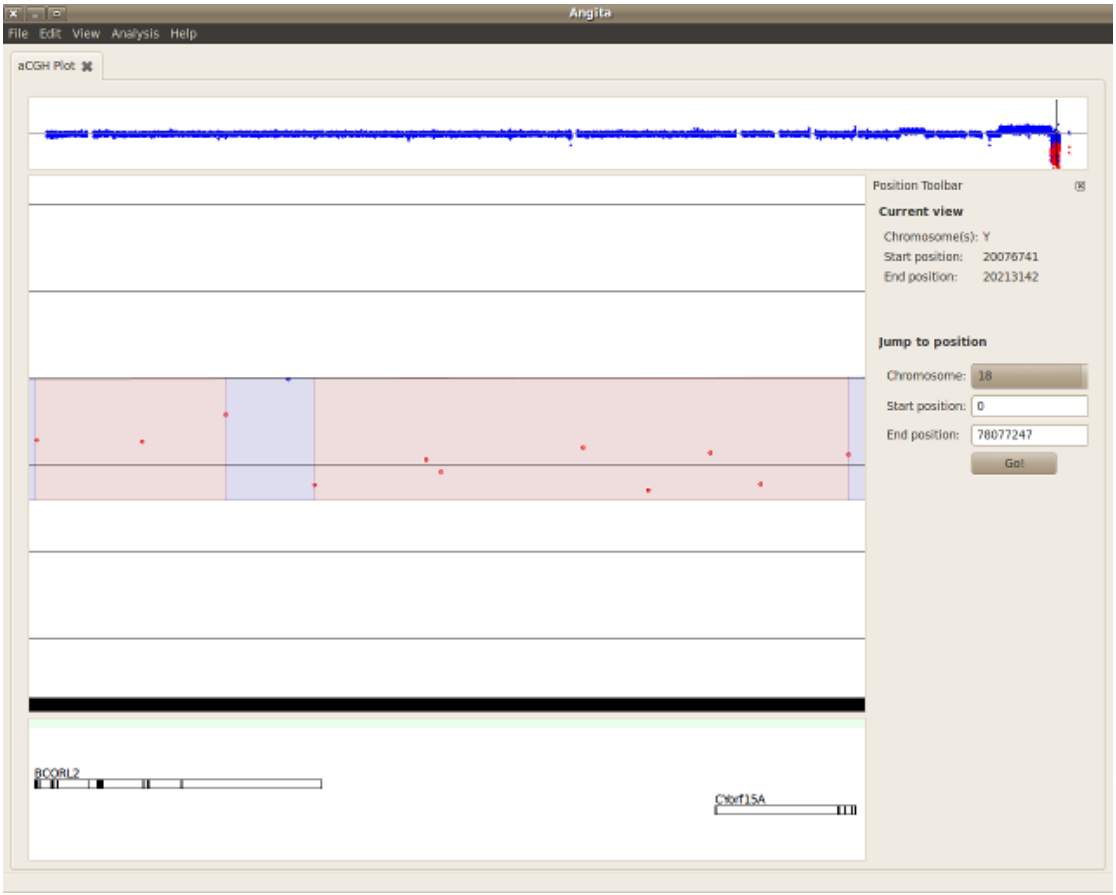


Figure 5.3: Finished analysis

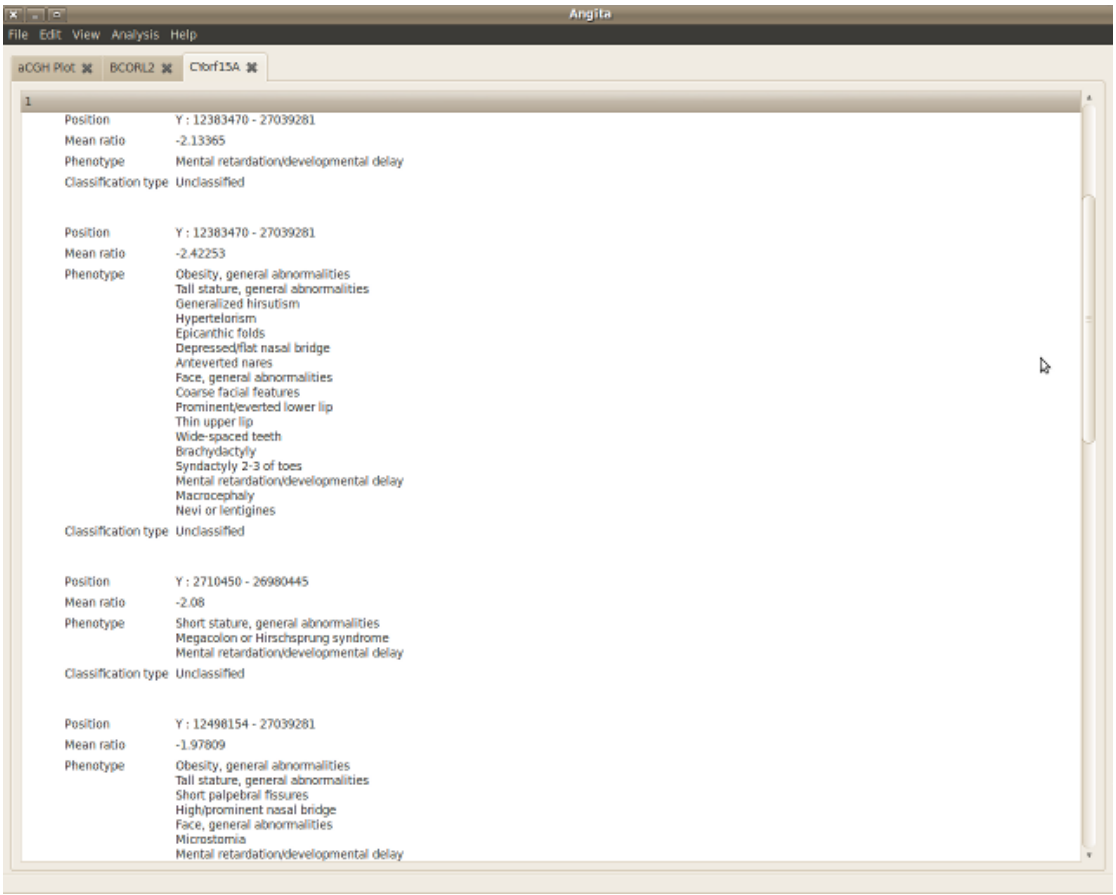


Figure 5.4: DECIPHER data for CYorf15A gene

Chapter 6

Export to PDF

It is possible to export context information shown in a tab to a PDF file. Go to *File* → *Export tab to file...* (Figure 6.1). The product will ask for a location to save the PDF. A sample PDF showing the results of such an export is attached to this document.

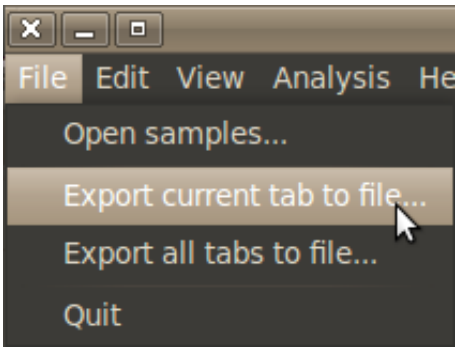


Figure 6.1: Export tab information to file

References

[1] Little, Brown, and company. *A Dictionary of Greek and Roman biography and mythology*. Smith, William, Sir, 1867. URL: http://quod.lib.umich.edu/cgi/t/text/text-idx?c=moa;idno=ACL3129.0001.001.

[2] NKI. About the netherlands cancer institute, 2010. URL: http://www.nki.nl/Research/About+the+Netherlands+Cancer+Institute/.

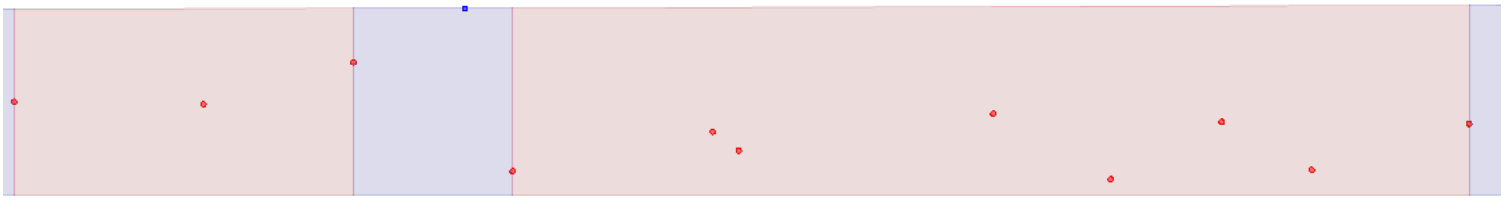
[3] VUMC. About the vumc, 2010. URL: http://identity20.com/media/OSCON2005/.

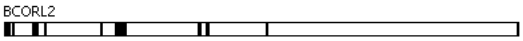
Appendix A

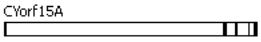
Sample exported PDF

Generated Angita Report

27-06-2010 18:51:03







Chromosome(s) : Y
Start position: 20076862
End position: 20213263

BCORL2

Position	Y : 20077586 - 20124410
Band	q11.222
Strand	-1
Description	BCoR-like protein 2 (BCL-6 corepressor-like protein 2) [Source:UniProtKB/Swiss-Prot;Acc:Q8N888]

DECYPHER	
Position	Y : 12383470 - 27039281
Mean ratio	-2.13365
Phenotype	Mental retardation/developmental delay
Classification type	Unclassified

Position	Y : 12383470 - 27039281
Mean ratio	-2.42253
Phenotype	Obesity, general abnormalities Tall stature, general abnormalities Generalized hirsutism Hypertelorism Epicanthic folds Depressed/flat nasal bridge Anteverted nares Face, general abnormalities Coarse facial features Prominent/everted lower lip Thin upper lip Wide-spaced teeth Brachydactyly Syndactyly 2-3 of toes Mental retardation/developmental delay Macrocephaly Nevi or lentigines
Classification type	Unclassified

Position	Y : 20087973 - 20105719
Mean ratio	-5.55
Phenotype	Short stature, general abnormalities Megacolon or Hirschsprung syndrome Mental retardation/developmental delay
Classification type	Unclassified

Position	Y : 2710450 - 26980445
Mean ratio	-2.08
Phenotype	Short stature, general abnormalities Megacolon or Hirschsprung syndrome Mental retardation/developmental delay
Classification type	Unclassified

Position	Y : 12498154 - 27039281
Mean ratio	-1.97809

	Obesity, general abnormalities Tall stature, general abnormalities Short palpebral fissures High/prominent nasal bridge Face, general abnormalities Microstomia Mental retardation/developmental delay
Phenotype	Macrocephaly
Classification type	Unclassified
Position	Y : 20083423 - 20384698
Mean ratio	-3.65
	Short stature, general abnormalities Megacolon or Hirschsprung syndrome
Phenotype	Mental retardation/developmental delay
Classification type	Unclassified
Position	Y : 12383470 - 27039281
Mean ratio	-2.2718
	Tall stature, general abnormalities Large ears Deafness, sensorineural Large nose Face, general abnormalities Coarse facial features Mental retardation/developmental delay SEIZURES, general abnormalities
Phenotype	Macrocephaly
Classification type	Unclassified
Position	Y : 12383470 - 27039281
Mean ratio	-2.28175
	Tall stature, general abnormalities
Phenotype	Mental retardation/developmental delay
Classification type	Unclassified
Position	Y : 12452733 - 27039281
Mean ratio	-2.25708

	Obesity, general abnormalities Tall stature, general abnormalities Hair growth pattern, general abnormalities Generalized hirsutism Sparse/decreased eyebrows Long/prominent eyelashes Ptosis of eyelids Face, general abnormalities Thick lower lip Speech delay Kyphosis Brachydactyly Mental retardation/developmental delay
Phenotype	Microcephaly
Classification type	Unclassified

DGV	
Location	Y : 19512769 - 21351380
Variation type	CopyNumber
Observed gains	1
Observed losses	0
Reference	Perry et al. (2008)
URL	http://projects.tcag.ca/cgi-bin/variation/xview?source=hg19&view=variation&id=variation_315

CYorf15A

Position	Y : 20188623 - 20211528
Band	q11.222
Strand	1
Description	Uncharacterized protein CYorf15A [Source:UniProtKB/Swiss-Prot;Acc:Q9BZA5]

OMIM	
Title	Chromosome Y open reading frame 15A
Date entered	23-3-4
Status	provisional
Disorders	
Comments	
URL	http://www.ncbi.nlm.nih.gov/omim/400031

DECYPHER	
Position	Y : 12383470 - 27039281
Mean ratio	-2.13365
Phenotype	Mental retardation/developmental delay
Classification type	Unclassified
Position	Y : 12383470 - 27039281
Mean ratio	-2.42253
Phenotype	Obesity, general abnormalities Tall stature, general abnormalities Generalized hirsutism Hypertelorism Epicanthic folds Depressed/flat nasal bridge Anteverted nares Face, general abnormalities Coarse facial features Prominent/everted lower lip Thin upper lip Wide-spaced teeth Brachydactyly Syndactyly 2-3 of toes Mental retardation/developmental delay Macrocephaly Nevi or lentigines
Classification type	Unclassified
Position	Y : 2710450 - 26980445
Mean ratio	-2.08
Phenotype	Short stature, general abnormalities Megacolon or Hirschsprung syndrome Mental retardation/developmental delay
Classification type	Unclassified
Position	Y : 12498154 - 27039281
Mean ratio	-1.97809

Phenotype	Obesity, general abnormalities Tall stature, general abnormalities Short palpebral fissures High/prominent nasal bridge Face, general abnormalities Microstomia Mental retardation/developmental delay Macrocephaly
Classification type	Unclassified

Position	Y : 20083423 - 20384698
Mean ratio	-3.65
Phenotype	Short stature, general abnormalities Megacolon or Hirschsprung syndrome Mental retardation/developmental delay
Classification type	Unclassified

Position	Y : 20193911 - 20201274
Mean ratio	-5.3
Phenotype	Short stature, general abnormalities Megacolon or Hirschsprung syndrome Mental retardation/developmental delay
Classification type	Unclassified

Position	Y : 12383470 - 27039281
Mean ratio	-2.2718
Phenotype	Tall stature, general abnormalities Large ears Deafness, sensorineural Large nose Face, general abnormalities Coarse facial features Mental retardation/developmental delay SEIZURES, general abnormalities Macrocephaly
Classification type	Unclassified

Position	Y : 12383470 - 27039281
Mean ratio	-2.28175
Phenotype	Tall stature, general abnormalities Mental retardation/developmental delay
Classification type	Unclassified

Position	Y : 12452733 - 27039281
Mean ratio	-2.25708
Phenotype	Obesity, general abnormalities Tall stature, general abnormalities Hair growth pattern, general abnormalities Generalized hirsutism Sparse/decreased eyebrows Long/prominent eyelashes Ptosis of eyelids Face, general abnormalities Thick lower lip Speech delay Kyphosis Brachydactyly Mental retardation/developmental delay Microcephaly
Classification type	Unclassified

DGV	
Location	Y : 19512769 - 21351380
Variation type	CopyNumber
Observed gains	1
Observed losses	0
Reference	Perry et al. (2008)
URL	http://projects.tcag.ca/cgi-bin/variation/xview?source=hg19&view=variation&id=variation_315