Graph based analysis of biological networks in the context of experimental results

Jan Baumbach

Diploma thesis in Applied Computer Science in the Natural Sciences

Supervisors:
Prof. Dr. R. Hofestädt ¹
Dr. J. Köhler ²

¹ Faculty of Technology, University of Bielefeld (Germany)
² BAB Department, Rothamsted Research Harpenden (UK)

Bielefeld, 11/5/2005
Acknowledgment

This work was written as my diploma thesis at the Faculty of Technology at the University of Bielefeld under the supervision of Prof. Dr. R. HofestädDt whom I wish to thank for his help and advise. The research presented here was undertaken while I was a visiting student at Rothamsted Research (RRes) in Harpenden (UK), an Institute of the UK Biotechnology and Biology Science Research Council (BBSRC) and I am very much indebted to Dr. J. Köhler for giving me the opportunity to work with him under his guidance.

Although he is not involved with this work, I wish to thank Dr. T. Schmitt-John for teaching me developmental biology in his theoretical and practical laboratory courses which aroused my interests in the problems of molecular biology, microarrays and laboratory techniques.

I enjoyed my time at the Biomathematics and Bioinformatics (BAB) division at RRes. I have learned a lot in the group meetings and the discussions with my colleagues (especially Prof. Dr. C. Rawlings and Prof. P. Verrier). The working conditions were excellent and I was very pleased to be invited to work there. Special thanks to Prof. P. Verrier for the accomplishment of the application case and his very helpful revisions.

Without the support of my whole family, especially my grandparents and my mother, I could not have written this diploma thesis and I want to thank all who are interested in my life and my work and who supported me with all their knowledge and their help. Thanks to my best friend Josch who was always very helpful and understanding in all circumstances and to my girl friend Birte.

I would like to extend my special thanks to my father, who I owe very much.
Contents

1. INTRODUCTION ................................................................................................................ 5
   1.1 OVERVIEW ................................................................................................................... 5
   1.2 MOTIVATION ................................................................................................................. 7
   1.3 AIMS .............................................................................................................................. 10

2. BACKGROUND ................................................................................................................. 12
   2.1 BIOLOGICAL BACKGROUND AND EXPERIMENTAL TECHNIQUES ......................... 12
       2.1.1 cDNA Microarray ................................................................................................... 12
       2.1.2 Expressed Sequence Tags ...................................................................................... 13
       2.1.3 Nuclear Magnetic Resonance Spectroscopy ........................................................... 14
       2.1.4 Mass Spectrometry .................................................................................................. 14
       2.1.5 Ion mobility spectrometry ........................................................................................ 15
   2.2 ONDEX .......................................................................................................................... 18
   2.3 THEORETICAL BACKGROUND ..................................................................................... 21

3. RELATED WORK ............................................................................................................ 23

4. REQUIREMENT ANALYSIS .......................................................................................... 27
   4.1 LARGE GRAPHS .......................................................................................................... 27
   4.2 GRAPH LAYOUT .......................................................................................................... 28
   4.3 VISUALISATION .......................................................................................................... 30
   4.4 GENERIC ARCHITECTURE .......................................................................................... 31
   4.5 FILTERS ....................................................................................................................... 32
   4.6 VALIDATION OF APPROACH ..................................................................................... 32

5. GRAPH LIBRARY EVALUATION ................................................................................ 33

6. PRINCIPLES AND METHODS .................................................................................... 37
   6.1 DATA STRUCTURES ..................................................................................................... 37
   6.2 ARCHITECTURE .......................................................................................................... 42
   6.3 ALGORITHMS ............................................................................................................. 45
       6.3.1 FastCircularLayout ................................................................................................. 45
       6.3.2 MicroArrayFilter .................................................................................................... 47
       6.3.3 HideConceptClassFilter ....................................................................................... 53
7. IMPLEMENTATION........................................................................................................ 55
  7.1 UML CLASS DIAGRAM.................................................................................................. 55
  7.2 INTERFACES.................................................................................................................. 58
    7.2.1 GeneralGraphLibraryAdapter ................................................................................ 58
    7.2.2 GeneralLayouter .................................................................................................... 59
    7.2.3 GeneralFilter........................................................................................................... 60
  7.3 EXAMPLE.......................................................................................................................... 61
  7.4 TRANSPATH PARSER....................................................................................................... 68
  7.5 SYSTEM REQUIREMENTS .............................................................................................. 70
    7.5.1 Software................................................................................................................... 71
    7.5.2 Hardware ................................................................................................................ 72
8. RESULTS............................................................................................................................ 73

9. APPLICATION CASE: MICROARRAY ANALYSIS .......................................................... 76
  9.1 METHODS ......................................................................................................................... 76
  9.2 DATA ANALYSIS AND VISUALISATION .................................................................. 78
    9.2.1 Key Pathways .......................................................................................................... 78
    9.2.2 Relevance of the Microarray Chip .......................................................................... 80
    9.2.3 Key Stress Genes ..................................................................................................... 81
  9.3 GENE ANNOTATION ....................................................................................................... 81

10. DISCUSSION .................................................................................................................... 82

11. OUTLOOK ......................................................................................................................... 86

11. APPENDIX ........................................................................................................................ 90
  11.1 BIOBIBLIOGRAPHY ..................................................................................................... 90
  11.2 GENE ANNOTATIONS .................................................................................................. 97
  11.3 FIGURES AND TABLES ................................................................................................. 100
    11.3.1 Figures ................................................................................................................ 100
    11.3.2 Tables .................................................................................................................. 105
1. Introduction

1.1 Overview

Research biologists test a scientific hypothesis with experimentation. Current practice is to use high-throughput experimental techniques to essentially assay the existence or quantity of a compound or the change in expression level of a gene (through mass spectrometry, ion mobility spectrometry or microarray experiments, for example). This results in large quantities of experimental data, much of which needs to be compared with known details of a compound, gene or enzyme. Currently, data on genomes, compounds, genes and proteins is held in a diverse range of public and proprietary databases which have to be searched independently of each other for information about one specific element. This searching out of information is both time-consuming and error prone even when only a few items are being sought because it is necessary to take the output of one search as input to another on a different database and frequently, relationships between data is hard and sometimes almost impossible to spot for any but the most experienced people. When handling high-throughput data, the problem becomes exceedingly difficult and occupies a large proportion of the research scientists time.

In this thesis a software framework (OVTK) has been developed to help biologists with the analysis and the interpretation of experimental results in the context of biological networks that have been extracted from multiple integrated data sources.

The following two sections outline the motivation and the aims of this work.

The next chapter provides some essential background information. It introduces the biological background (especially relevant molecular biological experimental techniques like cDNA microarrays, NMR analysis, mass spectrometry and ion mobility spectrometry) and the ONDEX data warehouse system which is used to extract and to integrate biological networks from scientific data sources. In general (biological) networks are represented through graphs. For this reason the theoretical background of graphs will be given in the following section.

To give an overview on related work, existing frameworks are presented, along with their strengths and limitations and where they differ from this approach in Chapter 3. Related Work.

Chapter 4. Requirement Analysis is dedicated to concrete specifications of the requirements for a framework to meet the aims of this project. To assist a biologist analysing experimental
results in the context of biological networks, a useful framework has to have the ability e.g. to handle, filter, visualise and layout large graphs consisting of hundreds of thousands of nodes and edges.

Before designing and developing new layout algorithms, existing graph libraries were evaluated in order to test their performance in visualising and layouting biological graphs. The results will be discussed in Chapter 5. Graph Library Evaluation in detail. One can see that the capabilities of the different libraries varied largely, a fact which did lead to an additional requirement: that the system should be developed using a generic architecture which enables the use of different graph libraries.

To develop methods that fulfil the requirements it is necessary to define data structures and to develop algorithms which can exploit these data structures to their full potential. Based on this, an architecture has to be designed which allows to use the data structures and algorithms in a standardised way and to simplify the integration of additional advanced algorithms. In Chapter 6. Principles and Methods the used data structures are defined (based on the definitions from the Chapter 2.3 Theoretical Background) as well as the entire architecture. In addition two filter algorithms which were developed as part of this project and one layout algorithm is presented.

In Chapter 7. Implementation, there is a description of how the system implementation of the abstract data structures and algorithms was accomplished with the aid of an UML class diagram of the developed object oriented graph representation of biological networks. Furthermore, a generic architecture which allows to integrate several software libraries in order to benefit from their different advantages is described. Additionally, an example of a potential standard analysis workflow is used to illustrate how to work with the developed software package OVTK.

As already mentioned, ONDEX is a data warehouse system which helps to integrate distributed data sources into one database. ONDEX uses “small” tools (parser programs). These parsers have to be implemented for each data source separately. In the framework of this thesis a parser for the signal transduction database Biobase Transpath (Wingender, 2004) has been developed and implemented and its functionality is presented in Chapter 7.4 Transpath Parser. In Chapter 7.5 System Requirements the soft- and hardware requirements of the OVTK are described.

In Chapter 8. Results the results are summarised and compared to the original aims and requirements of this thesis. Then the main analysis and visualisation methods of the OVTK are outlined.
An application case in the context of microarray analysis results is presented in the Chapter 9. Application Case: Microarray Analysis. The approaches are validated by the use of OVTK for the analysis of the results given by an array experiment of Parani (Parani et al., 2004). The Chapter 10. Discussion summarises the results and discusses them in a wider context. In the outlook, approaches for the improvement of existing methods as well as ideas for new techniques are discussed.

1.2 Motivation

Many biological databases are databases which describe molecular biological pathways such as protein-protein, protein-DNA or protein-metabolite interactions. The database AraCyc (Mueller et al., 2003) contains data on the metabolism, signal transduction and gene regulatory networks of the plant Arabidopsis thaliana. It is an application of a computational symbolic theory, which is a database that represents knowledge of biochemical transactions in a well unified structured and formal way so that it is possible to run computational analysis on it. By integrating informal scientific theories in a symbolic representation, a new realm of analysis is opened that would be much too large and complex for scientists to draw conclusions from efficiently without computational assistance.

Experimental biological techniques generate a large amount of data. As an illustration, Figure 1 shows an example of a portion of a cDNA Microarray plate from the Faculty of Medicine of Imperial College, London, UK (Froguel, 2004) that shows a combination of Cy3 and Cy5 fluorescence being used to identify the amount of differential gene expression between two biological samples. Figure 2 (page 8) shows an example of the full yeast genome on a Microarray chip from the Stanford University, USA after hybridisation (Brown, 2004). As explained in detail in Chapter 2.1.1 cDNA Microarray (page 12) this experimental technique is used to determine which genes are underexpressed or overexpressed when two populations of genes are compared (Schena, 2000). A yellow spot means that the correlating gene is not differentially expressed, a green spot denotes a down-regulation and a red spot denotes an up-regulation of the corresponding gene.
Figure 1 - Microarray example from Faculty of Medicine, Imperial College UK, Genetics and Genomics (Froguel, 2004)

Figure 2 - Microarray example of "The Full Yeast Genome on a Chip" from Standford University (Brown, 2004)

Table 1 illustrates an extract of an (already normalised) result file of a Microarray experiment which was generated in the context of the exploration of the nitrous oxide stress of plants at Rothamsted Research, UK (refer Chapter 9. Application Case: Microarray Analysis, page...
This file contains more than 10 000 entries consisting of the spot name, the identifier, the locus link id, the expression level (“Log2-RbyG”), annotations and some additional informations (e.g. known sequences).

Table 1 - Microarray result file

<table>
<thead>
<tr>
<th>SpotName</th>
<th>ID</th>
<th>Loci</th>
<th>Log2-RbyG</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>T41531</td>
<td>102B12T7</td>
<td>At3g29075</td>
<td>-0.37</td>
<td>glycine-rich protein</td>
</tr>
<tr>
<td>AI100731</td>
<td>102B8XP</td>
<td>At3g10350</td>
<td>-0.08</td>
<td>ATPase-related; similar to ATPase GB:CAB54139 (Solanum tuberosum), Pfam HMM hit: arsenite activated ATPase (arsA); contains non-consensus GA donor splice site at intron 5</td>
</tr>
<tr>
<td>T22227</td>
<td>103A14T7</td>
<td>At5g04810</td>
<td>0.19</td>
<td>pentatricopeptide (PPR) repeat-containing protein; contains Pfam profile: PF01535 PPR repeat</td>
</tr>
<tr>
<td>T22228</td>
<td>103A17T7</td>
<td>At2g05630</td>
<td>-0.11</td>
<td>microtubule-associated protein related</td>
</tr>
<tr>
<td>AI100735</td>
<td>103B2XP</td>
<td>At1g33475</td>
<td>-0.25</td>
<td>RRM-containing protein related; temporary automated functional assignment</td>
</tr>
<tr>
<td>T22240</td>
<td>103C10T7</td>
<td></td>
<td>-0.17</td>
<td>glycine-rich protein</td>
</tr>
<tr>
<td>T21817</td>
<td>103C16T7</td>
<td>At1g14400</td>
<td>-0.04</td>
<td>ubiquitin-conjugating enzyme 1 (UBC1); E2; identical to gi:431259, SP:P25865</td>
</tr>
<tr>
<td>T21822</td>
<td>103C22T7</td>
<td>At2g40830</td>
<td>-0.25</td>
<td>RING-H2 finger protein (RHC1a); annotation temporarily based on supporting cDNA gi</td>
</tr>
<tr>
<td>T22246</td>
<td>103C8T7</td>
<td>At4g17390</td>
<td>-0.35</td>
<td>60S ribosomal protein L15 (RPL15B)</td>
</tr>
<tr>
<td>T22250</td>
<td>103D19T7</td>
<td>At3g55010</td>
<td>-0.15</td>
<td>phosphoribosylformylglycinamidine cyclo-ligase precursor</td>
</tr>
<tr>
<td>R83996</td>
<td>103D1T7</td>
<td>At5g60980</td>
<td>0.07</td>
<td>NTF2-containing RNA-binding protein, putative; G3BP ras-GTPase-activating protein SH3-domain binding protein, Mus musculus, EMBL:MMU65313</td>
</tr>
<tr>
<td>T22254</td>
<td>103D23T7</td>
<td>At3g54580</td>
<td>-0.28</td>
<td>ATP synthase epsilon chain, mitochondrial identical to ATP synthase epsilon chain, mitochondrial SP:Q96253 from [Arabidopsis thaliana]</td>
</tr>
<tr>
<td>T22261</td>
<td>103E13T7</td>
<td>At5g57710</td>
<td>0.01</td>
<td>101 kDa heat shock protein; HSP101-related protein</td>
</tr>
<tr>
<td>T22264</td>
<td>103E18T7</td>
<td>At3g47860</td>
<td>-0.41</td>
<td>expressed protein</td>
</tr>
<tr>
<td>T22270</td>
<td>103E4T7</td>
<td></td>
<td>0.12</td>
<td>F-box family protein similar to ESTs gb</td>
</tr>
</tbody>
</table>

After the accomplishment of experiments like these it is desirable to know how the different elements from the results (mostly genes) are linked together, how they interact and to find out more information about specific genes that are exhibiting some interesting features such as up-regulation. Often there is interest in finding combined linkages and correlations between the involved elements. The main difficulty here is the large quantity of results and their possible interactions with others, complicating the visualisation and the biological analysis.
1.3 Aims

The objectives of this project are the visualisation and the analysis of information obtained from biological experiments in the context of knowledge extraction from molecular biology databases which are then represented through graphs.

As an illustration, consider a protein interaction graph and a gene regulation graph. They might have been extracted e.g. from Biobase Transpath and Transfac (Wingender, 2004) with a linkage (mapping) between the two graphs for all known DNA sequences for proteins (Figure 3 and Figure 4).

![Figure 3 - Legend to Figure 4 and Figure 5](image)

![Figure 4 - Abstract example for a mapping between 2 graphs](image)

To provide a better visualisation and to reduce the size of the (possibly) very densely populated graph it might be useful to apply special relevance filters, for example ‘filter by the organism’, ‘filter by the cell type’, ‘filter by the kind (class) of the nodes’, ‘filter by the type of the links (relations) between elements’ and so on.

Furthermore, it must be possible to present the graph in the context of the results of biological experiments and to use these results to apply filters and to arrange the graph in a suitable layout that exposes the information more clearly (Figure 5).
Figure 5 - Abstract example of a graph after filtering by organism (mouse) and after the colorisation and the setting of the node size according to the expression levels given by an experimental result.
2. Background

In this chapter the necessary background, concepts and tools are introduced. This includes the biological background (especially some experimental techniques), the ONDEX system that is used for the data integration process, and some basic graph definitions.

2.1 Biological Background and Experimental Techniques

Various methods are available for the detection and quantification of substances in biological tissues.

First, the main focus lies on the analysis of gene expression in cells using microarray experiments. Although both, cDNA (Duggan et al., 1999) and oligonucleotide (Bodrossy and Sessitsch, 2004) arrays are capable of analysing patterns of gene expression, differences exist between the methods. Since the results of both techniques are very similar, this chapter focuses primarily on the technical aspects of the cDNA microarrays.

The second part of this section introduces expressed sequence tags (EST), nuclear magnetic resonance (NMR) spectroscopy and the detection of biological substances (especially metabolites) using mass spectrometry (MS) and ion mobility spectrometry (IMS) methods.

2.1.1 cDNA Microarray

DNA microarrays are a powerful tool for the high throughput detection and quantification of mRNA. Microarrays have been developed for the analysis of whole genome gene expression. Depending on the availability of appropriate probe sets, they enable the detection of up to several thousand genes (depending on the design of the probe) in a single assay (Schena, 2000).

Figure 6 illustrates the procedure of cDNA microarray analysis. Templates for genes of interest are obtained and amplified with the aid of the Polymerase Chain Reaction (PCR). Following purification, aliquots are printed on coated glass slides. Total RNA from both the test and reference sample is fluorescently labelled with either Cye3-dUTP (green) or Cye5-dUTP (red) using a single round of reverse transcription. The fluorescent targets are mixed and allowed to hybridise to the clones on the array. Laser excitation of the incorporated
targets yields a characteristic light spectrum, which is measured using a laser microscope. Information about the clones, including gene name, clone identifier, intensity values and intensity ratios is attached to each target. Data from a single hybridisation experiment is viewed as a normalised ratio (Cye3 divided by Cye5) in which significant deviations from 1 (no change) are indicative of increased (>1) or decreased (<1) levels of gene expression relative to the reference sample (Duggan et al., 1999; Southern et al., 1999).

Figure 6 - cDNA microarray schema (Duggan et al., 1999)

2.1.2 Expressed Sequence Tags

As a result of the rapid progress on genome sequencing projects (such as the human genome project) thousands of genes have been identified and millions of partial fragments of genes termed expressed sequence tags (ESTs) were sequenced. Most of these genes are only partially characterised and their functions as yet unknown (Alizadeh et al., 2004). ESTs have applications in the discovery of new genes, mapping of the genome and identification of coding regions in genomic sequences (Adams et al., 1991).

In the context of this project, ESTs gain in importance to the probe selection for cDNA microarrays. In addition, the methods developed in this thesis can also be applied to the functional annotation of ESTs. Both is challenging due to high sequence redundancy and
potential cross-hybridisation between paralogous genes. In organisms with limited genomic information, like marine organisms, this challenge is even greater due to annotation uncertainty (Chen et al., 2004). Therefore, databases try to compile genomic sequences into non redundant sets, such as NCBIs UniGene database (Feolo et al., 2000).

### 2.1.3 Nuclear Magnetic Resonance Spectroscopy

Nuclear magnetic resonance (NMR) is a phenomenon which occurs when the nuclei (cores) of certain atoms are immersed in a static magnetic field and exposed to a second oscillating magnetic field. Some atomic cores experience this phenomenon, and others do not, dependent if they possess a so called spin. A spin is a fundamental property of nature (like electrical charge or mass). It comes in multiples of 1/2 and can be + or -. Protons, electrons, and neutrons possess spin. Individual unpaired electrons, protons, and neutrons each possesses a spin of 1/2.

Most of the detectable matters using NMR are molecules. Molecules are composed of atoms. Atoms have a shell and a core which is composed of protons. The protons possesses a property called spin which can be thought of as a small magnetic field, that can be used to cause the core to produce a NMR signal. Not all atomic cores possess a spin. Some of the atomic cores often used in NMR analysis are H (hydrogen), P (phosphor) and Na (natrium).

NMR spectroscopy is the use of the NMR phenomenon to analyse physical, chemical, and biological properties. As a consequence, NMR spectroscopy finds applications in the structural analysis of proteins in molecular biology to predict the three-dimensional structures and dynamic behavior of proteins (Liu and Hsu, 2005). Furthermore, NMR allows the quantification of large amounts of different metabolites in one single experiment (Dunn et al., 2005). These quantifications can be analysed using the OVTK.

### 2.1.4 Mass Spectrometry

Mass spectrometry (MS) has application in both, Proteomics and Metabolomics. It is an instrumental method for identifying the chemical constitution of a substance by separating gaseous ions according to differing in their mass and charge. The principle behind mass spectrometry is as follows: The assay (for example an extraction of intercellular material) is injected into the mass spectrometer, evaporated and ionised under a very high vacuum. The
molecules fragment and produce positive ions, which are passed through a combination of electromagnetic fields and analysed. The analyser uses electrical or magnetic fields, or combination of both, to move the ions from the region where they are produced, to a detector, where they produce a signal which is amplified. Since the motion and separation of ions is based on electrical or magnetic fields, it is the mass to charge ratio and not only the mass, which is of importance.

MS/MS is the combination of two or more mass spectrometry experiments. The aim is either to get structure information by fragmenting the ions isolated during the first experiment, or to achieve better selectivity and sensitivity for quantitative analysis. MS/MS is done: either by coupling multiple analysers or, with an ion trap, by doing various experiments within the trap (Mann et al., 1993).

For example the “Matrix-assisted laser desorption/ionisation-time of flight-mass spectrometry” (MALDI-TOF-MS) is used in analytic routine for protein identification (refer Figure 7). The result of this experimental technique is a mass spectra which is unique for individual proteins. The comparison with known organism-specific protein spectra in a “bio-physical” database allows the identification of proteins contained in the analysed assay (Aebersold and Mann, 2003). These proteins, their interactions and involvements in biological pathways can be analysed using the OVTK.

![Figure 7 - Schematic representation of a MALDI-TOF mass spectrometry (Aebersold and Mann, 2003)](image)

2.1.5 Ion mobility spectrometry

The term ion mobility spectrometry (IMS) refers to the method of characterising chemical substances using gas-phase mobilities of ions in weak electric fields. The drift time of ions formed using ionisation sources and electrical shutters is measured. Ion mobilities are
characteristic for specific substances. For this reason it provides means for the detection and identification of biological substance (especially metabolites). The drift velocity is related to the electric field strength by the mobility, so that the mobility is proportional to the inverse drift time.

![Figure 8 - Schematic illustration of an ion mobility spectrometer (Baumbach et al., 2005)](image)

Ion mobility spectrometry combines both high analytical sensitivity and low technical complexity with high speed data collection. Compared with mass spectrometry (MS), the free path of the ion swarm is smaller than the size of the instrument. Therefore, ions drifting under such conditions experience a separation process which is based on different drift velocities of ions with different masses and geometrical structures. Collections of these ions on a Faraday-plate delivers a characteristic time dependent signal corresponding to the mobility of the arriving ions.

The most important parts of the spectrometer are the ionisation and reaction region and the drift region of the instrument (see Figure 8). The external and homogenous electric field is established within the drift tube using drift rings. The carrier gas takes sample molecules within the ionisation region. The ionisation of the analytes occurs by ionisation on collisions of the analytes with ionised carrier gas molecules or by application of partial discharge or ultraviolet (UV) ionisation sources. A so called drift gas flows from the Faraday-Plate towards the ionisation region. If the shutter is closed, no ions reach the drift region. The drift gas protects the drift region and no neutral analyte molecules should enter the drift region. If the shutter is held closed all analyte molecules are washed out on the gas outlet. During the shutter opening time, some ions enter the drift region.
In contrast to mass spectrometry, during several collisions with the surrounding gas, the sample molecules reach a steady drift velocity. If no chemical reactions occur in the ideal case a total separation is reached at the Faraday-plate.

The time dependent voltage during a time interval measured from the half of the shutter opening pulse is called an ion mobility spectrum (Baumbach et al., 2005; Ruszanyi et al., 2004). The application of IMS in molecular biology is illustrated in Figure 9, Figure 10 and Figure 11. They show IMS spectra used for the identification of emitted substances from different tissues (e.g. lung) and organisms (human and *Escherichia coli*) in air.

---

**Figure 9** - IMS-chromatogram of *human* exhaled air. Inlet: single spectrum at a fixed retention time showing signals of the major constitutions Ammonia, Ethanol and Acetone (Baumbach et al., 2005)

**Figure 10** - IMS-chromatogram (left) of *human* exhaled air of a patient suffering chronic obstructive pulmonary disease (COPD) and two typical single spectra (right). (Baumbach et al., 2005)
The partial high throughput techniques outlined in this chapter are used by research biologists to test a hypothesis with experimentation. Some of them produce a huge amount of data that has to be analysed. This thesis concentrates on the treatment and the handling of experimental results obtained from microarray experiments but the functionality of the approach is analogue to the other techniques mentioned above.

2.2 ONDEX

For the extraction, creation and normalisation of biological networks (graphs) and the identification of mappings (linkages) between the elements of different networks the Datawarehousing system ONDEX (ONtological inDEXing) is used (ONDEX, 2004; Köhler et al., 2005).

To be able to support different application scenarios for the processing of a wide range of life science data originating from external sources, ONDEX consists of two main components for database integration and text mining. These two components are linked through interfaces and data structures that enable them to interoperate and exchange data and methods as illustrated in Figure 12.
First, controlled vocabularies (CVs, refer *Chapter 6.1 Data Structures*, page 37) such as Transpath and Transfac (Wingender, 2004), AraCyc (Mueller *et al*., 2003), Brenda (Schomburg *et al*., 2004), NCBI Taxonomy (Wheeler *et al*., 2003) and MetaCyc (Krieger *et al*., 2004) are imported into the data repository (Data Integration) creating a dataset concept for each biological entity, such as genes, proteins, treatments, transcription factors, metabolites, cell types, organisms etc. and a dataset relation for every connection between two concepts (Rüegg *et al*., 2004). This is done with the aid of “small” parser programs which extract the data from the external data sources (refer *Chapter 7.4 Transpath Parser*, page 68).

The next step is the execution of different mapping algorithms to detect synonymous and homonymous concepts, co-occurrences of concepts and relations, homologous genes in different species etc. For every successful mapping a new linkage (dataset relation) is created and added to the data repository. The effect of these network mappings is the creation of one large graph from the single graphs imported and extracted from the CVs (Rüegg *et al*., 2004).
In the following the database backend of ONDEX is described. Figure 13 shows the Entity Relationship diagram of the ONDEX database (Elmasri and Navathe, 2002) which has been implemented in PostgreSQL (Boenigk, 2003). The most important and already mentioned entities located in the center of the diagram are Concept and Relation. These two entities are connected by the relations (concerning ER diagrams) From_concept and To_concept.

Relations and concepts are typed and for this reason there are entities Concept_class and Relation_type that reference themselves via the relationship Is_specialisation_of (e.g. a Transcription factor is a specialisation of a Protein or the relation Activates is a specialisation of Binds_to). So if this hierarchical organisation is considered on its own it spans a root tree (refer Chapter 2.3 Theoretical Background, page 21).

Furthermore every concept and every relation links to the CV it has been extracted from. There are references Element_of, From_element_of and To_element_of respectively that link the concept item to the entity CV.

Accessions are unique identifiers which unambiguously determine a concept but solely within a specific CV. Concepts might have different names and accessions (e.g. NM or ATG...
numbers) in different databases (CVs). This information is stored in the tables for the entities Concept_name and Concept_accession.

Moreover concepts and relations might have attached values. For example enzymes have a $k_m$ value which determines the affinity of an enzyme to its substrate and a turnover number which determines the enzymes substrate turnover per second. But e.g. a transcriptionsfactor have a position weight matrix (a PWM, which helps to predict its DNA binding sites) and no $k_m$ value (Kel et al., 2004). Thus it is necessary to have a generalised data structure (entity GDS) to store concept or relation specific values in a generalised way. Because of the different data structures and units care must be taken of comparator possibilities ($<$, $>$, $=$) during implementation (Philippi, 2003; ONDEX, 2004; Köhler et al., 2004 and Köhler et al., 2003).

2.3 Theoretical Background

As mentioned earlier, controlled vocabularies, ontologies and graphs are central to the work of this project. For this reason formal definitions (Tollis et al., 1999; Köhler et al., 2003; Köhler et al., 2004; Jungnickel, 1994 and Battista et al., 1994) of these are given below as background to the subject and will be referred to within the rest of this thesis.

**Definition 1:** A graph $G$ consists of a set of vertices and a set of edges, where each edge joins an unordered pair of vertices. The set of vertices of $G$ is denoted by $V(G)$ and the set of edges is denoted by $E(G)$.

$$G = (V(G), E(G)), E(G) = \{(V(G) \times V(G))\}$$

**Definition 2:** Two vertices are said to be adjacent if there is an edge joining them.

**Definition 3:** A vertex $v_i$ is directly reachable from the vertex $v_j$ if they are adjacent.

**Definition 4:** A vertex $v_i$ is reachable from the vertex $v_j$ if there is a path of adjacent vertices that starts with $v_i$ and stops with $v_j$, where a path from $v_i$ to $v_n$ is defined as: $(v_1,v_2), (v_2,v_3), \ldots, (v_{n-1},v_n) \mid v_i \in V(G)$. 
**Definition 5:** A graph is fully connected if every $v_i \in V(G)$ is reachable from every other $v_j \in V(G)$.

**Definition 6:** A graph is
undirected if: $e_k = (v_i,v_j)$, $e_l = (v_j,v_i)$ → $e_k = e_l$
directed if: $e_k = (v_i,v_j)$, $e_l = (v_j,v_i)$ → $e_k \neq e_l$

**Definition 8:** A direct loop is an edge that joins a vertex to itself: $e_k = (v_i,v_i)$.

**Definition 9:** A loop is a path from $v_i$ to $v_i$.

**Definition 10:** A graph is acyclic if there are no loops in $G$.

Figure 14 illustrates an undirected and acyclic graph (with one direct loop) while Figure 15 shows a directed and cyclic graph.

![Figure 14 - Simple undirected and acyclic graph](image1)

![Figure 15 - Simple directed and cyclic graph](image2)

**Definition 11:** A graph is called a tree if it is acyclic and fully connected.

**Definition 12:** A graph is called a root tree if it is a tree extended by a special vertex called root vertex which is reachable from all other vertices in the tree over one and only one directed path, where
directed path from $v_1$ to $v_n$ is defined as: $(v_1,v_2), (v_2,v_3), \ldots, (v_{n-1},v_n) \mid v_i \in V(G)$

Comment: In ONDEX the root vertex of all concept classes would be the *Concept_class Thing* (refer Chapter 2.2 ONDEX, page 18).
3. Related Work

There exist a large number of experimental techniques for characterising molecular biological interactions. This has lead to the development of a variety of software applications to normalise and to analyse the resulting large-scale data sets and to integrate them into one data structure to apply multiple algorithms to them. For these interactions multipurpose graph visualisation libraries and software tools are available such as Tom Sawyer Software Analysis, Visualisation and Layout (http://www.tomsawyer.com) which is used for example by the Center for Biomolecular and Medical Informatics at the University of Massachusetts Lowell (TomSawyer, 2005), yFiles (http://www.yworks.com) which is used e.g. at the Cincinnati Children's Hospital Medical Center, US or the Whitehead Institute for Biochemical Research, US (Wiese et al., 2001; Becker and Rojas, 2001 and Sirava et al., 2001) or PIMrider (http://pim.hybrigenics.com) that beside the visualisation and layouting of graphs additionally links e.g. to the transcription factor database Transfac (Formstecher et al., 2005 and Wingender, 2004).

A number of tools for microarray-based gene expression profiling and other analysis on gene transcriptional processes such as the Affymetrix Microarray Suite and Data Mining Tool (http://www.affymetrix.com/products/software/specific/mas.affx), Silicon Genetics GeneSpring (http://www.silicongenetics.com) or dChip from the Harvard School of Public Health (http://biosun1.harvard.edu/complab/dchip) are available for normalisation, clustering and visualisation (Xu et al., 2004).

Although there is a need for software that combines both, an implementation of an abstract model that allows to integrate parameters and attributes of elements in the graph for layouting as well as filtering (Karp, 2001) and a visualisation of biomolecular interactions. The attributes can be extracted from experimental results such as microarrays.

To meet the above requirements various approaches have been taken; one of the best established solutions is an open source software environment called Cytoscape Shannon et al., 2003. The main strategy of Cytoscape is to display large molecular interaction graphs to identify interesting subgraphs using, for example, gene expression data to reduce the size of the graphs and to scope of the modelling problem to a single subgraph, providing an entry point for lower-level modelling efforts (Shannon et al., 2003).

As demonstrated in Figure 16 it is possible to import a biological network into Cytoscape that is then represented as a graph consisting of nodes (vertices) and edges. Aside from a
visualisation, it is possible to use different layout styles to provide a better overview and to reduce the graph size by applying filters which can make use of additional knowledge (element-specific attributes). These attributes are mapped to elements (vertices or edges) and can be viewed or modified by the help of a Visual Mapper. Annotations are available through a separate server and the functionality of Cytoscape is extendable by the help of a plug-in mechanism to add more functionality.

![Figure 16 - Schematic overview of the Cytoscape Core architecture.](image)

In Cytoscape an attribute is a single parameter of a vertex or an edge whereas an annotation represents a link to a hierarchic classification that specifies the elements in a more general way so that vertices or edges now have types (e.g. “enzyme” or “regulates”). The hierarchical configuration can be extracted e.g. from the Gene Ontology database (GOC, 2001) which stores a controlled vocabulary that can be applied to all organisms for the description of the corresponding cellular components, molecular functions and biological processes. Currently, these data must be externally parsed into annotations or attributes. “One solution to this problem is data federation, in which a relational database management system serves as middleware providing transparent access to a number of heterogeneous data sources.” (Shannon et al., 2003). This is exactly the way that ONDEX and the implemented OVT...
follow in order that OVTK is able to filter a graph using the whole hierarchy of annotations. Furthermore Cytoscape only uses the yFiles library and is not able to profit from the advantages of several other graph libraries.

Another interesting approach to integrate data sources into a large-scale network has been given by Verschelde implemented in the package LinkBase (Verschelde et al., 2004) which helps to extract data from heterogeneous data sources that contain biomedical information such as gene annotations, biological processes or diseases. In contrast to Cytoscape LinkBase is focused on the data integration process by mapping the data sources to a proprietary biomedical ontology that has been developed for the purposes of making computers understand medical natural language. LinkBase disregards the visualisation aspect.

It comprehends various aspects of medicine that are represented via concepts inter-connected by typed relations (similar to ONDEX). LinkBase contains 543 relation types. They are divided into different groups, including spatial, temporal, and process-related link types. Unlike ONDEX, LinkBase does not provide concept classes or anything equivalent. Concepts and relations are language independent. They are cross-referenced to over 3 billion terms in 16 supported languages. Terms can be linked to concepts, criteria, and relation types via an intersection table which allows to define homonyms (single terms that have different meanings: criterion/concept/link type) and synonyms (multiple terms associated with one single meaning). The principle is the same as in ONDEX. With the help of these mapping procedures additional linkages between concepts or relations of the imported data sources can be created. Contrary to ONDEX no sequence data of genes are used during mapping. Another advantage of ONDEX and the OVTK when compared to LinkBase, are the graph visualisation and analysis methods that are developed in this thesis (especially in the context of experimental results).

The visualisation and the analysis of the integrated data are very important to assist scientific research. Chapter 4.1 Large Graphs (page 27) illustrates that graphs extracted from biological data sources can be very large (substantial numbers of vertices and edges) so that one difficulty is finding an appropriate presentation in the form of a graph (layouting). In the literature there are a number of layout algorithms available (Adai et al., 2004; ONDEX, 2004; Chalmers, 1996; Hobbs and Rodgers, 1998 and Han et al., 2004). Many implementations of these algorithms can be downloaded from the internet (Wiese et al., 2001; Becker and Rojas, 2001; Sirava et al., 2001; Tom Sawyer, 2005 and O'Madadhain et al., 2004). As illustrated in Chapter 7. Implementation (page 55) the OVTK makes use of the JUNG and the yFiles graph
libraries which already have many layout algorithms implemented (O'Madadhain et al., 2004; Sirava et al., 2001 and Becker and Rojas, 2001). Furthermore for OVTK a very fast linear runtime layout algorithm has been developed and implemented, known as the FastCircularLayout which is based on the general idea of circular layouts implemented for example in the yFiles (Wiese et al., 2001) or the Tom Sawyer (TomSawyer, 2005) libraries.
4. Requirement Analysis

In this chapter the requirements necessary to design a software which is able to handle the very big integrated graphs, to visualise, to layout and to filter them will be presented. In addition it deals with the requirement to have a generic architecture which allows to profit from several already implemented graph libraries and with the demand to validate the approaches with the aid of real experimental results.

4.1 Large Graphs

As illustrated in Table 2, graphs derived from ONDEX can be very large because ONDEX combines a number of databases (CVs). Moreover the size of the graphs increases due to the additional relations added by the mapping software during the data warehousing process of the ONDEX suite.

<table>
<thead>
<tr>
<th>Biological database (CV):</th>
<th>Concepts:</th>
<th>Relations:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell Ontology</td>
<td>679</td>
<td>937</td>
</tr>
<tr>
<td>Enzyme Nomenclature Committee</td>
<td>4,606</td>
<td>4,600</td>
</tr>
<tr>
<td>Drastic Insight Database</td>
<td>5,438</td>
<td>9,007</td>
</tr>
<tr>
<td>Biobase Transfac</td>
<td>13,945</td>
<td>9,179</td>
</tr>
<tr>
<td>Gene Ontology</td>
<td>16,704</td>
<td>23,308</td>
</tr>
<tr>
<td>AraCyc</td>
<td>18,909</td>
<td>18,424</td>
</tr>
<tr>
<td>MetaCyc</td>
<td>21,345</td>
<td>34,384</td>
</tr>
<tr>
<td>Medical Subject Headings Descriptors</td>
<td>22,568</td>
<td>39,681</td>
</tr>
<tr>
<td>Biobase Transpath</td>
<td>45,243</td>
<td>59,825</td>
</tr>
<tr>
<td>NCBI Taxonomy</td>
<td>226,076</td>
<td>226,076</td>
</tr>
<tr>
<td>Brenda</td>
<td>286,983</td>
<td>691,799</td>
</tr>
<tr>
<td>KEGG</td>
<td>2,170,188</td>
<td>3,073,502</td>
</tr>
<tr>
<td>Sum of all CVs</td>
<td>2,832,684</td>
<td>4,190,722</td>
</tr>
</tbody>
</table>

Large graphs (with more than 20,000 concepts and more than 20,000 relations) are very complex and can not easily be arranged in a clear way for the user to see the connectivity. Moreover, merely displaying such large graphs can take hours of CPU. Scrolling and exploring the graph in a window, labelling the concepts and relations and colouring relations and/or
concepts increases the complexity and massively increases the cpu requirements. Features such as antialiasing or ‘fit to content’ increases the user idle time additionally. Figure 17 illustrates a graph imported from ONDEX that shows the Biobase Transfac and Transpath data after import into ONDEX (Wingender, 2004). It becomes clear that large graphs do not enable the user to gain any great understanding other than one of complexity.

![Figure 17 - Large graph, extracted from Biobase Transpath and Transfac (Wingender, 2004), layouted orthogonal with the aid of the yFiles (Wiese et al., 2001)](image)

To improve this a major requirement is the ability to handle large graphs in an efficient manner.

### 4.2 Graph Layout

The layout of a graph is the placement of nodes and edges of a graph in a two- or three-dimensional space. There are many layout algorithms and methods known from the literature, which are implemented in various graph libraries using different programming languages. Due to the platform independence of Java it was decided to concentrate on graph libraries developed for this object-oriented programming language. The following figures (Figure 18 -
Figure 20) show the same graph after applying different layout algorithms displayed in the same visualisation style (using the yFiles graph library; Wiese et al., 2001).

Figure 18 - A graph organically layouted

Figure 19 - The graph from Figure 18 hierarchically layouted
What is the best layout in the particular context and situation? Because of context sensitivity for the choice of a sensible layout style the only workable solution is to provide as many layout algorithms as possible. Furthermore, it is important to make sure that the application of these layouts is fast enough when drawing large graphs and to make it easy to switch to different layout styles.

The main problem is the high idle time during the layout process which leads to problems in switching between different layouts at runtime. On account of this, another requirement is to find out which Java graph libraries exist and to evaluate them to find those that are efficient and provide a breadth of algorithms. It will also be necessary to develop new algorithms with a special attention to efficiency where these are needed.

### 4.3 Visualisation

Visualisation is colouring, shape setting and the labelling of nodes and edges together with antialiasing techniques to improve understanding. Also included is a set of tools for exploring a graph (scrolling, zooming and automatic navigation). Visualisation is not layouting and is independent of how the nodes and edges are arranged (their coordinates in display space). Figure 21 shows four different visualisations created by three different Java graph libraries of four different graphs.

Biological networks and pathways may have a number of different elements, element types, relationships etc. Therefore it is important that many abilities to present the graph are provided in a system (for example to apply different shapes to enzymes, genes, transcription factors etc.) to increase clarity for the user.
Figure 21 - Examples of different visualisations by different Java graph libraries - top left: BioLayout (Goldovsky et al., 2004) - top right: GINY (Bähr et al., 2004) - bottom left and right: yFiles (Wiese et al., 2001)

4.4 Generic Architecture

In the framework of this thesis a number of Java graph libraries have been evaluated. The results will be discussed in Chapter 5. Graph Library Evaluation (page 33) in detail. One can see that the abilities of the different libraries are extremely varied. Large performance and feature differences within and between the graph library packages exist. Anticipating the need to exploit different components from several graph libraries and the need to easily adopt new products as they become available, the best way is to try and integrate more than one library into the proposed system so that visualisation, layout and filtering can be done selectively using different graph libraries or a newly designed software. Thus we have the additional
requirement for a generic architecture which allows the use of a diverse range of graph libraries.

### 4.5 Filters

As already discussed in the *Chapter 4.1 Large Graphs* (page 27) biological networks represented through graphs can be very large and difficult to handle. To increase speed, decrease memory load and to provide a visualisation with an increased clarity, it is essential to hide nodes and edges from the graph that do not play a role in the considered context. This can be accomplished through the use of filters to take specific sub-views of the data underlying the graph.

It should be possible to apply one or more of the following filters in combination:

- hide all concepts of a specific concept class (including all specialisations if needed).
- show all concepts of a specific concept class (including all specialisations if needful).
- hide all relations of a specific relation type (including all specialisations if needful).
- show all relations of a specific relation type (including all specialisations if needful).
- hide all concepts with no connections to other concepts.
- set node size and colour according to specific experimental results (e.g. to expression levels given by a microarray result).
- find shortest paths between the highlighted elements and hide other elements.
- show n-neighbourhood of highlighted elements and hide other elements.

All these filters should lead to a simplification of the view of the initial full graph and focus the visible elements to the specific context of interest. Thus, this additional requirement is the ability to apply filters on the graph.

### 4.6 Validation of Approach

As discussed above, filters can be used to show subgraphs of ONDEX data in the context of experimental results to increase clarity enabling the user to concentrate on the specific biological question.

The approaches described in the previous chapters need to be validated with a practical data interpretation task using real data extracted from an experiment.
5. Graph Library Evaluation

Before designing and developing new algorithms the available libraries have to be evaluated to see how they work and if they are able to visualise, layout and filter biological graphs. This work has been done in the framework of this thesis. Criteria have been established for the evaluation. These are arranged in two categories: Knock out (k.o.) criteria and other criteria. If a library did not fulfill a k.o. condition the evaluation of this library was stopped as it would not fulfill enough conditions for use in OVTK.

Knock out criteria:
1. Support of directed and undirected graphs.
2. Ability to append objects (labels, colors, icons, references to Java objects) to nodes and edges.
3. Support of layout algorithms.
4. The software is available as a library, that can be embedded in this development.

Other criteria:
5. Source code available.
6. Ability to handle large graphs.
7. License type.
8. Availability of support.

The following list (Rügg, 2004) contains all Java graph libraries which have been evaluated:

a. BioLayout (http://maine.ebi.ac.uk:8000/services/biolayout)
b. gef (http://gef.tigris.org)
c. GINY (http://csbi.sourceforge.net)
d. GraphMaker (http://www.bluemarsh.com/java/graphmaker)
e. Grappa (http://www.research.att.com/~john/Grappa)
f. Hypergraph (http://hypergraph.sourceforge.net)
g. Interactive Graph Drawing (http://www.cs.rpi.edu/research/groups/pb/graphdraw)
h. InterViewer (http://interviewer.inha.ac.kr)
i. JDigraph (https://jdigraph.dev.java.net)
j. jGABL (http://www.math.tu-berlin.de/jGABL)
To compare the evaluation results a table (Table 3) has been created in which every line represents a library and every column a condition that indicates compliance (“X”), non compliance (“-“) or not evaluated (“n.d.”). In the column for criteria 7 the entries represent the specific license type of the library. In the column for criteria 8 (Availability of support) an “X” means no detailed support whereas sometimes the specific number of developers is given.
Table 3 - Graph library evaluation, rows: libraries, columns: criteria (X = criteria complied, “-“ = criteria not complied, n.d. = no data, os = open source, bsd = under BSD license, gpl = under GPL license, lgpl = under LGPL license, com = commercial)

<table>
<thead>
<tr>
<th></th>
<th>1.</th>
<th>2.</th>
<th>3.</th>
<th>4.</th>
<th>5.</th>
<th>6.</th>
<th>7.</th>
<th>8.</th>
</tr>
</thead>
<tbody>
<tr>
<td>b.</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>n.d.</td>
<td>bsd</td>
<td>X</td>
</tr>
<tr>
<td>d.</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>n.d.</td>
<td>gpl</td>
<td>1 developer</td>
</tr>
<tr>
<td>e.</td>
<td>X</td>
<td>n.d.</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>n.d.</td>
<td>os</td>
<td>X</td>
</tr>
<tr>
<td>g.</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>n.d.</td>
<td>X</td>
<td>n.d.</td>
<td>Os</td>
<td>2 developers</td>
</tr>
<tr>
<td>h.</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>n.d.</td>
</tr>
<tr>
<td>i.</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>n.d.</td>
<td>Os</td>
<td>n.d.</td>
</tr>
<tr>
<td>j.</td>
<td>n.d.</td>
<td>n.d.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>n.d.</td>
<td>lgpl</td>
<td>1 developer</td>
</tr>
<tr>
<td>k.</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>n.d.</td>
<td>Os</td>
<td>X</td>
</tr>
<tr>
<td>l.</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>n.d.</td>
<td>gpl</td>
<td>n.d.</td>
</tr>
<tr>
<td>m.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Os</td>
<td>4 developers</td>
</tr>
<tr>
<td>o.</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>n.d.</td>
<td>lgpl</td>
<td>1 developer</td>
</tr>
<tr>
<td>p.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>bsd</td>
<td>4 developers</td>
</tr>
<tr>
<td>q.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>com</td>
<td>X</td>
</tr>
<tr>
<td>r.</td>
<td>n.d.</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>os</td>
<td>1 developers</td>
</tr>
<tr>
<td>s.</td>
<td>n.d.</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>os</td>
<td>-</td>
</tr>
<tr>
<td>u.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>com</td>
<td>X</td>
</tr>
<tr>
<td>v.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>n.d.</td>
<td>X</td>
<td>-</td>
<td>n.d.</td>
<td>2 developers</td>
</tr>
<tr>
<td>w.</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>n.d.</td>
<td>os</td>
<td>X</td>
</tr>
</tbody>
</table>

As illustrated in Table 3 only four libraries (blue underlayed) fulfill all conditions:

1. yFiles (commercial with education license)
2. Tom Sawyer (commercial with education license)
3. JUNG (open source)
4. Prefuse (open source under bsd license)

To find out which is the best of these four libraries, the provided layout algorithms were compared and these are detailed in Table 4. The evaluation has been done on a Computer with two AMD 64 Bit CPUs with a clock frequency of 3 Ghz, 8 GB RAM and 500 GB of free disk space and a graph extracted from WordNet with the help of the ONDEX PostgreSQL database with 20 000 edges and 22 054 responding nodes.
### Table 4 - Graph libraries and the idle times while layouting

<table>
<thead>
<tr>
<th></th>
<th>Circular</th>
<th>Hierarchical</th>
<th>Symmetric</th>
<th>Organic</th>
</tr>
</thead>
<tbody>
<tr>
<td>yFiles</td>
<td>0m 2s</td>
<td>0m 3s</td>
<td>2m 6s</td>
<td>2m 15s</td>
</tr>
<tr>
<td>TomSawyer</td>
<td>0m 2s</td>
<td>1m 29s</td>
<td>3m 2s</td>
<td>2m 24s</td>
</tr>
<tr>
<td>JUNG</td>
<td>0m 4s</td>
<td>5m 23s</td>
<td>15m 43s</td>
<td>11m 33s</td>
</tr>
<tr>
<td>Prefuse</td>
<td>0m 2s</td>
<td>&gt; 60m</td>
<td>8m 13s</td>
<td></td>
</tr>
</tbody>
</table>
6. Principles and Methods

This chapter describes the developed data structures, which are extensions to the given basic data structures defined in Chapter 2.3 Theoretical Background (page 21), the selected generic architecture and some algorithms developed in this project.

6.1 Data Structures

Firstly it is necessary to decide which type of graph is required. For example because of the modelling of self-regulated genes in a pathway the resulting graphs might be cyclic and for recently discovered genes or treatments the graph does not have to be fully connected. In addition we need directed as well as undirected graphs (a transcription factor activates a gene but a ligand binds to a protein and the protein to the ligand). See Figure 22 for illustration.

![Graph diagrams](image)

Figure 22 - Simple cyclic graph (top), not fully connected graph (middle), directed and undirected mix graph (bottom)

In the following, concepts are the vertices (nodes) in a graph and relations are the edges.
Apart from this, concepts and relations are typed and built hierarchically in sub- and super-classes (see also Chapter 2.2 ONDEX, page 18). Figure 23 illustrates a simple example for the organisation of concept classes containing enzymes, transcription factors and proteins.

![Figure 23 - Concept Classes are built hierarchically (tree).](image)

To visualise a graph and apply filters and layouts we need a precise definition of a visible graph. As already mentioned concepts and relations are linked to their specific concept class (or relation type) but they have many more values attached: an id, a name, synonyms, their accessions, a sequence and a structure (if known), a link to the database where the element has been extracted from, a color, their x- and y-coordinates, a flag for their visibility and a size. The simplified graph in Figure 24 shows this enhancement by using only the x- and y-coordinates, the ids and the concept classes.

![Figure 24 - Simple graph with attached values](image)

For the knowledge representation of the data in the database of ONDEX, ontologies are used and defined for the usage as a basic principle for visualiseable graphs.
**Definition 13:** An integrated ontology is a 12-tuple $O(C, R, CA, CV, CC, RT, P, ca, cv, cc, rt, id)$ that consists of:

- a finite, not empty, distinct set of Concepts $C(O)$
- a finite, not empty set of Relations: $R(O) \subseteq C(O) \times C(O)$ [refer Definition 1]
- a finite set of Concept Accessions $CA(O)$
- a finite, not empty set of Controlled Vocabularies $CV(O)$
- a tree consisting of Concept Classes $CC(O)$ [refer Definition 12]
- a tree consisting of Relation Types $RT(O)$ [refer Definition 12]
- the function $ca$ which assigns concept accessions to concepts:
  - $ca$: $C(O) \rightarrow (ca_1 \times \ldots \times ca_n), ca_j \in CA(O)$
- the totally defined functions $cv, cc, rt$ that assign CVs, concept classes and relation types to concepts or relations:
  - $cv$: $C(O) \cup R(O) \rightarrow CV(O)$
  - $cc$: $C(O) \rightarrow V(CC(O))$ [refer Definition 1]
  - $rt$: $R(O) \rightarrow V(RT(O))$ [refer Definition 1]
- the bijective function $id$ which assigns an unique identifier to every concept and every relation with:
  - $id$: $C(O) \rightarrow N$

- and the additional properties of an ontology $P(O)$ consisting of:

  - a finite set of Concept Names $CN(O)$
  - a finite set of Sequences $SEQ(O)$
  - a finite set of Structures $STR(O)$
- and the functions $cn$, $seq$ and $str$ that optionally link concept names, aminoacid/nucleotide sequences and protein structures to concepts:
  - $cn$: $C(O) \rightarrow (cn_1 \times \ldots \times cn_n) | cn_j \in CN(O)$
  - $seq$: $C(O) \rightarrow SEQ(O)$
  - $str$: $C(O) \rightarrow STR(O)$

In order to be able to layout and display an ontology the structure given in Definition 13 has to be extended accordingly. This extension is given below with the definition of a visible graph.
**Definition 14:** A visible graph $G$ is a 7-tuple $G(O, CO, colour, size, visibility, x, y)$ that consists of:

- an integrated ontology $O$
- a finite, not empty set of Colours $CO(G)$
- the functions $co$ (color), $s$ (size), $v$ (visibility), $x/y$ (coordinates) which affect the way concepts and relations are visualised:
  - $co: C(O) \cup R(O) \rightarrow CO(G)$
  - $s: C(O) \rightarrow R$
  - $v: C(O) \cup R(O) \rightarrow \{true, false\}$
  - $x: C(O) \rightarrow R$
  - $y: C(O) \rightarrow R$

Based on this extended definition, a filter and a layout can be defined straight forward.

**Definition 15:** A filter is a function $f(G): G \rightarrow G$ that replaces the functions $co$, $s$ and $v$ of the graph $G$.

**Definition 16:** A layout is a function $g(G): G \rightarrow G$ that replaces the functions $x$ and $y$ of the graph $G$.

The above definitions of visible graphs as well as filters and layouts are the formal basis of graph based analysis and visualisation (Köhler et al., 2005). The next step is to transform these formal definitions into an object oriented data model which can be used to generate data structures in an object-oriented (programming) language. To illustrate how to achieve a suitable data structure based on Definition 14 we start at the ONDEX ER diagram (see also Chapter 2.2 ONDEX, page 18) and extract a general class diagram (Figure 25).

As can be seen in Figure 26, the structure of the class diagram is very similar to the ONDEX ER diagram. It can be used for the implementation of the necessary functionality to generate, filter, layout and visualise an ONDEX graph.
Figure 25 - Data structure (overview) with underlying ONDEX ER diagram (Köhler et al., 2004), see page 20

Figure 26 - Data structure (overview)
6.2 Architecture

To comply with the requirement that several graph libraries may be needed for full visualisation and to minimise the overhead for integrating new filter and layout algorithms, the generic modular architecture of the ONDEX Visualisation ToolKit (OVTK) component of ONDEX is centered on an internal representation of graphs that is independent of any graph library (Figure 27). The Internal Graph Object is therefore handled by the following interfaces and adapters:

a) The import interface (Interface to ONDEX) is used for initializing the Internal Graph Object by importing data from the relational database back-end of ONDEX. In order to minimise the amount of data to import, and thus to reduce the size of resulting graphs as well as associated costs such as network traffic, memory utilisation and run-time of the analysis.
algorithms, only user-specified subsets of the data sources are imported from ONDEX. The selection of data to import for graph based analysis and visualisation by the user is based on criteria such as source database, concept class, taxonomy etc.

b) The export interface (not display in Figure 27) converts the Internal Graph Object for instance into an XML representation for data exchange with external software packages. This is used for example to export graph objects as executable Petri Nets to Cell Illustrator (Doi et al., 2003).

c) The layout interface (GeneralLayouter) allows to make use of layout algorithms available in different graph libraries mapped through an Internal Graph Object. Since every graph library has its own data structure for graph representation, implementations of this interface translate the Internal Graph Object into these graph library specific representations. After the application of a layout algorithm to a graph, the resulting coordinates for its elements are transferred back to the Internal Graph Object by the respective interface implementation. Custom layout algorithms also implement the layout interface, but usually work directly on the Internal Graph Object.

d) The graph library adapter (GeneralGraphLibraryAdapter) is used to encapsulate different graph libraries for the purpose of painting. This enables different graph libraries to be used for layouting (calculation of x and y coordinates) and for painting (displaying the graphs on the computer screen). As already mentioned, there are significant differences in the performance and quality of layout algorithms and graph visualisations (see section 5. Graph Library Evaluation, page 33). Whereas in small graphs the performance for painting the graphs on the canvas can be neglected, when dealing with large graphs and layout algorithms such as the FastCircleLayout (see page 45), the performance bottleneck shifts from layouting to painting.

e) Various filters were developed which can be freely combined by applying them consecutively. In this context the filter interface (GeneralFilter) provides a common infrastructure for the integration of filtering algorithms. Most of these filters are relatively simple: yet they are very powerful, because they exploit the rich information that is associated with the data structures. Such filters operate, for instance, on concept classes (CC), relation types (RT), branching factor of concepts or by combinations of these filters.
In summary, Graph analysis and visualisation with the OVTK works on an Internal Graph Object which may be connected to arbitrary graph libraries as well as layout and filter algorithms by means of several interfaces and adapters. With this architecture a graph is generated from data imported from the ONDEX backend and subsequently passed to algorithm independently of its origin. The results of the application of an algorithm are transferred back into the Internal Graph Object which then may be processed again by the available filter and layout algorithms. In this manner, arbitrary graph analysis and visualisation processes are supported in order to provide the user with a wide range of possibilities for his specific investigation and exploration of the data in ONDEX and how it relates to his experimental data.
6.3 Algorithms

The development of application-specific layout and filter algorithms can be used to exploit the semantically rich information that is held in the database integration component. These can be used to increase efficiency and to present information in application specific ways that will be more clear and therefore more readily interpretable by users.

The following algorithms are described in pseudo code which is based on the data structures presented in Chapter 6.1 Data Structures, page 37.

6.3.1 FastCircularLayout

![Figure 28 - FastCircularLayout applied on a subgraph extracted from Biobase Transfac and Transpath (Wingender, 2004), AraCyc (Mueller et al., 2003) and Drastic Insight Database (Newton et al., 2002)](image)

One efficient layout algorithm is the so called FastCircularLayout. This separates all visible concepts by their concept class and arranges them in discrete circles which are evenly distributed over a given circular area (Figure 28). In contrast to the layout algorithms available in the “off the shelf” graph libraries, the FastCircularLayout can, due to its linear time efficiency, also layout very large graphs consisting of several millions of elements in acceptable times on a desktop PC. The algorithm is described in the following pseudo code.
**Algorithm:**

\[ \text{bigR} \leftarrow \text{radius of big circle, smallR} \leftarrow \text{radius of small circle} \]

FastCircularLayout(O, bigR, smallR):

\[ \text{usedClasses} \leftarrow \text{new list} \]

for every class \( \in \text{CC}(O) \) do

\[ \text{ac} \leftarrow x \mid x: \text{cc}(x) = \text{class} \]

if \( \text{ac.size} > 0 \) do

\[ \text{usedClasses.add}(\text{class}) \]

\[ \text{bigAngle} \leftarrow (2\pi) / \text{usedClasses.size} \]

\[ \text{bigCircleNumber} \leftarrow 0 \]

\[ \text{lastX} \leftarrow 0 \]

\[ \text{lastY} \leftarrow 0 \]

for every class \( \in \text{CC}(O) \) in \text{usedClasses} do

\[ \text{actualX} \leftarrow \text{bigR} \times \cos(\text{bigCircleNumber} \times \text{bigAngle}) \]

\[ \text{actualY} \leftarrow \text{bigR} \times \sin(\text{bigCircleNumber} \times \text{bigAngle}) \]

\[ \text{ac} \leftarrow x \mid x: \text{cc}(x) = \text{class} \]

\[ \text{smallAngle} \leftarrow (2\pi) / \text{ac.size} \]

\[ i \leftarrow 0 \]

for every \( c \in \text{C}(O) \) in \text{ac} do

\[ i \leftarrow i++ \]

\[ x(c) \leftarrow \text{actualX} + \text{smallR} \times \cos(i \times \text{smallAngle}) \]

\[ y(c) \leftarrow \text{actualY} + \text{smallR} \times \sin(i \times \text{smallAngle}) \]

\[ \text{bigCircleNumber} \leftarrow \text{bigCircleNumber}++ \]
Efficiency:

In all cases all concepts classes and concepts in the graph have to be explored which leads to a linear time cost function:

\[ m = |CC(O)| \text{, } n = |C(O)| \]

first loop: \( O(m) \)

second loop:
  - outer loop: \( O(m) \)
  - inner loop: \( O(n - \text{Concepts handled before}) \rightarrow O(n/m) \) [if considered all loops]

\( \rightarrow \) both loops: \( O(m) + O(m)O(n/m) = O(m) + (mn/m) = O(m + n) \)

6.3.2 MicroArrayFilter

The application-specific microarray filter makes use of the types of the relations as domain knowledge for efficient data processing. The first step of the MicroArrayFilter is to set the size and colour of concepts according to the expression levels given by a microarray result (illustrated in Figure 29 and Figure 30). The concepts to be processed are identified by a comparison between all concept accessions \( CA(O) \) and all SpotIDs presented in the microarray result file as well as the relation between concepts and their accessions (function \( ca \)). On being applied, the MicroArrayFilter only shows concepts within a specified distance from concepts which are contained in the microarray result by the process of recursively expanding concepts. The outcome of this filter operation is an accumulation of highlighted concepts in the context to their expression level and their surrounding neighbourhood (the analysis is stopped if two neighbourhoods are brought into contact). The details of the MicroArrayFilter are illustrated with the following abstract example and its pseudo-code description afterwards.
Example:

Figure 29 shows a simple abstract graph.

As mentioned, the first step of the MicroArrayFilter is to set the concept size and colour according to the expression levels given by a microarray result (Figure 30).

The next step is to hide all relations and to hide all that concepts in the graph which are not colourised. This leads to an invisible graph (dashed elements) excepting that concepts which are disregulated in the microarray experiment (Figure 31).
The MicroArrayFilter unhides all concepts within the specified distance of 2 (in this example) from concepts which are contained in the microarray result by the process of recursively expanding concepts (neighbourhood propagation, Figure 32 and Figure 33).

Figure 32 - Step 3: Neighbourhood propagation (for the red concept): Show all elements in the neighbourhood of the red concept with distance $\leq 2$ and stop recursive process earlier if a visible concept is reached. Note that the distances from the red concept are written into the concepts in its neighbourhood.

Figure 33 - Step 4: Neighbourhood propagation for the green concept
As mentioned, the outcome of the filter process is an accumulation of highlighted concepts in the context to their expression level and their surrounding neighbourhood. For a better overview of the results, the invisible (dashed) elements have been deleted from the graph (Figure 34).

![Figure 34 - Result: The graph from Figure 29 after application of the MicroArrayFilter.](image)

Two paths between the green and the red node (brown underlayed) can be seen easily in the filtered graph (Figure 35).

![Figure 35 - There are two paths (brown) between the green and the red node.](image)
Algorithm:

\[\text{mList} \leftarrow \text{list of MicroArrayResults (where the elements in the list are called a spot with an associated id [spot.spotid] and an expression level [spot.expressionLevel])}\]

\[\text{cut} \leftarrow \text{cutOff value} \quad // \text{for neighbourhood propagation}\]

\[\text{downCo} \leftarrow \text{green} \quad // \text{downRegulatedColor}\]

\[\text{upCo} \leftarrow \text{red} \quad // \text{downRegulatedColor}\]

\[
\text{function MicroArrayFilter(O, mList, cut, downCo, upCo):}\n\]

\[\quad \text{for all } c \in \text{C(O) with } v(c) = \text{true do}\]

\[\quad \quad v(c) \leftarrow \text{false}\]

\[\quad \text{inBoth} \leftarrow \text{new list}\]

\[\quad \text{for every spot } \in \text{mList do}\]

\[\quad \quad \text{for every } c \in \text{C(O) with spot.spotid } \in \text{ca(c) do}\]

\[\quad \quad \quad \text{inBoth.add}(c)\]

\[\quad \quad \quad s(c) \leftarrow \text{spot.expressionLevel}\]

\[\quad \quad \quad \text{if spot.expressionLevel } < 0 \text{ do}\]

\[\quad \quad \quad \quad \text{co}(c) \leftarrow \text{downCo}\]

\[\quad \quad \quad \text{else}\]

\[\quad \quad \quad \quad \text{co}(c) \leftarrow \text{upCo}\]

\[\quad \quad \text{for every } c \in \text{inBoth do}\]

\[\quad \quad \quad \text{propagateNeighbours}(c, \text{null, null, cut})\]

\[
\text{function propagateNeighbours(c, class, type, cut):}\n\]

\[\quad \text{if } v(c) = \text{false and cut } > 0 \text{ do}\]

\[\quad \quad v(c) \leftarrow \text{true}\]

\[\quad \quad \text{for all } r \in \text{R(O) with } r = (s, c) \mid s \in \text{C(O) do}\]

\[\quad \quad \quad \text{if rt}(r) \neq \text{type and cc}(c) \neq \text{class do}\]

\[\quad \quad \quad \quad \text{propagateNeighbours}(s, \text{cc}(c), \text{rt}(r), \text{cut}--)\]

\[\quad \quad \text{for all } r \in \text{R(O) with } r = (c, t) \mid t \in \text{C(O) do}\]

\[\quad \quad \quad \text{if rt}(r) \neq \text{type and cc}(c) \neq \text{class do}\]

\[\quad \quad \quad \quad \text{propagateNeighbours}(t, \text{cc}(c), \text{rt}(r), \text{cut}--)\]
**Efficiency:**

The worst case is a graph where all concepts are direct neighbours (or in the n-neighbourhood of every other concept with cut > n) and if all concepts are assigned to a spotID from the microarray file (m = n).

\[ m = |mList|, \quad n = |C(O)|, \quad \text{with} \quad m \leq n \]

**first loop:** \( O(n) \)

**second loop:**
- outer loop: \( O(m) \)
- inner loop: \( O(n) \)

\( \rightarrow \) both (first and second) loops: \( O(n) + O(m)O(n) = O(n+mn) = O(n(1 + m)) = O(nm) \)

**function:** \( O(n) \)

**third loop:** \( O(m) \)

all 3 loops: \( O(nm) + O(m)O(n) = O(nm) + O(nm) = O(2nm) = O(nm) \)

with \( m \leq n \): \( O(n^2) \)

In practice, a cutOff value of 4 leads to good results. Applying the filter took between a few seconds up to a few minutes even when used in the context of several large integrated databases such as Aracyc, Brenda and Transpath.
6.3.3 HideConceptClassFilter

The HideConceptClassFilter is simple: To provide a better overview it is useful to remove all concepts of a specific class and all outgoing and incoming relations from or to these concepts from view (Figure 36 and Figure 37).

Figure 36 - Subgraph of Biobase Transfac and Transpath (Wingender, 2004), AraCyc (Mueller et al., 2003) and Drastic Insight Database (Newton et al., 2002) layouted in FastCircularLayout before applying the HideConceptClassFilter

Figure 37 - The same graph after applying the HideConceptClassFilter on the Concept class “Treatment”
Algorithm:

class $\leftarrow cc \in CC(O)$

FastCircularLayout(O, class):
    for every $c \in C(O)$ with $cc(c) = \text{class}$ do
        $v(c) \leftarrow \text{false}$

Efficiency:

The worst case is a graph where all concepts are of the same class and have to be hidden.

$n = |C(O)|$

$\Rightarrow O(n)$
7. Implementation

This chapter concentrates on the implementation of the presented formal approaches in Java. To illustrate how the ONDEX Visualisation ToolKit (OVTK) works and how it is implemented, the UML class diagram of the Internal Graph Object (created with Poseidon for UML Community Edition) is used as well as the basic interfaces GeneralFilter, GeneralLayouter and GeneralGraphLibraryAdapter (see also Chapter 6.2 Architecture, page 42). Following this, an example is given to describe a short workflow through the application possibilities of the OVTK. This is followed by a description of the implementation of a Biobase Transpath parser that is used within the data integration process of ONDEX and finally the system requirements of the OVTK are detailed.

The class names and the main structure of the Internal Graph Object is strongly oriented on the ONDEX ER diagram and the discussed data structures (see Chapter 6.1 Data Structures, page 37).

The classes Hashtable, Vector, JComponent, Jpanel and JFrame are part of the Java Development Kit (JDK) and their descriptions and specifications can be found in the Java documentation at http://java.sun.com/j2se/1.4.2/docs/api (Deitel, 2005).

7.1 UML Class Diagram

Figure 38 shows the UML class diagram of the Internal Graph Object and all of its referring elements. The core is a class named ONDEXGraph referencing all concepts and relations in the imported graph by the help of hashtables (which have an almost constant access time) and an object of type GeneralGraphData which links (through hashtables) to other informations according to the needs of the graph such as concept classes, relation types, concept accessions, concept names, mapping methods, CVs, sequence types, format types and attribute names.
Figure 38 - UML class diagram of the Internal Graph Object of the OVTK
Objects of the type *Concept* represent concepts and these reference their concept class (class *ConceptClass* which links to its specialisation *ConceptClass* and via a list, so called vectors to the attended concepts). Furthermore, concepts might have nucleotide or amino acid sequences (e.g. if they represent genes or proteins) or a structure (if they represent proteins). They reference to classes *Sequence* and *Structure* that will be described in detail below. In addition concepts store from which data source they are imported through the link to an object of the type *CV*.

Concepts as well as relations may have attached values e.g. $k_m$-values, turnover numbers or $v_{max}$-values (for example enzyme representing concepts). To handle these values in an abstract generalised way, would have the advantage that it is possible to apply filters to the graph using them. A Generalised Data Structure (GDS) which stores all these values in an abstract way is used. The *GDS* objects are categorised in so called attribute names to whose they link to (class: *Attribute_Name*). They store classifying data such as datatype and unity of the referencing *GDS* and implement required methods such as *greaterThan*, *smallerThan* and *equals*.

Because of heterogeneous names and accessions (identifiers) in different databases (CVs) two classes *Concept_Acc* and *Concept_Name* are used which are referenced by the specific concepts. One concept can have more than one concept accession or concept name in different databases. For this reason every concept references its concept accessions and names (hashtable) and they in turn link to the *CV* they are annotated in.

Sequences and structures are special values that might be attached to concepts as mentioned above. They are typed and reference their *Sequence_Type* or *Format_Type*. Sequence types link to their specialisation, similar to concept classes and relation types.

Due to the fact that nodes in a graph are represented by concepts, they might have incoming and outgoing edges (relations) which connect two concepts. To accommodate this, the class *Relation* is used whose instantiated objects reference to a source concept (*from_concept*), a destination concept (*to_concept*) and the *CVs* this concepts arise from. Every concept stores in a hashtable the incoming and the outgoing relations to speed up filtering and layout processes.

In addition, relations are typed and they link to their relation type (class: *Relation_Type*) which in a manner similar to concept classes, references its specialisation relation type and its attended relations. To remember the reason why the relation was created, the mapping method is stored and referenced as well.

All these objects are created and can be reached over the class *ONDEXGraph* which is the Java class equivalent to the visible graph from Chapter 6.1 Data Structures (page 37).
7.2 Interfaces

All, the specific library adapters, the filters as well as the layouters basically work on an object instance of the class \textit{ONDEXGraph}. To provide the required generic architecture, interfaces are used to transform the \textit{ONDEXGraph} into a graph representation the different graph libraries use and to apply the results to the \textit{ONDEXGraph} object after finishing the specific calculations (see also Chapter 4.4 Generic Architecture, page 31 and Chapter 6.2 Architecture, page 42).

7.2.1 GeneralGraphLibraryAdapter

The description of the \textit{GeneralGraphLibraryAdapter} is illustrated in Figure 39 in the form of an UML class diagram.

![Figure 39 - Interface GeneralGraphLibraryAdapter](image_url)

To provide the library independence in the context of visualisation, the interface \textit{GeneralGraphLibraryAdapter} has to be implemented by all library adapters which have a description (\textit{description}) and a name (\textit{name}) and two variables used to determine if edges and vertices have to be labelled in the visualised graph or not (\textit{showEdgeLabels} and \textit{showVertexLabels}).

First, the adapter must have a reference to the \textit{ONDEXGraph} object that has to be visualised, which can be done with the \textit{setGraph} method.

If the \textit{paintGraph} method is called the \textit{ONDEXGraph} object will be translated into a library specific representation and will be painted into a \textit{JComponent} which can be displayed inside a \textit{JPanel} or \textit{JFrame}. 
Some data sources, such as KEGG (Kaneshisa and Goto, 2000), provide precomputed coordinations for some concepts which can be applied by using the `applyPrecomputedPositions` method. After using a filter some concepts or relations might have changed their visualisation specific values (color, size or visibility) so they have to be updated inside the `JComponent` (methods: `updateConcept` and `UpdateRelation`). Some graph libraries provide the facility of animated visualisations. To stop the animation a method (`stopAnimation`) can be called. When choosing a concept or a relation externally (e.g. from a list of microarray results) it might be advantageous to set the selected element via a method (`setMarked`) or to move the visible area to this element (`zoomTo`). Furthermore it is possible to zoom into the visualisation and to get the actual zoom value (methods: `zoom` and `getActualZoom`) and to set an unified `GraphMouseListener` which listens if an element is clicked or moved (`setGraphMouseListener`).

### 7.2.2 GeneralLayouter

The description of the `GeneralLayouter` is illustrated in Figure 40 in the form of an UML class diagram.

**Figure 40 - Abstract class GeneralLayouter**

To provide the library independence in the context of layouting and to provide the possibility to use self developed layout algorithms, the abstract class `GeneralLayouter` has to be implemented by all layouter classes. Every layouter has a name (`name`). Some layouters depend on special values (e.g. the size of the small and the big circle in the FastCircularLayouter, see page 45). To be able to handle them in a normalised way, in one list, the descriptions of the itemised values (the vector `value_descriptions`) have to be stored, so they can be presented to the user. To speed up the usage of the menu prompt the standard
values can be given in the vector `standardValues` in the same order as in the vector `values_descriptions`.

`SetGraph` and `getGraph` are preimplemented methods of this abstract class. They set and get the protected internal `ONDEXGraph` reference the layouter uses.

The method `setValues` assigns a list of values to the layouter whose values have to be in the same order as in the `value_descriptions` and are used to modify the layout in a user defined way.

`ApplyLayout` starts the layout process. If an external library is used to layout, this method transforms the graph into the library dependent data structure, starts the layout process and extracts the coordinates of the concepts and applies the results back to `ONDEXGraph` object.

### 7.2.3 GeneralFilter

It would be an advantage to use all developed filters in a general normalised way. Therefore, an abstract class is used which has to be implemented by all filter classes (Figure 41).

Every filter has a name (name) and might make use of some special values (like the MicroArrayFilter, see page 47). This mechanism works similar to the value assignment of the `GeneralLayouter` and includes the two vectors `value_descriptions` and `standardValues` and the method `setValues`. The `setGraph` and `getGraph` methods are likewise uniform to the `GeneralLayouter` and the method `applyFilter` is likewise similar to `applyLayout`.

With the help of `setGeneralGraphLibraryAdapter` a reference to the currently used graph library adapter can be assigned which is necessary to execute the `updateConcept` or `updateRelation` methods from the `GeneralGraphLibraryAdapter` (e.g. if concepts are hidden).
7.3 Example

To illustrate how to work with the OVTK application, how the graphical user interface appears and to illustrate how powerful the different filters and layouts are, a short example of a potential standard analysis workflow is given below.

After starting the OVTK a window appears that prompts for configuration information concerning the ONDEX Postgres database back-end (such as the location of the Postgres server and its port, the database name, the username and password and also, if some details for the different data sources have to be shown or if the sequences and structures have to be imported as well (Figure 42).

![Figure 42 - Start screen of the OVTK (request for configuration informations)](image)

After connecting to the Postgres database server and collecting some details of the available CVs (if the box was checked in the step before) the user has to select which concepts and relations from which data sources have to be imported into the local memory (Figure 43).

![Figure 43 - Assortment of data source to import from ONDEX to the OVTK](image)
When “Start import” has been clicked the OVTK runs some SQL statements on the ONDEX database server to extract the subgraph of ONDEX that has been chosen in the step before and the main windows then appear. In the top left of Figure 44 you can see a window visualising the present subgraph (at startup layouted in the FastCircularLayout, refer page 45). In the window in its right the main menu is displayed which gives the user the ability to initiate some actions on the graph (to zoom, apply filters and layouts, to change the visualising graph library, to load and save the graph, to stop the animation and to re-layout or to import some more CVs and to add them to the present graph). Below is a console window which shows some general details of the OVTK (welcome message, exceptions, load and save details etc.), some database specific details (connection status, present actions etc.) and some information concerning the graph (actual layout, number of visible concepts and relations before and after the applying of a filter etc.). The window in the bottom left shows some details of concepts or relations if they have been selected by the user.
As illustrated in Figure 44 the initial graph extracted from ONDEX can be very large and it is difficult to identify single elements but some relationships between the different concept classes can be seen. To bring the already visible graph in the context to given experimental result filters, such as the MicroArrayFilter (see page 47) can be used to highlight important elements and to decrease the graph size. Figure 45 shows the configuration window of the MicroArrayFilter which allows the user to choose the filename of the microarray file, the minimum and the maximum considered expression level and additionally a node size multiplier to distinguish relevant elements by increasing their size. Furthermore, the user can choose if a table of the extracted results from the microarray file with some additional details shall appear on the screen and if this filter has to be applied just to the visible subgraph or to the whole graph that is held in memory. Additionally, it has to be decided if the focus-on-highlighted-analysis has to be used (if set to false, the recursive function propagateNeighbours is not called, see the description of the filter algorithm at page 47) and which cut-off has to be used.

![MicroArrayFilter user interface](image)

**Figure 45 - The MicroArrayFilter user interface**

The filter makes use of the ImportWizard as a tool which allows the user to specify how the microarray file is to be parsed (interpreted): Tab-, space- or comma delimited and which rows represent which values (e.g. the spotids or the expression levels etc.), see Figure 46.
After having applied the filter and repainted the graph as illustrated in Figure 47 the graph size has been decreased (because of the neighbourhood propagation). The concepts in the graph which have a reference to a concept accession that has been found in the microarray file are highlighted according to their expression level. The table at the top right displays information to show if a spot-id has been found in the graph and if the element is up- or down regulated or if the expression level lies outside the minimum or maximum threshold.

The big yellow vertical bar in the figure are the relations between the concepts of the concept class “Treatment” and that of the class “Gene” which are some numerous as to appear as a solid line.
To provide a better overview, decrease the graph size and to show an example of the results of the HideConceptClassFilter (see page 53) we will apply it to the graph of Figure 47. The user interface of this filter is illustrated in Figure 48. The user can choose one concept class to hide all concepts of that type in the graph. In our example we choose the class “Treatment”.

**Figure 47 - Graph after applying the MicroArrayFilter (note the information table at the top right)**

**Figure 48 - The HideConceptClass filter user interface**
The resulting graph can be seen in Figure 49.

![Figure 49 - The graph after hiding the concepts that represent treatments](image)

Now the overview has been increased and we can look at the graph displayed with different layout algorithms. In Figure 50 the same graph as in Figure 49 is shown but using the YCircularLayout (Wiese et al., 2001) layout algorithm.

![Figure 50 - The graph layouted in CircularLayout](image)

It is obvious that there are many concepts with no connections (relations) to others so that no conclusions can be drawn about their relevance in the considered context. To hide them the RemoveAllUnconnectedFilter can be used. This sets all concepts with no visible incoming or outgoing relations to invisible and leads to a relatively small and clearly arranged graph with a focus on the elements extracted from the microarray file and some relevant elements in their
neighbourhood that have been drawn from the databases imported from ONDEX (see Figure 51).

In addition, Figure 51 shows the graph visualised using the anti-aliasing feature of the yFiles (Wiese et al., 2001) and with labelled concepts. After selecting a concept (by clicking on it), the details frame below the graph frame gives some information about the tagged concept (id, name, concept accessions, concept class, incoming and outgoing relations etc.).

Figure 51 - Graph applying the RemoveAllUnconnectedFilter (top pane). Details frame for selected elements (bottom pane).

Figure 52 illustrates that incoming and outgoing relations to or from other concepts are highlighted and additionally described in the details frame. To follow a graph linkage (a relation) from one concept to another it is not necessary to do this by tracing the route in the graph window. An alternative is provided by clicking on the description of the relation in the details window which causes the zoomTo function of the GraphLibraryAdapter (see Chapter 7.2.1 GeneralGraphLibraryAdapter, page 58) to be called and the specific concept be selected in the graph layout.
Figure 52 - Path from a down-regulated gene over a ‘protein’ to a ‘protein complex’ (hierarchical layout)

The hierarchical layout where all concepts of different concept classes are arranged in long vertical linear alignment can help to increase the clarity of the overview and to identify paths between elements of interest in a specific context (see Figure 52).

7.4 Transpath Parser

In the framework of the data integration process of the ONDEX data warehousing system a Biobase Transpath (Wingender, 2004) parser was implemented as part of this project. Transpath is a signal transduction database. It is based on flatfiles of details of molecules, reactions and genes. Figure 53 illustrates an example of a typical entry in the molecule flatfile. Every entry in these flatfiles is introduced and terminated with a “/” record that signals the begin and the end of a single dataset. Values flanked by “<” and “>” are identifiers which link to other Transpath entries (either in the same flatfile or in another one) or to external data sources (such as Biobase Transfac (Wingender, 2004), KEGG (Kaneshisa and Goto, 2000) etc.).
Figure 53 - Example of a typical entry in a Transpath flatfile which contains information about molecules

Every line starts with a two capital letter tag which determine the type of the attribute in the line. The following table (Table 5) gives an overview of the most important type identifiers. The original and complete descriptions can be found in the Biobase Transpath documentations at http://www.biobase.de/biobase/transpath/3.4_demo/doc/index.html.

Table 5 - Line type identifiers and their description for aTranspath molecules flatfile

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>Accession number</td>
</tr>
<tr>
<td>AS</td>
<td>Secondary accession numbers</td>
</tr>
<tr>
<td>NA</td>
<td>Name</td>
</tr>
<tr>
<td>SY</td>
<td>Synonyms</td>
</tr>
<tr>
<td>TY</td>
<td>Type (family, group, complex etc.)</td>
</tr>
<tr>
<td>DR</td>
<td>External database hyperlink</td>
</tr>
<tr>
<td>CC</td>
<td>Comments</td>
</tr>
<tr>
<td>CX</td>
<td>Complex the molecule is engaged in</td>
</tr>
<tr>
<td>XA</td>
<td>Outgoing reaction</td>
</tr>
<tr>
<td>XB</td>
<td>Incoming reaction</td>
</tr>
<tr>
<td>XC</td>
<td>Reaction catalyzed by the molecule</td>
</tr>
<tr>
<td>XI</td>
<td>Reaction inhibited by the molecule</td>
</tr>
<tr>
<td>RN</td>
<td>Reference publication</td>
</tr>
<tr>
<td>ID</td>
<td>Description</td>
</tr>
<tr>
<td>----</td>
<td>----------------------</td>
</tr>
<tr>
<td>GE</td>
<td>Encoding gene</td>
</tr>
<tr>
<td>OS</td>
<td>Species</td>
</tr>
<tr>
<td>SQ</td>
<td>Sequence (nucleotide or aminoacid)</td>
</tr>
<tr>
<td>ST</td>
<td>Complex or modified form of ...</td>
</tr>
</tbody>
</table>

During parsing, all flatfiles are read and all entries are imported into objects of suitable types which are derived from the classes Molecule, Reaction and Gene. Afterwards they are inserted into three hashtables (for the molecules, the encoding genes and the reactions). For example if a molecule has an encoding gene the object of type Molecule contains a reference to the according object of type Gene. At the end of this process all datasets are in the local memory.

The next step is to create tab delimited files for every table in the ONDEX Postgres database (refer page 18) such as Concept, Relation, GDS, Concept_Name, Concept_Acc, Structure and Sequence. For example every molecule, reaction and gene will be inserted in the file for the table Concept with the accession number (AS) as unique identifier. If, for example, a molecule has an encoding gene, then an entry in the file for the table Relation will be inserted with `element_of` set to the id of Transpath in the table CV (“TP”), `from_concept` set to the id of the molecule, `to_concept` set to the id of the gene and `of_type_FK` set to the identifier of the `Relation_type` “is_encoded_by”.

The generated tab delimited files will be imported using the “import” command of the relational Postgres database management system and the data is inserted into the database. Once the data from Transpath is extracted and imported into the ONDEX database back-end it is accessable from the OVTK.

### 7.5 System Requirements

The following sections of this chapter describe which software and which hardware is required to run the ONDEX Visualisation ToolKit (OVTK) successfully and what is recommended for optimal use of the whole the OVTK functionality.
7.5.1 Software

Because the OVTK is a Java program the Java virtual machine (the runtime environment) is needed to execute it.

<table>
<thead>
<tr>
<th>Software</th>
<th>License</th>
<th>Url</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun Java Runtime Environment</td>
<td>Sun License (free)</td>
<td><a href="http://java.sun.com/j2se/1.4/">http://java.sun.com/j2se/1.4/</a></td>
</tr>
</tbody>
</table>

The Java graph library JUNG (Java Universal Network/Graph Framework; O'Madadhain et al., 2004) version 1.5 and above, plus the yFiles (Wiese et al., 2001) are required to layout and visualise graphs and there are two Apache libraries and the Cern Colt library required for the JUNG library. The PostgreSQL JDBC database driver is used to connect to the PostgreSQL database of ONDEX to import data. All these Java archives have to be accessible for the OVTK over the Java Class Path.

<table>
<thead>
<tr>
<th>Software</th>
<th>License</th>
<th>Url</th>
</tr>
</thead>
<tbody>
<tr>
<td>JUNG 1.5</td>
<td>Open Source</td>
<td><a href="http://jung.sourceforge.net/">http://jung.sourceforge.net/</a></td>
</tr>
<tr>
<td>Apache Jakarta Commons Collections 3.1</td>
<td>Apache License (free)</td>
<td><a href="http://jakarta.apache.org/commons/collections/">http://jakarta.apache.org/commons/collections/</a></td>
</tr>
<tr>
<td>Cern Colt Scientific Library 1.2.0</td>
<td>Open Source</td>
<td><a href="http://dsd.lbl.gov/~hoschek/colt/">http://dsd.lbl.gov/~hoschek/colt/</a></td>
</tr>
<tr>
<td>Apache Xerces 2</td>
<td>Apache License (free)</td>
<td><a href="http://xml.apache.org/xerces2-j/">http://xml.apache.org/xerces2-j/</a></td>
</tr>
<tr>
<td>PostgreSQL JDBC Driver</td>
<td>Open Source</td>
<td><a href="http://jung.sourceforge.net/">http://jung.sourceforge.net/</a></td>
</tr>
<tr>
<td>yFiles</td>
<td>Yworks License (commercial)</td>
<td><a href="http://www.yworks.com/">http://www.yworks.com/</a></td>
</tr>
</tbody>
</table>

The OVTK works without any data import from the ONDEX database but it is strongly recommended to use this feature and to install the ONDEX Suite and to let it run on a PostgreSQL 7.2 DBMS (database management system). The DBMS and ONDEX do not have to be necessarily installed locally (network access is required).

<table>
<thead>
<tr>
<th>Software</th>
<th>License</th>
<th>Url</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL 7.2</td>
<td>Open Source</td>
<td><a href="http://www.postgresql.org/download/">http://www.postgresql.org/download/</a></td>
</tr>
<tr>
<td>ONDEX</td>
<td>GPL</td>
<td><a href="http://sourceforge.net/projects/ondex/">http://sourceforge.net/projects/ondex/</a></td>
</tr>
</tbody>
</table>
7.5.2 Hardware

In general the OVTK is executable on every platform with a running Sun Java Runtime Environment.

<table>
<thead>
<tr>
<th>Table 9 - Hardware required</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>Intel compatible</td>
</tr>
</tbody>
</table>

But the hardware requirements are very high (especially the memory usage). For this reason using OVTK for large data collections on a machine with less than 3 GB of RAM is not recommended.

<table>
<thead>
<tr>
<th>Table 10 - Hardware recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>Intel compatible</td>
</tr>
</tbody>
</table>

Note: The hardware requirements presented here are for the state of this thesis as it was finished. Jan Taubert (a visiting student at Rothamsted Research) had a special look on it in the framework of his thesis and especially the memory usage has been decreased (for example the application case (refer Chapter 9. Application Case: Microarray Analysis, page 76) can now be handled with 1 Gigabyte memory).
8. Results

This diploma thesis has been concerned with the problems of the visualisation and analysis of information obtained from biological experimental results in the context of knowledge extracted from molecular biological databases represented as biological networks (see Chapter 1.3 Aims, page 10).

To understand the essential functionality and algorithms that have to be provided by software to solve these problems, a requirement analysis (see Chapter 4. Requirement Analysis, page 27) was undertaken. The core needs have been determined and these are summarised in the following list:

1. The ability to handle large graphs.
2. The possibility to apply diverse layout algorithms to graphs.
3. Graph visualisations.
4. A generic architecture to combine the advantages of different Java graph libraries.
5. The facility to apply several filters to graphs in any combination.
6. The use of real experimental results and the validation of the approach using real data

The OVTK has the ability to handle large graphs by the means of a well organised data structure and very fast layout algorithms. For example, the FastCircularLayout which can be applied on a graph with more than 83,000 concepts and 96,000 relations (see Figure 44, page 62) in less than one second on a desktop PC with a 2 Ghz AMD 32 Bit CPU and 2 GB RAM. This is very fast compared with the idle times measured for several other layout algorithms (refer Chapter 5. Graph Library Evaluation, page 33).

Graphs laid out by the OVTK can be visualised with the aid of different Java graph libraries including several features (e.g. antialiasing, node and edge labeling, colouring etc.). They can be explored with the help of the diverse possibilities, as described in Chapter 7. Implementation (page 61).

With the help of the OVTK a number of layout algorithms can be applied to the graph (for example the FastCircularLayout, Figure 47, CircularLayout, Figure 51, TreeLayout, Figure 54, HierarchicalLayout, Figure 52). Due to the generic architecture (Chapter 6.2 Architecture, page 42) the OVTK can make use of the potential layout algorithms that already exist and are
offered in different Java graph libraries and the OVTK can use self developed and implemented layout algorithms.

Filters (e.g. the MicorArrayFilter, page 47 or the HideConceptClassFilter, page 53 and some others such as the HideRelationTypeFilter or the RemoveAllUnconnectedFilter etc.) can be applied to the whole graph or just to the visible subgraph. What makes OVTK useful and powerful is the set of diverse filter mechanisms and algorithms and the ability to combine them and to display the results in several layouts and visualisation styles.

The requirement for the use of experimental results has been achieved within the limits of filters (e.g. the MicroArrayFilter, page 47, which finds linkages from the given experimental data to elements in the graph which has been extracted from ONDEX).

The validation of the approaches with real data has been accomplished with the use of ONDEX and the OVTK for the analysis of the results given by the experiment of Parani (Parani et al., 2004) and is described in Chapter 9. Application Case: Microarray Analysis (page 76).

Figure 54 - The OVTK meets all requirements mentioned in Chapter 4. Requirement Analysis (page 27). The graph shown here is the graph from Figure 51 (page 29) displayed in a YTreeLayout (Wiese et al., 2001)
Altogether the OVTK meets all requirements. Figure 54 illustrates once again how the OVTK graphical user interface appears. With the help of the OVTK it is possible to import subgraphs from the ONDEX Postgres database backend into the local memory, to load and to save the imported graph from/into a file from/to disk (Java standard object stream or XML format), to visualise, layout and filter the graph, to highlight special elements, to explore the graph and to navigate between its components easily.
9. Application Case: Microarray Analysis

To illustrate how the interoperation between the OVTK and the different ONDEX components can be used in a practical data interpretation task, a gene expression experiment was selected as a model.

This application has been carried out by Prof. P. Verrier at Rothamsted Research in cooperation with Jan Taubert who enhanced the OVTK. In the framework of his thesis he improved the graphical user interface (GUI) and used the potential of the generic architecture (refer Chapter 6.2 Architecture, page 42) for further developments of filter algorithms (subtree statistics filter and subtree filter, see below) to gain more information from the graph after the application of the MicroArrayFilter (Köhler et al., 2005).

9.1 Methods

The choice of the experiment was dictated by the availability of expression data, the quality of the experimental design and the use of a genome-wide array. To assess the added value of the ONDEX and OVTK analysis, a study reporting a relatively large number of differentially expressed genes with ‘unknown’ function was identified. The experiment selected was that of Parani (Parani et al., 2004) in which the Affymetrix full-genome Arabidopsis thaliana chip ATH1 (comprising 24,000 genes) was used to determine the expression levels of A. thaliana ecotype Columbia which had been irrigated with 0.1mM and 1.0mM of sodium nitrosulphide (SNP) after first bolting (28 days after planting). The leaves were harvested after 3 hours and plants were treated during the light period. SNP is a nitrous oxide (NO) donor and is used to stimulate plant NO response pathways. Three replicates were performed and a contra-indicative treatment with an NO-scavenger was also performed. The experiment was designed to identify novel genes involved in NO signalling which is known to be important in plant stress response. In their original study, Parani found by statistical analysis that from the 342 up-regulated and 80 down-regulated genes there were 126 novel genes with unknown functions. The majority of these differentially expressed genes were specific to NO treatments as the reverse trend was observed with the scavenging treatment.

In order to demonstrate the potential of the OVTK, the NO treatment (1.0mM) up- and down-regulated expression levels were used to explore the results from the experiments. An
ONDEX database was built by importing and integrating the following data sources: AraCyc (Mueller et al., 2003), MetaCyc (Krieger et al., 2004), KEGG (Kaneshisa and Goto, 2000), DRASTIC Insight (Newton et al., 2002), Transfac and Transpath (Wingender, 2004).

The integrated ontology generated from the concepts and relations in AraCyc, Transfac, Transpath and DRASTIC were loaded into the OVTk visualisation component. Loading the 200,000 concepts and 100,000 relations from the ONDEX database backend into the OVTk takes about 3 minutes. Then the concepts and relations are layed out using the FastCircularLayout as described above (see page 45). The layouting process takes less than a second, painting takes a few seconds. The microarray results were then imported, and the nodes sized and colored according to the expression level of the microarray results (see Figure 55).

Figure 55 - Visualisation after application of the MicroArrayFilter (see page 47) with the integrated data of the heterogeneous databases (AraCyc, Transfac, Transpath and DRASTIC). Concepts from several concept classes (transcription factors and genes) are highlighted according to their expression level in the microarray experiment. The layout was generated using the FastCircularLayout algorithm (see page 45).

The GUI has been improved by Jan Taubert (also note the legend at bottom right).
By applying the MicroArrayFilter (which took less than 10 seconds at a cutOff of 10, refer Chapter 6.3.2 MicroArrayFilter, page 47) the number of concepts displayed was reduced to 700 elements that were directly or indirectly linked to the microarray results. By using the semantic relation information displayed in the details panel, the user can apply various graph layouts, filters and selections as well as drag and zoom features to rapidly explore the data in the context of the concepts in the database.

9.2 Data Analysis and Visualisation

9.2.1 Key Pathways

In the next step, the subtree statistics filter (developed and implemented by Jan Taubert with the aid of the GeneralFilter, page 60) was applied to calculate a linkage table summarising all metabolic pathways with at least one differentially expressed gene according to the microarray experiments. This identified that 50PATHAC (lignin biosynthesis) and 70PATHAC (jasmonic acid biosynthesis) were the pathways showing greatest activity in terms of linked entities (refer Table 11).

<table>
<thead>
<tr>
<th>Pathway</th>
<th>Gene</th>
<th>Protein</th>
<th>Enzyme</th>
<th>EC</th>
<th>Reaction</th>
<th>Treatment</th>
</tr>
</thead>
<tbody>
<tr>
<td>102PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>146PATHAC</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>170PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>176PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>165PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>167PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>159PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>181PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>174PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>163PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>164PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>51PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>41PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>186PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>37PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>179PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>50PATHAC</td>
<td>6</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>44PATHAC</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
The lignin biosynthesis pathway and the results from OVTK are shown in Figure 56. In their original analysis Parani also noted that there was considerable activity in the lignin pathway. In the OVTK, the lignin pathway was identified solely from visualisation prompting us that this may be the most significant change. In order to follow-up this information, the subtree filter (from Jan Taubert) which allowed to narrow down to only the reactions which are associated with active genes and related entities was applied.

![Lignin Pathway Diagram](image)

Figure 56 - The lignin pathway in the OVTK showing only elements where the genes are up- or down-regulated. Note the equivalence relation between gene from the AraCyc gene concept and the DRASTIC treatment gene concept. The unlabelled nodes are reaction concepts that have not been named. The concepts at the bottom are DRASTIC treatment types.

It is noteworthy that this pathway information was located rapidly and has been supplemented by important information from the DRASTIC database treatments concept to show that the three genes At1g09500, At1g72680 and At2g33590 are implicated in plant defense to abscissic acid (ABA), salt, drought and wound.

The second most important pathway (70PATHAC) is jasmonic acid biosynthesis which was not previously identified in the original analysis.
9.2.2 Relevance of the Microarray Chip

Not everything that is relevant is spotted on the microarray chip: reference to Figure 55 reveals that there are a number of transcription factors associated with the microarray genes of interest. On close inspection it was noticed that one gene had a number of transcription factors associated with it. By selecting a subtree filter on this gene to gain a clearer view (see Figure 57) it was observed that an equivalent gene was also found in DRASTIC treatments. Furthermore, it was concluded that the gene in question is not included in the microarray probe set and the OVTK has revealed this potentially interesting gene because one of the transcription factors associated with it has been observed in *Arabidopsis*.

![Figure 57](image)

*Figure 57 - Even though it was not spotted on the microarray chip, At5g52310 was implicated with six transcription factors (bottom right) in which one was found to be associated with a differentially expressed gene. In addition, the associated treatments from the DRASTIC database (bottom left) are shown through the equivalence relation and related to other differentially expressed genes (triangles).*
9.2.3 Key Stress Genes

By using the subtree filter statistics it was possible to identify the key stress genes which were also seen in the microarray results. Six treatments (from the DRASTIC database) that were linked to more than 25 genes in the analysed microarray data set were identified. Of these treatments, ABA, sodium chloride and drought all relate to water shortage stress and Yariv phenylglycoside treatment relates to cell stress. This possibly indicates that NO is active in the response to both drought and cell wall repair mechanisms which is a potentially novel observation not discussed in the original publication.

Table 12 - Genes in the microarray experiment associated with different types of stress

<table>
<thead>
<tr>
<th>Stress type</th>
<th>Number of genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>cold</td>
<td>25</td>
</tr>
<tr>
<td>ABA (abscisic acid)</td>
<td>27</td>
</tr>
<tr>
<td>flagellin 22 (flg22)</td>
<td>32</td>
</tr>
<tr>
<td>sodium chloride</td>
<td>33</td>
</tr>
<tr>
<td>drought / dehydration / wilt</td>
<td>34</td>
</tr>
<tr>
<td>Yariv phenylglycoside (beta-D-Glc)3</td>
<td>55</td>
</tr>
</tbody>
</table>

9.3 Gene Annotation

Parani reported a list of 126 ‘novel’ genes with ‘unknown’ functions. The associated accession numbers of these ‘unknowns’ were used as a BLAST query with stringent parameters to search against all sequences integrated in the ONDEX database component. A BLAST E-value of $10^{-8}$ was used which typically results in BLAST hits with a sequence identity >40%. This resulted in additional putative annotations being obtained for 6 of these 8 ‘unknowns’. The full list of all ‘unknowns’ was then presented to the OVTK system to display known relationships. As a result of this simple operation, functional annotations could be assigned to about another 40 genes. These new functional annotations can be considered high confidence because they are supported by experimental evidence stored in the DRASTIC databases. The annotations are included as supplementary material (see Appendix, Chapter 11.2 Gene annotations, page 97).
10. Discussion

The result of this thesis is a software framework (OVTK) that has been developed to help biologists with the analysis and the interpretation of experimental results in the context of biological networks that have been extracted from several integrated data sources. The integration problem is accomplished with the aid of the data warehousing system ONDEX. The practical outcome of the integration process is a relational database backend that holds the integrated biological data sources in its memory stored with the aid of an ontology based datastructure. The ONDEX data warehousing process has to be executed at regular time intervalls to take care of changes in the distributed biological data sources. This process potentially takes a very long time to finish (days or up to weeks in worst case). The integrated ontologies from the ONDEX database backend are extended to visible graphs (Definition 13 and Definition 14, Chapter 6.1 Data Structures, page 39). These biological networks represented as graphs can be very large, depending on the amount of data imported and the cross-linkings between the disparate data sets. For this reason they are difficult to handle in several respects. The OVTK has been implemented in the object oriented programming language Java. An object oriented representation of visible graphs used for the storage and the visualisation has been developed and implemented. Because of the size of the extracted graphs and the usage of memory-inefficient Java-integrated standard data structures (such as Hashtables and Vectors) the memory requirements are exceedingly high and have to be decreased in future work. Due to the platform independence of Java, the graphical user interfaces (GUIs) designed with Java are relative inefficient. Unfortunately, this aspect can not be improved without loosing the platform (especially the operation system) independence of the OVTK. The development of the GUI has started in the framework of first test environments and was developed incrementally. For this reason it is not very user-friendly. To increase usability and utility for research biologists, the GUI has to be improved to provide the whole functionality of the OVTK to end-users. The biggest conceptual problem in this project was the very large graph size with more then 2.8 millions of concepts and over 4 millions of relations (refer Chapter 4.1 Large Graphs, page 27). This leads to very high user idle times during the layout, filter and visualisation process. Concerning the above mentioned GUI-inefficient methods of Java, the visualisation
cannot be improved significantly. However, the layout and filter techniques have potential for further optimisation.

Before starting the development and implementation of novel layout algorithms, existing Java graph libraries have been evaluated. Most of them offer pre-implmented layout algorithms that have been tested with the aid of one and the same graph. One can see (in Chapter 5. Graph Library Evaluation, page 33) that the potential of the evaluated libraries is very heterogenous. To profit from the advantages of various libraries, a generic architecture has been developed and implemented to standardise the transformation of the internal graph object into the specific internal data structures of the integrated graph libraries.

Indeed, the application of standard layouts (offered by several graph libraries) to graphs works fine if they are not too large. But up from a graph size of 20,000 concepts and 30,000 relations, the time required for layouting is not acceptable for user interactions. On account of this, the generic architecture has been used to develop and implement a very fast layout algorithm (the FastCircularLayouter) whose application to graphs with hundreds of thousands of concepts and relations takes less than a few seconds.

The current implementation of the OVTK handles graph visualisation and the calculations of graph layouts separately. This does have some disadvantages. For example, in iterative graph layout algorithms (termed incremental layouts; O'Madadhain et al., 2004) the calculation of graph layouts and the visualisation of the graphs are implemented as alternating steps: the result is an animation in which every step depends on the foregoing. At the moment the OVTK is has no ability to support these methods.

The generic architecture standardises and simplifies the development and implementation of filter algorithms to increase speed, decrease memory usage and to provide a visualisation with an increased clarity. It is essential to hide nodes and edges from the graph that do not play a role in a given context. Two exemplary filter algorithms that have been developed in this project were presented in Chapter 6.3 Algorithms (page 45).

Filters, such as the MicroArrayFilter (see Chapter 6.3.2 MicroArrayFilter, page 47) can be used to display a graph in the context of experimental results. This thesis concentrated on the handling of experimental results obtained from microarray experiments, but the functionality of the approach is analogue to the analysis of experimental results of other techniques (such as mass spectrometry or ion mobility spectrometry). The principle of the MicroArrayFilter is to flag, filter and highlight those concepts in a graph that are relevant in an experimental result. Further on, it sets the node size and color according to a specific quantification. Thus, the filter algorithm can be extended very easily to handle experimental results obtained from mass
spectrometry, ion mobility spectrometry or other experimental techniques which consist of pairs of data (one accession used for the identification of the according concept in the graph and a quantitative experimental result).

All things considered, the OVTK can be used for the visualisation and the analysis of biological networks represented as graphs. The graphs can be extracted from ONDEX which integrates various biological data sources. The visualisation, layouting and filtering is independent from a specific graph library. Own developed algorithms can be implemented. Furthermore, experimental results can be used during the filtering process so that, using the OVTK, biological networks can be analysed in the context of experimental results.

Compared with other solutions such as LinkBase (Verschelde et al., 2004) or Cytoscape (Shannon et al., 2003) this approach profits from the combination of data integration of biological data sources and the visualisation and analysis of the resulting biological networks.

As already mentioned, the data integration process is accomplished with the aid of parsers (such as the Transpath parser, refer Chapter 7.4 Transpath Parser, page 68) and advanced methods to link equivalent entries of different heterogeneous data sources. Currently, parsers for about 15 data sources have been developed and implemented at Rothamsted Research and at the University of Bielefeld to assist the integration of many biological databases into the ONDEX data warehouse.

One of the most “OVTK-like” and best established solutions is the open source software Cytoscape (see Chapter 3. Related Work, page 23). Cytoscape’s strategy is to display large molecular biological graphs to identify interesting subgraphs using, for example, microarray results to reduce the size of the graphs, providing an entry point for lower-level modelling efforts. Similar to the OVTK it is possible to import a biological network into Cytoscape which is then visualised as a graph. In contrast to Cytoscape, the OVTK is able to import more than one graph because of its close coupling to the ONDEX data warehouse. As mentioned, the imported graphs are aligned with the aid of mapping algorithms (refer Chapter 2.2 ONDEX, page 18) which results in one large graph that represents all imported data sources.

Like the OVTK, Cytoscape provides the ability to apply filter algorithms to the visualised graph. These filter algorithms profit from typed nodes and edges. The node and edge types are stored in a hierarchical structure which can be extracted from Gene Ontology database (GOC, 2001). Gene Ontology stores a controlled vocabulary for the description of corresponding cellular components, molecular functions and biological processes. Currently, these data must be externally parsed. Using ONDEX and OVTK, this step is not necessary. The hierarchical structure (concept classes and relation types) is already stored in the ONDEX database.
backend and is used for example by the HideConceptClassFilter (refer Chapter 6.3.3 HideConceptClassFilter, page 53). The OVTK is able to filter a graph using the complete hierarchy. Furthermore, OVTK provides a generic architecture (refer Chapter 6.2 Architecture, page 42). In Cytoscape, the application of different layout algorithms and visualizations is supported using exclusively the yFiles and GINY graph libraries (Wiese et al., 2001).

The latest version of Cytoscape (2.1) is able to support large graphs consisting of more than hundred of thousands of elements. This was tested by loading the graph from the application case (Chapter 9. Application Case: Microarray Analysis, page 76) into Cytoscape. Whereas the import of the graph worked successfully, the creating of a view (which is the first step of data analysis and visualization in Cytoscape) has been interrupted after waiting more than 1 hour (Köhler et al., 2005).

Another approach (LinkBase) to integrate data sources into a database backend has been given by Verschelde (Verschelde et al., 2004). It helps to extract data from heterogeneous biomedical data sources. The principle is the same as in ONDEX: mapping procedures create additional linkages between concepts or relations of the imported data sources. In contrast to ONDEX no sequence data are used for the mapping process. Another advantage of ONDEX and the OVTK when compared to LinkBase, are the graph visualization and analysis methods.
11. Outlook

Naturally there are some more things which can be done. First of all a more user-friendly GUI (Graphical User Interface) of the OVTK could be developed and implemented. That would help to increase the usability for biologists. Jan Taubert (a visiting student at Rothamsted Research) already started the enhancement of the GUI (refer Chapter 9. Application Case: Microarray Analysis, especially Figure 55, page 77).

The next point is the development and implementation of more filter algorithms. In general, the MicroArrayFilter (page 47) is able to handle experimental results obtained from every experimental technique (with very simple modifications) which consist of pairs of data in form of one unique identifier and one quantification level. Jan Taubert implemented two additional filters used in the application case (Chapter 9. Application Case: Microarray Analysis, page 76).

Due to the generic architecture, the design and implementation of new filter and layout algorithms on the one hand and the binding of the OVTK to other visualising graph libraries on the other hand is relative simple. Due to this, the next step should be to provide support for more graph libraries in the OVTK, such as Tom Sawyer (TomSawyer, 2005). After finishing the graph library evaluation and during the implementation phase of the OVTK, the implementation of especially one very interesting graph library has been improved: WilmaScope (Dwyer and Eckersley, 2003). WilmaScope is a Java application which creates real time three-dimensional animations and layouts of graphs. The development of a GeneralGraphLibraryAdapter (refer Chapter 7.2.1 GeneralGraphLibraryAdapter, page 58) is of particular interest to optimise the clarity of the overview of large graphs. A disadvantage of WilmaScope is the platform dependency, since it uses an operation system dependent Java3D library.

The highest potential to increase the clarity of the overview of graphs might lie in the development of additional layout algorithms. In this case, the design of layouts with a more biological or experiment-oriented (and less pure graph-based) background is recommended. Standard layout algorithms can not make conventional textbook-like drawings.

Li and Kurata (Li and Kurata, 2005) developed a force-directed grid layout algorithm. This algorithm can be applied on metabolic pathways where metabolites are nodes and enzymes are edges (in contrast to this project where enzymes are nodes as well). Li and Kurata proposed a new algorithms where a graph is treated as a system of interacting nodes which are placed on a square grid. The nodes interact according to a specified cost function, which is
designed based on the topological structure of the network. In such a system, closely related nodes attract each other and remotely related nodes repulse each other. This facilitates the forming of cluster structures as illustrated in Figure 58.

Figure 58 - A grid layout of the yeast cell cycle regulatory network. The rounded rectangles are proteins, the rectangles metabolites, the parallelograms mRNAs and the ovals represent events. The solid black circles are complexes or modified molecules. The arrows indicate various reactions or regulations. Functional modules are marked by color blocks (Li and Kurata, 2005).

Another approach developed in this thesis but not implemented yet, would be a force-directed layout algorithm. Imagine an iterative layouter which implements two-dimensional force-induced movements of nodes. Connected nodes of the same colour are affected by forces of attraction and all other nodes are influenced by forces of repulsion (see Figure 59). These forces can be set fixed or in relation to a quantification level given by an experimental result. Green nodes represent flagged (highlighted) nodes while blue nodes represent the remaining. To avert a direct contact of nodes a minimum distance should be set. Additionally, a central gravitation field placed in the middle of the drawing area could be applied which attracts the flagged (green) nodes.
When applying these forces to the nodes of a graph and after running an iterative animation the graph should develop in a specific way: the green nodes accumulate in the centre of the graph and the remaining allocate around them (see Figure 60).

This method could also be used to filter the graph. Anticipating that all the green nodes are in the centre of the graph two circles can be drawn: one (smaller) circle directly around the green nodes and a bigger one around the first one. In the next step all elements outside this second circle can be hidden. This would lead to a decreased graph size without hiding green elements.

Figure 60 - Abstract example of a possibly resulting graph after the application of the CentralMicroArrayLayouter
Last not least, the memory efficiency of the OVTK has a high potential for improvement. Jan Taubert started working towards this end within his diploma thesis. By using the Cern Colt library for Java (http://dsd.lbl.gov/~hoschek/colt) he was able to decrease the memory usage drastically. The application case (refer Chapter 9. Application Case: Microarray Analysis, page 76) can now be processed with less than 1 GB of RAM.
11. Appendix

11.1 Bibliography


http://www.techfak.uni-bielefeld.de/~arueegg/graphdrawing.html


http://www.tomsawyer.com

[Verschelde et al., 2004] Verschelde, J.-L., Casella Dos Santos, M., Deray, T., Smith, B.,
Language Processing and Biomedical Data-mining. Journal of Integrative

L., Pontius, J. U., Schuler, G. D., Schriml, L. M., Sequeira, E., Tatusova, T. A. and

and Automatic Layout of Graphs. Proceedings of the 9th International Symposium on


different microarray data analysis programs and description of a database for
microarray data management. DNA Cell Biology, 23(10): 643-651.
11.2 Gene annotations

Supplement to Chapter 9.3 Gene Annotation (page 81)

<table>
<thead>
<tr>
<th>Accession</th>
<th>Hit accession</th>
<th>Identity</th>
<th>E value</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT1G05575</td>
<td>DRA_4401</td>
<td></td>
<td></td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
</tr>
<tr>
<td>AT1G19020</td>
<td>DRA_4362</td>
<td></td>
<td></td>
<td>stress response to low oxygen</td>
</tr>
<tr>
<td>AT1G19020</td>
<td>DRA_4362</td>
<td></td>
<td></td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
</tr>
<tr>
<td>AT1G19180</td>
<td>DRA_4363</td>
<td></td>
<td></td>
<td>stress response to drought / dehydration / wilt</td>
</tr>
<tr>
<td>AT1G19180</td>
<td>DRA_4363</td>
<td></td>
<td></td>
<td>stress response to ABA (abscisic acid)</td>
</tr>
<tr>
<td>AT1G19180</td>
<td>DRA_4363</td>
<td></td>
<td></td>
<td>stress response to sodium chloride</td>
</tr>
<tr>
<td>AT1G19180</td>
<td>DRA_4363</td>
<td></td>
<td></td>
<td>stress response to Pseudomonas syringae pv tomato</td>
</tr>
<tr>
<td>AT1G19180</td>
<td>DRA_4363</td>
<td></td>
<td></td>
<td>stress response to Peronospora parasitica</td>
</tr>
<tr>
<td>AT1G19180</td>
<td>DRA_4363</td>
<td></td>
<td></td>
<td>stress response to ethylene</td>
</tr>
<tr>
<td>AT1G19180</td>
<td>DRA_4363</td>
<td></td>
<td></td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
</tr>
<tr>
<td>AT1G21520</td>
<td>DRA_4699</td>
<td></td>
<td></td>
<td>stress response to sodium chloride</td>
</tr>
<tr>
<td>AT1G21550</td>
<td>TP_MO0000018780</td>
<td>34,44</td>
<td>1.2E-12</td>
<td>CaM(h)</td>
</tr>
<tr>
<td>AT1G21550</td>
<td>TP_MO0000032590</td>
<td>34,44</td>
<td>1.2E-12</td>
<td>CaM (r)</td>
</tr>
<tr>
<td>AT1G21550</td>
<td>TP_MO0000034799</td>
<td>34,44</td>
<td>1.2E-12</td>
<td>Calcium-binding</td>
</tr>
<tr>
<td>AT1G21550</td>
<td>TP_MO0000034798</td>
<td>33,77</td>
<td>5.8E-12</td>
<td>CaM (Calmodulin)</td>
</tr>
<tr>
<td>AT1G21550</td>
<td>TP_MO0000035556</td>
<td>24,69</td>
<td>8.1E-06</td>
<td>CnB2(m); calcineurin B subunit isoform 2; protein phosphatase 2B regulatory subunit 2</td>
</tr>
<tr>
<td>AT1G21550</td>
<td>DRA_608</td>
<td></td>
<td></td>
<td>stress response to salicylic acid</td>
</tr>
<tr>
<td>AT1G21550</td>
<td>DRA_608</td>
<td></td>
<td></td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
</tr>
<tr>
<td>AT1G23710</td>
<td>DRA_4739</td>
<td></td>
<td></td>
<td>stress response to turnip mosaic potyvirus (TuMV)</td>
</tr>
<tr>
<td>AT1G23710</td>
<td>DRA_4739</td>
<td></td>
<td></td>
<td>stress response to cucumber mosaic virus (CMV)</td>
</tr>
<tr>
<td>AT1G23710</td>
<td>DRA_4739</td>
<td></td>
<td></td>
<td>stress response to potato virus X potexvirus (PVX)</td>
</tr>
<tr>
<td>AT1G23710</td>
<td>DRA_4739</td>
<td></td>
<td></td>
<td>stress response to oilseed rape tobamovirus (ORMV)</td>
</tr>
<tr>
<td>AT1G23710</td>
<td>DRA_4739</td>
<td></td>
<td></td>
<td>stress response to turnip vein clearing tobamovirus (TVCV)</td>
</tr>
<tr>
<td>AT1G23710</td>
<td>DRA_4739</td>
<td></td>
<td></td>
<td>stress response to cold</td>
</tr>
<tr>
<td>AT1G67810</td>
<td>DRA_1447</td>
<td></td>
<td></td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
</tr>
<tr>
<td>AT1G67920</td>
<td>DRA_4371</td>
<td></td>
<td></td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
</tr>
<tr>
<td>AT1G68620</td>
<td>BR_Prot_Q97TVW1</td>
<td>37,68</td>
<td>5.8E-12</td>
<td>triacylglycerol lipase</td>
</tr>
<tr>
<td>AT1G68620</td>
<td>BR_Prot_Q60123</td>
<td>36,13</td>
<td>1.2E-09</td>
<td>carboxylesterase</td>
</tr>
<tr>
<td>AT1G68620</td>
<td>BR_Prot_CAF19872</td>
<td>34,62</td>
<td>7.3E-07</td>
<td>triacylglycerol lipase</td>
</tr>
<tr>
<td>AT1G68620</td>
<td>BR_Prot_Q9TUK0</td>
<td>33,96</td>
<td>1.5E-07</td>
<td>triacylglycerol lipase</td>
</tr>
<tr>
<td>AT1G68620</td>
<td>BR_Prot_Q8NKS0</td>
<td>33,49</td>
<td>2.1E-17</td>
<td>carboxylesterase</td>
</tr>
<tr>
<td>AT1G68620</td>
<td>BR_Prot_Q9X6Z3</td>
<td>31,34</td>
<td>3.6E-06</td>
<td>triacylglycerol lipase</td>
</tr>
<tr>
<td>AT1G68620</td>
<td>BR_Prot_Q7P136</td>
<td>30,85</td>
<td>8.4E-11</td>
<td>carboxylesterase</td>
</tr>
<tr>
<td>AT1G68620</td>
<td>BR_Prot_O77540</td>
<td>30,3</td>
<td>6.2E-06</td>
<td>carboxylesterase</td>
</tr>
<tr>
<td>AT1G68620</td>
<td>BR_Prot_Q9L178</td>
<td>29,07</td>
<td>1.3E-06</td>
<td>carboxylesterase</td>
</tr>
<tr>
<td>AT1G68620</td>
<td>BR_Prot_Q97VT8</td>
<td>27,83</td>
<td>1.5E-12</td>
<td>triacylglycerol lipase</td>
</tr>
<tr>
<td>AT1G68620</td>
<td>DRA_694</td>
<td></td>
<td></td>
<td>stress response to ABA (abscisic acid)</td>
</tr>
<tr>
<td>AT1G68620</td>
<td>DRA_694</td>
<td></td>
<td></td>
<td>stress response to sodium chloride</td>
</tr>
<tr>
<td>AT1G68620</td>
<td>DRA_694</td>
<td></td>
<td></td>
<td>stress response to drought / dehydration / wilt</td>
</tr>
<tr>
<td>AT1G76600</td>
<td>DRA_4456</td>
<td></td>
<td></td>
<td>stress response to low oxygen</td>
</tr>
<tr>
<td>Gene ID</td>
<td>Accession</td>
<td>Stress Response</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT1G76600</td>
<td>DRA_4456</td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT1G78410</td>
<td>DRA_4450</td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT1G79270</td>
<td>DRA_4411</td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G10940</td>
<td>DRA_3056</td>
<td>stress response to low oxygen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G10940</td>
<td>DRA_3056</td>
<td>stress response to drought / dehydration / wilt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G10940</td>
<td>DRA_3056</td>
<td>stress response to sodium chloride</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G10940</td>
<td>DRA_3056</td>
<td>stress response to chitin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G10940</td>
<td>DRA_3056</td>
<td>stress response to BTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G10940</td>
<td>DRA_3056</td>
<td>stress response to Peronospora parasitica</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G10940</td>
<td>DRA_3056</td>
<td>stress response to salicylic acid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G10940</td>
<td>DRA_3056</td>
<td>stress response to ethylene</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G17500</td>
<td>DRA_474</td>
<td>stress response to low oxygen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G17500</td>
<td>DRA_474</td>
<td>stress response to ethylene</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G20870</td>
<td>DRA_730</td>
<td>stress response to ethylene</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G22860</td>
<td>DRA_2917</td>
<td>stress response to potato virus X potexvirus (PVX)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G22860</td>
<td>DRA_2917</td>
<td>stress response to turnip mosaic potyvirus (TuMV)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G22860</td>
<td>DRA_2917</td>
<td>stress response to cucumber mosaic virus (CMV)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G22860</td>
<td>DRA_2917</td>
<td>stress response to oilseed rape tobamovirus (ORMV)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G22860</td>
<td>DRA_2917</td>
<td>stress response to turnip vein clearing tobamovirus (TVCV)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G22860</td>
<td>DRA_2917</td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G23170</td>
<td>DRA_4965</td>
<td>stress response to IAA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G24100</td>
<td>DRA_4586</td>
<td>stress response to sodium chloride</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G24800</td>
<td>DRA_4447</td>
<td>stress response to cold</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G28400</td>
<td>DRA_4447</td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G36220</td>
<td>DRA_4477</td>
<td>stress response to IAA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G36220</td>
<td>DRA_4477</td>
<td>stress response to cold</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G36220</td>
<td>DRA_4477</td>
<td>stress response to low oxygen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G36220</td>
<td>DRA_4477</td>
<td>stress response to Peronospora parasitica</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G36220</td>
<td>DRA_4477</td>
<td>stress response to BTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G36220</td>
<td>DRA_4477</td>
<td>stress response to brassinosteroid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G36220</td>
<td>DRA_4477</td>
<td>stress response to hydrogen peroxide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G39650</td>
<td>DRA_4455</td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT2G41730</td>
<td>DRA_4861</td>
<td>stress response to low oxygen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT3G02840</td>
<td>DRA_4452</td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT3G02840</td>
<td>TP_T05491</td>
<td>44.22 3.5E-72 strong elicitor-induced increase in the ELI17 mRNA levels was detectable as early as 5 min after addition of elicitor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT3G02840</td>
<td>DRA_4429</td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT3G14200</td>
<td>TP_MO000022544</td>
<td>41.98 5.2E-10 inhibitors; intracellular inhibitors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT3G14200</td>
<td>TP_MO000022537</td>
<td>41.98 5.2E-10  p58(b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT3G14200</td>
<td>TP_MO000038491</td>
<td>41.79 1.1E-07 Tid-1(h); chaperones; Hsp; Tid-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT3G14200</td>
<td>TP_MO000034990</td>
<td>40.58 2.6E-09 DnaJ (Hsp40) homolog, subfamily B, member 5; Dnajb5; heat shock protein cognate 40; heat shock protein hsp40-3 ; Hsc40; Hsp40-3.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT3G14200</td>
<td>TP_MO000021654</td>
<td>40.58 6.4E-08 Heat shock protein; co-factors; Hsp40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT3G19660</td>
<td>DRA_4421</td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT3G25780</td>
<td>DRA_4427</td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT3G59900</td>
<td>DRA_4428</td>
<td>stress response to ethylene</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G02940</td>
<td>DRA_2627</td>
<td>stress response to oilseed rape tobamovirus (ORMV)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G02940</td>
<td>DRA_2627</td>
<td>stress response to cucumber mosaic virus (CMV)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G02940</td>
<td>DRA_2627</td>
<td>stress response to potato virus X potexvirus (PVX)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gene Accession</td>
<td>Description</td>
<td>P-Value</td>
<td>E-Value</td>
<td>Function</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>---------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>AT4G02940</td>
<td>DRA_2627</td>
<td>Stress response to turnip vein clearing tobamovirus (TVCV)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G02940</td>
<td>DRA_2627</td>
<td>Stress response to turnip mosaic potyvirus (TuMV)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G02940</td>
<td>DRA_2627</td>
<td>Stress response to ABA (abscisic acid)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G02940</td>
<td>DRA_2627</td>
<td>Stress response to heat stress</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q86AE2</td>
<td>Stress response to senescence</td>
<td>38.05</td>
<td>8E-85</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_P54533</td>
<td>Alcohol dehydrogenase</td>
<td>26.64</td>
<td>3.3E-06</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_AAS42920</td>
<td>Dihydrolipoamide dehydrogenase</td>
<td>26.42</td>
<td>8.7E-07</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q819K6</td>
<td>Dihydrolipoamide dehydrogenase</td>
<td>26.42</td>
<td>5.1E-07</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q59299</td>
<td>Dihydrolipoamide dehydrogenase</td>
<td>26.24</td>
<td>3.9E-07</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q42711</td>
<td>Monodehydroascorbate reductase (NADH)</td>
<td>25.85</td>
<td>3E-07</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q97Z19</td>
<td>Dihydrolipoamide dehydrogenase</td>
<td>25.76</td>
<td>1.8E-07</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_P75393</td>
<td>Dihydrolipoamide dehydrogenase</td>
<td>25.24</td>
<td>8.7E-07</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q8CxEE6</td>
<td>Dihydrolipoamide dehydrogenase</td>
<td>25.21</td>
<td>5.6E-06</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q04829</td>
<td>Dihydrolipoamide dehydrogenase</td>
<td>24.6</td>
<td>4.3E-06</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q48419</td>
<td>Dihydrolipoamide dehydrogenase</td>
<td>24.07</td>
<td>5.6E-06</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q9Z773</td>
<td>Dihydrolipoamide dehydrogenase</td>
<td>23.39</td>
<td>6E-08</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_AAS14059</td>
<td>Dihydrolipoamide dehydrogenase</td>
<td>23.11</td>
<td>3.9E-07</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q8FSK7</td>
<td>Dihydrolipoamide dehydrogenase</td>
<td>22.92</td>
<td>5.6E-06</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_CAE28304</td>
<td>Dihydrolipoamide dehydrogenase</td>
<td>21.9</td>
<td>1.3E-07</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>DRA_2475</td>
<td>Stress response to salicylic acid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G05020</td>
<td>DRA_4519</td>
<td>Stress response to light</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_O22975</td>
<td>Epoxide hydrolase</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q9SU43</td>
<td>Epoxide hydrolase</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q81000</td>
<td>Triacylglycerol lipase</td>
<td>34.51</td>
<td>5E-11</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q818B9</td>
<td>Triacylglycerol lipase</td>
<td>33.64</td>
<td>3.3E-10</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q94K11</td>
<td>Epoxide hydrolase</td>
<td>33.3</td>
<td>1.1E-08</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_O23227</td>
<td>Epoxide hydrolase</td>
<td>33.3</td>
<td>1.1E-08</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q8MKP3</td>
<td>Epoxide hydrolase</td>
<td>31.74</td>
<td>1E-35</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q9V322</td>
<td>Epoxide hydrolase</td>
<td>31.34</td>
<td>1.1E-37</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q9DCD4</td>
<td>Epoxide hydrolase</td>
<td>28.87</td>
<td>2.9E-06</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q81D60</td>
<td>Epoxide hydrolase</td>
<td>26.35</td>
<td>1.7E-06</td>
</tr>
<tr>
<td>AT4G05020</td>
<td>BR_Prot_Q9BV23</td>
<td>Epoxide hydrolase</td>
<td>23.15</td>
<td>1.5E-07</td>
</tr>
<tr>
<td>AT4G24160</td>
<td>DRA_1973</td>
<td>Stress response to hydrogen peroxide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G24160</td>
<td>DRA_1973</td>
<td>Stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G27410</td>
<td>DRA_2470</td>
<td>Stress response to ABA (abscisic acid)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G27410</td>
<td>DRA_2470</td>
<td>Stress response to drought / dehydration / wilt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G27410</td>
<td>DRA_2470</td>
<td>Stress response to cold</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G27410</td>
<td>DRA_2470</td>
<td>Stress response to sodium chloride</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G27652</td>
<td>DRA_4357</td>
<td>Stress response to hydrogen peroxide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G27652</td>
<td>DRA_4357</td>
<td>Stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G27654</td>
<td>DRA_4295</td>
<td>Stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT4G33040</td>
<td>TP_MO000021425</td>
<td>Arsenate reductase; GLRX; glutaredoxin; glutaredoxin (thioltransferase).</td>
<td>31.52</td>
<td>2.5E-08</td>
</tr>
<tr>
<td>AT5G12010</td>
<td>DRA_4659</td>
<td>Stress response to sodium chloride</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT5G14730</td>
<td>DRA_4469</td>
<td>Stress response to low oxygen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT5G14730</td>
<td>DRA_4469</td>
<td>Stress response to hydrogen peroxide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT5G22060</td>
<td>TP_MO000034990</td>
<td>DnaJ (Hsp40) homolog, subfamily B, member 5; Dnajb5; heat shock protein cognate 40; heat shock protein hsp40-3 ; Hsc40; Hsp40-3.</td>
<td>59.42</td>
<td>1.5E-18</td>
</tr>
<tr>
<td>AT5G22060</td>
<td>TP_MO000021654</td>
<td>Heat shock protein 40kD</td>
<td>53.62</td>
<td>1.4E-16</td>
</tr>
<tr>
<td>AT5G22060</td>
<td>TP_MO000022544</td>
<td>41.77</td>
<td>2.4E-08</td>
<td>p58(h); inhibitors; intracellular inhibitors</td>
</tr>
<tr>
<td>AT5G22060</td>
<td>TP_MO000022537</td>
<td>41.77</td>
<td>2.4E-08</td>
<td>p58(b); inhibitors; intracellular inhibitors</td>
</tr>
<tr>
<td>AT5G22060</td>
<td>TP_MO000021654</td>
<td>29.71</td>
<td>3.3E-18</td>
<td>Hsp40(h)</td>
</tr>
<tr>
<td>AT5G22060</td>
<td>TP_MO000034990</td>
<td>28.45</td>
<td>3.9E-19</td>
<td>DnaJ (Hsp40) homolog, subfamily B, member 5; Dnajb5; heat shock protein cognate 40; heat shock protein hsp40-3; Hsc40; Hsp40-3.</td>
</tr>
<tr>
<td>AT5G22060</td>
<td>TP_MO000038491</td>
<td>26.59</td>
<td>1.1E-21</td>
<td>DnaJ (Hsp40) homolog</td>
</tr>
<tr>
<td>AT5G51550</td>
<td>DRA_2811</td>
<td>stress response to Heterodera schachtii</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT5G44590</td>
<td>DRA_4304</td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT5G61820</td>
<td>DRA_2431</td>
<td>stress response to ABA (abscisic acid)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT5G61820</td>
<td>DRA_2431</td>
<td>stress response to nitric oxide (NO donor NOR-3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT5G61820</td>
<td>DRA_2431</td>
<td>stress response to sodium chloride</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT5G61820</td>
<td>DRA_2431</td>
<td>stress response to hydrogen peroxide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT5G61820</td>
<td>DRA_2431</td>
<td>stress response to drought / dehydration / wilt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT5G62520</td>
<td>DRA_4324</td>
<td>stress response to Yariv phenylglycoside (beta-D-Glc)3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AT5G65300</td>
<td>DRA_4580</td>
<td>stress response to drought / dehydration / wilt</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11.3 Figures and Tables

11.3.1 Figures

FIGURE 1 - MICROARRAY EXAMPLE FROM FACULTY OF MEDICINE, IMPERIAL COLLEGE UK, GENETICS AND GENOMICS (FROGUEL, 2004) .............................................. 8
FIGURE 2 - MICROARRAY EXAMPLE OF "THE FULL YEAST GENOME ON A CHIP" FROM STANDFORD UNIVERSITY (BROWN, 2004) .............................................. 8
FIGURE 3 - LEGEND TO FIGURE 4 AND FIGURE 5 ........................................................ 10
FIGURE 4 - ABSTRACT EXAMPLE FOR A MAPPING BETWEEN 2 GRAPHS .................. 10
FIGURE 5 - ABSTRACT EXAMPLE OF A GRAPH AFTER FILTERING BY ORGANISM (MOUSE) AND AFTER THE COLORISATION AND THE SETTING OF THE NODE SIZE ACCORDING TO THE EXPRESSION LEVELS GIVEN BY AN EXPERIMENTAL RESULT ............................................................ 11
FIGURE 6 - CDNA MICROARRAY SCHEMA (DUGGAN ET AL., 1999) ......................... 13
FIGURE 7 - SCHEMATIC REPRESENTATION OF A MALDI-TOF MASS SPECTROMETRY (AEBERSOLD AND MANN, 2003) ......................................................... 15
FIGURE 8 - SCHEMATIC ILLUSTRATION OF AN ION MOBILITY SPECTROMETER (BAUMBACH ET AL., 2005) ................................................................. 16
FIGURE 9 - IMS-CHROMATOGRAM OF HUMAN EXHALED AIR. INLET: SINGLE SPECTRUM AT A FIXED RETENTION TIME SHOWING SIGNALS OF THE
MAJOR CONSTITUTIONS AMMONIA, ETHANOL AND ACETONE (BAUMBACH ET AL., 2005) ........................................................................................................................................................................... 17

FIGURE 10 - IMS-CHROMATOGRAM (LEFT) OF HUMAN EXHALED AIR OF A PATIENT SUFFERING CHRONIC OBSTRUCTIVE PULMONARY DISEASE (COPD) AND TWO TYPICAL SINGLE SPECTRA (RIGHT). (BAUMBACH ET AL., 2005)................................................................................................................................. 17

FIGURE 11 - IMS CHROMATOGRAM OF EMITTED AIR OF ESCHERICHIA COLI (LEFT) AND RELATED SINGLE SPECTRA (RIGHT). (BAUMBACH ET AL., 2005) .......................................................................................................................................... 18

FIGURE 12 - OVERVIEW: ONDEX ARCHITECTURE AND ITS COMPONENTS. THE BIG ARROW AT THE LEFT SIDE INDICATES A TYPICAL DATA INTEGRATION AND ANALYSIS WORKFLOW ................................................................................... 19


FIGURE 14 - SIMPLE UNDIRECTED ANDACYCLIC GRAPH........................................ 22

FIGURE 15 - SIMPLE DIRECTED AND CYCLIC GRAPH ........................................ 22

FIGURE 16 - SCHEMATIC OVERVIEW OF THE CYTOSCAPE CORE ARCHITECTURE. THE CYTOSCAPE WINDOW IS THE PRIMARY VISUAL AND PROGRAMMATIC INTERFACE TO CYTOSCAPE AND CONTAINS THE NETWORK GRAPH AND ATTRIBUTE DATA STRUCTURES. CORE METHODS THAT OPERATE ON THESE STRUCTURES ARE GRAPH EDITING, GRAPH LAYOUT, ATTRIBUTE-TO-VISUAL MAPPING, AND GRAPH FILTERING. ANNOTATIONS ARE AVAILABLE THROUGH A SEPARATE SERVER (SHANNON ET AL., 2003). ............................................................................................ 24

FIGURE 17 - LARGE GRAPH, EXTRACTED FROM BIOBASE TRANSPATH AND TRANSFAC (WINGENDER, 2004), LAYOUTED ORTHOGONAL WITH THE AID OF THE YFILES (WIESE ET AL., 2001) ........................................................................................................ 28

FIGURE 18 - A GRAPH ORGANICALLY LAYOUTED ..................................................... 29

FIGURE 19 - THE GRAPH FROM FIGURE 18 HIERARCHICALLY LAYOUTED ........ 29

FIGURE 20 - THE GRAPH FROM FIGURE 18 LAYOUTED IN A CIRCULAR LAYOUT .......................................................................................................................................... 30

FIGURE 21 - EXAMPLES OF DIFFERENT VISUALISATIONS BY DIFFERENT JAVA GRAPH LIBRARIES - TOP LEFT: BIOLAYOUT (GOLDOVSKY ET AL., 2004) -

FIGURE 22 - SIMPLE CYCLIC GRAPH (TOP), NOT FULLY CONNECTED GRAPH (MIDDLE), DIRECTED AND UNDIRECTED MIX GRAPH (BOTTOM)............. 37
FIGURE 23 - CONCEPT CLASSES ARE BUILT HIERARCHICALLY (TREE). ............. 38
FIGURE 24 - SIMPLE GRAPH WITH ATTACHED VALUES.............................................. 38
FIGURE 25 - DATA STRUCTURE (OVERVIEW) WITH UNDERLYING ONDEXER DIAGRAM (KÖHLER ET AL., 2004), SEE PAGE 20.......................... 41
FIGURE 26 - DATA STRUCTURE (OVERVIEW)................................................................... 41
FIGURE 27 - GENERIC ARCHITECTURE ............................................................................. 42
FIGURE 28 - FASTCIRCULARLAYOUT APPLIED ON A SUBGRAPH EXTRACTED FROM BIOBASE TRANSFAC AND TRANSPATH (WINGENDER, 2004), ARACYC (MUELLER ET AL., 2003) AND DRASTIC INSIGHT DATABASE (NEWTON ET AL., 2002)......................................................................................................................... 45
FIGURE 29 - AN ABSTRACT GRAPH................................................................................. 48
FIGURE 30 - STEP 1: SET CONCEPT SIZE AND COLOUR ACCORDING TO MICROARRAY RESULTS ................................................................................. 48
FIGURE 31 - STEP 2: HIDE ALL ELEMENTS EXCEPTING THE COLORISED CONCEPTS .................................................................................................................. 49
FIGURE 32 - STEP 3: NEIGHBOURHOOD PROPAGATION (FOR THE RED CONCEPT): SHOW ALL ELEMENTS IN THE NEIGHBOURHOOD OF THE RED CONCEPT WITH DISTANCE ≥ 2 AND STOP RECURSIVE PROCESS EARLIER IF A VISIBLE CONCEPT IS REACHED. NOTE THAT THE DISTANCES FROM THE RED CONCEPT ARE WRITTEN INTO THE CONCEPTS IN ITS NEIGHBOURHOOD................................................................. 49
FIGURE 33 - STEP 4: NEIGHBOURHOOD PROPAGATION FOR THE GREEN CONCEPT.................................................................................................................. 49
FIGURE 34 - RESULT: THE GRAPH FROM FIGURE 29 AFTER APPLICATION OF THE MICROARRAYFILTER. ....................................................................................... 50
FIGURE 35 - THERE ARE TWO PATHS (BROWN) BETWEEN THE GREEN AND THE RED NODE.................................................................................................................. 50
FASTCIRCULARLAYOUT BEFORE APPLYING THE HIDECONCEPTCLASSFILTER ................................................................. 53

FIGURE 37 - THE SAME GRAPH AFTER APPLYING THE HIDECONCEPTCLASSFILTER ON THE CONCEPT CLASS “TREATMENT”……53

FIGURE 38 - UML CLASS DIAGRAM OF THE INTERNAL GRAPH OBJECT OF THE OVTK............................................................................................................................... 56

FIGURE 39 - INTERFACE GENERALGRAPHLIBRARYADAPTER.......................................................... 58

FIGURE 40 - ABSTRACT CLASS GENERALLAYOUTER .................................................. 59

FIGURE 41 - ABSTRACT CLASS GENERALFILTER ......................................................... 60

FIGURE 42 - START SCREEN OF THE OVTK (REQUEST FOR CONFIGURATION INFORMATIONS) .......................................................................................................... 61

FIGURE 43 - ASSORTMENT OF DATA SOURCE TO IMPORT FROM ONDEX TO THE OVTK ....................................................................................................................................... 61

FIGURE 44 – OVTK START DISPLAY AFTER DATA IMPORT........................................... 62

FIGURE 45 - THE MICROARRAYFILTER USER INTERFACE ............................................. 63

FIGURE 46 – MICROARRAY DATA IMPORT BY MEANS OF THE IMPORTWIZARD ...................................................................................................................................... 64

FIGURE 47 - GRAPH AFTER APPLYING THE MICROARRAYFILTER (NOTE THE INFORMATION TABLE AT THE TOP RIGHT)........................................................................... 65

FIGURE 48 - THE HIDECONCEPTCLASS FILTER USER INTERFACE .............................. 65

FIGURE 49 - THE GRAPH AFTER HIDING THE CONCEPTS THAT REPRESENT TREATMENTS ............................................................................................................... 66

FIGURE 50 - THE GRAPH LAYOUTED IN CIRCULARLAYOUT ....................................... 66

FIGURE 51 - GRAPH APPLYING THE REMOVEALLUNCONNECTEDFILTER (TOP PANE). DETAILS FRAME FOR SELECTED ELEMENTS (BOTTOM PANE). ........ 67

FIGURE 52 - PATH FROM A DOWN-REGULATED GENE OVER A ‘PROTEIN’ TO A ‘PROTEIN COMPLEX’ (HIERARCHICAL LAYOUT) ................................................. 68

FIGURE 53 - EXAMPLE OF A TYPICAL ENTRY IN A TRANSPATH FLATFILE WHICH CONTAINS INFORMATION ABOUT MOLECULES ................................. 69

FIGURE 54 - THE OVTK MEETS ALL REQUIREMENTS MENTIONED IN CHAPTER 4. REQUIREMENT ANALYSIS (PAGE 27). THE GRAPH SHOWN HERE IS THE GRAPH FROM FIGURE 51 (PAGE 29) DISPLAYED IN A YTREELAYOUT (WIESE ET AL., 2001) ...................................................................................................................................... 74


FIGURE 57 - EVEN THOUGH IT WAS NOT SPOTTED ON THE MICROARRAY CHIP, AT5G52310 WAS IMPLICATED WITH SIX TRANSCRIPTION FACTORS (BOTTOM RIGHT) IN WHICH ONE WAS FOUND TO BE ASSOCIATED WITH A DIFFERENTIALLY EXPRESSED GENE. IN ADDITION, THE ASSOCIATED TREATMENTS FROM THE DRASTIC DATABASE (BOTTOM LEFT) ARE SHOWN THROUGH THE EQUIVALENCE RELATION AND RELATED TO OTHER DIFFERENTIALLY EXPRESSED GENES (TRIANGLES)............................... 80


FIGURE 59 - GRAVITY FORCES USED BY THE CENTRALMICROARRAYLAYOUTER (GREEN: HIGHLIGHTED NODES ACCORDING TO MICROARRAY RESULT, BLUE: OTHER NODES)................. 88
11.3.2 Tables

TABLE 1 - MICROARRAY RESULT FILE ............................................................................. 9
TABLE 2 - BIOLOGICAL DATABASES WHICH HAVE BEEN IMPORTED INTO
ONDEX AND THEIR SIZE (NUMBER OF CONCEPTS AND RELATIONS)............. 27
TABLE 3 - GRAPH LIBRARAY EVALUATION, ROWS: LIBRARIES, COLUMNS:
CRITERIAS (X = CRITERIA COMPLIED, “-” = CRITERIA NOT COMPLIED, N.D.
= NO DATA, OS = OPEN SOURCE, BSD = UNDER BSD LICENSE, GPL = UNDER
GPL LICENSE, LGPL = UNDER LGPL LICENSE, COM = COMMERCIAL)........ 35
TABLE 4 - GRAPH LIBRARIES AND THE IDLE TIMES WHILE LAYOUTING ........ 36
TABLE 5 - LINE TYPE IDENTIFIERS AND THEIR DESCRIPTION FOR ATRANSPATH
MOLECULES FLATFILE ............................................................................................. 69
TABLE 6 - NECESSARY SOFTWARE............................................................................ 71
TABLE 7 - REQUIRED JAVA ARCHIVES (JARS) ............................................................ 71
TABLE 8 - RECOMMENDED SOFTWARE...................................................................... 71
TABLE 9 - HARDWARE REQUIRED .............................................................................. 72
TABLE 10 - HARDWARE RECOMMENDED ................................................................ 72
TABLE 11 - GENERATED LINKAGE LIST FOR METABOLIC PATHWAYS SHOWING
THE NUMBER OF ENTITIES LINKED TO THE SPECIFIC PATHWAYS
(50PATHAC = LIGNIN BIOSYNTHESIS, 70PATHAC = JASMONIC ACID
BIOSYNTHESIS). ........................................................................................................ 78
TABLE 12 - GENES IN THE MICROARRAY EXPERIMENT ASSOCIATED WITH
DIFFERENT TYPES OF STRESS.............................................................................. 81
Erklärung

Hiermit versichere ich, dass ich meine Arbeit vollständig unter Anleitung verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entlehnt sind, durch die Angabe von Quellen als Entlehnungen kenntlich gemacht habe.

Ort, Datum  Unterschrift